# Linguagens e Máquinas:
# Uma Introdução aos Fundamentos da Computação

*Newton José Vieira*
Departamento de Ciência da Computação
Instituto de Ciências Exatas
Universidade Federal de Minas Gerais

Belo Horizonte, 04/05/2004

# Prefácio

> For me, the first challenge for computing science is to discover how to maintain order in a finite, but very large, discrete universe that is intricately intertwined. And a second, but not less important challenge is how to mould what you have achieved in solving the first problem, into a teachable discipline: it does not suffice to hone your own intellect (that will join you in your grave), you must teach others how to hone theirs. The more you concentrate on these two challenges, the clearer you will see that they are only two sides of the same coin: teaching yourself is discovering what is teachable.
>
> E.W. Dijkstra.

Este livro foi escrito para ser utilizado principalmente por alunos de cursos de graduação na área de Computação, como Ciência da Computação, Matemática Computacional, Engenharia de Computação, Sistemas de Informação, e outros. Ele pode também ser utilizado por alunos de pós-graduação, eventualmente complementado com outras referências que abordem alguns assuntos com maior profundidade ou que apresentem alguns tópicos não cobertos aqui. Além disso, ele pode ser útil para profissionais da área de Computação em geral, tanto para aqueles que desejem fazer uma revisão, quanto para aqueles que queiram ter um primeiro contato com a área.

Os textos que abordam os assuntos aqui apresentados podem ser divididos em três grupos. Há aqueles com uma abordagem bastante abstrata e formal, orientados para leitores com uma base matemática forte. Alguns deles se preocupam em apontar ou desenvolver algumas aplicações, mas o foco é a exploração das estruturas matemáticas em si. Em um segundo grupo, há aqueles com uma abordagem menos abstrata, com uma preocupação em explicar de forma intuitiva as estruturas matemáticas envolvidas, mas que ainda se valem de formalismos e de demonstrações de teoremas para assegurar um mínimo de concisão, precisão e rigor. Estes se dirigem a leitores que não precisam ter uma base matemática forte, mas que, ainda assim, precisam ter uma certa facilidade para lidar com objetos matemáticos. Finalmente, há aqueles que, além de priorizar uma explicação intuitiva das estruturas envolvidas, evita ao máximo o uso de formalismos e de demonstrações de teoremas. O objetivo principal de tais livros é atrair leitores sem a mínima propensão para lidar com formalismos e demonstrações de teoremas. Devido ao fato de evitar o uso de formalismo, tais textos costumam ser muito prolixos, com uma taxa muito baixa de conteúdo por página, além de terem, aqui e ali, problemas de precisão e até mesmo de clareza.

Com relação aos três grupos mencionados no parágrafo anterior, este livro pode ser classificado como estando no segundo grupo, mas, ainda que se beneficiando do formalismo matemático para efeitos de concisão e precisão, ele tem uma preocupação especial de apresentar todos os conceitos apelando para a intuição do leitor, principalmente aquele incluido no público alvo mencionado acima. Deve-se ressaltar ainda que aqueles que não tiverem interesse nas demonstrações de teoremas, ainda assim podem se beneficiar da leitura do livro, visto que poderão encontrar bastante material, como métodos, algoritmos, etc., que pode ser assimilado para uso em uma ampla gama de situações.

Os conceitos matemáticos necessários para uma melhor compreensão do assunto são revisados no Capítulo 1, a saber: os conceitos de conjuntos, relações, funções, conjuntos

enumeráveis, definições recursivas e grafos, além das principais técnicas para prova de teoremas, dentre as quais, ressaltando-se a de indução matemática. O leitor que já tiver estudado tais tópicos pode ir direto para o Capítulo 2. Por outro lado, para aquele que sinta necessidade de uma revisão, o conteúdo deste capítulo é suficiente como base para entendimento do resto do livro.

Como se pode depreender pelos títulos dos capítulos, o assunto foi desenvolvido a partir do conceito de *máquinas*. Esta opção, que não é a mais comum na literatura, foi escolhida por levar a uma abordagem mais intuitiva para o aluno médio de Computação, como constatado pelo autor a partir de uma experiência de mais de 12 anos lecionando uma disciplina com o conteúdo deste livro no curso de bacharelado em Ciência da Computação da Universidade Federal de Minas Gerais.

As Linguagens Formais e Autômatos, são, dentro da área de Teoria da Computação, das mais profícuas do ponto de vista prático. Ao contrário do que se pode imaginar à primeira vista, vários dos conceitos aqui estudados têm aplicações práticas, não apenas em ambientes complexos e sofisticados, mas também em ambientes relativamente simples e corriqueiros. Por exemplo, as máquinas de estado-finito, tema do Capítulo 2, podem ser usadas como ferramentas de modelagem em problemas os mais diversos, como, por exemplo, o da concepção de analisadores léxicos, confeção de algoritmos para busca em texto, projeto de máquinas para venda de produtos (jornais, refrigerantes, etc.), projeto do funcionamento de elevadores, etc.

O conteúdo do livro foi definido para ser lecionado em uma disciplina de um semestre. Dada esta limitação de prazo, e como o assunto é bastante extenso, optou-se por dar ênfase aos aspectos que têm uma maior contrapartida do ponto de vista prático. Assim, por exemplo, a parte relativa a máquinas de estado-finito, no Capítulo 2, recebeu um espaço maior, visto que as mesmas encontram aplicações em grande quantidade de situações, como ficou ressaltado acima. Já no Capítulo 3, que trata dos autômatos com pilha, é dada uma maior ênfase a gramáticas e conceitos correlatos, visto que estes têm maior aplicabilidade prática do que os autômatos com pilha propriamente (pelo menos em sua forma original). Na verdade, os autômatos costumam ser, na prática, obtidos a partir de gramáticas.

No Capítulo 4 são apresentadas as máquinas de Turing, no intuito de procurar dar ao leitor uma noção dos componentes fundamentais de uma máquina que faz *computação*. As máquinas que têm um poder computacional entre o de autômato com pilha e o de máquina de Turing, os denominados autômatos linearmente limitados, recebem um tratamento bastante superficial, por não terem muita importância do ponto de vista prático e não terem a relevância teórica das máquinas de Turing.

Finalmente, no Capítulo 5, que trata de decidibilidade, o foco é o de procurar dar ao leitor uma noção dos limites do conceito de computação. Isto é feito através da apresentação de alguns problemas de enunciado muito simples para os quais não existem soluções computacionais (algoritmos), dentre os quais o tradicional *problema da parada*.

O livro contém cerca de 380 exercícios formulados. Ao final de cada sessão de cada capítulo, são formulados exercícios para prática e consolidação dos conceitos estudados na sessão. E ao final de cada capítulo é ainda apresentada uma sessão de exercícios sobre todo o assunto coberto no capítulo. Tais exercícios diferem dos exercícios de final de sessão por terem, em geral, um nível de dificuldade um pouco maior e por não serem tão direcionados a determinados conceitos. Por exemplo, em um exercício de final de sessão,

normalmente é apontado o método, técnica ou algoritmo a ser utilizado para conseguir certo objetivo; já em um exercício de final de capítulo, o método, técnica ou algoritmo a ser utilizado não é apontado, devendo ser escolhido ou determinado pelo estudante.

Uma outra característica marcante deste texto é a existência de um capítulo com soluções de exercícios selecionados. Este é um capítulo importante, visto que dá aos alunos a oportunidade de conferir ou comparar soluções, além de dar uma oportunidade ao autor de abordar tópicos interessantes que não cabem naturalmente no texto normal.

Ao final de cada capítulo, após a seção de exercícios propostos, é apresentada uma bibliografia que contém a origem dos diversos conceitos, assim como sugestões para leituras complementares.

Do ponto de vista de conteúdo existem vários livros que cobrem os tópicos aqui discutidos. Destes, deve ser especialmente ressaltado o de Hopcroft e Ullman[HU79], fonte a partir da qual, não apenas a presente publicação, mas também a esmagadora maioria das outras, buscou inspiração e exemplo. Alguns outros bons textos, do ponto de vista didático, com abordagens alternativas à aqui utilizada são os de Hopcroft, Motwani e Ullman[HMU01] (que é a segunda edição do texto citado acima), Martin[Mar91], Linz[Lin97], Sipser[Sip97], Greenlaw e Hoover[GH98], Floyd e Beigel[FB94] e Kozen[Koz97].

# Agradecimentos

Vários professores e alunos da Universidade Federal de Minas Gerais e de outras instituições ajudaram na produção deste livro, tanto sugerindo melhorias na apresentação, quanto apontando erros em diversas versões preliminares. Sem dúvida nenhuma o texto ficou muito melhor a partir de tais sugestões e da correção dos erros apontados. Evidentemente, tais pessoas, tanto as que serão apontadas explicitamente a seguir, quanto as outras, não podem ser responsabilizadas pelas imperfeições e erros porventura ainda presentes.

Faço um agradecimento especial ao professor José Luis Braga, da Universidade Federal de Viçosa, por ter dado valiosíssimas sugestões a partir da leitura de uma primeira versão. Ao professor Marcos Alexandre Castilho, agradeço por ter usado versões preliminares deste texto na Universidade Federal do Paraná. Em especial, agradeço ao aluno do professor Marcos, Jonatan Schröeder, por ter encontrado um erro em um dos exemplos do Capítulo 3, e por ter apresentado uma solução elegante, prontamente adotada neste texto. Agradeço aos alunos de pós-graduação em Ciência da Computação da UFMG, Paulo Sérgio Silva Rodrigues, Camillo Jorge Santos Oliveira, Luiz Filipe Menezes Vieira, Marcos Augusto Menezes Vieira, Carlos Maurício Seródio Figueiredo e Vilar Fiúza da Câmara Neto por terem tido o trabalho de ler com atenção partes do texto e de anotar os erros encontrados. A estes dois últimos, Carlos Maurício e Vilar, em particular, agradeço por me apontarem uma redundância em um dos métodos do Capítulo 2.

Ao Departamento de Ciência da Computação da UFMG agradeço pelo ambiente propício e por fornecer parte dos recursos computacionais utilizados.

Newton José Vieira
04 de maio de 2003

# Conteúdo

# Capítulo 1

# Conceitos Preliminares

> In a way, math isn't the art of answering mathematical questions, it is the art of asking the right questions.
>
> Gregory Chaitin.

Neste capítulo serão revisados, de forma sucinta, os conceitos matemáticos necessários para o entendimento dos capítulos restantes.

Inicialmente, na Seção 1.1, será abordado o problema fundamental relativo ao uso dos computadores, qual seja, o da Representação, com o intuito de fazer transparecer a importância dos conceitos matemáticos a serem introduzidos, tanto no nível de modelagem quanto no de representação propriamente dito. Em seguida, na Seção 1.2, serão revisados superficialmente os principais conceitos da Lógica de Predicados, culminando com a apresentação de algumas técnicas de prova que devem ser assimiladas, não apenas por aqueles que devem ser capazes de provar teoremas, mas também por aqueles que devam ter a capacidade de ler e entender uma prova. As noções básicas que, juntamente com a Lógica de Predicados, constituem o alicerce da Matemática Discreta, em geral, e dos fundamentos da Computação, em particular, quais sejam, as noções de Conjuntos, Relações e Funções, serão introduzidas nas Seções 1.3, 1.4 e 1.5. O conceito de Conjunto Enumerável, que servirá como base para uma primeira argumentação com relação à existência de funções não computáveis, será visto na Seção 1.6. Ferramentas importantes para quem trabalha com conjuntos enumeráveis, o conceito de Definição Recursiva e a técnica de prova por Indução Matemática, serão apresentados em seguida, nas Seções 1.7 e 1.8. Os Grafos, estruturas matemáticas úteis para modelagem de problemas reais ou abstratos, e que permeia todas as áreas da Computação, tanto do ponto de vista teórico quanto prático, serão introduzidos na Seção 1.9. O conceito de Linguagem Formal, a partir do qual será desenvolvido todo o material do restante do texto, será desenvolvido na Seção 1.10. Para finalizar este capítulo preliminar, será apresentada a noção de Gramática, um dos formalismos mais utilizados para a definição de linguagens formais. Nestas duas últimas seções, os conceitos de Linguagem Formal e Gramática serão vistos de forma sucinta, nas suas formas mais gerais; nos capítulos seguintes, eles serão retomados em conexão com classes específicas de linguagens.

| *Entidade* | *Modelo matemático* | *Representação* |
|---|---|---|
| mês | número inteiro no intervalo $[1, 12]$ | um dos caracteres 1, 2, 3 4, 5, 6, 7, 8, 9, 0, A ou B |
| remuneração | número real positivo | número real na base 10 |
| presença | vetor de números, um para cada dia do mês | seqüência de números reais na base 10 |
| FP | relação | tabela de seqüências de símbolos c/ nome, salário, etc. |
| cálculo de FP | algoritmo | programa |

Figura 1.1: A matemática entre a entidade e a representação.

## 1.1 Representação

Quando se pretende resolver um problema por computador, uma tarefa importante é *representar* as entidades envolvidas, sejam elas concretas ou não. A representação de uma entidade na forma de um programa, na forma de entrada para um programa ou na forma de saída de um programa é, muitas vezes, constituída de *seqüências de símbolos*[1]. Considere, por exemplo, uma aplicação referente à folha de pagamento de uma empresa. A entidade correspondente ao processo de cálculo da folha de pagamento é representada por uma seqüência de símbolos em uma linguagem de programação, denominada programa; a entrada para tal programa é constituída de seqüências de símbolos representando vários tipos de entidades, como o mês em questão, ano, empregados, horas trabalhadas para cada empregado, etc.; a saída do programa é constituída de seqüências de símbolos representando os empregados, horas trabalhadas, remuneração, etc.

Muitas vezes, é útil considerar, entre uma entidade representada e a seqüência de símbolos que a representa, a existência de um terceiro elemento: *o modelo matemático* correspondente à entidade representada. A Figura 1.1 mostra alguns exemplos (lá FP abrevia folha de pagamento).

Na coluna *Representação* da Figura 1.1, o elemento fundamental é a *seqüência de símbolos*. Ela é utilizada na representação de qualquer tipo de entidade, de maneira a propiciar o processamento computacional que a envolve. Se não se considerar a concretização das seqüências de símbolos em um meio físico, isto é, se as mesmas forem consideradas apenas do ponto de vista lógico, elas também podem ser modeladas matematicamente. Em tal modelagem, em geral as seqüências de símbolos são consideradas como componentes do que se denomina *linguagens formais*. .

Uma boa parte deste texto versará sobre linguagens formais. Isto será importante, tanto para introduzir técnicas bem fundamentadas para construção de algoritmos para um amplo espectro de aplicações, quanto para caracterizar o conceito de computabilidade. Neste último aspecto, será visto, por exemplo, que existe uma infinidade de funções que *não* são computáveis; em particular, serão vistos problemas com enunciados bastante simples para os quais não existe algoritmo e, portanto, nem programa em qualquer linguagem de programação.

Para fazer este estudo das linguagens formais, assim como para considerar modelos

[1]Embora, cada vez mais se esteja usando recursos gráficos bi e tridimensionais, som, etc. De qualquer forma, em algum nível, mesmo tais entidades são representadas mediante seqüências de símbolos

alternativos para o conceito de computabilidade, são necessários alguns elementos de matemática, que serão revisados no restante deste capítulo. Tais elementos são também úteis na etapa intermediária de modelagem ilustrada na Figura 1.1.

A revisão começará na próxima seção com uma síntese dos principais elementos de Lógica Matemática utilizados na construção de provas de teoremas.

## 1.2 Prova de Teoremas

O objetivo de quem escreve uma prova é mostrar, sem deixar margens a dúvidas, que determinada afirmativa é verdadeira. Uma prova pode ser mais ou menos formal, e mais ou menos concisa, dependendo do tipo de leitor para o qual a ela é construída.

Assim, por exemplo, se a prova deve ser verificada via um programa de computador, ela deve ser absolutamente formal. Para isto, ela deve explicitar todas as hipóteses nas quais se baseia, assim como todos os passos de inferência utilizados, de forma que ela possa ser verificada mecanicamente. Com isto, a prova é, em geral, muito grande e bastante ilegível.

Por outro lado, se a prova é produzida para ser lida apenas por especialistas em determinado assunto, ela pode ser bem concisa, fazendo referência a resultados conhecidos pelos especialistas, sem prová-los de novo. Neste caso, a prova é informal, o que não significa que não seja escrita em um estilo que assegure um certo rigor e relativa clareza.

Entre os dois extremos apontados acima, existem as provas que buscam convencer pessoas que não são especialistas no assunto de que trata o resultado. Neste caso, a prova também não é formal, mas é mais prolixa, abordando aspectos que o não especialista não apreende com facilidade.

Em síntese, quando se escreve uma prova para pessoas lerem, ela deve ser informal, e seu estilo e nível de detalhe devem depender da audiência intencionada. Em geral, a prova é expressa utilizando-se uma língua natural (português, no nosso caso) intercalada com algum formalismo matemático. Mas o vocabulário empregado é normalmente bastante limitado, com o objetivo de evitar ambiguidades. Em particular, existem certas palavras e expressões que ocorrem com freqüência e que têm um significado padrão, intimamente relacionado com a própria conceituação de prova, como, por exemplo, "se ... então", "contradição", "portanto", etc.

Além de conter um vocabulário limitado, com determinados termos tendo um significado bem definido, uma prova é estruturada segundo uma ou mais *técnicas básicas de prova*. A seguir, será feito um apanhado conciso da terminologia envolvida na prova de teoremas e serão apresentadas as principais técnicas básicas de prova.

Dentre os termos utilizados em provas, destacam-se aqueles para os *conectivos lógicos*:

- negação: $\neg$, não;
- conjunção: $\wedge$, e;
- disjunção: $\vee$, ou;
- condicional: $\rightarrow$, se ... então;

| *Negação* | |
|---|---|
| $\alpha$ | $\neg\alpha$ |
| V | F |
| F | V |

| *Conjunção* | | |
|---|---|---|
| $\alpha$ | $\beta$ | $\alpha \wedge \beta$ |
| V | V | V |
| V | F | F |
| F | V | F |
| F | F | F |

| *Disjunção* | | |
|---|---|---|
| $\alpha$ | $\beta$ | $\alpha \vee \beta$ |
| V | V | V |
| V | F | V |
| F | V | V |
| F | F | F |

| *Condicional* | | |
|---|---|---|
| $\alpha$ | $\beta$ | $\alpha \rightarrow \beta$ |
| V | V | V |
| V | F | F |
| F | V | V |
| F | F | V |

| *Bicondicional* | | |
|---|---|---|
| $\alpha$ | $\beta$ | $\alpha \leftrightarrow \beta$ |
| V | V | V |
| V | F | F |
| F | V | F |
| F | F | V |

Figura 1.2: Tabelas da verdade para conectivos lógicos.

- bicondicional: $\leftrightarrow$, se, e somente se;
- quantificador universal: $\forall$, para todo;
- quantificador existencial: $\exists$, existe.

Para cada conectivo, estão mostradas duas notações. A primeira é a mais utilizada nos textos de Lógica Matemática modernos. A segunda consta de expressões em português que denotam os mesmos conectivos. A primeira notação será utilizada em grande parte desta seção, favorecendo concisão e clareza da apresentação. No restante do livro, serão mais utilizadas as expressões em português, como é mais usual.

Os significados dos cinco primeiros conectivos podem ser dados pelas denominadas *tabela da verdade* mostradas na Figura 1.2. Nestas tabelas, $\alpha$ e $\beta$ representam afirmativas quaisquer, sendo que cada uma destas afirmativas pode ser verdadeira (V) ou falsa (F), ou seja, os *valores-verdade* para $\alpha$ e $\beta$ podem ser V ou F. Para todas as combinações possíveis de V e F para $\alpha$ e $\beta$, cada tabela apresenta o valor resultante da composição via o conectivo respectivo.

**Exemplo 1** Sejam as seguintes afirmativas:

- $0 > 1$;
- *2 é um número par*;
- *2 é um número primo*;
- *todo número é um quadrado perfeito*.

Sabe-se que os valores-verdade de tais afirmativas são, respectivamente: F, V, V e F. Assim, a afirmativa "*2 é um número par* $\wedge$ *2 é um número primo*" é verdadeira, de acordo com a primeira linha da tabela da verdade para o conectivo "$\wedge$". Por outro lado, a afirmativa "$0 > 1 \wedge$ *2 é um número primo*" é falsa, de acordo com a terceira linha da

mesma tabela. Já a afirmativa "$0 > 1 \vee$ *2 é um número primo*" é verdadeira, de acordo com a terceira linha da tabela da verdade para o conectivo "$\vee$". A afirmativa "*2 é um número par* $\rightarrow$ *todo número é um quadrado perfeito*" é falsa, pela segunda linha da tabela para o conectivo "$\rightarrow$". □

O significado do conectivo "$\rightarrow$" pode parecer estranho à primeira vista, pois a expressão "se ... então" é utilizada no dia a dia com um sentido diferente. Uma maneira de explicar o significado exibido pela tabela da Figura 1.2, é dada pelo fato de que, em matemática, a expressão "se $\alpha$ e $\beta$ então $\alpha$" deve ser verdadeira para qualquer combinação de valores-verdade para $\alpha$ e $\beta$, como mostra a tabela:

| $\alpha$ | $\beta$ | $\alpha \wedge \beta$ | $(\alpha \wedge \beta) \rightarrow \alpha$ |
|---|---|---|---|
| V | V | V | V |
| V | F | F | V |
| F | V | F | V |
| F | F | F | V |

A primeira linha desta tabela diz que "$(\alpha \wedge \beta) \rightarrow \alpha$" deve ser verdadeira quando "$\alpha \wedge \beta$" e $\alpha$ são verdadeiras, justificando a primeira linha da tabela para o conectivo "$\rightarrow$" na Figura 1.2. As linhas restantes dizem que "$(\alpha \wedge \beta) \rightarrow \alpha$" deve ser verdadeira quando "$\alpha \wedge \beta$" é falsa, tanto no caso em que $\alpha$ é verdadeira, quanto no caso em que é falsa; isto justifica as duas últimas linhas da tabela para o conectivo "$\rightarrow$"[2].

Outros termos equivalentes a $\alpha \rightarrow \beta$, além de "se $\alpha$ então $\beta$", são: "$\alpha$ é *condição suficiente* para $\beta$", e "$\beta$ é *condição necessária* para $\alpha$". E "$\alpha$ é *condição necessária e suficiente* para $\beta$" quer dizer o mesmo que $\alpha \leftrightarrow \beta$.

Dizer que $\forall x P(x)$ é verdadeira é dizer que para todo membro $x$ do *universo de discurso* em consideração, a afirmativa $P(x)$ é verdadeira. Informalmente, $P(x)$ é qualquer afirmativa matemática envolvendo a variável $x$. Como mostrado em exemplos abaixo, $P(x)$ pode ser uma expressão em uma linguagem puramente matemática (formal) ou uma mistura de português e termos matemáticos (informal). Dizer que $\exists x P(x)$ é verdadeira é dizer que para algum (um ou mais de um) membro $x$ do *universo de discurso* em consideração, a afirmativa $P(x)$ é verdadeira. Algumas vezes o universo $U$ para o qual se faz a afirmativa é explicitado na forma[3] $\forall x \in U P(x)$ ou $\exists x \in U P(x)$.

**Exemplo 2** Considere a afirmativa: todo número natural par ao quadrado é par. Uma notação um pouco mais formal para tal afirmativa, supondo que $\mathbf{N}$ é o conjunto dos números naturais[4], seria: $\forall n[n \in \mathbf{N} \rightarrow (n \text{ é par } \rightarrow n^2 \text{ é par })]$; ou, alternativamente: $\forall n \in \mathbf{N}(n \text{ é par } \rightarrow n^2 \text{ é par })$. Mais formalmente: $\forall n \in \mathbf{N}[\exists k \in \mathbf{N}(n = 2k) \rightarrow \exists k \in \mathbf{N}(n^2 = 2k)]$. □

Uma afirmativa que é sempre verdadeira, independentemente dos valores-verdade assumidos para as subafirmativas que a compõem, é dita ser *válida*. E aquela que é sempre

[2]Esta explicação aparece no livro Mendelson, E., *Introduction to Mathematical Logic*, 3rd ed., Wadsworth & Brooks/Cole, 1987.

[3]A expressão $x \in A$ denota que o elemento $x$ pertence ao conjunto $A$. Elementos de Teoria dos Conjuntos serão apresentados na próxima seção.

[4]O conjunto dos números naturais é o conjunto dos números inteiros não negativos.

| | |
|---|---|
| *Idempotência* | |
| $\alpha \vee \alpha \equiv \alpha$ | $\alpha \wedge \alpha \equiv \alpha$ |
| *Identidade* | |
| $\alpha \vee \bot \equiv \alpha$ | $\alpha \wedge \top \equiv \alpha$ |
| $\alpha \vee \top \equiv \top$ | $\alpha \wedge \bot \equiv \bot$ |
| *Comutatividade* | |
| $\alpha \vee \beta \equiv \beta \vee \alpha$ | $\alpha \wedge \beta \equiv \beta \wedge \alpha$ |
| *Associatividade* | |
| $(\alpha \vee \beta) \vee \gamma \equiv \alpha \vee (\beta \vee \gamma)$ | $(\alpha \wedge \beta) \wedge \gamma \equiv \alpha \wedge (\beta \wedge \gamma)$ |
| *Distributividade* | |
| $\alpha \vee (\beta \wedge \gamma) \equiv (\alpha \vee \beta) \wedge (\alpha \vee \gamma)$ | $\alpha \wedge (\beta \vee \gamma) \equiv (\alpha \wedge \beta) \vee (\alpha \wedge \gamma)$ |
| *Complementação* | |
| $\alpha \vee \neg\alpha \equiv \top$ | $\alpha \wedge \neg\alpha \equiv \bot$ |
| $\neg\top \equiv \bot$ | $\neg \bot \equiv \top$ |
| *Leis de De Morgan* | |
| $\neg(\alpha \vee \beta) \equiv \neg\alpha \wedge \neg\beta$ | $\neg(\alpha \wedge \beta) \equiv \neg\alpha \vee \neg\beta$ |

Tabela 1.1: Propriedades elementares dos conectivos $\vee$ e $\wedge$.

falsa é dita ser uma *contradição*. Com isto, pode-se dizer que uma afirmativa é válida se, e somente se, a negação dela é uma contradição. Em alguns contextos pode ser útil ter um símbolo específico, $\top$, que é interpretado como sendo sempre verdadeiro, e um símbolo específico, $\bot$, que é interpretado como sendo sempre falso. Desta forma, uma afirmativa válida tem o mesmo valor-verdade que $\top$, ou seja V, e uma contradição tem o mesmo valor-verdade que $\bot$, ou seja, F.

**Exemplo 3** A seguinte tabela mostra que a afirmativa $[(\alpha \rightarrow \beta) \wedge \alpha] \rightarrow \beta$ é válida, já que contém apenas V's para tal afirmativa:

| $\alpha$ | $\beta$ | $\alpha \rightarrow \beta$ | $(\alpha \rightarrow \beta) \wedge \alpha$ | $[(\alpha \rightarrow \beta) \wedge \alpha] \rightarrow \beta$ |
|---|---|---|---|---|
| V | V | V | V | V |
| V | F | F | F | V |
| F | V | V | F | V |
| F | F | V | F | V |

São também exemplos de afirmativas válidas ($a$ denota um elemento do universo em consideração):

- $P(a) \rightarrow \exists x P(x)$;
- $\forall x P(x) \leftrightarrow \neg\exists x \neg P(x)$;
- $\exists x P(x) \leftrightarrow \neg\forall x \neg P(x)$. □

Duas afirmativas $\alpha$ e $\beta$ são ditas *logicamente equivalentes*, escreve-se $\alpha \equiv \beta$, quando o valor-verdade para cada uma delas é o mesmo, independentemente dos valores-verdades das sub-afirmativas componentes. A importância deste conceito está em que se $\alpha \equiv \beta$, tanto faz fazer referência a $\alpha$ quanto a $\beta$. A Tabela 1.1 mostra algumas equivalências lógicas elementares referentes aos conectivos $\vee$ e $\wedge$. Como se pode notar, as propriedades da disjunção e da conjunção vêm aos pares. E mais: elas podem ser obtidas umas das

| *Condicional* | |
|---|---|
| $\alpha \to \beta \equiv \neg\alpha \vee \beta$ | $\alpha \to \beta \equiv \neg\beta \to \neg\alpha$ |
| $\alpha \to (\beta \to \gamma) \equiv (\alpha \wedge \beta) \to \gamma$ | $\neg(\alpha \to \beta) \equiv \alpha \wedge \neg\beta$ |
| *Bicondicional* | |
| $\alpha \leftrightarrow \beta \equiv (\alpha \to \beta) \wedge (\beta \to \alpha)$ | $\alpha \leftrightarrow \beta \equiv (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$ |
| *Quantificadores* | |
| $\forall x P(x) \equiv \neg\exists x \neg P(x)$ | $\exists x P(x) \equiv \neg\forall x \neg P(x)$ |
| $\neg\forall x P(x) \equiv \exists x \neg P(x)$ | $\neg\exists x P(x) \equiv \forall x \neg P(x)$ |

Tabela 1.2: Algumas afirmativas logicamente equivalentes.

outras substituindo-se $\vee$, $\wedge$, $\top$ e $\bot$ por $\wedge$, $\vee$, $\bot$ e $\top$, respectivamente. A *dual* de uma afirmativa é justamente o resultado de se fazer tais substituições na mesma. Pode-se mostrar que a dual de uma equivalência lógica é também uma equivalência lógica.

Já a Tabela 1.2, apresenta equivalências lógicas importantes que são utilizadas em provas, muitas vezes sem menção explícita. As duas primeiras linhas apresentam equivalências importantes referentes ao conectivo condicional, que são utilizadas com bastante freqüência. A terceira linha exibe equivalências relativas ao conectivo bicondicional. A quarta linha mostra como os conectivos $\forall$ e $\exists$ são definíveis um em função do outro, e a última mostra duas equivalências relativas a tais quantificadores, também muito utilizadas.

O conectivo $\to$, também chamado de *implicação material*, tem relação íntima na matemática clássica com o conceito de *conseqüência lógica*, também chamado de *implicação lógica*. Uma afirmativa $\alpha$ é dita ser conseqüência lógica de um conjunto de afirmativas $\Gamma$, $\Gamma \Rightarrow \alpha$, quando $\alpha$ é verdadeira sempre que as afirmativas de $\Gamma$ são verdadeiras. O problema de provar um teorema $\alpha$ é justamente o problema de mostrar que $\Gamma \Rightarrow \alpha$, onde $\Gamma$ consta das hipóteses que podem ser usadas na prova.

**Exemplo 4** Suponha, como hipótese, que $\alpha \to \beta$ e $\alpha$ sejam verdadeiras. Neste caso, $\beta$ também é, como pode ser verificado na tabela de "$\to$" na Figura 1.2 (página 4): para cada linha em que $\alpha \to \beta$ e $\alpha$ são verdadeiras (no caso, apenas a primeira linha), $\beta$ também é. Assim, tem-se que[5] $\{\alpha \to \beta, \alpha\} \Rightarrow \beta$. □

Em uma prova obtém-se, sucessivamente, uma série de afirmativas de tal forma que cada afirmativa é conseqüência lógica das anteriores. Uma forma de obter uma afirmativa que seja conseqüência lógica de outras é mediante uma *regra de inferência*. O exemplo anterior justifica uma das regras de inferência utilizadas em provas de teoremas: a regra *modus ponens*. A Figura 1.3 ilustra várias regras de inferência, onde cada uma delas é apresentada na forma

$$\frac{\begin{array}{c} premissa_1 \\ premissa_2 \\ \vdots \\ premissa_n \end{array}}{conclusão}$$

[5] $\{\alpha \to \beta, \alpha\}$ é o conjunto constituído dos elementos $\alpha \to \beta$ e $\alpha$.

$$\frac{\alpha \quad \alpha \to \beta}{\beta} \qquad \frac{\alpha \quad \neg\alpha \vee \beta}{\beta} \qquad \frac{\neg\beta \quad \alpha \to \beta}{\neg\alpha}$$

$$\frac{\alpha \to \beta \quad \neg\alpha \to \beta}{\beta} \qquad \frac{\alpha \to \beta \quad \beta \to \gamma}{\alpha \to \gamma} \qquad \frac{\alpha \leftrightarrow \beta \quad \beta \leftrightarrow \gamma}{\alpha \leftrightarrow \gamma}$$

Figura 1.3: Algumas regras de inferência.

Evidentemente, tem-se que

$$\{premissa_1, premissa_2, \ldots, premissa_n\} \Rightarrow conclusão.$$

Seja $\Gamma$ um conjunto de afirmativas quaisquer. A relação entre o conectivo $\to$ e a implicação lógica é[6]: se $\Gamma \cup \{\alpha\} \Rightarrow \beta$ então $\Gamma \Rightarrow \alpha \to \beta$[7]. Em outras palavras: pode-se concluir que $\alpha \to \beta$ é consequência lógica do conjunto de hipóteses $\Gamma$ se $\beta$ é conseqüência lógica do conjunto de hipóteses $\Gamma$ acrescido da hipótese adicional $\alpha$. Isto justifica uma das técnicas de prova mais utilizadas:

**Técnica de prova direta para a condicional:**

Para provar $\alpha \to \beta$, supor $\alpha$ e provar $\beta$.

**Exemplo 5** Seja o problema de provar, para qualquer número natural $n$, que se $n$ é par, então $n^2$ é par.

Usando a técnica de prova direta para a condicional: Suponha que $n$ é um natural par. Neste caso, $n = 2k$ para algum número natural $k$, e $n^2 = (2k)^2 = 4k^2 = 2(2k^2)$. Logo, $n^2$ é par. Conclui-se: se $n$ é um natural par, então $n^2$ é par. □

Uma afirmativa da forma $\neg\beta \to \neg\alpha$ é dita ser a *contrapositiva* de $\alpha \to \beta$. Como diz a Tabela 1.2, $\alpha \to \beta \equiv \neg\beta \to \neg\alpha$. Assim, pode-se utilizar a técnica de prova para a condicional na seguinte forma:

**Técnica de prova pela contrapositiva:**

Para provar $\alpha \to \beta$, supor $\neg\beta$ e provar $\neg\alpha$.

No Exemplo 5, foi utilizado implicitamente um resultado relativo ao quantificador universal. Tal resultado é: se $\Gamma \Rightarrow P(a)$ e $a$ não ocorre em $\Gamma$, então $\Gamma \Rightarrow \forall x P(x)$. Isto justifica usar a seguinte técnica para provar $\forall x P(x)$:

**Técnica de prova para a universal:**

Para provar $\forall x P(x)$, supor um $a$ arbitrário, que não aparece em nenhuma hipótese a ser utilizada, e provar $P(a)$.

Ou então: para provar $\forall x \in A P(x)$, supor um $a \in A$ arbitrário, que não aparece em nenhuma hipótese a ser utilizada, e provar $P(a)$.

[6] $A \cup B$ é a *união* dos conjuntos $A$ e $B$, ou seja, é o conjunto constituído dos elementos de $A$ e de $B$.

[7] Este resultado, bastante intuitivo, é denominado *teorema da dedução* nos textos de Lógica Matemática.

**Exemplo 6** Segue uma prova da afirmativa do Exemplo 5 pondo em maior evidência a aplicação da técnica de prova para a universal. Será provado, então, que $\forall n \in \mathbf{N}$ se $n$ é par, então $n^2$ é par.

Seja $n$ um número natural arbitrário. Suponha que $n$ é par. Neste caso, $n = 2k$ para algum número natural $k$, e $n^2 = (2k)^2 = 4k^2 = 2(2k^2)$. Logo, se $n$ é par, $n^2$ é par. E como $n$ é um natural arbitrário, conclui-se que $\forall n \in \mathbf{N}$ se $n$ é par, então $n^2$ é par. □

Uma outra técnica bastante utilizada é a da prova *por contradição*. Suponha que se queira provar $\alpha$, a partir de um conjunto de hipóteses $\Gamma$. Ao invés de provar $\alpha$ diretamente a partir de $\Gamma$, supõe-se a negação de $\alpha$ como hipótese adicional e tenta-se chegar a uma contradição. Esta técnica é baseada no fato de que se $\Gamma \cup \{\neg\alpha\} \Rightarrow c$, onde $c$ é uma contradição qualquer, então $\Gamma \Rightarrow \alpha$[8]. Tem-se, então:

**Técnica de prova por contradição:**

Para provar $\alpha$, supor $\neg\alpha$ e derivar uma contradição.

**Exemplo 7** Será provado, por contradição, que existe uma infinidade de números primos.

Suponha que existe uma quantidade limitada de números primos $p_1, p_2, \ldots, p_n$, para algum natural $n$. Seja o número $k = (p_1 p_2 \cdots p_n) + 1$. Ora, tal número não é divisível por nenhum dos números primos $p_1, p_2, \ldots, p_n$. Logo, $k$ é divisível por algum outro primo diferente de $p_1, p_2, \ldots, p_n$ ou então $k$ é primo. Em qualquer destes dois casos, tem-se a existência de um primo diferente de $p_1$, $p_2$, $\ldots$, e $p_n$. Isto contradiz a suposição original de que existe uma quantidade limitada de números primos. Logo, existe uma infinidade de números primos. □

Observe que no exemplo acima mostrou-se que *existe* um número primo diferente de qualquer um dos $p_i$'s, sem mostrar um exemplar. Este é um exemplo de prova de uma afirmativa da forma $\exists x P(x)$, não construtiva: não é exibido (construído) um $x$ tal que $P(x)$. Uma outra técnica seria:

**Técnica de prova por construção:**

Para provar $\exists x P(x)$, provar $P(a)$ para um certo $a$ específico.

Ou então: para provar $\exists x \in A P(x)$, provar $P(a)$ para um certo $a \in A$ específico.

Tal técnica é baseada no fato de que se $\Gamma \Rightarrow P(a)$, então $\Gamma \Rightarrow \exists x P(x)$. Uma generalização desta técnica, baseada no fato mais geral de que se $\Gamma \Rightarrow P(a_1) \vee P(a_2) \vee \cdots \vee P(a_n)$, então $\Gamma \Rightarrow \exists x P(x)$, seria: para provar $\exists x P(x)$, provar $P(a_1) \vee P(a_2) \vee \cdots \vee P(a_n)$ para $a_1$, $a_2$, $\ldots$, e $a_n$ específicos.

**Exemplo 8** Será provado que, para qualquer número natural $n$, existe um número natural com no mínimo $n$ divisores distintos. (Observe como são usadas as técnicas de prova para universal e por construção.)

Seja um número natural arbitrário $n$. Pelo resultado do Exemplo 7, existem $n$ primos $p_1, p_2, \ldots, p_n$. Ora, um número natural com $n$ divisores distintos seria $p_1 p_2 \ldots p_n$. □

[8]Tal resultado segue do teorema da dedução no qual se baseia a técnica de prova para a condicional. Por este teorema, se $\Gamma \cup \{\neg\alpha\} \Rightarrow c$, então $\Gamma \Rightarrow \neg\alpha \rightarrow c$. Ora, $\neg\alpha \rightarrow c$, sendo $c$ uma contradição qualquer, é logicamente equivalente a $\alpha$!

Uma outra técnica, que já foi utilizada no Exemplo 7, é o da *análise de casos*. Ela é baseada no fato de que se $\Gamma \Rightarrow \alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_n$ e $\Gamma \cup \{\alpha_1\} \Rightarrow \beta$, $\Gamma \cup \{\alpha_2\} \Rightarrow \beta$, ..., e $\Gamma \cup \{\alpha_n\} \Rightarrow \beta$ então $\Gamma \Rightarrow \beta$:

**Técnica de prova por análise de casos:**

Para provar $\beta$, provar primeiro que $\alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_n$. Em seguida:

1) supor $\alpha_1$ e provar $\beta$;
2) supor $\alpha_2$ e provar $\beta$;

   $\vdots$

$n$) supor $\alpha_n$ e provar $\beta$.

**Exemplo 9** Seja o problema de provar que $min(x, y) + max(x, y) = x + y$ para quaisquer números reais $x$ e $y$.

Sejam $x$ e $y$ dois números reais arbitrários. Sabe-se que $x < y$, $x = y$ ou $x > y$. Serão considerados cada um destes três casos. No caso em que $x < y$, tem-se que $min(x, y) = x$ e $max(x, y) = y$; se $x = y$, $min(x, y) = max(x, y) = x = y$; e no caso em que $x > y$, $min(x, y) = y$ e $max(x, y) = x$. Em qualquer um dos três casos, $min(x, y) + max(x, y) = x + y$. Portanto, $min(x, y) + max(x, y) = x + y$ para quaisquer números reais $x$ e $y$. □

Na primeira técnica de prova vista acima, mostrou-se como provar $\alpha \rightarrow \beta$: supor $\alpha$ e provar $\beta$, ou então supor $\neg\beta$ e provar $\neg\alpha$. Uma outra forma de provar $\alpha \rightarrow \beta$ seria por contradição: supor $\neg(\alpha \rightarrow \beta)$, ou seja, $\alpha \wedge \neg\beta$ e derivar uma contradição. Já para provar $\alpha \leftrightarrow \beta$, baseando-se no fato de que $\alpha \leftrightarrow \beta \equiv (\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$, basta provar ambos, $\alpha \rightarrow \beta$ e $\beta \rightarrow \alpha$, usando tais técnicas.

**Técnica de prova para a bicondicional:**

Para provar $\alpha \leftrightarrow \beta$, provar ambos, $\alpha \rightarrow \beta$ e $\beta \rightarrow \alpha$.

**Exemplo 10** A seguir, prova-se que $\forall n \in \mathbf{N}$ $n$ é par se, e somente se, $n^2$ é par.

Seja $n$ um número natural arbitrário. Basta provar que $n$ é par se, e somente se, $n^2$ é par.

($\rightarrow$) Suponha que $n$ é par. Neste caso, $n = 2k$ para algum número natural $k$, e $n^2 = (2k)^2 = 4k^2 = 2(2k^2)$. Logo, se $n$ é par, $n^2$ é par.

($\leftarrow$) A prova será feita pela contrapositiva. Para isto, suponha que $n$ é ímpar. Neste caso, $n = 2k + 1$ para algum número natural $k$, e $n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 2(2k^2 + 2k) + 1$. Logo, se $n$ é ímpar, $n^2$ é ímpar. □

Quando se deseja provar uma cadeia de equivalências da forma $\alpha_1 \leftrightarrow \alpha_2 \leftrightarrow \cdots \leftrightarrow \alpha_n$[9], onde $n \geq 3$, basta provar a seqüência de implicações:

$$\alpha_1 \rightarrow \alpha_2, \alpha_2 \rightarrow \alpha_3, \ldots, \alpha_n \rightarrow \alpha_1.$$

Uma outra técnica de prova que será muito utilizada neste livro é a *prova por indução*, que será abordada na Seção 1.8.

[9]O conectivo $\leftrightarrow$ é associativo, isto é, $(\alpha \leftrightarrow \beta) \leftrightarrow \gamma \equiv \alpha \leftrightarrow (\beta \leftrightarrow \gamma)$, como pode ser facilmente verificado, o que justifica a ausência de parênteses.

## Exercícios

1. Mostre que as seguintes afirmativas são válidas:
   (a) $(\alpha \wedge \beta) \rightarrow \alpha$.
   (b) $\alpha \rightarrow (\alpha \vee \beta)$.
   (c) $(\alpha \wedge \neg\alpha) \rightarrow \beta$.
   (d) $\alpha \rightarrow (\beta \vee \neg\beta)$.
   (e) $(\alpha \rightarrow \beta) \vee \alpha$.
   (f) $(\alpha \rightarrow \beta) \vee (\beta \rightarrow \alpha)$.

2. Mostre as seguintes equivalências lógicas[10]:
   (a) $(\alpha \rightarrow \beta) \equiv (\neg\beta \rightarrow \neg\alpha)$.
   (b) $(\alpha \vee \beta) \rightarrow \gamma \equiv [(\alpha \rightarrow \gamma) \wedge (\beta \rightarrow \gamma)]$.
   (c) $\alpha \rightarrow (\beta \wedge \gamma) \equiv [(\alpha \rightarrow \beta) \wedge (\alpha \rightarrow \gamma)]$.

3. Mostre as seguintes implicações lógicas:
   (a) $\{\alpha, \neg\alpha\} \Rightarrow \gamma$.
   (b) $\{\alpha \rightarrow \gamma\} \Rightarrow (\alpha \wedge \beta) \rightarrow \gamma$.
   (c) $\{\neg\alpha \rightarrow \beta, \neg\beta\} \Rightarrow \alpha$.

4. Prove que se $x > 0$ e $x < y$, onde $x$ e $y$ são números reais, então $x^2 < y^2$.

5. Prove que se $x^2 + y = 13$ e $y \neq 4$, onde $x$ e $y$ são reais, então $x \neq 3$.

6. Prove que, para todo número real $x$, se $x > 2$, então existe um real $y$ tal que $y + (1/y) = x$.

7. Prove que, para todo número natural $x$, se $x$ não é um quadrado perfeito, $\sqrt{x}$ é um número irracional.

8. Prove que se $n$ é um número inteiro não divisível por 3, então $n^2 = 3k + 1$ para algum inteiro $k$.

## 1.3 Conjuntos

Um *conjunto* é uma abstração matemática que captura o conceito de uma coleção de objetos. Os objetos de um conjunto, também chamados de *elementos* ou *membros* do conjunto, podem ser também conjuntos. Para se dizer que um objeto *a pertence a* um conjunto $A$, ou seja, é membro de $A$, será usada a notação $a \in A$; e para dizer que *a não* pertence a $A$, será usada a notação $a \notin A$.

Alguns conjuntos finitos podem ser definidos listando-se seus elementos entre chaves, separados por vírgulas. A ordem dos elementos na lista é irrelevante, pois dois conjuntos com os mesmos elementos são considerados *iguais*, ou seja, são *o mesmo conjunto*.

[10]Para mostrar que $\alpha \equiv \beta$ pode-se, por exemplo, usar as tabelas da verdade para $\alpha$ e $\beta$.

**Exemplo 11** Um exemplo de conjunto de objetos homogêneos seria o conjunto dos planetas do sistema solar:

$$\{\textit{mercúrio}, \textit{vênus}, \textit{terra}, \textit{marte}, \textit{júpiter}, \textit{saturno}, \textit{urano}, \textit{netuno}, \textit{plutão}\}.$$

Um outro seria o conjunto dos números inteiros. O seguinte conjunto contém números, planetas e conjuntos:

$$\{10, \textit{marte}, \{0\}, \{\textit{terra}, 1, 2, 3\}\}.$$

Observe que os seguintes conjuntos são iguais, ou seja, são o mesmo conjunto:

$$\{1, 2\} = \{2, 1\} = \{1, 2, 1\} = \{2, 1 + 1, 2 - 1, \sqrt{4}\}.$$

□

O conjunto que não contém membros, o *conjunto vazio*, é denotado por $\emptyset$. Assim, $\emptyset = \{\}$. No outro extremo existem os *conjuntos infinitos*, que contêm uma quantidade ilimitada de elementos. Dentre estes, alguns são importantes para merecer uma notação especial:

- $\mathbf{N}$, o conjunto dos números naturais (inteiros não negativos);
- $\mathbf{Z}$, o conjunto dos números inteiros;
- $\mathbf{R}$, o conjunto dos números reais;
- $\mathbf{Q}$, o conjunto dos números racionais: os números reais que podem ser expressos na forma $m/n$, onde $m$ e $n$ são números inteiros.

Existem várias outras formas de definir conjuntos, além de listar seus elementos entre chaves. Uma delas é utilizar uma expressão da forma $\{x \mid P(x)\}$, que quer dizer *o conjunto de todos os elementos $x$ tais que $x$ satisfaz a propriedade $P$*. É muito comum definir "o conjunto dos elementos do conjunto $A$ que satisfazem a propriedade $P$"[11]. Por isto, é comum o uso da notação $\{x \in A \mid P(x)\}$ para denotar o mesmo conjunto que $\{x \mid x \in A \text{ e } P(x)\}$.

**Exemplo 12** O conjunto dos números naturais ímpares pode ser denotado por $\{k \mid k = 2n + 1 \text{ e } n \in \mathbf{N}\}$.

O conjunto dos números reais entre 0 e 1, incluindo 0 e 1, pode ser denotado por $\{k \in \mathbf{R} \mid 0 \leq k \leq 1\}$. □

Um *conjunto unitário* é um conjunto que contém um único membro. Exemplos: {terra}, {10}, $\{\emptyset\}$. Este último conjunto tem o conjunto vazio como seu único elemento.

Um conjunto $A$ é dito estar *contido* em um conjunto $B$, $A \subseteq B$, se todo elemento de $A$ é elemento de $B$, ou seja:

$$A \subseteq B \leftrightarrow \text{para todo } x, \text{ se } x \in A \text{ então } x \in B.$$

[11] Sendo $A$ um conjunto bem conhecido, garante-se a existência do conjunto definido. Evita-se, com isto, o surgimento de paradoxos como o de Russell.

Neste caso, $A$ é dito ainda ser um *subconjunto* de $B$. Um conjunto $A$ que está contido em $B$, mas que não é igual a $B$, é dito ser um subconjunto *próprio* de $B$; neste caso, escreve-se $A \subset B$. Assim,

$$A \subset B \leftrightarrow A \subseteq B \text{ e } A \neq B.$$

A *união* de dois conjuntos $A$ e $B$, $A \cup B$, é conjunto constituído dos elementos de $A$ e de $B$, ou seja:

$$A \cup B = \{x \mid x \in A \text{ ou } x \in B\}.$$

A *interseção* de dois conjuntos $A$ e $B$, $A \cap B$, é conjunto constituído dos elementos comuns de $A$ e $B$, ou seja:

$$A \cap B = \{x \mid x \in A \text{ e } x \in B\}.$$

A *diferença* entre dois conjuntos $A$ e $B$, $A - B$, é o conjunto constituído dos elementos de $A$ que não pertencem a $B$, ou seja:

$$A - B = \{x \mid x \in A \text{ e } x \notin B\}.$$

**Exemplo 13** Sejam os conjuntos $A = \{0, 1, 2, 3\}$ e $B = \{2, 3, 4\}$. Então:

- $A, B \subseteq \mathbf{N}$;
- $A, B \subset \mathbf{N}$;
- $\mathbf{N} \subset \mathbf{Z} \subset \mathbf{Q} \subset \mathbf{R}$;
- $A \cup B = \{0, 1, 2, 3, 4\}$;
- $A \cap B = \{2, 3\}$;
- $A - B = \{0, 1\}$;
- $B - A = \{4\}$;
- $A \cup \mathbf{N} = \mathbf{N}$;
- $A \cap \mathbf{N} = A$;
- $A - \mathbf{N} = \emptyset$;
- $\mathbf{N} - A = \{k \in \mathbf{N} \mid k \geq 4\}$. □

O *complemento* de um conjunto $A$ com relação a um *conjunto universo* $U$ é $U - A$. Em um determinado contexto, fixando-se um certo conjunto $U$ como sendo o conjunto universo, passa-se a expressar o complemento de um conjunto $A$ por $\overline{A}$. Neste caso, dizer que $x \in \overline{A}$ é equivalente a dizer que $x \in U - A$, e equivalente também a dizer que $x \notin A$.

A Tabela 1.3 mostra algumas propriedades das operações de união, concatenação e diferença. Deve ser observada a similaridade desta tabela com a Tabela 1.1 (página 6). De forma análoga ao que lá acontece com os conectivos $\vee$ e $\wedge$, aqui as propriedades da união e da interseção vêm aos pares, sendo que elas podem ser obtidas umas das outras substituindo-se $\cup$, $\cap$, $U$ e $\emptyset$ por $\cap$, $\cup$, $\emptyset$ e $U$, respectivamente. A *dual* de uma equação

| | | |
|---|---|---|
| *Idempotência* | | |
| $A \cup A = A$ | | $A \cap A = A$ |
| *Identidade* | | |
| $A \cup \emptyset = A$ | | $A \cap \emptyset = \emptyset$ |
| *Comutatividade* | | |
| $A \cup B = B \cup A$ | | $A \cap B = B \cap A$ |
| *Associatividade* | | |
| $(A \cup B) \cup C = A \cup (B \cup C)$ | | $(A \cap B) \cap C = A \cap (B \cap C)$ |
| *Distributividade* | | |
| $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ | | $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ |
| *Complementação* | | |
| $A \cup \overline{A} = U$ | | $A \cap \overline{A} = \emptyset$ |
| $\overline{U} = \emptyset$ | | $\overline{\emptyset} = U$ |
| *Leis de De Morgan* | | |
| $\overline{A \cup B} = \overline{A} \cap \overline{B}$ | | $\overline{A \cap B} = \overline{A} \cup \overline{B}$ |
| Diferença | | |
| $A - \emptyset = A$ | $A - A = \emptyset$ | $\emptyset - A = \emptyset$ |

Tabela 1.3: Propriedades da união, interseção e diferença.

envolvendo conjuntos, $E$, é justamente o resultado de se fazer tais substituições em $E$. Pode-se mostrar que a dual de uma identidade é também uma identidade.

Em geral, para provar que dois conjuntos $A$ e $B$ são iguais, prova-se que $A \subseteq B$ e $B \subseteq A$. No entanto, para provar propriedades genéricas sobre conjuntos, como as da Tabela 1.3, pode ficar mais conciso e simples utilizar as propriedades abaixo, decorrentes diretamente das definições dos operadores:

$$\begin{aligned} x \in A \cup B &\leftrightarrow x \in A \vee x \in B \\ x \in A \cap B &\leftrightarrow x \in A \wedge x \in B \\ x \in A - B &\leftrightarrow x \in A \wedge x \notin B \\ x \notin A &\leftrightarrow \neg x \in A \end{aligned}$$

Utilizando-se tais propriedades, "traduz-se" o problema em um problema de manipulação lógica, que pode ser feita utilizando as equivalências lógicas conhecidas e a última regra de inferência da Figura 1.3 (página 8). Veja o próximo exemplo.

**Exemplo 14** Segue uma prova da propriedade distributiva da união sobre a interseção.

Para provar $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$, basta provar que para todo $x$, $x \in A \cup (B \cap C) \leftrightarrow x \in (A \cup B) \cap (A \cup C)$. Para isto, basta provar que, para um elemento arbitrário $x$, $x \in A \cup (B \cap C) \leftrightarrow x \in (A \cup B) \cap (A \cup C)$. Seja então um $x$ arbitrário. Tem-se:

$$\begin{aligned} x \in A \cup (B \cap C) &\leftrightarrow x \in A \text{ ou } x \in B \cap C && \text{pela definição de } \cup \\ &\leftrightarrow x \in A \text{ ou } (x \in B \text{ e } x \in C) && \text{pela definição de } \cap \\ &\leftrightarrow (x \in A \text{ ou } x \in B) \text{ e } (x \in A \text{ ou } x \in C) \\ & && \text{pela distributividade de "ou" sobre "e"} \\ &\leftrightarrow x \in A \cup B \text{ e } x \in A \cup C && \text{pela definição de } \cup \\ &\leftrightarrow x \in (A \cup B) \cap (A \cup C) && \text{pela definição de } \cap. \end{aligned}$$

Observe como a última regra de inferência da Figura 1.3 é utilizada aqui de forma implícita. □

As leis de De Morgan para conjuntos são intimamente relacionadas com as leis de De Morgan para os conectivos lógicos "e" e "ou". Durante a prova de tais leis para conjuntos, certamente serão utilizadas as leis respectivas para os conectivos lógicos, como poderá ser verificado solucionando-se o exercício 4(a) do final desta seção, na página 16.

Dois conjuntos $A$ e $B$ são ditos *disjuntos* se, e somente se, $A \cap B = \emptyset$.

Dados os conjuntos $A_1, A_2, \ldots, A_n$, sendo $n \geq 1$, define-se:

$$\bigcup_{i=1}^{n} A_i = A_1 \cup A_2 \cup \cdots \cup A_n,$$

e

$$\bigcap_{i=1}^{n} A_i = A_1 \cap A_2 \cap \cdots \cap A_n.$$

Uma *partição* de um conjunto $A$ é um conjunto $\{B_1, B_2, \ldots, B_n\}$, $n \geq 1$, constituído de conjuntos $B_i$, tal que:

(a) $B_i \neq \emptyset$, para $1 \leq i \leq n$;

(b) $B_i$ e $B_j$ são disjuntos, para $1 \leq i < j \leq n$; e

(c) $\bigcup_{i=1}^{n} B_i = A$.

Neste caso, diz-se ainda que os conjuntos $B_1, B_2, \ldots, B_n$ *particionam* o conjunto $A$.

**Exemplo 15** Os conjuntos $\mathbf{Q}$, dos números racionais, e $\mathbf{R} - \mathbf{Q}$, dos números irracionais, particionam $\mathbf{R}$, ou seja, $\{\mathbf{Q}, \mathbf{R} - \mathbf{Q}\}$ é uma partição de $\mathbf{R}$. □

O *conjunto potência* de um conjunto $A$, $\mathcal{P}(A)$, é o conjunto de todos os subconjuntos de $A$, ou seja,

$$\mathcal{P}(A) = \{X \mid X \subseteq A\}.$$

Em particular, $\emptyset \in \mathcal{P}(A)$ e $A \in \mathcal{P}(A)$.

O número de elementos de um conjunto finito $A$ será denotado por $|A|$. Por exemplo, $|\{\emptyset, a, \{a, b, c, d\}\}| = 3$. Um outro exemplo: $|\mathcal{P}(A)| = 2^{|A|}$, se $A$ é finito.

Um conjunto de dois elementos é denominado um *par não ordenado*. Um *par ordenado*, também chamado de *dupla*, cujo primeiro elemento é $a$ e segundo é $b$, é denotado por $(a, b)$. A propriedade básica de um par ordenado é que $(a, b) = (c, d)$ se, e somente se, $a = c$ e $b = d$. Uma $n$-upla (lê-se "ênupla") de elementos $a_1$, $a_2$, ..., $a_n$, nesta ordem, é denotada por $(a_1, a_2, \ldots, a_n)$[12]. Termos alternativos para 3-upla, 4-upla, etc., são tripla, quádrupla, quíntupla, sêxtupla, etc.

O *produto cartesiano* de dois conjuntos $A$ e $B$, $A \times B$, é o conjunto de todos os pares ordenados tais que o primeiro elemento pertence a $A$ e o segundo pertence a $B$, ou seja,

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

[12]Formalmente, para $n > 2$, uma $n$-nupla seria uma dupla cujo primeiro elemento é a $n-1$-upla constituída dos $n-1$ primeiros elementos da $n$-upla, e cujo segundo elemento é o último elemento da $n$-upla.

Generalizando, o produto cartesiano de $n$ conjuntos $A_1$, $A_2$, ..., $A_n$ é o conjunto:

$$A_1 \times A_2 \times \cdots \times A_n = \{(a_1, a_2, \ldots, a_n) \mid a_1 \in A_1, a_2 \in A_2, \ldots, a_n \in A_n\}.$$

O produto cartesiano $A \times A \times A \times \cdots \times A$ ($n$ vezes) é denotado, alternativamente, por $A^n$. Assume-se que $A^0 = \emptyset$ e $A^1 = A$.

**Exemplo 16** Sejam $A = \{1, 2\}$ e $B = \{2, 3\}$. Tem-se:

- $A \times B = \{(1,2), (1,3), (2,2), (2,3)\}$;
- $A \times A = A^2 = \{(1,1), (1,2), (2,1), (2,2)\}$;
- $A \times B \times A = \{(1,2,1), (1,2,2), (1,3,1), (1,3,2), (2,2,1), (2,2,2), (2,3,1), (2,3,2)\}$;
- $A^3 = \{(1,1,1), (1,1,2), (1,2,1), (1,2,2), (2,1,1), (2,1,2), (2,2,1), (2,2,2)\}$. □

Evidentemente, $|A_1 \times A_2 \times \cdots \times A_n| = |A_1||A_2| \cdots |A_n|$, se $A_1$, $A_2$, ..., $A_n$ são finitos.

# Exercícios

1. Sejam os conjuntos $A = \{n \in \mathbf{N} \mid n \leq 8\}$ e $B = \{n \in \mathbf{Z} \mid -5 \leq n \leq 5\}$. Liste os elementos dos conjuntos seguintes:
   (a) $A \cap B$.
   (b) $C = \{n \in A \cup B \mid n = 2k \text{ para algum } k \in \mathbf{Z}\}$.
   (c) $D = (A - B) \cup (B - A)$.
   (d) $[(A \cap C) - (A \cap D)] \times [(A \cap D) - (A \cap C)]$.
2. Que condição os conjuntos $A$ e $B$ devem satisfazer para que $A - B = B - A$? E para que $A \cup B = A \cap B$?
3. Que condição os conjuntos $A$, $B$ e $C$ devem satisfazer para que $A \cup B = A \cup C$ e $B \neq C$?
4. Prove que:
   (a) $\overline{A \cup B} = \overline{A} \cap \overline{B}$.
   (b) se $A \cap B = A \cup B$ então $A = B$.
   (c) $(A - B) \cup (B - A) = (A \cup B) - (A \cap B)$.
   (d) $A - (A - B) = A \cap B$.
   (e) $(A - B) - C = A - (B \cup C)$.
   (f) $(A - B) - C = (A - C) - (B - C)$.
   (g) $A \times (B \cap C) = (A \times B) \cap (A \times C)$.
   (h) $(A \cap B) \times (C \cap D) = (A \times C) \cap (B \times D)$.
5. Liste todas as partições dos conjuntos $\{1, 2\}$, $\{1, 2, 3\}$ e $\{1, 2, 3, 4\}$. Desenvolva um método sistemático para gerar todas as partições de um conjunto $\{1, 2, \ldots, n\}$.

## 1.4 Relações

Uma *relação* de $n$ argumentos sobre os conjuntos $A_1$, $A_2$, ..., $A_n$ é um subconjunto de $A_1 \times A_2 \times \cdots \times A_n$. As relações de dois argumentos são denominadas relações *binárias*, as de três argumentos são denominadas relações *ternárias*, etc.

**Exemplo 17** Um exemplo de relação binária seria:

$$\{(a, d) \mid a \in A \text{ e } d \in D \text{ e } a \text{ está matriculado em } d\}.$$

onde $A$ é o conjunto de todos os alunos de certo curso e $D$ é o conjunto das disciplinas do curso.

As relações $<$, $\leq$, etc., sobre os reais são exemplos de relações binárias sobre $\mathbf{R}^2$.

Um exemplo de relação ternária: $\{(x, y, z) \in \mathbf{N}^3 \mid x < y < z\}$. □

O *domínio* de uma relação binária $R \subseteq A \times B$ é o conjunto $A$, e o *contra-domínio* de $R$ é o conjunto $B$. A *imagem* de $R$ é o conjunto $\{y \mid (x, y) \in R \text{ para algum } x\}$. Uma relação sobre $A^2$ é também dita ser uma relação sobre $A$ simplesmente. A *relação inversa* de $R \subseteq A \times B$ é a relação $R^{-1} \subseteq B \times A$ dada por $R^{-1} = \{(y, x) \mid (x, y) \in R\}$. Para relações binárias, é comum a notação $xRy$ ao invés de $(x, y) \in R$.

**Exemplo 18** Seja a relação $<$ sobre $\mathbf{N}$. Tanto o domínio como o contra-domínio de $<$ são o conjunto $\mathbf{N}$. A inversa de $<$ é a relação $>$, isto é, $<^{-1}=>$, pois tem-se que $x < y \Leftrightarrow y > x$ para quaisquer $(x, y) \in \mathbf{N}^2$. A imagem de $<$ é o conjunto $\mathbf{N} - \{0\}$, pois nenhum número natural é menor que 0, e qualquer outro número natural possui algum menor que ele (0, por exemplo). A imagem de $>$ é $\mathbf{N}$. □

Uma relação binária $R \subseteq A^2$ é dita ser:

(a) *reflexiva*, se $xRx$ para todo $x \in A$;

(b) *simétrica*, se $xRy$ implica $yRx$ para todo $x, y \in A$;

(c) *transitiva*, se $xRy$ e $yRz$ implica $xRz$ para todo $x, y, z \in A$.

**Exemplo 19** Tanto a relação $<$ com a relação $>$ do Exemplo 18 não são reflexivas nem simétricas; e ambas são transitivas. Já as relações $\leq$ e $\geq$ sobre $\mathbf{N}$ são reflexivas, não são simétricas e são transitivas. A relação $\subseteq$ também é reflexiva, não simétrica e transitiva.

A relação *é irmão de* sobre o conjunto das pessoas do mundo em certo instante

- não é reflexiva: uma pessoa não é irmã de si mesma;
- é simétrica: se fulano é irmão de beltrano, então beltrano é irmão de fulano; e
- não é transitiva: quando fulano é irmão de beltrano, beltrano é irmão de fulano (simetria), mas fulano não é irmão de fulano. □

Uma *relação de equivalência* é uma relação binária reflexiva, simétrica e transitiva. Uma relação de equivalência $R$ sobre um conjunto $A$ divide $A$ em *classes de equivalência*. Tais classes de equivalência formam uma partição do conjunto $A$. A classe de equivalência que contém o elemento $x$ é denotada por $[x]$, e é definida como $[x] = \{y \mid xRy\}$. Pode-se provar que $[x] = [y]$ se, e somente se, $xRy$.

**Exemplo 20** Um exemplo simples de relação de equivalência é a relação de identidade sobre um conjunto $C$, $\iota_C = \{(x, x) \mid x \in C\}$. O conjunto $C$ é particionado em classes equivalência unitárias $[x] = \{x\}$.

Um outro exemplo de relação de equivalência é a relação $(mod\, n) = \{(x, y) \in \mathbf{N}^2 \mid x \bmod n = y \bmod n\}$[13]. A partição de $\mathbf{N}$ induzida pela relação $(mod\, n)$ tem $n$ classes de equivalência, uma para cada resto de 0 a $n-1$: $\{\{0, n, 2n, \ldots\}, \{1, n+1, 2n+1, \ldots\}, \ldots, \{n-1, 2n-1, 3n-1, \ldots\}\}$

Mais um exemplo: a relação $\mathcal{A} \subseteq P \times P$, onde $P$ é o conjunto das pessoas do mundo, tal que

$$\mathcal{A} = \{(p, q) \in P^2 \mid p \text{ e } q \text{ fazem aniversário no mesmo dia}\}$$

é uma relação de equivalência que particiona $P$ em 366 classes de equivalência, uma para cada dia do ano. □

Os conceitos de fechos de uma relação sob as propriedades reflexiva, simétrica e/ou transitiva, definidos a seguir, terão aplicação nos próximos capítulos. O *fecho reflexivo* de uma relação $R \subseteq A \times A$ é a relação $S$ tal que:

(a) $R \subseteq S$;

(b) $S$ é reflexiva; e

(c) $S$ está contida em qualquer outra relação com as propriedades (a) e (b), ou seja, se $R \subseteq T$ e $T$ é reflexiva, então $S \subseteq T$.

De forma análoga se define *fecho simétrico* e *fecho transitivo*.

**Exemplo 21** Seja a relação $<$ sobre $\mathbf{N}$. Então: seu fecho reflexivo é $\leq$; seu fecho simétrico é a relação $\{(x, y) \in \mathbf{N} \mid x < y \text{ ou } x > y\}$; seu fecho transitivo é $<$, pois esta é transitiva; seu fecho transitivo e reflexivo é $\leq$; seu fecho reflexivo e simétrico é a relação $\{(x, y) \mid x, y \in \mathbf{N}\}$; e assim por diante. □

# Exercícios

1. Diga que propriedades, dentre reflexividade, simetria e transitividade, têm cada uma das relações:

   (a) $\subset$ sobre conjuntos.

   (b) $\{(x, y) \in \mathbf{R}^2 \mid x = y^2\}$.

   (c) $\{(x, y) \in \mathbf{N}^2 \mid x \text{ é divisível por } y\}$.

   (d) $\{((x_1, x_2), (y_1, y_2)) \in (\mathbf{N} \times \mathbf{N})^2 \mid x_1 \leq y_1 \text{ e } x_2 \leq y_2\}$.

   (e) $\{(x, y) \in \mathbf{R}^2 \mid x - y \text{ é um inteiro}\}$.

2. Sejam duas relações binárias $R_1$ e $R_2$. Diga que propriedades as relações $R_1 \cup R_2$ e $R_1 \cap R_2$ devem ter nos seguintes casos:

[13] $x \bmod n$ é o resto da divisão de $x$ por $n$

(a) $R_1$ e $R_2$ são reflexivas.

(b) $R_1$ e $R_2$ são simétricas.

(c) $R_1$ e $R_2$ são transitivas.

3. Mostre que uma relação $R$ sobre um conjunto é simétrica se, e somente se, $R = R^{-1}$.

4. Mostre que uma relação $R$ sobre um conjunto é reflexiva se, e somente se, $R^{-1}$ é reflexiva.

5. Uma relação $R$ sobre $A$ é dita ser *anti-simétrica* se, e somente se, se $xRy$ e $yRx$ então $x = y$, para todo $x, y \in A$. Mostre que uma relação $R$ sobre $A$ é anti-simétrica se, e somente se, $R \cap R^{-1} \subseteq \iota_A$.

6. Seja a relação $R = \{((x_1, x_2), (y_1, y_2)) \in (\mathbf{N} \times \mathbf{N})^2 \mid x_1/x_2 = y_1/y_2\}$. Prove que $R$ é uma relação de equivalência. Quais são as classes de equivalência de $R$?

7. Seja a relação $R = \{(a, b), (b, b), (c, d), (d, c), (d, a)\}$ sobre $A = \{a, b, c, d, e\}$. Determine todos os 7 fechos de $R$ (reflexivo, simétrico, etc.).

## 1.5 Funções

Uma *função parcial* $f : A \to B$ (lê-se "$f$ de $A$ para $B$") é uma relação binária $f \subseteq A \times B$ tal que se $(x, y) \in f$ e $(x, z) \in f$ então $y = z$. Uma notação mais utilizada para dizer que $(x, y) \in f$ é $f(x) = y$. Se não existe $y$ tal que $f(x) = y$, diz-se que $f$ é *indefinida* para o argumento $x$. Se para todo $x \in A$ existe $y$ tal que $f(x) = y$, diz-se que a função é *total*. Ou seja, uma função total $f : A \to B$ é uma função parcial definida para todo argumento $x \in A$.

Uma *função de n argumentos* é uma função da forma $f : A_1 \times A_2 \times \ldots \times A_n \to B$. Ao invés de escrever $f((a_1, a_2, \ldots, a_n))$ é mais comum escrever $f(a_1, a_2, \ldots, a_n)$. Diz-se que $a_1$ é o primeiro argumento, $a_2$ o segundo, etc.

**Exemplo 22** Um exemplo de função total é a soma sobre os reais, $+ : \mathbf{R}^2 \to \mathbf{R}$, pois a soma $x + y$[14] sempre existe para quaisquer reais $x$ e $y$. Já a divisão, $/ : \mathbf{R}^2 \to \mathbf{R}$ não é total, visto que não é definida quando o segundo argumento é 0. □

Sejam duas funções $f : A \to B$ e $g : C \to D$. A *composição* de $g$ e $f$ é a função $g \circ f : A \to D$ tal que

$$g \circ f(x) = \begin{cases} g(f(x)) & \text{se } f \text{ é definida para } x \text{ e } g \text{ é definida para } f(x) \\ \textit{indefinido} & \text{caso contrário.} \end{cases}$$

**Exemplo 23** Sejam as funções $f : \mathbf{Z} \to \mathbf{N}$ tal que[15] $f(n) = |n| + 1$ e $g : \mathbf{N} \to \mathbf{Z}$ tal que $g(n) = 1 - n$. Tem-se que $g \circ f : \mathbf{Z} \to \mathbf{Z}$ é tal que $(g \circ f)(n) = g(f(n)) = g(|n| + 1) =$

[14] Aqui está sendo utilizada a notação infixada $x + y$ ao invés da notação prefixada $+(x, y)$. De forma similar, outras funções comuns da matemática serão utilizadas também na forma infixada, ou seja, com o nome da função entre os argumentos.

[15] $|x|$ denota o valor absoluto de $x$, quando $x$ é um número, e denota o número de elementos de $x$ quando $x$ é um conjunto.

$1 - (|n| + 1) = -|n|$. Por outro lado, $f \circ g : \mathbf{N} \to \mathbf{N}$ é tal que $(f \circ g)(n) = f(g(n)) = f(1 - n) = |1 - n| + 1$. □

O termos *domínio*, *contra-domínio* e *imagem*, como definidos na Seção 1.4, página 17, se aplicam a funções. Assim, $A$ é o domínio e $B$ o contra-domínio de uma função $f : A \to B$. A imagem é $\{y \in B \mid f(x) = y \text{ para algum } x \in A\}$.

Seja uma função total $f : A \to B$. Diz-se que $f$ é:

- *injetora*, se para $x, y \in A$ quaisquer, $x \neq y \to f(x) \neq f(y)$;
- *sobrejetora*, se $B$ é a imagem de $f$;
- *bijetora*[16], se é injetora e sobrejetora.

**Exemplo 24** A função $f : \mathbf{N} \to \mathbf{N}$ tal que $f(x) = 2x$ é injetora, mas não sobrejetora. A função $g : \mathbf{Z} \to \mathbf{N}$ tal que $g(x) = |x|$ é sobrejetora, mas não injetora. A função $h : \mathbf{Z} \to \mathbf{N}$ tal que

$$h(x) = \begin{cases} 2x & \text{se } x \geq 0 \\ -(2x + 1) & \text{se } x < 0 \end{cases}$$

é injetora e sobrejetora; logo, é bijetora. □

A função *inversa* de $f : A \to B$, $f^{-1} : B \to A$, existe somente no caso em que $f$ é injetora, e é tal que $f^{-1}(f(x)) = x$ para todo $x \in A$. Se $f$ for injetora mas não sobrejetora, $f^{-1}$ será uma função parcial definida apenas para os elementos da imagem de $f$. Note que, para uma função injetora $f : A \to B$, $(f^{-1} \circ f) = \iota_A$.

# Exercícios

1. Sejam $A$ e $B$ dois conjuntos finitos. Qual é o número máximo de elementos que uma relação $R \subseteq A \times B$ pode ter para ela ser uma função?
2. Que condição deve ser satisfeita por duas funções $f$ e $g$ para que $f \cup g$ seja uma função?
3. Verifique se as seguintes funções são injetoras, sobrejetoras e/ou bijetoras e, para cada uma, se possível, determine a função inversa:
   (a) $f : \mathbf{N} \to \mathbf{N}$ tal que $f(n) = n \div 2$, onde "$\div$" é divisão inteira.
   (b) $g : \mathbf{N} \to \mathbf{N}$ tal que $g(n) = n(n + 1)/2$.
   (c) $h : \mathbf{N} \to \mathbf{N}$ tal que $h(n) = n - 1$, se $n$ for ímpar, e $h(n) = n + 1$, caso contrário.
4. Sejam $A$ e $B$ conjuntos finitos. Determine quantas funções de $A$ para $B$ existem para cada um dos seguintes tipos de funções:
   (a) Totais.
   (b) Parciais.

[16] Também chamada de *correspondência um-para-um*.

(c) Injetoras.

(d) Sobrejetoras.

(e) Bijetoras.

5. Sejam duas funções totais $f : A \to B$ e $g : B \to C$. Especifique que propriedades deve ter a função $g \circ f$ nos seguintes casos:

   (a) $f$ e $g$ são injetoras.

   (b) $f$ e $g$ são sobrejetoras.

   (c) $f$ e $g$ são bijetoras.

6. Mostre que qualquer função pode ser representada como a composição de duas funções, uma sobrejetora e outra injetora.

7. Sejam $f$ e $g$ funções sobre $\mathbf{R}$ tais que $f(n) = an+b$ e $g(n) = cn+d$. Se $g \circ f = f \circ g$, que relação existirá entre os coeficientes $a$, $b$, $c$ e $d$?

## 1.6 Conjuntos Enumeráveis

Para determinar se dois conjuntos finitos $A$ e $B$ têm o mesmo tamanho, basta "contar" o número de elementos de cada um e verificar se dá o mesmo número. Uma outra abordagem seria determinar se existe uma função bijetora de $A$ para $B$ (ou vice-versa). No caso de conjuntos infinitos, evidentemente não é possível contar o número de seus elementos, e a noção de "tamanho" como conhecida no dia a dia não se aplica. Tal noção, no entanto, pode ser substituída pela noção de cardinalidade, definida abaixo, que permite uma útil hierarquização dos conjuntos infinitos. Em particular, com tal noção é possível mostrar, por exemplo, que existem mais funções do que programas em qualquer linguagem de programação; ou seja, o conjunto de todas as funções (que é infinito) é "maior" do que o conjunto de todos os programas (também infinito), o que permite concluir que existem funções que *não* são programáveis em qualquer linguagem de programação.

Dois conjuntos $A$ e $B$ são ditos ter a mesma *cardinalidade*, isto é, $card(A) = card(B)$, se existe uma função bijetora de $A$ para $B$. Se $A$ é um conjunto finito, sua cardinalidade pode ser imaginada como o número de elementos do mesmo, ou seja, $card(A) = |A|$. Na Seção 1.3, foi dito que um conjunto infinito é aquele que tem uma "quantidade ilimitada" de elementos. Mais precisamente, um conjunto infinito pode ser definido como sendo aquele que tem um subconjunto próprio de mesma cardinalidade. Alternativamente, pode-se definir um conjunto $A$ como sendo finito quando ele tem a mesma cardinalidade que $\{k \in \mathbf{N} \mid k \leq n\}$ para algum $n \in \mathbf{N}$; neste caso, diz-se que $|A| = n + 1$.

**Exemplo 25** O conjunto dos naturais, $\mathbf{N}$ é infinito, pois:

(a) $\mathbf{N} - \{0\} \subset \mathbf{N}$; e

(b) $f : \mathbf{N} \to \mathbf{N} - \{0\}$ tal que $f(x) = x + 1$ é uma função bijetora.

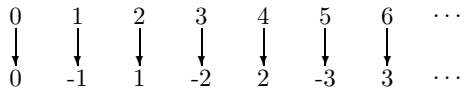



Figura 1.4: Função bijetora de $\mathbf{N}$ para $\mathbf{Z}$.

Seja $P$ o conjunto dos naturais pares (incluindo 0)[17], um outro conjunto infinito. Qual conjunto é maior, $P$ ou $\mathbf{N}$? Por um lado, tem sentido dizer que $\mathbf{N}$ é maior, pois $\mathbf{N}$ contém todos os elementos de $P$ mais todos os naturais ímpares. Por outro lado, existe uma função bijetora $f : \mathbf{N} \to P$, onde $f(x) = 2x$. Assim, $P$ e $\mathbf{N}$ têm a mesma cardinalidade. □

Um conjunto é *enumerável*[18] se tem a mesma cardinalidade que $\mathbf{N}$[19]. Um conjunto é dito *contável* se é finito ou enumerável. Neste texto só serão estudados conjuntos contáveis, principalmente enumeráveis.

O exemplo 25 mostra que o conjunto dos naturais pares é enumerável. De forma análoga pode-se mostrar que o conjunto dos naturais ímpares é enumerável[20]. O seguinte exemplo mostra que um conjunto aparentemente maior que $\mathbf{N}$ tem a mesma cardinalidade que $\mathbf{N}$.

**Exemplo 26** A função $f : \mathbf{N} \to \mathbf{Z}$ tal que

$$f(x) = \begin{cases} x/2 & \text{se } x \text{ é par} \\ -(x+1)/2 & \text{se } x \text{ é ímpar} \end{cases}$$

demonstra que $\mathbf{Z}$ tem a mesma cardinalidade que $\mathbf{N}$ e, portanto, é enumerável. A Figura 1.4 apresenta esquematicamente a relação entre os elementos de $\mathbf{N}$ e de $\mathbf{Z}$ dada pela função $f$. □

A relação de cardinalidade é uma relação reflexiva, simétrica e transitiva, e, portanto, uma relação de equivalência. Assim, por exemplo, do fato de que $card(\mathbf{N}) = card(P)$, onde $P$ é o conjunto dos naturais pares, e do fato de que $card(\mathbf{N}) = card(\mathbf{Z})$, segue-se que $card(\mathbf{Z}) = card(P)$.

O seguinte teorema pode facilitar a demonstração de que determinados conjuntos são contáveis.

**Teorema 1** *As seguintes afirmativas são equivalentes:*

1. *O conjunto $A$ é contável.*
2. *Existe uma função injetora de $A$ para $\mathbf{N}$.*
3. *$A = \emptyset$ ou existe uma função sobrejetora de $\mathbf{N}$ para $A$.* □

[17] Neste texto, zero será considerado como um número par.
[18] *Denumerable* ou *countably infinite*.
[19] A cardinalidade de $\mathbf{N}$ é denotada por $\aleph_0$ ($\aleph$ é a primeira letra do alfabeto hebraico.)
[20] Isto implica que $\aleph_0 + \aleph_0 = \aleph_0$.

| | 1 | 2 | 3 | 4 | 5 | ⋯ |
|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 3 | 6 | 10 | |
| 1 | 2 | 4 | 7 | 11 | | |
| 2 | 5 | 8 | 12 | | | |
| 3 | 9 | 13 | | | | |
| 4 | 14 | | | | | |
| ⋮ | | | | | | |

Figura 1.5: Função sobrejetora de $\mathbf{N}$ para $QP$.

Segue uma aplicação do teorema 1

**Exemplo 27** O conjunto dos números racionais não negativos, $QP$, é enumerável, como mostra a função sobrejetora $g : \mathbf{N} \to QP$, que será definida de forma a espelhar a correspondência mostrada esquematicamente na Figura 1.5. Nesta figura, os números naturais são dispostos em uma matriz infinita com as linhas numeradas 0, 1, 2, ... e as colunas 1, 2, 3, ... Observe que, começando na posição (0,1) da matriz, os naturais são dispostos nas diagonais inversas, em seqüência, a partir de 0. Com este esquema, *todo natural* terá uma posição na matriz. Supondo que o par (linha $i$, coluna $j$) represente o número racional $i/j$, observe que cada racional possui múltiplas representações. Por exemplo, o número 0 tem inúmeras representações: 0/1, 0/2, 0/3, etc. O número 1: 1/1, 2/2, etc. A matriz representa uma função $g$ tal que $g(k) = i/j$, onde $k$ é o natural especificado na linha $i$, coluna $j$. Isto é suficiente para concluir que o conjunto $QP$ é enumerável[21].

Antes de determinar $g$, será determinada a função $f : \mathbf{N} \times \mathbf{N} - \{0\} \to \mathbf{N}$, onde $f(i,j) = k$ se $g(k) = i/j$. Observe, inicialmente, que $f(i,j) = S + i$, onde $S$ é o primeiro natural da diagonal inversa em que se situa $k$ (na linha 0). Para os elementos de uma diagonal inversa, $i + j$ é constante e é exatamente a quantidade de números naturais dispostos na diagonal. Assim, $S$ é a quantidade de números naturais já colocados antes da diagonal, que é $\sum_{k=0}^{i+j-1} k = (i+j)(i+j-1)/2$. Logo, $f(i,j) = (i+j)(i+j-1)/2 + i$. A inversa desta função dá o par $(i,j)$ para o qual $g(k) = i/j$, para cada natural $k$. Dado um número $k$, deve-se, assim, determinar $i$ e $j$ tais que $k = (i+j)(i+j-1)/2 + i$, ou seja, $2k - 2i = (i+j)(i+j-1)$. Veja que $i + j$ deve ser o *maior número natural tal que* $(i+j)(i+j-1) \leq 2k$. Assim, designando-se por $n$ o maior número natural tal que $n(n-1) \leq 2k$, determina-se $i$ através de $i = k - n(n-1)/2$ e, em seguida, determina-se $j$ através de $j = n - i$. □

Além do Teorema 1, os seguintes resultados também podem ser úteis para determinar se um conjunto é ou não contável:

(a) Todo subconjunto de conjunto contável é contável.

(b) $A \times B$ é contável, se $A$ e $B$ são contáveis.

(c) $A \cup B$ é contável, se $A$ e $B$ são contáveis.

[21] Deste exemplo segue que $\mathbf{N} \times \mathbf{N}$ é enumerável e, com isto, que $\aleph_0 \times \aleph_0 = \aleph_0$.

| | 0 | 1 | 2 | 3 | $\cdots$ |
|---|---|---|---|---|---|
| $C_0$ | $\boxed{\in \text{ ou } \notin}$ | $\in$ ou $\notin$ | $\in$ ou $\notin$ | $\in$ ou $\notin$ | |
| $C_1$ | $\in$ ou $\notin$ | $\boxed{\in \text{ ou } \notin}$ | $\in$ ou $\notin$ | $\in$ ou $\notin$ | |
| $C_2$ | $\in$ ou $\notin$ | $\in$ ou $\notin$ | $\boxed{\in \text{ ou } \notin}$ | $\in$ ou $\notin$ | |
| $C_3$ | $\in$ ou $\notin$ | $\in$ ou $\notin$ | $\in$ ou $\notin$ | $\boxed{\in \text{ ou } \notin}$ | |
| $\vdots$ | | | | | |

Figura 1.6: Matriz de diagonalização para subconjuntos de $\mathbf{N}$.

A seguir é apresentado um exemplo de conjunto não contável. Tal conjunto, em certo sentido, tem tantos elementos que é impossível construir uma função bijetora dos naturais para ele. A técnica utilizada na demonstração é conhecida como *diagonalização de Cantor*.

**Exemplo 28** Seja $S$ o conjunto de todos os subconjuntos de $\mathbf{N}$. Será mostrado, por contradição, que $S$ não é contável.

Suponha que $S$ é contável. Como $S$ é infinito, seja então uma função bijetora $f : \mathbf{N} \to S$ tal que $f(i) = C_i$; ou seja, a função $f$ "enumera" os elementos de $S$: $S = \{C_0, C_1, C_2, \ldots\}$. Observe que está-se assumindo apenas a *existência* de $f$, sem impor uma função $f$ específica. O fundamental é que cada subconjunto de $\mathbf{N}$ deve ser algum $C_i$ para algum $i \in \mathbf{N}$. Seja, então, o conjunto $X$ tal que para cada $i \in \mathbf{N}$, $i \in X$ se, e somente se, $i \notin C_i$. Ora, tal conjunto é um subconjunto de $\mathbf{N}$ e, no entanto, é diferente de cada conjunto $C_i$. Isto implica que a suposição da existência da função bijetora $f$ é incorreta, ou seja, $S$ não é contável. □

É ilustrativo ver em que consiste o argumento da *diagonalização* utilizado no exemplo anterior. Para isto, observe que se pode dispor a seqüência de conjuntos $C_0, C_1, C_2, \ldots$ como índices das linhas de uma matriz e a seqüência de naturais $0, 1, 2, \ldots$ como índices das colunas, como mostrado na Figura 1.6. O elemento na posição $(C_i, k)$ da matriz é "$\in$" se $k \in C_i$, e é "$\notin$" se $k \notin C_i$. O conjunto $X$ do Exemplo 28 é formado observando-se a *diagonal* da matriz: $i \in X \leftrightarrow i \notin C_i$. Diz-se, então, que $X$ difere de $C_i$ na diagonal, para cada $i \in \mathbf{N}$.

**Exemplo 29** Seja o conjunto de todas as funções $f : \mathbf{N} \to \mathbf{N}$. Será mostrado, por contradição, usando a técnica da diagonalização, que tal conjunto não é contável.

Suponha que o conjunto em questão é enumerável (sabe-se que não é finito). Então, as funções podem ser enumeradas em seqüência $f_0, f_1, f_2, \ldots$. (Veja a Figura 1.7, que explicita a matriz para o argumento da diagonalização.) Considere agora a função $g : \mathbf{N} \to \mathbf{N}$ tal que $g(i) = f_i(i) + 1$. Mas $g$ é diferente de $f_i$ para todo $i \in \mathbf{N}$, pois $g$ difere de $f_i$ para o argumento $i$. Logo, a suposição de que o conjunto das funções $f : \mathbf{N} \to \mathbf{N}$ é enumerável não é correta. □

Neste ponto, é conveniente retornar ao problema de representar uma entidade já modelada matematicamente por meio de um número, ou um conjunto, ou uma função,

| | 0 | 1 | 2 | 3 | $\cdots$ |
|---|---|---|---|---|---|
| $f_0$ | $\boxed{f_0(0)}$ | $f_0(1)$ | $f_0(2)$ | $f_0(3)$ | |
| $f_1$ | $f_1(0)$ | $\boxed{f_1(1)}$ | $f_1(2)$ | $f_1(3)$ | |
| $f_2$ | $f_2(0)$ | $f_2(1)$ | $\boxed{f_2(2)}$ | $f_2(3)$ | |
| $f_3$ | $f_3(0)$ | $f_3(1)$ | $f_3(2)$ | $\boxed{f_3(3)}$ | |
| $\vdots$ | | | | | |

Figura 1.7: Matriz de diagonalização para funções $\mathbf{N} \to \mathbf{N}$.

ou uma relação, ou um grafo, etc., via uma seqüência de símbolos finita, problema este mencionado na Seção 1.1, que é um problema básico da computação.

Um número natural pode ser representado na base 1 (unário), 2 (binário), etc. Do ponto de vista tecnológico, pelo menos até agora, a base mais conveniente tem sido a base 2. Do ponto de vista matemático, no entanto, qualquer base serve. Por exemplo, para representar um número natural $n$ na base 1, pode-se usar uma seqüência de $n+1$ símbolos. Pode-se determinar que, para representar um número natural $n$ na base $b$, para $n \geq 2$, usa-se uma seqüência de $\lfloor \log_b n \rfloor + 1$ símbolos. Assim, vê-se que a representação de um número em uma base maior que 1 é exponencialmente mais concisa que a representação em unário.

Já os números reais não podem ser todos representados em qualquer base, já que os números irracionais não podem ser representados. Isto segue do fato de que o conjunto dos números reais não é enumerável: se cada número real pudesse ser representado por uma seqüência de símbolos finita, então, dado que os naturais também podem, seria possível ter uma função bijetora dos naturais para os reais. Uma implicação disto, é que nenhum computador, mesmo com memória ilimitada, pode manipular todos os números reais. Como os computadores têm memória limitada, eles sequer conseguem manipular todos os números racionais; na realidade eles só conseguem manipular um subconjunto dos racionais, normalmente denominados de números de ponto flutuante.

Generalizando, se um conjunto não é contável, então seus elementos não podem ser (todos) representados por sequências finitas de símbolos tomados de um conjunto finito de símbolos. Por outro lado, se um conjunto é contável, seus elementos podem ser (todos) representados por sequências finitas de símbolos tomados de um conjunto finito de símbolos.

# Exercícios

1. Para cada um dos conjuntos abaixo, prove que ele é, ou que não é, enumerável:

   (a) $\{n \in \mathbf{N} \mid n \, mod 10 = 0\}$.

   (b) $\{(n_1, n_2, n_3) \mid n_1, n_2, n_3 \in \mathbf{N}\}$.

   (c) $\{n \in \mathbf{R} \mid 0 < n < 1\}$.

2. O conjunto das funções de $\mathbf{N}$ para $\{0,1\}$ pode ser visto como o conjunto de todos os problemas de decisão (problemas cuja resposta é sim ou não)[22]. Prove que tal conjunto não é enumerável.

3. Uma função total $f : \mathbf{N} \to \mathbf{N}$ é dita *monotônica crescente* se $f(n+1) > f(n)$ para todo $n \in \mathbf{N}$. Prove que o conjunto das funções monotônicas crescentes não é contável.

4. Mostre que todo subconjunto de um conjunto enumerável é contável.

5. Mostre que a união, a interseção e o produto cartesiano de dois conjuntos contáveis são conjuntos contáveis.

6. Sejam $F$ um conjunto finito e $E$ um conjunto enumerável. O conjunto das funções totais $f : F \to E$ é enumerável?

## 1.7 Definições Recursivas

Uma propriedade importante dos conjuntos enumeráveis é que eles podem ser definidos através de uma *definição recursiva* (ou *indutiva*). Uma definição recursiva especifica como um conjunto contável pode ser *gerado* a partir de um subconjunto do mesmo aplicando-se determinadas *operações* um número finito de vezes. Uma definição recursiva de um conjunto $A$ consta de três partes:

(a) base: especificação de um conjunto base $B \subset A$;

(b) passo recursivo: especificação de um elenco de operações que, se aplicadas a elementos de $A$, geram elementos de $A$;

(c) fechamento: afirmação que os únicos elementos de $A$ são aqueles que podem ser obtidos a partir dos elementos de $B$ aplicando-se um número finito de vezes as operações especificadas em (b).

O conjunto $B$ deve ser um conjunto contável. Ele pode, inclusive, ter sido definido recursivamente.

**Exemplo 30** O conjunto $\mathbf{N}$ pode ser definido assim, a partir de $\{0\}$, usando-se a operação $s$ (sucessor):

(a) $0 \in \mathbf{N}$;

(b) se $n \in \mathbf{N}$, então $s(n) \in \mathbf{N}$;

(c) só pertence a $\mathbf{N}$ o número que pode ser obtido de acordo com (a) e (b). □

De forma equivalente, pode-se omitir o item (c) de uma definição recursiva e dizer que o conjunto definido é *o menor conjunto* que pode ser obtido por meio de (a) e (b). Neste texto, o item (c) não será explicitado, mas suposto implicitamente quando se disser que o conjunto é definido *recursivamente* por (a) e (b).

Funções também podem ser definidas recursivamente; afinal, funções são conjuntos!

[22] Uma introdução a problemas de decisão será feita na Seção 1.12.

**Exemplo 31** A função fatorial, $fat : \mathbf{N} \to \mathbf{N}$ é definida recursivamente por:

(a) $fat(0) = 1$;

(b) $fat(n) = n \times fat(n-1)$, para $n \geq 1$. □

Evidentemente, a definição do exemplo anterior poderia ser colocada no formato apresentado no início desta seção:

(a) $(0, 1) \in fat$;

(b) se $n \geq 1$ e $(n-1, k) \in fat$, então $(n, nk) \in fat$;

(c) só pertence a $fat$ o par que pode ser obtido conforme (a) e (b).

No estilo do Exemplo 31, que será adotado aqui, o passo $(b)$ da definição mostra como obter o valor $f(n_1, n_2, \ldots, n_k)$ a partir de valores "mais simples", isto é, de valores $f(n'_1, n'_2, \ldots, n'_k)$ tais que pelo menos um $n'_j$ é menor que $n_j$ e nenhum $n'_j$ é maior que $n_j$. Segue mais um exemplo.

**Exemplo 32** Utilizando a representação de número natural dada pela definição recursiva do Exemplo 30, onde a representação de um número $n > 0$ é dada por $s(s(\cdots s(0)\cdots))$, onde aparecem $n$ $s$'s, é apresentada a seguir uma definição recursiva da operação de soma sobre $\mathbf{N}$:

(a) $n + 0 = n$, para todo $n \in \mathbf{N}$;

(b) $m + s(n) = s(m + n)$, para todo $n \in \mathbf{N}$.

Observe que a soma $m + s(n)$ é obtida a partir da soma "mais simples" $m + n$. A partir do seguinte trecho dá para perceber como uma soma é "construída" (colchetes são usados ao invés de alguns parênteses apenas para efeitos de maior legibilidade):

$$\begin{aligned} s(0) + s(s(s(0))) &= s[s(0) + s(s(0))] \text{ por (b)} \\ &= s[s[s(0) + s(0)]] \text{ por (b)} \\ &= s[s[s[s(0) + 0]]] \text{ por (b)} \\ &= s[s[s[s(0)]]] \text{ por (a)} \end{aligned}$$

□

Antecipando um pouco do assunto a ser tratado na Seção 1.10, segue um exemplo de definição recursiva da sintaxe de uma linguagem.

**Exemplo 33** A seguir será definida uma (sintaxe de uma) linguagem para a Lógica Proposicional, LP, ou seja, a parte da Lógica Matemática vista informalmente na Seção 1.2, excluídos os conectivos $\forall$ e $\exists$. Para isto, será suposto um conjunto de *variáveis proposicionais* para expressar afirmativas primitivas (indivisíveis ou não compostas) constituído dos símbolos: $p$, $q$ e $r$, com ou sem índices. Para os conectivos serão utilizados os símbolos apresentados na Seção 1.2. E serão também utilizados os símbolos auxiliares "(" e ")". A linguagem LP é, então, definida recursivamente assim:

(a) cada variável proposicional pertence a LP;

(b) se $\alpha$ e $\beta$ pertencem a LP, então também pertencem a LP:

- $\neg\alpha$;
- $(\alpha \wedge \beta)$;
- $(\alpha \vee \beta)$;
- $(\alpha \rightarrow \beta)$;
- $(\alpha \leftrightarrow \beta)$.

Exemplos de afirmativas pertencentes a LP: $p$, $q$, $\neg p$, $(p \rightarrow q)$, $((p \rightarrow q) \wedge \neg p)$. As duas primeiras foram obtidas da base da definição, a terceira e a quarta aplicando-se o passo recursivo da definição uma única vez, e a última aplicando-se o passo recursivo a partir da terceira e da quarta. Observe que, por esta definição, *não* são exemplos de afirmativas pertencentes a LP: $pq$, $p \wedge q$, $(p)$, $\neg(p)$. Não há como gerar tais seqüências a partir das variáveis proposicionais aplicando-se as regras de composição de $(b)$. □

Diversos outros exemplos de definições recursivas serão vistos nas próximas seções e capítulos.

# Exercícios

1. Dê uma definição recursiva de $f : \mathbf{N} \rightarrow \mathbf{N}$, onde $f(n) = \sum_{k=1}^{n} k$.

2. Faça uma definição recursiva do número de elementos do conjunto potência de conjuntos finitos.

3. Faça uma definição recursiva das representações dos números naturais na base 2, sem zeros à esquerda, de forma que cada número tenha uma única representação.

4. Faça uma definição recursiva dos números da série de Fibonacci: 0, 1, 1, 2, 3, 5, 8, 13, ...

5. Continuando no estilo do Exemplo 32, página 27, faça uma definição recursiva da operação de multiplicação sobre $\mathbf{N}$.

6. No Exemplo 33, foi apresentada uma sintaxe muito simples e conveniente do ponto de vista formal, mas que não é adequada na prática, pois o uso exaustivo de parênteses leva a afirmativas longas e ilegíveis. O procedimento usual para resolver esse problema é assumir prioridades para os conectivos e utilizar as leis associativas da conjunção e da disjunção para omitir parênteses. Por outro lado, a colocação de parênteses em excesso às vezes é tolerado. Faça uma definição recursiva para uma linguagem LP$'$ em que parênteses podem ser omitidos, mas que também podem ser colocados em excesso. (Observe que a interpretação de uma afirmativa existe à parte da mesma; a definição recursiva deverá gerar apenas as afirmativas *sintaticamente* aceitáveis, sem preocupação com as regras de prioridades dos conectivos, que existirão à parte.)

## 1.8 Indução Matemática

A maioria dos resultados a serem apresentados nos capítulos vindouros serão provados mediante evocação do denominado *princípio de indução matemática*. Tal princípio espelha a definição recursiva dos números naturais, como se pode observar a seguir.

> Princípio de indução matemática: *Seja uma propriedade $P$ sobre os naturais. Então, caso*
>
> - *$P$ se verifique para o número 0, e*
> - *para um natural $n$ arbitrário, se $P$ se verifica para $n$, então $P$ se verifica para $n+1$,*
>
> *pode-se concluir que $P$ se verifica para todo número natural.*[23]

Utilizando-se tal princípio pode-se provar, *por indução sobre* $n$, que uma propriedade $P$ se verifica para todo número $n \in \mathbf{N}$, em três passos:

(1) *base* da indução: provar que $P$ se verifica para o número zero;

(2) *hipótese de indução*: supor que $P$ se verifica para $n$, onde $n$ é um número natural arbitrário; e

(3) *passo indutivo*: provar que $P$ se verifica para $n+1$.

**Exemplo 34** Um resultado frequentemente utilizado por quem trabalha em computação é o fato de que, para todo $n \in \mathbf{N}$, $\sum_{k=0}^{n} k = n(n+1)/2$, resultado este que já foi utilizado no Exemplo 27, página 23. Segue uma prova do mesmo por indução sobre $n$.

Inicialmente, veja que $\sum_{k=0}^{0} k = 0 = 0(0+1)/2$. Suponha, como hipótese de indução, que $\sum_{k=0}^{n} k = n(n+1)/2$ para um número natural $n$ arbitrário. Basta provar, então, que $\sum_{k=0}^{n+1} k = (n+1)(n+2)/2$. Ora, $\sum_{k=0}^{n+1} k = \sum_{k=0}^{n} k + (n+1) = n(n+1)/2 + (n+1)$, pela hipótese de indução. Desenvolvendo: $n(n+1)/2 + (n+1) = [n(n+1) + 2(n+1)]/2 = (n+1)(n+2)/2$. Logo, pelo princípio da indução, $\sum_{k=0}^{n} k = n(n+1)/2$. □

Vale ressaltar que o princípio da indução é a base para se provar uma afirmativa aplicável a todos os elementos de um conjunto enumerável, já que, como existe uma função bijetora dos naturais para tal conjunto, pode-se "transformar" uma afirmativa sobre os elementos do conjunto enumerável infinito em uma afirmativa sobre os naturais (ou vice-versa). Só que, normalmente, ao invés de fazer a transformação, raciocina-se com a afirmativa original, adaptando-se o princípio de indução. Assim, por exemplo, para provar que $P$ se verifica para todo natural $n \geq k$, basta:

(1) *base* da indução: provar que $P$ se verifica para o número k;

(2) *hipótese de indução*: supor que $P$ se verifica para $n$, sendo $n \geq k$ arbitrário; e

(3) *passo indutivo*: provar que $P$ se verifica para $n+1$.

[23] Mais formalmente: $[P(0) \wedge \forall n(P(n) \rightarrow P(n+1))] \rightarrow \forall n P(n)$.

(Neste caso, a transformação seria trivial: uma afirmativa sobre $n$, onde $n \geq k$, é o mesmo que uma afirmativa sobre $n + k$, onde $n \geq 0$). Segue um exemplo.

**Exemplo 35** Segue uma demonstração, por indução sobre $n$, que $n! > 2^n$ para todo $n \geq 4$.

Inicialmente, para $n = 4$ tem-se: $n! = 24 > 16 = 2^n$. Seja um $n \geq 4$ arbitrário, e suponha, como hipótese de indução, que $n! > 2^n$. Deduz-se:

$$\begin{array}{ll} (n+1)! = (n+1) \times n! & \text{pela definição de } \textit{fatorial} \\ > (n+1) \times 2^n & \text{pela hipótese de indução, pois } n+1 > 0 \\ > 2 \times 2^n & \text{pois } n \geq 4 \\ = 2^{n+1}. & \end{array}$$

Logo, $(n + 1)! > 2^{n+1}$. Conclui-se que $n! > 2^n$ para todo $n \geq 4$. □

Existe uma versão do princípio de indução, e consequente formato de prova por indução que pode ser mais fácil e/ou conveniente de ser usada em algumas circunstâncias. Tal versão, chamada de princípio de indução *forte*, diz que, caso

- para um natural arbitário $n$, se $P$ se verifica para todo $k < n$, então $P$ se verifica para $n$,

pode-se concluir que $P$ se verifica para todo número natural.[24] Uma prova por indução baseada neste princípio teria os passos:

(1) *hipótese de indução*: supor que $P$ se verifica para todo $k < n$, onde $n$ é um número natural arbitrário; e

(2) *passo indutivo*: provar que $P$ se verifica para $n$.

**Exemplo 36** Seja o problema de provar que todo número natural $n \geq 2$ é primo ou produto de números primos.

A prova será feita por indução forte sobre $n$. Para este efeito, seja $n \geq 2$ arbitrário e suponha, como hipótese de indução, que todo número natural $2 \leq k < n$ é primo ou produto de números primos. Basta, então, provar que $n$ é primo ou produto de primos. Se $n$ é um número primo, a afirmativa é trivialmente verdadeira. Caso contrário, por definição de número primo, $n = i \times j$, sendo $2 \leq i, j < n$. Neste caso, pela hipótese de indução, ambos, $i$ e $j$, são primos ou produtos de números primos. Conclui-se que $n$ é primo ou produto de números primos. Logo, pelo princípio de indução, todo número natural $n \geq 2$ é primo ou produto de números primos. □

O exemplo a seguir ilustra a aplicação do princípio de indução a entidades que não envolvem diretamente os números naturais.

**Exemplo 37** Seja o conjunto das afirmativas da linguagem LP definido recursivamente no Exemplo 33, página 27. Seja $\text{na}(\alpha)$ o número de abre parênteses e $\text{nf}(\alpha)$ o número de fecha parênteses da afirmativa $\alpha$. Será provado, por indução que $\text{na}(\alpha) = \text{nf}(\alpha)$ para todo $\alpha \in LP$.

Seja o *grau* de uma afirmativa o número de conectivos lógicos da mesma, o qual pode ser definido recursivamente assim:

[24] Mais formalmente: $\forall n[(\forall k < n P(k)) \rightarrow P(n)] \rightarrow \forall n P(n)$.

- o grau de uma variável proposicional é zero;
- o grau de $(\alpha \oplus \beta)$ é um a mais que a soma dos graus de $\alpha$ e $\beta$, onde $\oplus \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$.
- o grau de $\neg\alpha$ é um a mais que o grau de $\alpha$.

Será feita indução (forte) sobre o grau das afirmativas. Seja $n$ um natural arbitrário e suponha, como hipótese de indução, que $\text{na}(\alpha) = \text{nf}(\alpha)$ para todo $\alpha$ de grau menor que $n$. Basta, então, mostrar que $\text{na}(\alpha) = \text{nf}(\alpha)$ para todo $\alpha$ de grau $n$. Considera-se dois casos:

*Caso 1:* $n = 0$. Neste caso, $\alpha$ é uma variável proposicional e, portanto, $\text{na}(\alpha) = 0 = \text{nf}(\alpha)$.

*Caso 2:* $n > 0$. Este caso pode ser subdividido em dois:

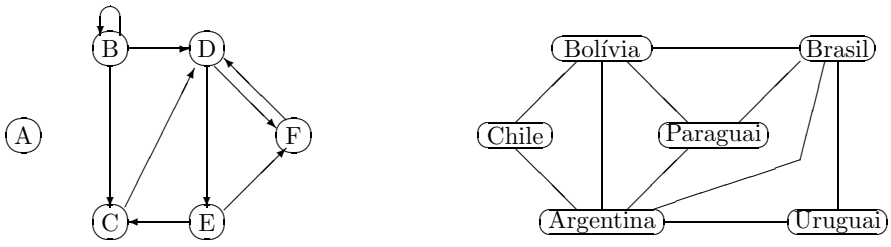
2.1 $\alpha = \neg\gamma$. O grau de $\gamma$ é $n - 1$ e, portanto, pela hipótese de indução, $\text{na}(\gamma) = \text{nf}(\gamma)$. Segue-se que $\text{na}(\alpha) = \text{nf}(\alpha)$, pois $\text{na}(\alpha) = \text{na}(\gamma)$ e $\text{nf}(\alpha) = \text{nf}(\gamma)$.

2.2 $\alpha = (\gamma_1 \oplus \gamma_2)$. Os graus de $\gamma_1$ e de $\gamma_2$ são menores que $n$ e, portanto, pela hipótese de indução, $\text{na}(\gamma_1) = \text{nf}(\gamma_1)$ e $\text{na}(\gamma_2) = \text{nf}(\gamma_2)$. Segue-se que $\text{na}(\alpha) = \text{nf}(\alpha)$, pois $\text{na}(\alpha) = \text{na}(\gamma_1) + \text{na}(\gamma_2) + 1$ e $\text{nf}(\alpha) = \text{nf}(\gamma_1) + \text{nf}(\gamma_2) + 1$.

Segue-se que $\text{na}(\alpha) = \text{nf}(\alpha)$ para todo $\alpha \in LP$. □

# Exercícios

1. Prove por indução que $|\mathcal{P}(A)| = 2^{|A|}$ para todo conjunto finito $A$.
2. Prove por indução que, para todo número natural $n \geq 0$:
   (a) $\sum_{k=0}^{n} k^2 = n(n+1)(2n+1)/6$.
   (b) $\sum_{k=0}^{n} k^3 = [n(n+1)/2]^2$.
   (c) $2^{2n} - 1$ é divisível por 3.
   (d) $n^3 - n$ é divisível por 6.
   (e) $7^n - 1$ é divisível por 6.
3. Prove por indução que, para todo número natural $n \geq 1$:
   (a) $\sum_{k=1}^{n} [k(k+1)] = n(n+1)(n+2)/3$.
   (b) $\sum_{k=1}^{n} 2^k = 2(2^n - 1)$.
   (c) $\sum_{k=1}^{n} [1/k(k+1)] = n/(n+1)$.
   (d) $n^3 + (n+1)^3 + (n+2)^3$ é divisível por 9.
4. Seja $F$ a função de Fibonacci definida recursivamente assim:
   (a) $F(0) = 0$; $F(1) = 1$;
   (b) $F(n) = F(n-1) + F(n-2)$ para $n \geq 2$.




(a) Um grafo dirigido. (b) Um grafo não dirigido.

Figura 1.8: Exemplos de grafos dirigido e não dirigido.

Prove por indução que

$$F(n) = \frac{1}{\sqrt{5}}\left[\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right].$$

5. Prove que o número de abre parênteses de qualquer prefixo de qualquer afirmativa da linguagem LP do Exemplo 33, página 27, é maior ou igual ao número de fecha parênteses.

## 1.9 Grafos

Um grafo é uma estrutura matemática que contém dois tipos de entidades: vértices e arestas. Existem dois tipos básicos de grafos: os dirigidos e os não dirigidos. Nos grafos dirigidos as arestas (dirigidas) são pares ordenados de vértices, e nos grafos não dirigidos as arestas (não dirigidas) são pares não ordenados de vértices.

Na representação gráfica de um grafo, um vértice é, em geral, representado por meio de uma curva fechada, como um círculo, uma oval, etc. Em um grafo dirigido, uma aresta é representada por meio de uma seta ligando as representações dos dois vértices da aresta no sentido do primeiro para o segundo vértice. E em um grafo não dirigido, uma aresta é representada por meio de uma linha (reta ou curva) ligando as representações dos dois vértices da aresta. As Figuras 1.8(a) e (b) mostram exemplos de representações gráficas de um grafo dirigido e de um grafo não dirigido. No grafo não dirigido da Figura 1.8(b) existe uma aresta conectando dois vértices $v$ e $v'$ se, e somente se, o país $v$ tem fronteira com o país $v'$.

Sintetizando, um grafo é um par $G = (V, A)$, onde $V$ é um conjunto de vértices e $A$ é um conjunto de arestas. Se $G$ é dirigido, $A$ é um conjunto de pares ordenados de elementos de $V$; e se $G$ não é dirigido, $A$ é um conjunto de pares não ordenados de elementos de $V$. Assim, o grafo dirigido da Figura 1.8(a) é um par $(V, A)$, onde $V = \{\mathrm{A}, \mathrm{B}, \mathrm{C}, \mathrm{D}, \mathrm{E}, \mathrm{F}\}$ e $A = \{(\mathrm{B}, \mathrm{B}),\ (\mathrm{B}, \mathrm{C}),\ (\mathrm{B}, \mathrm{D}),\ (\mathrm{C}, \mathrm{D}),\ (\mathrm{D}, \mathrm{E}),\ (\mathrm{D}, \mathrm{F}),\ (\mathrm{E}, \mathrm{C}),\ (\mathrm{E}, \mathrm{F}),\ (\mathrm{F}, \mathrm{D})\}$. E o grafo não dirigido da Figura 1.8(b) é um par $(V', A')$, onde $V' = \{$Argentina, Bolívia, Brasil, Chile, Paraguai, Uruguai$\}$ e $A' = \{\{$Argentina, Bolívia$\}$, $\{$Argentina, Brasil$\}$, $\{$Argentina, Chile$\}$, $\{$Argentina, Paraguai$\}$, $\{$Argentina, Uruguai$\}$, $\{$Bolívia, Brasil$\}$, $\{$Bolívia, Chile$\}$, $\{$Bolívia, Paraguai$\}$, $\{$Brasil, Paraguai$\}$, $\{$Brasil, Uruguai$\}\}$.

Além dos dois tipos básicos de grafos exemplificados acima, existem variações. Por exemplo, existem os grafos mistos, que têm ambos os tipos de arestas, dirigidas e não

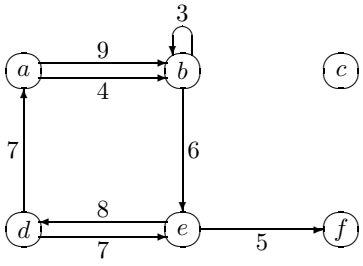



Figura 1.9: Exemplo de grafo dirigido rotulado.

dirigidas. Em alguns contextos, os grafos contêm, não um conjunto de arestas, mas um *multi-conjunto*[25]. Neste último caso, podem existir várias arestas para um único par de vértices.

Um grafo dirigido *rotulado* é uma tripla $(V, A, R)$ em que:

- $V$ é um conjunto finito de vértices;
- $A \subseteq V \times R \times V$ é um conjunto arestas (rotuladas); e
- $R$ é um conjunto de rótulos.

A representação gráfica é similar à de um grafo não rotulado. A diferença é que, para uma aresta $(a, r, b)$, coloca-se, além da seta de $a$ para $b$, o rótulo $r$ adjacente à seta, como mostra o exemplo a seguir. Observe que agora podem existir várias arestas que saem do mesmo vértice e entram em um vértice comum; basta que seus rótulos sejam diferentes. Um grafo não dirigido rotulado pode ser definido de forma análoga.

**Exemplo 38** A Figura 1.9 dá a representação gráfica de um grafo rotulado $(V, A, \mathbf{N})$ tal que:

- $V = \{a, b, c, d, e, f\}$;
- $A = \{(a, 9, b), (a, 4, b), (b, 3, b), (b, 6, e), (d, 7, a), (d, 7, e), (e, 8, d), (e, 5, f)\}$. □

Os vértices de uma aresta são ditos *adjacentes*. Para uma aresta $x = (a, b)$, em um grafo dirigido não rotulado, ou $x = (a, r, b)$, em um grafo dirigido rotulado, diz-se que $x$ *sai* do vértice $a$ e *entra* em $b$, e também que a aresta $x$ é uma aresta *de $a$ para $b$*. O *grau* de um vértice é o número de arestas que o contêm. Em um grafo dirigido, o *grau de entrada* de um vértice é o número de arestas que entram nele, e o grau de saída é o número de arestas que saem dele.

**Exemplo 39** Para o grafo da Figura 1.8(a), os graus de entrada e de saída de A são zero; o grau de entrada de B é 1 e o de saída de B é 3, sendo 4 o grau de B. No grafo da Figura 1.9, o grau de entrada de $b$ é 3 e o grau de saída é 2, sendo 5 o grau de $b$. Na Figura 1.8(b), o vértice Brasil tem grau 4. □

[25]Um multi-conjunto é um conjunto que admite repetições de elementos. Assim, por exemplo, $\{a\} \neq \{a, a\}$.

Um *caminho* de tamanho $n$ de $a$ para $b$, em um grafo dirigido, é uma seqüência de vértices e arestas $v_0x_1v_1x_2v_2 \ldots v_{n-1}x_nv_n$ tal que $a = v_0$, $b = v_n$ e a aresta $x_i$ sai de $v_{i-1}$ e entra em $v_i$; $v_0$ é o *vértice inicial* e $v_n$ é o *vértice final* do caminho. Neste caso, diz-se ainda que o caminho *passa* pelos vértices $v_0$, $v_1$, etc., e pelas arestas $x_1$, $x_2$, etc. Se o grafo não é rotulado, pode-se representar um caminho usando-se apenas os vértices, na forma $v_0v_1v_2 \ldots v_{n-1}v_n$, assumindo-se que há uma aresta de um vértice de $v_i$ para $v_{i+1}$ para $0 \leq i \leq n-1$. Quando $n = 0$, o caminho consta apenas do vértice $v_0 = a = b$, e é dito ser um *caminho nulo*. Um *caminho fechado* é um caminho não nulo em que os vértices inicial e final são o mesmo, isto é, $v_0 = v_n$. Um *ciclo* é um caminho fechado em que $v_i \neq v_j$ para todo $0 \leq i < j \leq n$, exceto para $i = 0$ e $j = n$, e em que cada aresta não ocorre mais de uma vez. Um ciclo de tamanho 1 é denominado *laço*. Um *caminho simples* é um caminho sem vértices repetidos. Define-se caminho, caminho nulo, caminho fechado, ciclo, laço e caminho simples para grafos não dirigidos de forma análoga.

**Exemplo 40** Seja o grafo da Figura 1.9. São exemplos de caminhos:

- $a$: é caminho nulo e é caminho simples;
- $a(a, 4, b)b(b, 3, b)b$: é um caminho de tamanho 2;
- $b(b, 3, b)b$: é um ciclo de tamanho 1 e, portanto, um laço;
- $a(a, 9, b)b(b, 6, e)e(e, 8, d)d(d, 7, a)a$: é um ciclo de tamanho 4;
- $d(d, 7, e)e(e, 8, d)d(d, 7, e)e(e, 8, d)d$: é um caminho fechado de tamanho 4; não é ciclo.

No grafo da Figura 1.8(b), Brasil Paraguai Argentina Chile é um caminho simples de tamanho 3; Brasil Bolívia Brasil é um caminho fechado, mas não é ciclo. Brasil Bolívia Argentina Brasil é um ciclo.

Note que o tamanho de um caminho é o número arestas do mesmo, que é igual ao número vértices menos um. □

Um grafo que não tem ciclos é dito ser um *grafo acíclico*.

Um grafo não dirigido é dito *conexo* se existe caminho de qualquer vértice a qualquer outro. E um grafo dirigido em que existe caminho de qualquer vértice a qualquer outro é dito ser *fortemente conexo*. Assim, os grafos das Figura 1.8a e 1.9 não são fortemente conexos, e o da Figura 1.8b é conexo.

Um tipo de grafo muito comum em computação é a *árvore*. Uma árvore pode ser definida como sendo um grafo acíclico conexo. Na maioria das aplicações, existe um vértice especial denominado *raiz*. Supondo que os vértices são tomados de um universo $U$, pode-se definir recursivamente árvore como sendo uma tripla $(V, A, r)$ tal que:

(a) $(\{v\}, \emptyset, v)$ é uma árvore para qualquer $v \in U$;

(b) se $(V, A, r)$ é uma árvore, $v \in V$ e $v' \in U - V$, então $(V \cup \{v'\}, A \cup \{\{v, v'\}\}, r)$ é uma árvore.

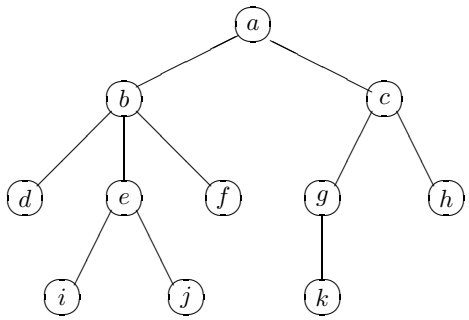



Figura 1.10: Exemplo de árvore com raiz.

Se $r$ é a raiz e $v$ um vértice qualquer de uma árvore, pode-se mostrar que existe um, e apenas um, caminho simples de $r$ para $v$. Se o caminho simples de $r$ para $v$ passa por um vértice $v'$ (que pode ser $r$ ou $v$), diz-se que $v'$ é *ancestral* de $v$ e que $v$ é *descendente* de $v'$. Neste caso, se $\{v', v\}$ é uma aresta da árvore, diz-se ainda que $v'$ é *o ancestral imediato*, ou *pai*, de $v$ e que $v$ é *um descendente imediato*, ou *filho*, de $v'$. Os filhos do mesmo pai são ditos *irmãos*. Um vértice sem filhos é denominado *folha* e um vértice que não é folha é chamado vértice *interno*. O *nível* de um vértice $v$ de uma árvore de raiz $r$ é o tamanho do caminho simples de $r$ para $v$. A *altura* de uma árvore é o maior dentre os níveis de seus vértices. Se o maior número de filhos de vértices de uma árvore é $n$, diz-se que a árvore é $n$-ária (binária, ternária, etc.).

Graficamente, uma árvore é, em geral, representada com a raiz no topo, os vértices adjacentes à raiz logo abaixo, os filhos destes últimos mais abaixo, etc.

**Exemplo 41** A Figura 1.10 apresenta a representação gráfica de uma árvore ternária com raiz $a$ e arestas $\{a, b\}$, $\{a, c\}$, $\{b, d\}$, $\{b, e\}$, $\{b, f\}$, $\{c, g\}$, $\{c, h\}$, $\{e, i\}$, $\{e, j\}$, $\{g, k\}$. São ancestrais de $f$: $f$, $b$ e $a$. O pai de $f$ é $b$ e o de $b$ é a raiz $a$. Os filhos de $b$ são $d$, $e$ e $f$. São descendentes de $b$: $b$, $d$, $e$, $f$, $i$ e $j$. As folhas da árvore são: $d$, $f$, $h$, $i$, $j$ e $k$. A raiz $a$ tem nível 0, $b$ e $c$ têm nível 1; $d$, $e$, $f$, $g$ e $h$ têm nível 2; e $i$, $j$ e $k$ têm nível 3, que é a altura da árvore. □

Em algumas aplicações pode ser útil considerar árvores dirigidas e/ou com rótulos. Tipicamente, em uma árvore dirigida uma aresta sempre sai do pai e entra em um filho, ou então sempre sai de um filho e entra no pai.

Em muitas aplicações são comuns o uso de árvores em que os vértices são rotulados, já que o uso dos próprios rótulos como vértices é impedido por existirem vértices diferentes com o mesmo rótulo. Neste caso, pode-se considerar a existência de uma função que associa a cada vértice o seu rótulo. Na representação gráfica, normalmente não se coloca os nomes dos vértices; coloca-se apenas os rótulos.

São comuns também aplicações em que os filhos de cada vértice são ordenados, ou seja, fala-se em *primeiro* filho (o mais à *esquerda*), *segundo*, etc. Neste último caso, diz-se que se trata de uma *árvore ordenada* (dirigida ou não). O ordenamento dos filhos de cada vértice induz um ordenamento dos vértices em geral, como definido a seguir. Para isto, define-se antes o conceito de *ancestral comum mínimo* de dois vértices, $v$ e $v'$, como sendo o ancestral, $u$, de ambos, $v$ e $v'$, tal que todo ancestral comum de $v$ e $v'$ é ancestral de $u$.

Tem-se, então, que um vértice $v$ *está à esquerda* de $v'$ se, e somente se, $v$ não é ancestral ou descendente de $v'$ e, além disto,

(a) $v$ e $v'$ são irmãos e $v$ está à esquerda de $v'$; ou

(b) sendo $u$ o ancestral de $v$ e $u'$ o ancestral de $v'$ que são filhos do ancestral comum mínimo de $v$ e $v'$, então $u$ está à esquerda de $u'$.

A *fronteira* de uma árvore ordenada é a seqüência das folhas na ordem "está à esquerda" que se acaba de definir.

**Exemplo 42** Suponha que a árvore representada graficamente na Figura 1.10 seja ordenada segundo a disposição apresentada na própria figura. Então, sendo $v$, $v'$, $u$ e $u'$ como na definição de "está à esquerda":

- $d$ está à esquerda dos irmãos $e$ e $f$; $e$ está à esquerda de $f$.
- O ancestral comum mínimo de $v = i$ e $v' = f$ é $b$; logo, $i$ está à esquerda de $f$, pois $u = e$ está à esquerda de $u' = f$.
- O ancestral comum mínimo de $v = e$ e $v' = k$ é $a$; logo, $e$ está à esquerda de $k$, pois $u = b$ está à esquerda de $u' = c$.
- A fronteira é $dijfkh$. □

# Exercícios

1. Mostre que todo grafo não dirigido possui um número par de vértices de grau ímpar.
2. Prove, por indução, que o número de arestas de uma árvore é igual ao número de vértices menos 1.
3. Sejam $v_1, v_2, \ldots, v_n$ os vértices de um grafo, e seja $grau(v_i)$ o grau do vértice $v_i$. Prove que $\sum_{i=1}^{n} grau(v_i) = 2k$, onde $k$ é o número de arestas do grafo.
4. Um grafo não dirigido é dito ser *completo* se há uma aresta para cada par de vértices distintos. Um grafo completo de $n$ vértices é denotado por $K_n$. Qual é o número de arestas de $K_n$?
5. Prove por indução que toda árvore binária de altura $k \geq 0$ possui, no máximo:
   (a) $2^k$ folhas.
   (b) $2^{k+1} - 1$ vértices.
6. Mostre que as seguinte afirmativas são equivalentes:
   (a) É uma árvore.
   (b) É acíclico e o número de vértices é um a mais do que o número de arestas.
   (c) Tem um único caminho simples de qualquer vértice para qualquer outro.

   *Sugestão*: prove que $(a) \rightarrow (b) \rightarrow (c) \rightarrow (a)$.

## 1.10 Linguagens Formais

Uma linguagem formal, ao contrário de uma linguagem natural, é tal que:

(a) tem uma sintaxe bem definida, de tal forma que, dada uma sentença, é sempre possível saber se ela pertence ou não à linguagem; e

(b) tem uma semântica precisa, de tal forma que não contém sentenças sem significado ou ambíguas.

As linguagens formais são úteis, não apenas na matemática, mas também nas áreas que utilizam a matemática como ferramenta, como, por exemplo, as Engenharias, a Física, a Química e a Computação. No caso da Computação, em particular, as linguagens formais têm uma importância ímpar, pois a maioria dos profissionais da área lidam diretamente com uma ou mais no dia a dia.

Exemplos de linguagens formais, ou concretizações diretas das mesmas, são as linguagens Java, C, Pascal, HTML, etc. Em princípio, se um programador ou analista projeta um programa ou sistema que envolve um diálogo com o usuário, ele tem o problema de projetar a linguagem (formal) de comunicação. Desde o nível de instruções de máquina até os níveis mais altos da programação de um computador, as linguagens formais são uma presença constante.

Nesta seção será vista uma definição de linguagem formal bastante geral, sem tocar na parte de semântica. À primeira vista isto pode parecer uma limitação, mas o fato é que uma abordagem puramente sintática é suficiente para caracterização do conceito de "computabilidade", um dos objetivos deste texto, assim como para servir como base para uma gama ampla de aplicações, como ficará claro no decorrer do livro. Além disto, a especificação e processamento da sintaxe de linguagens formais já envolve um material bastante extenso, que pode anteceder um estudo posterior de semântica. Por outro lado, muitas vezes consegue-se uma estrutura sintática rica o suficiente para capturar todos os aspectos relevantes da linguagem, não havendo a necessidade de considerar uma semântica à parte.

Toda *linguagem*[26] tem um *alfabeto* associado. Um alfabeto é um conjunto finito não vazio de elementos que serão referidos como *símbolos*. Uma *palavra*[27] sobre um alfabeto $\Sigma$ é uma seqüência finita de símbolos de $\Sigma$.[28] O *tamanho* de uma palavra $x$, $|x|$, é o número de símbolos que a compõem. Em particular, existe a *palavra vazia*, constituída de zero símbolos; tal palavra será designada por $\lambda$. Assim, $|\lambda| = 0$.

**Exemplo 43** Dois exemplos de alfabetos particularmente importantes são: $\Sigma = \{\mathtt{1}\}$ e $\Gamma = \{\mathtt{0}, \mathtt{1}\}$.

São palavras sobre $\Gamma$: $\lambda$, `0`, `1`, `00`, `01`, `10`, `11`, `000`, etc. Com este alfabeto pode-se *representar* qualquer número. Uma possibilidade é utilizar a codificação na base 2: 0

[26] Daqui para frente, quando se disser "linguagem", quer-se dizer "linguagem formal", a menos que se diga o contrário.

[27] *String*, *word*.

[28] Formalmente, uma seqüência é uma função $f : \{1, 2, 3, \ldots, n\} \to \Sigma$. Ela é representada normalmente por $f(1)f(2)f(3)\cdots f(n)$. Por exemplo, a seqüência $f : \{1, 2, 3\} \to \{0, 1\}$ tal que $f(1) = 1$, $f(2) = 0$ e $f(3) = 1$ é representada por 101.

é representado por uma infinidade de palavras: 0, 00, etc; 1 é representado por uma infinidade de palavras: 1, 01, etc.; apenas $\lambda$ não representa algum número.

São palavras sobre $\Sigma$: $\lambda$, 1, 11, 111, etc. Observe que com este alfabeto também consegue-se representar qualquer número natural! Por exemplo, basta representar o número $n$ por uma palavra de tamanho $n$. Neste caso, cada palavra $x$ representaria o número $|x|$. Comparando a representação do parágrafo anterior com a deste, note que a menor palavra $x$ usada para representar um número $n$ na base 2 tem tamanho $\lfloor \log_2 n \rfloor + 1$, e a palavra $y$ usada para representar o mesmo número $n$ na base 1 tem tamanho $n$. Assim, nesta segunda representação a palavra usada é exponencialmente maior. □

Seja $a$ um símbolo qualquer. A notação $a^n$, onde $n \in \mathbf{N}$, será utilizada para designar a palavra constituída de $n$ $a$'s em seqüência. Assim, por exemplo, dado o alfabeto $\{0, 1\}$, são exemplos de palavras sobre tal alfabeto: $1^0 = \lambda$, $0^4 = 0000$, $1^3 0 1^2 = 11101 1$, etc.

Uma linguagem sobre um alfabeto $\Sigma$ é um conjunto de palavras sobre $\Sigma$. Denotando o conjunto de todas as palavras sobre $\Sigma$ por $\Sigma^*$, diz-se, então, que uma linguagem sobre $\Sigma$ é qualquer subconjunto de $\Sigma^*$.

**Exemplo 44** Seja o alfabeto $\Sigma = \{0, 1\}$. O conjunto de todas as palavras sobre $\Sigma$ é $\Sigma^* = \{\lambda, 0, 1, 00, 01, 10, 11, 000, \ldots\}$. São exemplos de linguagens sobre $\Sigma$:

- $\emptyset$. É a linguagem mais simples que existe; não contém palavras.
- $\{\lambda\}$. Contém uma única palavra: a palavra vazia.
- $\{0\}$. Contém uma única palavra: 0.
- $\{\lambda, 0\}$. Contém duas palavras: $\lambda$ e 0.
- $\{w \in \Sigma^* \mid \ 1 \leq |w| \leq 5\}$. Contém $\sum_{i=1}^{5} 2^i$ palavras.
- $\{0^n \mid n$ é um número primo$\}$. Esta linguagem é infinita, já que existe uma infinidade de números primos.
- $\{0^n 1^n \mid n \in \mathbf{N}\}$. Linguagem constituída de toda palavra de tamanho par cuja primeira metade só contém 0's e cuja segunda metade só contém 1's.
- $\Sigma^*$. Contém todas as palavras sobre o alfabeto $\Sigma$. □

Assim como as três últimas linguagens do exemplo acima, a maioria das linguagens de interesse são infinitas. Como fazer para especificar tais linguagens, se não dá para listar explicitamente todas as suas palavras? Na verdade, como será visto oportunamente, existem muitas opções para isto, cada uma delas possuindo contextos em que é mais apropriada.

Como uma linguagem é um conjunto, pode-se lançar mãos das operações sobre conjuntos, definidas na Seção 1.3. Assim, por exemplo, se $L_1$ e $L_2$ são linguagens sobre alfabetos $\Sigma_1$ e $\Sigma_2$, respectivamente, também são linguagens:

- $L_1 \cup L_2$, uma linguagem sobre $\Sigma_1 \cup \Sigma_2$;

- $L_1 \cap L_2$, uma linguagem sobre $\Sigma_1 \cap \Sigma_2$;
- $L_1 - L_2$, uma linguagem sobre $\Sigma_1$.

Além destas, levando-se em consideração que os elementos que constituem as linguagens são as palavras, existem outras operações, tanto sobre palavras, quanto sobre linguagens, que podem auxiliar na especificação de uma linguagem.

A *concatenação* de duas palavras $x = a_1a_2 \ldots a_m$ e $y = b_1b_2 \ldots b_n$ é a palavra $xy = a_1a_2 \ldots a_mb_1b_2 \ldots b_n$. Em particular, note que $\lambda w = w\lambda = w$ para qualquer palavra $w$. Tal definição implica que a concatenação é uma operação associativa: $x(yz) = (xy)z$ para quaisquer palavras $x$, $y$ e $z$. Assim, uma seqüência de concatenações poderá ser escrita sem parênteses. O *reverso* de uma palavra $w = a_1a_2 \ldots a_n$, $w^R$, é a seqüência dos símbolos de $w$ na ordem reversa, isto é, $w^R = a_na_{n-1} \ldots a_1$. Uma palavra $w$ tal que $w = w^R$ é um *palíndromo*. Seja uma palavra $w = xzy$, onde $x$, $y$ e $z$ podem ser $\lambda$ ou não. A palavra $z$ é uma *subpalavra* de $w$, $x$ é um *prefixo* de $w$, e $y$ é um *sufixo* de $w$. Em particular, $\lambda$ é prefixo, sufixo e subpalavra de qualquer palavra, e $w$ é prefixo, sufixo e subpalavra de qualquer palavra $w$. Seguem exemplos destes conceitos.

**Exemplo 45** Seja o alfabeto $\Sigma = \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$. Todas as palavras exemplificadas abaixo são palavras sobre tal alfabeto.

Estes são alguns exemplos de concatenações: $\lambda(\mathtt{ab}) = (\mathtt{ab})\lambda = \mathtt{ab}$; $(\mathtt{abc})(\mathtt{aabb}) = \mathtt{abcaabb}$. Alguns reversos: $\lambda^R = \lambda$; $\mathtt{a}^R = \mathtt{a}$; $(\mathtt{abcaabb})^R = \mathtt{bbaacba}$. Os prefixos de `abc` são: $\lambda$, `a`, `ab` e `abc`. Os sufixos de `abc` são: $\lambda$, `c`, `bc` e `abc`. As subpalavras de `abc` são: $\lambda$, `a`, `b`, `c`, `ab`, `bc` e `abc`. Alguns palíndromos: $\lambda$, `a`, `bb`, `ccc`, `aba`, `baab`. □

A *concatenação* de duas linguagens $L_1$ e$L_2$ é dada por:

$$L_1L_2 = \{xy \mid x \in L_1 \text{ e } y \in L_2\}.$$

Em particular, $\emptyset L = L\emptyset = \emptyset$ e $\{\lambda\}L = L\{\lambda\} = L$, para qualquer linguagem $L$.

**Exemplo 46** Sejam as linguagens $L_1 = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid |w| = 5\}$ e a linguagem $L_2 = \{\mathtt{0}y \mid y \in \{\mathtt{0}, \mathtt{1}\}^*\}$. Então:

- $L_1L_1 = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid |w| = 10\}$;
- $L_1L_2 = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ o sexto símbolo de $w$ é $\mathtt{0}\}$;
- $L_2L_1 = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ $w$ começa com $\mathtt{0}$ e $|w| \geq 6\}$;
- $L_2L_2 = \{\mathtt{0}y \mid y \in \{\mathtt{0}, \mathtt{1}\}^*$ e $y$ contém no mínimo um $\mathtt{0}\}$. □

Seja $L$ uma linguagem. A notação $L^n$ será utilizada para designar $LL \ldots L$ ($n$ vezes). Recursivamente:

(a) $L^0 = \{\lambda\}$;

(b) $L^n = L^{n-1}L$ para $n \geq 1$.

A operação *fecho de Kleene* de uma linguagem $L$, $L^*$, pode ser definida recursivamente assim:

(a) $\lambda \in L^*$;

(b) se $x \in L^*$ e $y \in L$ então $xy \in L^*$.

A partir desta definição e da definição de concatenação de linguagens, pode-se verificar que:

$$L^* = \bigcup_{n \in \mathbf{N}} L^n.$$

Ou seja, $L^* = L^0 \cup L^1 \cup L^2 \cup \cdots = \{\lambda\} \cup L \cup LL \cup \cdots$. Define-se também o *fecho positivo de Kleene* de $L$: $L^+ = LL^*$. Pode-se verificar que:

$$L^+ = \bigcup_{n \in \mathbf{N} - \{0\}} L^n.$$

Segue diretamente destas definições que $L^* = L^+ \cup \{\lambda\}$.

**Exemplo 47** Seguem algumas linguagens e seus fechos de Kleene:

- $\emptyset^* = \{\lambda\}$, e $\emptyset^+ = \emptyset$;
- $\{\lambda\}^* = \{\lambda\}^+ = \{\lambda\}$;
- $\{\mathtt{0}\}^* = \{\mathtt{0}^n \mid n \in \mathbf{N}\}$ e $\{\mathtt{0}\}^+ = \{\mathtt{0}^n \mid n \geq 1\}$;
- $\{\lambda, \mathtt{00}, \mathtt{11}\}^* = \{\lambda, \mathtt{00}, \mathtt{11}\}^+ = \{\lambda\} \cup \{\mathtt{00}, \mathtt{11}\}^+$. □

Com as operações definidas nesta seção, pode-se expressar (ou definir) de forma precisa algumas linguagens, como exemplificado a seguir.

**Exemplo 48** Seguem descrições informais, em português, para algumas linguagens sobre $\{\mathtt{0}, \mathtt{1}\}$, assim como descrições mais formais utilizando as operações definidas nesta seção:

(a) O conjunto das palavras que começam com `0`: $\{\mathtt{0}\}\{\mathtt{0}, \mathtt{1}\}^*$.

(b) O conjunto das palavras que contêm `00` ou `11`: $\{\mathtt{0}, \mathtt{1}\}^*\{\mathtt{00}, \mathtt{11}\}\{\mathtt{0}, \mathtt{1}\}^*$.

(c) O conjunto das palavras que terminam com `0` seguido de um número ímpar de `1`'s consecutivos: $\{\mathtt{0}, \mathtt{1}\}^*\{\mathtt{01}\}\{\mathtt{11}\}^*$.

(d) O conjunto das palavras de tamanho par que começam com `0` ou terminam com `0`: $(\{\mathtt{0}, \mathtt{1}\}\{\mathtt{0}, \mathtt{1}\})^* \cap [\{\mathtt{0}\}\{\mathtt{0}, \mathtt{1}\}^* \cup \{\mathtt{0}, \mathtt{1}\}^*\{\mathtt{0}\}]$.

(e) Como (d): $[\{\mathtt{0}\}\{\mathtt{0}, \mathtt{1}\}(\{\mathtt{0}, \mathtt{1}\}\{\mathtt{0}, \mathtt{1}\})^*] \cup [\{\mathtt{0}, \mathtt{1}\}(\{\mathtt{0}, \mathtt{1}\}\{\mathtt{0}, \mathtt{1}\})^*\{\mathtt{0}\}]$.

(f) O conjunto das palavras com um prefixo de um ou mais `0`'s seguido (imediatamente) de um sufixo de `1`'s de mesmo tamanho: $\{\mathtt{0}^n\mathtt{1}^n \mid n \geq 1\}$.

(g) O conjunto das palavras formadas por concatenações de palavras da forma $\mathtt{0}^n\mathtt{1}^n$ para $n \geq 1$: $\cup_{k \geq 1}\{\mathtt{0}^n\mathtt{1}^n \mid n \geq 1\}^k$. □

Como uma linguagem sobre um alfabeto $\Sigma$ é sempre um conjunto contável, pois é um subconjunto de $\Sigma^*$, que é enumerável, existe a possibilidade de se fazer uma definição recursiva, da forma mostrada na Seção 1.7. Mas a verdade é que, na prática, linguagens são raramente definidas desta forma. Existe um formalismo, que permite o uso de recursão, mas que foi especialmente projetado para definição de linguagens: a *gramática*. Na próxima seção será apresentada uma breve introdução às gramáticas. Em capítulos vindouros elas serão melhor estudadas e exploradas pouco a pouco.

# Exercícios

1. Qual é o número de prefixos, sufixos e subpalavras de uma palavra de tamanho $n$?

2. Descreva mais formalmente as seguintes linguagens sobre o alfabeto $\{\mathtt{0}, \mathtt{1}\}$:

   (a) O conjunto das palavras com, no mínimo, um `0`.

   (b) O conjunto das palavras de tamanho ímpar.

   (c) O conjunto das palavras com um prefixo de um ou mais `0`'s seguido (imediatamente) de um sufixo de zero ou mais `1`'s.

   (d) O conjunto dos palíndromos que não contenham símbolos consecutivos idênticos.

   (e) O conjunto das palavras de tamanho par cuja primeira metade é idêntica à segunda.

   Procure ser bem preciso e conciso.

3. Expresse as linguagens a seguir utilizando operações sobre conjuntos finitos de palavras de $\{\mathtt{0}, \mathtt{1}\}^*$. Considere o alfabeto como sendo $\{\mathtt{0}, \mathtt{1}\}$.

   (a) O conjunto das palavras de 10 símbolos.

   (b) O conjunto das palavras que têm de 1 a 200 símbolos.

   (c) O conjunto das palavras que não têm `00` como prefixo, mas têm `00` como sufixo.

   (d) O conjunto das palavras em que todo `0` é seguido de dois `1`'s consecutivos. Exemplos: $\lambda$, `1`, `1011111`, `11011101111`.

   (e) O conjunto das palavras com número par de `0`'s ou ímpar de `1`'s (ou ambos).

   (f) O conjunto das palavras que contêm um ou dois `1`'s, cujo tamanho é múltiplo de 3.

4. Sejam $A$, $B$ e $C$ linguagens sobre um alfabeto $\Sigma$. Mostre que:

   (a) $A(B \cup C) = (AB) \cup (AC)$.

   (b) nem sempre $A(B \cap C) = (AB) \cap (AC)$.

5. Mostre que se $\lambda \in L$ então $L^+ = L^*$ e se $\lambda \notin L$ então $L^+ = L^* - \{\lambda\}$.

6. Quando $L^*$ é finita?

7. Seja $L^R = \{w^R \mid w \in L\}$, onde $L$ é uma linguagem. Para que linguagens $L$, $L^R = L$?

8. Prove que $(w^R)^n = (w^n)^R$ para toda palavra $w$ e todo $n \in \mathbf{N}$.

9. Prove que $L^* = \bigcup_{n\in\mathbf{N}} L^n$. (*Sugestão:* para provar que $L^* \subseteq \bigcup_{n\in\mathbf{N}} L^n$, use indução sobre $|w|$, e para provar que $\bigcup_{n\in\mathbf{N}} L^n \subseteq L^*$, use indução sobre $n$.)

10. Prove por indução que:

    (a) $L^*L^* = L^*$.
    (b) $(L^*)^* = L^*$.
    (c) $(L_1 \cup L_2)^* L_1^* = (L_1 \cup L_2)^*$.
    (d) $(L_1 \cup L_2)^* = (L_1^* L_2^*)^*$.

11. Em que condições $L_1^* \cup L_2^* = (L_1 \cup L_2)^*$?

12. Dê definições recursivas para as seguintes linguagens:

    (a) $\{\mathtt{0}\}^*\{\mathtt{1}\}^*$.
    (b) $\{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbf{N}\}$.
    (c) $\{w \in \{\mathtt{0},\mathtt{1}\}^* \mid w \text{ contém } \mathtt{00}\}$.
    (d) $\{\mathtt{0}^0\mathtt{1}0^1\mathtt{1}0^2\mathtt{1}\ldots\mathtt{0}^n\mathtt{1} \mid n \in \mathbf{N}\}$.

## 1.11 Gramáticas

As gramáticas são um formalismo originalmente projetado para a definição de linguagens. Nesta seção serão apenas definido o conceito de gramática e apresentados alguns poucos exemplos. Nos próximos capítulos o estudo de gramáticas será retomado associado a classes específicas de linguagens e serão apresentados muitos outros exemplos.

Antes de dar uma definição precisa de gramática, serão apresentados os conceitos envolvidos de maneira informal. Como foi dito na Seção 1.7, uma definição recursiva provê um meio de construir (ou gerar) os elementos de um conjunto (enumerável). Similarmente, uma gramática mostra como *gerar* as palavras de uma linguagem.

O elemento fundamental das gramáticas é a *regra*[29]. Uma regra é um par ordenado $(u, v)$, tradicionalmente escrito na forma $u \to v$, onde $u$ e $v$ são palavras que podem conter símbolos de dois alfabetos disjuntos, um com símbolos denominados *variáveis*, ou *não terminais*, e outro com símbolos denominados *terminais*. As variáveis são símbolos auxiliares para a geração das palavras da linguagem, enquanto que o conjunto de terminais nada mais é do que o alfabeto da linguagem definida. Nos exemplos a seguir, serão usadas letras maiúsculas para variáveis e minúsculas para terminais. Segue um exemplo de regra:

$$\mathtt{a}AB \;\to\; \mathtt{ba}A$$

Tal regra especifica que: dada uma palavra que contenha a subpalavra $\mathtt{a}AB$, tal subpalavra pode ser substituída por $\mathtt{ba}A$. Assim, a partir da palavra $\mathtt{a}ABB\mathtt{a}AB$, aplicando-se a regra acima pode-se obter, diz-se *derivar*:

[29] Também chamada de *regra de produção*, ou *produção*, simplesmente.

- $\texttt{ba}AB\texttt{a}AB$, substituindo a primeira ocorrência de $\texttt{a}AB$;
- $\texttt{a}ABB\texttt{ba}A$, substituindo a segunda ocorrência de $\texttt{a}AB$.

A relação de derivação é denotada por $\Rightarrow$. Por exemplo, a cadeia de derivações:

$$\begin{aligned}\texttt{a}ABB\texttt{a}AB &\Rightarrow \texttt{ba}ABB\texttt{a}AB \quad \text{(aplicando-se a regra } \texttt{a}AB \rightarrow \texttt{ba}A\text{)}\\ &\Rightarrow \texttt{bba}AB\texttt{a}AB \quad \text{(aplicando-se a regra } \texttt{a}AB \rightarrow \texttt{ba}A\text{)}\\ &\Rightarrow \texttt{bba}AB\texttt{ba}A \quad \text{(aplicando-se a regra } \texttt{a}AB \rightarrow \texttt{ba}A\text{)}\end{aligned}$$

mostra uma derivação de $\texttt{bba}A\texttt{ba}A$ a partir de $\texttt{a}ABB\texttt{a}AB$.

Uma gramática consta de um conjunto de regras e da indicação de uma variável especial denominada *variável de partida*. Qualquer derivação deve começar com a palavra constituída apenas da variável de partida.

As palavras de variáveis e/ou terminais geradas a partir da variável de partida são chamadas *formas sentenciais* da gramática. Assim, uma regra pode ser aplicada a uma forma sentencial para gerar uma outra forma sentencial. Uma forma sentencial sem variáveis é denominada *sentença*. A linguagem definida pela gramática, também dita *gerada* pela gramática, é o conjunto de sentenças geradas pela gramática. Para uma gramática $G$, a linguagem gerada por ela é denotada por $L(G)$.

**Exemplo 49** Seja a gramática $G$ constituída da variável de partida $P$ e das regras:

1. $P \rightarrow \texttt{a}A\texttt{bc}$
2. $A \rightarrow \texttt{a}A\texttt{b}C$
3. $A \rightarrow \lambda$
4. $C\texttt{b} \rightarrow \texttt{b}C$
5. $C\texttt{c} \rightarrow \texttt{cc}$

Toda derivação de $G$ deve começar com a forma sentencial constituída da variável de partida $P$. Um exemplo de derivação:

$$\begin{aligned}P &\Rightarrow \texttt{a}A\texttt{bc} \quad \text{(regra 1)}\\ &\Rightarrow \texttt{abc} \quad \text{(regra 3)}\end{aligned}$$

Isto mostra que $\texttt{abc}$ é uma sentença da linguagem gerada por $G$: $\texttt{abc} \in L(G)$. Um outro exemplo de derivação:

$$\begin{aligned}P &\Rightarrow \texttt{a}A\texttt{bc} && \text{(regra 1)}\\ &\Rightarrow \texttt{aa}A\texttt{b}C\texttt{bc} && \text{(regra 2)}\\ &\Rightarrow \texttt{aaa}A\texttt{b}C\texttt{b}C\texttt{bc} && \text{(regra 2)}\\ &\Rightarrow \texttt{aaab}C\texttt{b}C\texttt{bc} && \text{(regra 3)}\\ &\Rightarrow \texttt{aaabb}CC\texttt{bc} && \text{(regra 4)}\\ &\Rightarrow \texttt{aaabb}C\texttt{b}C\texttt{c} && \text{(regra 4)}\\ &\Rightarrow \texttt{aaabb}C\texttt{bcc} && \text{(regra 5)}\\ &\Rightarrow \texttt{aaabbb}C\texttt{cc} && \text{(regra 4)}\\ &\Rightarrow \texttt{aaabbbccc} && \text{(regra 5)}\end{aligned}$$

Logo, $\mathtt{a}^3\mathtt{b}^3\mathtt{c}^3 \in L(G)$. Na verdade, pode-se mostrar que

$$L(G) = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 1\}.$$

□

Agora, define-se formalmente o que é gramática. Uma gramática é uma quádrupla $(V, \Sigma, R, P)$, onde:

(a) $V$ é um conjunto finito de elementos denominados *variáveis*;

(b) $\Sigma$ é um alfabeto; $V \cap \Sigma = \emptyset$;

(c) $R \subseteq (V \cup \Sigma)^+ \times (V \cup \Sigma)^*$ é um conjunto finito de pares ordenados denominados *regras*; e

(d) $P \in V$ é uma variável denominada *variável de partida*.

Observe que o lado esquerdo de uma regra não pode ser $\lambda$.

Seja uma gramática $G = (V, \Sigma, R, P)$. Diz-se que $x \Rightarrow y$ em $G$, onde $x, y \in (V \cup \Sigma)^*$, se há uma regra $u \to v \in R$ tal que $u$ ocorre em $x$ e $y$ é o resultado de substituir uma ocorrência de $u$ em $x$ por $v$. A relação $\stackrel{n}{\Rightarrow}$ é definida recursivamente assim, utilizando-se $\Rightarrow$:

(a) $x \stackrel{0}{\Rightarrow} x$ para todo $x \in (V \cup \Sigma)^*$;

(b) se $w \stackrel{n}{\Rightarrow} xuy$ e $u \to v \in R$ então $w \stackrel{n+1}{\Rightarrow} xvy$ para todo $w, x, y \in (V \cup \Sigma)^*$, $n \geq 0$.

Quando $x \stackrel{n}{\Rightarrow} y$, diz-se que "de $x$ deriva-se $y$ em $n$ passos", ou então que "há uma derivação de tamanho $n$ de $y$ a partir de $x$". Diz-se ainda que de $x$ deriva-se $y$ em zero ou mais passos, $x \stackrel{*}{\Rightarrow} y$, se existe $n \geq 0$ tal que $x \stackrel{n}{\Rightarrow} y$[30]. E de $x$ deriva-se $y$ em um ou mais passos, $x \stackrel{+}{\Rightarrow} y$, se existe $n \geq 1$ tal que $x \stackrel{n}{\Rightarrow} y$[31]. Com isto, pode-se definir o que é a linguagem gerada pela gramática $G$:

$$L(G) = \{w \in \Sigma^* \mid P \stackrel{*}{\Rightarrow} w\}.$$

**Exemplo 50** Seja a gramática $G$ do Exemplo 49. As duas derivações demonstram que $P \stackrel{2}{\Rightarrow} abc$ e $P \stackrel{9}{\Rightarrow} \mathtt{a}^3\mathtt{b}^3\mathtt{c}^3$. É fácil mostrar que todas as palavras da forma $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$, para $n \geq 1$, são geradas por $G$. Basta notar que tais palavras podem ser geradas por derivações da forma:

$$\begin{array}{ll}
P \Rightarrow \mathtt{a}A\mathtt{bc} & \text{(regra 1)} \\
\stackrel{k}{\Rightarrow} \mathtt{a}\mathtt{a}^k A(\mathtt{b}C)^k\mathtt{bc} & \text{(regra 2, } k \text{ vezes; } k \geq 0) \\
\Rightarrow \mathtt{a}\mathtt{a}^k(\mathtt{b}C)^k\mathtt{bc} & \text{(regra 3)} \\
\Rightarrow \mathtt{a}^{k+1}(\mathtt{b}C)^{k-1}\mathtt{b}^2\mathtt{C}\mathtt{c} & \text{(regra 4, 1 vez)} \\
\stackrel{2}{\Rightarrow} \mathtt{a}^{k+1}(\mathtt{b}C)^{k-2}\mathtt{b}^3\mathtt{C}^2\mathtt{c} & \text{(regra 4, 2 vezes)} \\
\vdots & \\
\stackrel{k}{\Rightarrow} \mathtt{a}^{k+1}\mathtt{b}^{k+1}\mathtt{C}^k\mathtt{c} & \text{(regra 4, } k \text{ vezes)} \\
\stackrel{k}{\Rightarrow} \mathtt{a}^{k+1}\mathtt{b}^{k+1}c^{k+1} & \text{(regra 5, } k \text{ vezes)}
\end{array}$$

[30]Usando a terminologia da Seção 1.4, $x \stackrel{*}{\Rightarrow} y$ é o fecho transitivo e reflexivo da relação $\Rightarrow$.

[31]$x \stackrel{+}{\Rightarrow} y$ é o fecho transitivo da relação $\Rightarrow$.

Isto mostra que pode-se derivar $\mathtt{a}^{k+1}\mathtt{b}^{k+1}c^{k+1}$, para $k \geq 0$, em $1+k+1+(1+2+\cdots+k)+k$ passos, ou seja, em $(2k+2)+k(k+1)/2$ passos, ou ainda, $(k+1)(k+4)/2$ passos. Tem-se, então, que:

$$P \overset{n(n+3)/2}{\Rightarrow} \mathtt{a}^n\mathtt{b}^n c^n \text{ para } n \geq 1.$$

Logo, conclui-se que $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 1\} \subseteq L(G)$. □

No Exemplo 50, mostrou-se como provar que um conjunto $A$ está contido na linguagem gerada por uma gramática $G$, ou seja, $A \subseteq L(G)$, mediante o que se denomina um *esquema de derivação*. Tal esquema indica como as palavras de $A$ podem ser geradas pela gramática $G$. Para provar que $A = L(G)$, resta então provar que as únicas palavras geradas por $G$ são as do conjunto $A$, ou seja, que $L(G) \subseteq A$. Infelizmente, não existe uma receita geral para se fazer isto. Para a gramática do Exemplo 49, pode-se notar, analisando-se o próprio esquema de derivação apresentado no Exemplo 50, que, qualquer que seja a ordem de aplicação das regras, não se consegue gerar outras sentenças que não as da forma $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$ para $n \geq 1$.

A mesma linguagem pode ser gerada por inúmeras gramáticas. Duas gramáticas $G$ e $G'$ são ditas *equivalentes* quando $L(G) = L(G')$. O problema de modificar uma gramática de forma que ela atenda a certos requisitos, mas sem alterar a liguagem gerada pela mesma, é importante em certos contextos, como, por exemplo, na construção de analisadores sintáticos de linguagens. Algumas técnicas de manipulação de gramáticas serão abordadas na Seção 3.4.3.

É muito comum uma gramática ter duas ou mais regras com o mesmo lado esquerdo. Por exemplo, a gramática do Exemplo 49, página 43, tem as regras 2 e 3 com o mesmo lado esquerdo: $A$. Neste caso, pode ser útil abreviar colocando-se apenas um lado esquerdo e colocando-se os lados direitos das várias regras separados por "|". As regras 2 e 3 do Exemplo 49 seriam substituídas por:

$$A \rightarrow \mathtt{a}A\mathtt{b}C \mid \lambda$$

Segue mais um exemplo de gramática. Neste exemplo, as regras têm a característica que os seus lados esquerdos contêm apenas e tão somente uma variável. Este tipo de gramática, muito importante em termos práticos, será estudado na Seção 3.4.

**Exemplo 51** Seja a gramática $G = (V, \Sigma, R, E)$, onde:

- $V = \{E, T, N, D\}$
- $\Sigma = \{+, -, (, ), 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- $R$ contém as regras:

$$E \rightarrow E + T \mid E - T \mid T$$
$$T \rightarrow (E) \mid N$$
$$N \rightarrow DN \mid D$$
$$D \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

A gramática $G$ gera expressões aritméticas contendo números na base decimal, operadores de soma e subtração, e parênteses. Através das três regras com $E$ do lado esquerdo pode-se gerar uma seqüência de somas e/ou subtrações de $T$'s (termos); exemplo:

$$\begin{aligned} E &\Rightarrow E + T && \text{(regra } E \to E + T\text{)} \\ &\Rightarrow E - T + T && \text{(regra } E \to E - T\text{)} \\ &\Rightarrow E - T - T + T && \text{(regra } E \to E - T\text{)} \\ &\Rightarrow T - T - T + T && \text{(regra } E \to T\text{)} \end{aligned}$$

Observe como as regras são *recursivas à esquerda*, levando à produção de uma seqüência *da direita para a esquerda*. Por outro lado, as regras responsáveis pela produção dos números das expressões são *recursivas à direita*, redundando na produção de números *da esquerda para a direita*. Por exemplo, para gerar um número de quatro dígitos pode-se derivar:

$$\begin{aligned} N &\Rightarrow DN && \text{(regra } N \to DN\text{)} \\ &\Rightarrow DDN && \text{(regra } N \to DN\text{)} \\ &\Rightarrow DDDN && \text{(regra } N \to DN\text{)} \\ &\Rightarrow DDDD && \text{(regra } N \to D\text{)} \end{aligned}$$

e, em seguida, derivar os quatro dígitos usando-se as regras com $D$ do lado esquerdo. Observe também que a geração de parênteses se dá através de *recursão* (no caso, indireta), a qual não pode ser classificada como recursão à esquerda nem à direita. Por exemplo, na derivação:

$$\begin{aligned} E &\Rightarrow E + T && \text{(regra } E \to E + T\text{)} \\ &\Rightarrow T + T && \text{(regra } E \to T\text{)} \\ &\Rightarrow (E) + T && \text{(regra } T \to (E)\text{)} \end{aligned}$$

a variável $E$ aparece (recursivamente) na forma sentencial entre "(" e ")". □

# Exercícios

1. Seja a gramática $(\{A, B\}, \{\mathtt{0}, \mathtt{1}\}, R, A)$, onde $R$ tem as três regras:

   $A \to BB$

   $B \to \mathtt{0}B\mathtt{1} | \lambda$

   Dê todas as derivações das seguintes palavras:

   (a) $\lambda$.

   (b) `01`.

   (c) `0101`.

   (d) `0011`.

   Que linguagem é gerada?

2. Considere a gramática $G'$ constituída da variável de partida $P$ e das regras:

   1. $P \to \mathtt{a}A\mathtt{b}D$
   2. $A \to \mathtt{a}A\mathtt{b}C$
   3. $A \to \lambda$
   4. $C\mathtt{b} \to \mathtt{b}C$
   5. $CD \to D\mathtt{c}$
   6. $D \to \mathtt{c}$

   obtida da gramática $G$ do Exemplo 49, página 43, modificando-se as regras 1 e 5, e acrescentando-se a regra 6. $L(G') = L(G)$? Explique sua resposta.

3. Construa gramáticas para as seguintes linguagens:

   (a) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid$ o número de $\mathtt{a}$'s em $w$ é par$\}$.
   (b) $\{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbf{N}\}$.
   (c) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w = w^R\}$.
   (d) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w = w^R$ e $w$ não contém símbolos consecutivos idênticos$\}$.
   (e) $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n \mid n \in \mathbf{N}\}$.

4. Para as gramáticas dos itens (b) e (c) do exercício anterior, determine o número de passos necessário para gerar uma palavra de tamanho $n$.

5. Seja a gramática $G = (\{A, B\}, \{\mathtt{a}, \mathtt{b}\}, R, A)$ em que $R$ é constituído das quatro regras:

   $$A \to \mathtt{a}A|B$$
   $$B \to \mathtt{b}B|\lambda$$

   Que linguagem é gerada por $G$? Prove sua resposta.

## 1.12 Problemas de Decisão

Ao longo deste texto serão tratados vários problemas cujas respostas são do tipo sim ou não. Nesta seção é apresentada uma introdução genérica e informal a este tipo de problema.

Um *problema de decisão* (PD) é uma questão que faz referência a um conjunto finito de parâmetros, e que, para valores específicos dos parâmetros, tem como resposta *sim* ou *não*. Seguem alguns exemplos.

**Exemplo 52** São exemplos de problemas de decisão:

(a) Determinar se o número 123654789017 é um número primo.

(b) Determinar se um número natural $n$ é um número primo.

(c) Determinar se existe um ciclo em um grafo $G$.

(d) Determinar se uma palavra $w$ é gerada por uma gramática $G$.

O primeiro PD não tem parâmetros, o segundo e o terceiro têm um parâmetro cada um, um natural $n$ e um grafo $G$, e o quarto tem dois parâmetros, uma palavra $w$ e uma gramática $G$. □

A questão referente a um PD pode ser vista como representando um *conjunto de questões*, uma para cada combinação possível dos valores que os parâmetros podem assumir. Cada questão obtida dando aos parâmetros valores específicos, ou seja, instanciando-se os parâmetros, é dita ser uma *instância* do PD. Em particular, um PD sem parâmetros contém uma única instância.

**Exemplo 53** O PD (b) do Exemplo 52, "determinar se um número natural $n$ é um número primo", é constituído do seguinte conjunto infinito de questões:

- Determinar se 0 é um número primo.
- Determinar se 1 é um número primo.
- Determinar se 2 é um número primo.
- etc.

O PD "determinar se 123654789017 é um número primo" é constituído de uma única instância, a qual é instância também do PD "determinar se um número natural $n$ é um número primo". □

Para cada instância de um PD existe uma resposta correta, *sim* ou *não*. Uma *solução para um PD*, também chamada de um *procedimento de decisão*, é um *algoritmo* que, para qualquer instância do PD, determina a resposta correta. Informalmente, um algoritmo é uma seqüência finita de instruções, cada uma executável por uma máquina em tempo finito, e cada uma produzindo o mesmo resultado para os mesmos dados. Mais para frente será apresentada uma proposta de formalização do conceito de algoritmo mediante a chamada *hipótese de Church-Turing*. Até lá, fica-se com esta definição informal.

Um PD que tem solução é dito ser *decidível*, e um PD que não tem solução é dito ser *indecidível*.

Pelo que ficou dito acima, se um PD tem um conjunto finito de instâncias então ele é decidível: pode-se construir um algoritmo que simplesmente consulte uma "tabela de respostas" pré-computadas. Assim, o estudo de PD's se torna mais interessante para PD's com uma infinidade de instâncias.

Observe que uma solução para o PD "determinar se um número natural $n$ é um número primo" serve também para solucionar o PD "determinar se 123654789017 é um número primo". Mas, mesmo que não existisse solução para o primeiro PD, existiria solução para o segundo! Mesmo não se sabendo a resposta! Para ressaltar este ponto, observe a famosa conjectura de Fermat, que só foi provada recentemente, 350 anos após enunciada:

Para qualquer número natural $n \geq 3$ não existem números naturais $a$, $b$ e $c$ tais que $a^n + b^n = c^n$.

O problema de determinar se esta conjectura está correta é um PD que tem solução, pois é um PD sem parâmetros e, portanto, constituído de uma única instância. Mesmo na época em que a conjectura não tinha sido provada, o PD respectivo já tinha solução, embora desconhecida... Considere os dois algoritmos: "*retorne sim*" e "*retorne não*"; um deles é a solução.

Um PD obtido de um outro, $P$, restringindo-se o conjunto de valores possíveis de um ou mais parâmetros de $P$, é dito ser uma *restrição* de $P$. Assim, por exemplo o PD "determinar se 123654789017 é um número primo" é uma restrição de "determinar se um número natural $n$ é um número primo". É evidente que se um PD tem solução, então suas restrições também têm. No entanto, pode acontecer de uma ou mais restrições de um PD terem soluções e o PD não ter. Por exemplo, a restrição do PD "determinar se uma palavra $w$ é gerada por uma gramática $G$":

"determinar se uma palavra $w$ é gerada por $G_0$",

onde $G_0$ é uma gramática em que o lado esquerdo de cada regra é constituído de uma variável[32], tem solução, mas, como será mostrado mais à frente, o PD original não tem solução.

## Exercícios

1. Diz-se que um PD $P$ é *redutível* a um PD $Q$, se existe um algoritmo $\mathcal{R}$ que, recebendo $x$ como entrada, produz um resultado $y$ tal que a resposta a $P$ para a entrada $x$ é idêntica ou complementar[33] à resposta a $Q$ para a entrada $y$, qualquer que seja a entrada $x$. Diz-se, com isto, que o algoritmo $\mathcal{R}$ pode ser usado para *reduzir* o problema $P$ ao problema $Q$.

   Seja $D$ um PD decidível e $I$ um PD indecidível. O que se pode dizer de um PD $X$, com relação à sua decidibilidade, se:

   (a) $D$ é redutível a $X$?
   (b) $X$ é redutível a $D$?
   (c) $I$ é redutível a $X$?
   (d) $X$ é redutível a $I$?

## 1.13 Exercícios

1. Sejam VP o conjunto das variáveis proposicionais e LP o conjunto das sentenças da Lógica Proposicional. Uma definição recursiva foi vista no Exemplo 33, página 27. Suponha a existência de uma função $v : \text{VP} \to \{V, F\}$ que atribua valores-verdade para as variáveis proposicionais. Defina recursivamente a função $\hat{v} : \text{LP} \to \{V, F\}$ de forma que $\hat{v}(\alpha)$ seja o valor-verdade de $\alpha$ para a atribuição de valores-verdade dada por $v$.

[32] Este tipo de gramática será definido na Seção 3.4 como sendo uma gramática livre do contexto.
[33] A resposta complementar a *sim* é *não*, e a *não* é *sim*.

2. Toda afirmativa da Lógica Proposicional é equivalente a uma afirmativa que usa apenas (se usar) os conectivos $\vee$ e $\neg$. Especifique como se pode obter, a partir de uma afirmativa qualquer, uma equivalente que use apenas (se usar) os conectivos $\vee$ e $\neg$. Estenda o método proposto para incluir os conectivos $\forall$ e $\exists$; no caso, além de $\vee$ e $\neg$, a afirmativa final pode ter apenas $\forall$.

3. Prove que para todo $n \in \mathbf{N}$ existe $k \in \mathbf{N}$ tal que $(k+1)^2 - k^2 > n$.

4. Dadas duas funções $f$ e $g$, ambas de $\mathbf{N}$ para $\mathbf{N}$, $f$ é dita ser de ordem $g$, ou seja, $f$ é $O(g)$, se e somente se, existem dois números naturais constantes $k$ e $n_0$ tais que para todo $n \geq n_0$, $f(n) \leq kg(n)$. Prove que $10n^2 + 100n + 1000$ é $O(n^2)$. Explicite que técnicas básicas de prova foram utilizadas.

5. Qual é o número de elementos de $A \cup B$, se $A$ e $B$ são conjuntos finitos? Generalize: qual é o número de elementos de $\bigcup_{i=1}^{n} A_i$, se $A_1$, $A_2$, ..., $A_n$ são conjuntos finitos? *Sugestão*: neste último caso, use indução matemática.

6. Prove que:

   (a) $\mathcal{P}(A) \cup \mathcal{P}(B) \subseteq \mathcal{P}(A \cup B)$.

   (b) $\mathcal{P}(A) \cap \mathcal{P}(B) = \mathcal{P}(A \cap B)$.

7. Defina *diferença simétrica* de conjuntos $A$ e $B$ por $A \triangle B = (A - B) \cup (B - A)$. Prove que:

   (a) $A \triangle B = (A \cup B) - (A \cap B)$.

   (b) $A \triangle (B \triangle C) = (A \triangle B) \triangle C$.

   (c) $(A - B) \triangle (B - A) = A \triangle B$.

   (d) $A \triangle B = A$ se, e somente se, $B = \emptyset$.

   Em que condições $A - B = A \triangle B$?

8. Seja uma relação binária sobre $A$. A *relação composta* $R \circ R$ é definida como sendo $\{(x, z) \mid (x, y) \in R \text{ e } (y, z) \in R \text{ para algum } y \in A\}$. A relação $R^n$, para $n \geq 1$, é definida recursivamente por:

   (a) $R^1 = R$;

   (b) $R^n = R^{n-1} \circ R$ para $n \geq 2$.

   Prove que para toda relação reflexiva e transitiva $R^n = R$ para todo $n \geq 1$.

9. Seja $R$ uma relação de equivalência sobre $A$, $A \neq \emptyset$. Mostre que existe uma função $f : A \to A$ tal que $xRy$ se, e somente se, $f(x) = f(y)$.

10. Seja $R$ uma relação de equivalência sobre $A$, $A \neq \emptyset$. Prove que $\{[a] \mid a \in A\}$ é uma partição de $A$.

11. Defina $f(A)$ onde $f$ é uma função e $A$ um conjunto, como sendo $f(A) = \{f(x) \mid x \in A\}$. Mostre que:

(a) $f(A \cup B) = f(A) \cup f(B)$.

(b) $f(A \cap B) \subseteq f(A) \cap f(B)$.

12. A *função característica* de um conjunto $A$ é a função $f_A : A \to \{0, 1\}$ tal que $f_A(x) = 1$ se, e somente se, $x \in A$. Mostre que, para todo elemento do conjunto universo:

    (a) $f_A(x) = 1 - f_{\overline{A}}(x)$.

    (b) $f_{A \cup B}(x) = f_A(x) + f_B(x) - f_A(x) f_B(x)$.

    (c) $f_{A \cap B}(x) = f_A(x) f_B(x)$.

    (d) $f_{A-B}(x) = f_A(x)(1 - f_B(x))$.

13. Mostre que há uma função injetora de $A$ para $B$ se, e somente se, há uma função sobrejetora de $B$ para $A$.

14. Seja um alfabeto qualquer $\Sigma$. Prove que $\Sigma^*$ é enumerável. Prove que $\mathcal{P}(\Sigma^*)$ não é enumerável. Ou seja, qualquer linguagem é enumerável, mas o conjunto de todas as linguagens não é! Conseqüência: como o conjunto dos compiladores é enumerável, existem linguagens para as quais não há compiladores.

15. Utilizando o método da diagonalização de Cantor, prove que não existe uma função bijetora de um conjunto não vazio para o conjunto potência do mesmo. Quais são as implicações deste resultado?

16. No Exemplo 43, página 37, verificou-se que números naturais podem ser representados com palavras exponencialmente menores na base 2, com relação à base 1. Generalize este resultado, isto é, determine a relação que existe entre as representações em duas bases diferentes quaisquer.

17. Faça uma definição recursiva da função $v : \{\mathtt{0}, \mathtt{1}\}^* \to \mathbf{N}$ tal que $v(w)$ é o número natural representado por $w$ na base 2. Por exemplo, $v(\mathtt{01}) = 1$, $v(\mathtt{101}) = 5$, $v(\mathtt{0010}) = 2$, etc.

18. Faça uma definição da operação de subtração sobre os naturais, onde $m - n$ é definida apenas quando $m \geq n$. Use para isto a operação *sucessor*. Defina também as operações: divisão, resto da divisão e máximo divisor comum, todas sobre os naturais. No caso da divisão, há truncamento para baixo. Na definição de uma operação, podem ser usadas outras já definidas nesta questão ou anteriormente neste texto.

19. Faça uma definição recursiva do *conjunto de ancestrais* de um vértice $v$ de uma árvore $(V, A, r)$. Considere que $v$ é ancestral dele mesmo.

20. Seja a seguinte definição recursiva da função $\psi : \mathbf{N} \to \mathbf{N}$:

    (a) $\psi(1) = 0$;

    (b) para $n \geq 2$, $\psi(n) = \psi(\lfloor n/2 \rfloor) + 1$.

Descreva o que vem a ser $\psi(n)$. *Sugestão*: determine $\psi(n)$ para alguns valores de $n$, a partir de 0 até ficar claro o que vem a ser $\psi(n)$.

21. Prove por indução:

    (a) $\sum_{k=1}^{n}[k(k+1)(k+2)] = n(n+1)(n+2)(n+3)/4$.

    (b) $3^n + 7^n - 2$ é divisível por 8.

    (c) $\prod_{k=2}^{n}(1 - 1/k) = 1/n$.

    (d) para $n \geq 2$, $\sum_{k=1}^{n}(1/\sqrt{k}) > \sqrt{n}$.

22. Suponha que a relação "é amigo de" é simétrica. Mostre, modelando o problema por meio de grafo, que em um grupo de duas ou mais pessoas sempre existem duas pessoas com o mesmo número de amigos.

23. Qual é a altura mínima de uma árvore de $n$ vértices?

24. Uma *árvore estritamente n-ária* é uma árvore em que cada vértice interno tem exatamente $n$ filhos. Responda às seguintes questões referentes a uma árvore estritamente $n$-ária:

    (a) Quantos vértices ela tem se tiver $i$ vértices internos?

    (b) Se ela tiver $k$ vértices, quantos são vértices internos e quantos são folhas?

    (c) Se ela tiver $k$ vértices, quais são a altura mínima e a altura máxima possíveis?

25. Seja a seguinte definição recursiva da linguagem $L$ sobre o alfabeto $\{\mathtt{0}, \mathtt{1}\}$:

    (a) $\lambda \in L$;

    (b) se $x \in L$, então $\mathtt{0}x \in L$ e $x\mathtt{1} \in L$;

    (c) nada mais pertence a $L$.

    Que linguagem é $L$? Prove sua resposta.

26. Faça definições recursivas das seguintes linguagens:

    (a) $\{w \in \{\mathtt{0}, \mathtt{1}\} \mid |w| \text{ é par}\}$.

    (b) $\{w \in \{\mathtt{0}, \mathtt{1}\} \mid w \text{ é palíndromo}\}$.

    (c) $\{w \in \{\mathtt{0}, \mathtt{1}\} \mid w \text{ contém } \mathtt{00}\}$.

    (d) $\{w \in \{\mathtt{0}, \mathtt{1}\} \mid w \text{ não contém } \mathtt{00}\}$.

    (e) $\{\mathtt{0}^{n^2} \mid n \in \mathbf{N}\}$.

    (f) $\{w \mid w \text{ é uma permutação dos dígitos 1, 2 e 3}\}$.

    (g) $\{w \mid w \text{ é uma permutação dos 10 dígitos decimais}\}$.

27. O conjunto das palavras sobre um alfabeto $\Sigma$, $\Sigma^*$, pode ser definido recursivamente a partir da operação de "justapor um símbolo à direita" assim:

(a) $\lambda \in \Sigma^*$;

(b) se $x \in \Sigma^*$ e $a \in \Sigma$ então $xa \in \Sigma^*$.

Usando esta mesma operação de justapor um símbolo à direita, defina recursivamente a operação de concatenação. Em seguida, prove por indução que $(xy)z = x(yz)$ para todo $x, y, z \in \Sigma^*$.

Usando a operação de justapor um símbolo à direita e a de justapor um símbolo à esquerda, defina recursivamente a operação reverso. Em seguida, prove por indução que $(xy)^R = y^R x^R$ para todo $x, y \in \Sigma^*$. Para isto, pode ser utilizado o resultado anterior. Mostre também que se $w$ é palíndromo então $w = vv^R$ ou $w = vav^R$ para algum $v \in \Sigma^*$ e $a \in \Sigma$.

28. Mostre que se $x$, $y$ e $xy$ são palíndromos, então existe uma palavra $z$ e naturais $k$ e $n$ tais que $x = z^k$ e $y = z^n$.

29. Mostre que se $X \subseteq L^*$ então $(L^* \cup X)^* = L^*$.

30. Mostre que o conjunto $\Sigma_1^*\Sigma_2^*$ é enumerável, onde $\Sigma_1$ e $\Sigma_2$ são alfabetos.

31. Seja um alfabeto $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. Seja a função $H_n : \Sigma^n \to \Sigma^n$ tal que $H_n(x, y)$ é o número de posições em que as palavras (de tamanho $n$) $x$ e $y$ diferem. Por exemplo, $H_2(01, 01) = 0$, $H_3(101, 111) = 1$, $H_6(101100, 111001) = 3$. Mostre que $H_n(x, y) \leq H_n(x, z) + H_n(z, y)$.

32. Construa gramáticas para as linguagens:

    (a) $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w \text{ não contém } \mathtt{00}\}$.

    (b) $\{\mathtt{0}^n\mathtt{1}^{2n+1}\mathtt{0}^n \mid n \in \mathbf{N}\}$.

    (c) $\{w\mathtt{0}w \mid w \in \{\mathtt{1}, \mathtt{2}\}^*\}$.

    (d) $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^k \mid 0 \leq n < k\}$.

33. Como já ficou dito, podem existir várias gramáticas para a mesma linguagem. Ao ser obtida uma gramática, pode ser conveniente, como será visto mais à frente, fazer algumas modificações na mesma, substituindo-se, adicionando-se e/ou eliminando-se algumas regras, sem alterar a linguagem gerada. Reflita se é possível alterar uma gramática, obtendo-se uma outra equivalente, de forma que a gramática resultante não contenha regras da forma $A \to B$, onde $A$ e $B$ são variáveis.

34. Reflita se é possível alterar uma gramática, obtendo-se uma outra equivalente, de forma que a gramática resultante não contenha regras da forma $A \to \lambda$, onde $A$ é variável.

## 1.14 Notas Bibliográficas

Quem quizer uma bibliografia de lógica matemática orientada para aplicações em Ciência da Computação, em nível de graduação, pode consultar Ben-Ari[BA01], Nissanke[Nis99],

Burke and Foxley[BF96] ou Souza[dS02]. Excelentes introduções a lógica são as de Hodges[Hod01], Suppes[Sup57] e Copi[Cop81]. Para aqueles com propensão a raciocínio de natureza mais abstrata (matemática), há uma ampla coleção de bons livros, dentre os quais indica-se os de Mendelson[Men87] e Enderton[End00]. Para aqueles interessados especificamente em aprender as técnicas e estratégias utilizadas para prova de teoremas, indica-se o excelente livro de Velleman[Vel94].

Aqueles que querem se aprofundar um pouco mais com relação aos conceitos de conjuntos, funções e relações, podem consultar Halmos[Hal91]. Outros textos que também abordam tais assuntos, e que são orientados para estudantes em nível de graduação, são os de Dean[Dea97], Rosen[Ros99], Epp[Epp90] e Grimaldi[Gri94], além do já citado livro de Velleman[Vel94]. Este último tem também uma excelente introdução aos conceitos de recursão, indução matemática e conjuntos enumeráveis.

Os livros de matemática discreta citados acima têm capítulos com introduções a grafos. Para os que querem se aprofundar um pouco mais, recomenda-se o livro de West[Wes96].

As gramáticas foram propostas por Chomsky[Cho56][Cho59].

Vários outros textos que abordam linguagens formais e autômatos contêm capítulos com uma revisão dos conceitos apresentados. Dentre eles, pode-se destacar [HMU01], [LP98], [Lin97], [Mar91] e [Mor97].

# Capítulo 2

# Máquinas de Estado-Finito

> A good idea has a way of becoming simpler and solving problems other than that for which it was intended.
>
> Robert E. Tarjan *apud* M. Shasha & C. Lazere[SL95]

As máquinas de estado-finito[1] são máquinas abstratas que capturam as partes essenciais de algumas máquinas concretas. Estas últimas vão desde máquinas de vender jornais e de vender refrigerantes, passando por relógios digitais e elevadores, até programas de computador, como alguns procedimentos de editores de textos e de compiladores. O próprio computador digital, se considerarmos que sua memória é limitada, pode ser modelado por meio de uma máquina de estado-finito. Embora existam máquinas abstratas conceitualmente mais poderosas que as de estado-finito, como as que serão apresentadas em capítulos posteriores, e que são mais adequadas para a modelagem de certas máquinas como, por exemplo, o computador, as máquinas de estado-finito são adequadas, tanto do ponto de vista teórico, quanto prático, para a modelagem de um amplo espectro de máquinas (mecânicas, eletrônicas, de *software*, etc.).

Existem, basicamente, dois tipos de máquinas de estado-finito: os transdutores e os reconhecedores (ou aceitadores) de linguagens. Os transdutores são máquinas com entrada e saída, e os reconhecedores são máquinas com apenas duas saídas possíveis; usualmente, uma delas significa "aceitação" da entrada, e a outra, "rejeição" da entrada. Nas próximas seções serão abordados inicialmente os reconhecedores de linguagens. Em seguida, serão apresentados de forma sucinta os dois tipos clássicos de transdutores.

As linguagens reconhecidas por máquinas de estado-finito são denominadas *linguagens regulares*. Existem duas notações bastante úteis, tanto do ponto de vista teórico quanto prático, para especificação de linguagens regulares: expressões regulares e gramáticas regulares. Tais notações serão abordadas após a apresentação dos dois tipos de transdutores mencionados.

Uma característica fundamental de uma máquina de estado-finito é que sua memória é limitada e exclusivamente organizada em torno do conceito de "estado". Embora uma máquina de estado-finito seja, tecnicamente falando, uma estrutura matemática, e o conjunto de estados um conjunto finito da estrutura, pode ser conveniente introduzir tais conceitos de uma maneira mais intuitiva do que formal, dado que este livro se destina a

[1] *Finite-state machines*.

alunos da área de Informática, que nem sempre têm uma maturidade matemática consolidada. Isto será feito na próxima seção, através de três exemplos.

As máquinas de estado-finito do tipo reconhecedor de linguagem são mais conhecidas como *autômatos finitos*[2]. E os transdutores são também conhecidos como autômatos finitos com saída. Assim, estes termos serão usados daqui para frente.

Após a apresentação dos exemplos, na próxima seção, o conceito de autômato finito será então desenvolvido nas Seções 2.2 e 2.3. Propriedades importantes dos autômatos finitos, com desdobramentos positivos dos pontos de vista teórico e prático, serão apresentados, uma parte já na Seção 2.2.3 e outra na Seção 2.4. Na Seção 2.5, serão apresentados os dois tipos de autômatos finitos com saída. As expressões regulares e as gramáticas regulares serão abordadas em seguida, nas Seções 2.6 e 2.7. Finalmente, será feito um apanhado geral do assunto na Seção 2.8.

## 2.1 Alguns Exemplos

Para dar uma idéia do que vem a ser autômato finito antes da sua definição formal, e também para introduzir os principais conceitos e a terminologia associada, serão usados três exemplos bem simples. O primeiro refere-se a um quebra-cabeças, o segundo à modelagem de uma máquina para resolver um certo tipo de problema matemático e o terceiro à modelagem de um elevador. Apesar de muito simples, eles dão uma noção do espectro de problemas para os quais se pode aplicar o conceito de autômato finito. Esta noção será ampliada nos exercícios e ao longo do capítulo.

### 2.1.1 Um quebra-cabeças

Um homem, um leão, um coelho e um repolho devem atravessar um rio usando uma canoa, com a restrição de que o homem deve transportar no máximo um dos três de cada vez de uma margem à outra. Além disso, o leão não pode ficar na mesma margem que o coelho sem a presença do homem, e o coelho não pode ficar com o repolho sem a presença do homem. O problema consiste em determinar se é possível fazer a travessia.

A solução de qualquer problema é sempre, ou deveria ser, antecedida por uma fase de modelagem, na qual

(a) as informações relevantes, que efetivamente têm influência na solução do problema, são identificadas, deixando-se de lado aquelas que não contribuem para a solução;

(b) as informações relevantes são estruturadas (ou seja, representadas), de forma a facilitar a posterior solução.

Estas duas etapas não precisam ocorrer nesta ordem, sendo comum as duas serem consideradas simultaneamente. E, de modo geral, quanto "menos" informações se capturar da realidade que se está modelando, mais fácil e/ou eficiente se tornará a etapa de solução. Este processo de identificar as informações relevantes e desprezar as que não têm relevância é denominado *abstração*. Assim, por exemplo, quando se diz que um autômato finito é

[2] *Finite automata* ou *Finite-state automata*.

uma *máquina abstrata*, está-se dizendo que ele expressa a parte essencial, desprezando-se detalhes de implementação que, no presente contexto, são considerados irrelevantes.

Assim, para o problema proposto, pode-se identificar como informações relevantes:

(a) em um dado instante, em que margem do rio estão o homem, o leão, o coelho, e o repolho;

(b) a seqüência de movimentações entre as margens que propiciou a situação indicada em (a).

Observe a "economia": não é necessário dizer em que margem está a canoa, pois ela estará onde o homem estiver; não é necessário representar uma situação em que a canoa está no meio do rio, pois isto não é importante para se chegar à solução; etc... A primeira informação pode ser representada escrevendo-se apenas o que está na margem esquerda, pois o resto estará na margem direita. Por exemplo, usando as letras $h$, significando "homem", $l$, significando "leão", $c$, significando "coelho" e $r$, significando "repolho", tal informação poderia ser dada por um conjunto contendo elementos de $\{h, l, c, r\}$. Por exemplo, $\{h, c, r\}$ representa a situação em que o homem, o coelho e o repolho estão na margem esquerda e o leão na margem direita; $\{r\}$ representa o repolho na margem esquerda e o homem, coelho e leão na margem direita. A segunda informação importante para o problema, seqüência de movimentações, pode ser representada por uma palavra $a_0 a_1 a_2 \ldots a_n$, onde cada $a_i$ pode ser `s`, `l`, `c` ou `r`. Supondo que todos estão na margem esquerda no início, tem-se que $a_i$ denota movimentação da esquerda para a direita se $i$ é par e denota movimentação da direita para a esquerda se $i$ é ímpar. Se $a_i$ é `s`, o homem está indo sozinho; se é `l`, está levando o leão; se é `c`, está levando o coelho; e se é `r`, está levando o repolho. Por exemplo, a palavra `csr` representa uma movimentação em que

1) $a_0 =$ `c`: o homem leva o coelho da esquerda para a direita,

2) $a_1 =$ `s`: o homem volta sozinho da direita para a esquerda, e

3) $a_2 =$ `r`: o homem leva o repolho da esquerda para a direita.

Observe que cada uma das situações deste exemplo, descrita por um conjunto, é uma "fotografia" dos elementos relevantes da realidade que estamos modelando. Tal fotografia é, em alguns contextos, denominada *estado*. Assim, em termos gerais, embora de forma ainda um pouco imprecisa, um estado pode ser imaginado como uma representação dos elementos essenciais de uma realidade que estamos modelando, sendo que a escolha dos elementos essenciais depende da aplicação. Já cada elemento da palavra que representa uma seqüência de movimentações especifica uma operação que propicia a *transição* de um estado para outro. Assim, o problema pode ser visto como "encontrar uma palavra que represente uma seqüência de transições que leve de um *estado inicial* ($\{h, l, c, r\}$, no exemplo) a um *estado final* ($\{\}$, no exemplo)".

Está apresentada na Figura 2.1 uma representação gráfica do espaço dos estados e transições possíveis para o exemplo, mediante o que se denomina um *diagrama de estados*. Nesta figura, cada estado é representado por uma oval e cada transição possível é representada por uma seta ligando o estado ao qual ela se aplica ao estado que resulta de sua aplicação. O estado inicial é ressaltado por meio de uma seta que o aponta e o estado final é ressaltado por meio de uma oval dupla. Mais precisamente, o espaço de

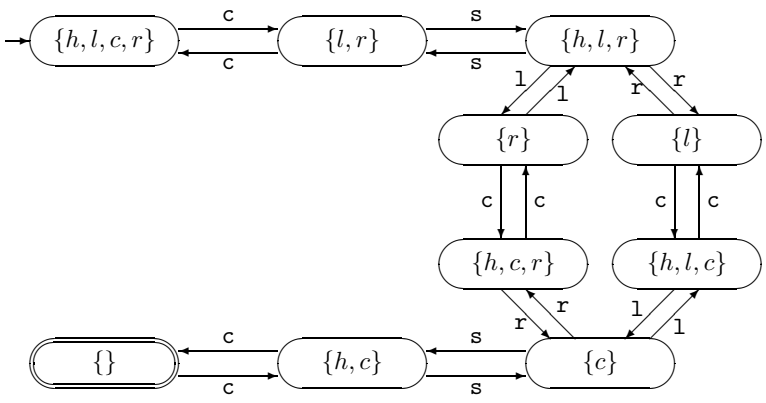



Figura 2.1: Diagrama de estados para Leão, Coelho e Repolho.

estados e transições pode ser modelado por meio de um grafo dirigido cujos vértices são os estados, cujas arestas são as transições, e que tem dois vértices destacados como inicial e final.

Se existe uma aresta de $e_1$ para $e_2$ com rótulo $a$ no diagrama de estados, será dito que há uma *transição de $e_1$ para $e_2$ sob $a$*.

O conjunto de todas as soluções para o quebra-cabeças é o conjunto das palavras correspondentes aos caminhos do estado inicial ao estado final. Cada palavra é formada concatenando-se os rótulos das arestas percorridas no caminho do estado inicial ao estado final. Pela Figura 2.1, fica evidente que existe um conjunto infinito de soluções, e que existem duas soluções de tamanho mínimo: `cslcrsc` e `csrclsc`.

Dada uma palavra $w$ de $\{\mathtt{s}, \mathtt{l}, \mathtt{c}, \mathtt{r}\}^*$, como saber se $w$ é solução? Evidentemente, deve-se verificar se o "caminho correspondente" a $w$ partindo do estado inicial chega ao estado final; se chegar, $w$ é solução, caso contrário, não é. No primeiro caso, diz-se que $w$ é *reconhecida*, ou *aceita*, e no segundo diz-se que não é reconhecida, é *rejeitada*.

Suponha que $x$ é um prefixo de $w$, ou seja, $w = xy$, para alguma palavra $y$. No processo de verificar se $w$ é reconhecida, quando tiver terminado o processamento de $x$, o que é necessário para se continuar o processo? Apenas duas informações:

- o estado atual; e
- $y$.

Este par é denomimado uma *configuração instantânea*. Uma notação útil é aquela associada à relação $\vdash$ (resulta em), que permite mostrar, passo a passo, a evolução das configurações instantâneas durante o processamento de uma palavra. Diz-se que $[e_1, w] \vdash [e_2, y]$ se existe uma transição de $e_1$ para $e_2$ sob $a$ e $w = ay$. Assim, por exemplo, tem-se:

$$[\{l, r\}, \mathtt{sllr}] \vdash [\{h, l, r\}, \mathtt{llr}] \vdash [\{r\}, \mathtt{lr}] \vdash [\{h, l, r\}, \mathtt{r}] \vdash [\{l\}, \lambda].$$

Esta cadeia de relacionamentos pode ser interpretada como *uma computação* (da máquina cujo diagrama de estados é mostrado na Figura 2.1) iniciada no estado $\{l, r\}$, processando `sllr`.

O diagrama de estados da Figura 2.1 tem uma característica marcante: a toda transição de um estado $e_1$ para um estado $e_2$ corresponde uma transição inversa, de $e_2$ para $e_1$ sob o mesmo símbolo. Os diagramas de estado, em geral, não precisam ter esta característica, como ficará evidenciado nos próximos exemplos. Uma outra característica do diagrama de estados da Figura 2.1, nem sempre presente, é a ocorrência de um único estado final. No entanto, existem duas outras características dignas de nota. A primeira é que para cada par (estado, símbolo) existe, no máximo, *uma* transição, ou seja, se existe uma transição do estado $e_1$ para o estado $e_2$ sob $a$, então não existe outra transição de $e_1$ para qualquer outro estado sob $a$. Esta característica confere um caráter de *determinismo* à máquina abstrata representada pelo diagrama: dada uma palavra $w$, existe um único estado que pode ser atingido a partir de qualquer estado, processando-se $w$. Esta característica também não é compartilhada por todos os diagramas de estados para autômatos finitos, como será visto na Seção 2.3.

Um outra característica do diagrama de estados da Figura 2.1, esta sim compartilhada por qualquer autômato finito, é que *o número de estados é finito*. O conjunto de estados funciona, na realidade, como uma memória em que cada estado representa o conjunto das palavras que levam do estado inicial até ele. Assim, no exemplo, o estado final {} representa todas as soluções. O único mecanismo de memória possuído por este tipo de máquina é o conjunto de estados.

### 2.1.2 Um probleminha de matemática

Seja o problema de projetar uma máquina que, dada uma seqüência de 0's e 1's, determine se o número representado por ela na base 2 é divisível por 6. O que se deseja é um projeto independente de implementação, ou seja, que capture apenas a essência de tal máquina, não importando se ela será mecânica, eletrônica, um programa, ou o que seja.

Este problema, aparentemente tão distinto do anterior, pode ser modelado de forma análoga. No exemplo anterior, uma palavra (representando uma seqüência de movimentos de uma margem à outra) é processada da esquerda para a direita, símbolo a símbolo. Aqui, analogamente, a máquina deverá processar a seqüência dígito a dígito, da esquerda para a direita. Assim, deve-se considerar, após lido um prefixo $x$ qualquer, estando o autômato em um determinado estado, para qual estado deve ser feita uma transição se o próximo dígito for 0, e para qual estado deve ser feita uma transição se o próximo dígito for 1. Ora, supondo que $\eta(x)$ é o número representado pela palavra $x$, sabe-se que a palavra $x\texttt{0}$ representa o número $2\eta(x)$, e a palavra $x\texttt{1}$ representa o número $2\eta(x)+1$. E mais, sabe-se que, sendo $r$ o resto da divisão por 6 do número $\eta(x)$:

- o resto da divisão por 6 do número $2\eta(x)$ é o resto da divisão por 6 de $2r$;
- o resto da divisão por 6 do número $2\eta(x)+1$ é o resto da divisão por 6 de $2r+1$.

Com base nisto, pode-se imaginar uma máquina com 6 estados, uma para cada resto de 0 a 5. De cada estado correspondente a resto $r$ ($0 \leq r \leq 5$), emanam duas transições: se o próximo dígito for 0, tem-se uma transição para o estado correspondente ao resto de $2r$ por 6, e se o próximo dígito for 1, tem-se uma transição para o estado correspondente ao resto de $2r+1$ por 6. O estado correspondente a resto 0 é estado inicial e final. A Figura 2.2 mostra o diagrama de estados correspondente.

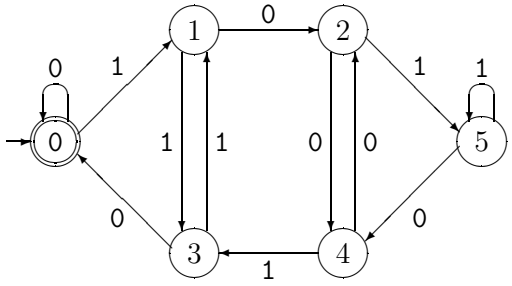



Figura 2.2: Diagrama de estados para Binário módulo 6.

Com este mesmo diagrama de estados da Figura 2.2, modificando-se apenas o conjunto de estados finais, obtém-se vários autômatos. Por exemplo, se os estados finais forem 1, 3 e 5, tem-se um autômato que reconhece toda palavra que representa um número cuja divisão por 6 tem resto ímpar.

### 2.1.3 Modelagem do funcionamento de um elevador

Seja um elevador destinado a servir a um prédio de 3 andares. O problema a ser abordado é o de modelar o funcionamento do elevador, o qual deverá satisfazer às seguintes condições:

(1) caso não haja chamada, o elevador fica parado onde estiver;

(2) o elevador dá prioridade ao chamado mais próximo no sentido em que estiver se movimentando;

(3) um chamado pode ser "desligado" manualmente. Assim, por exemplo, é possível existir uma chamada para um andar em certo instante e, logo em seguida, não existir mais, sem que o elevador se mova.

Nos exemplos anteriores, a identificação dos estados e transições é bem evidente a partir da identificação dos aspectos importantes do problema, ou seja, a partir de sua modelagem. Assim, inicialmente, listemos as informações relevantes do presente caso:

- os andares em que o elevador está sendo solicitado;
- o andar em que o elevador está;
- o sentido em que o elevador está se movendo.

A partir de tais informações, o elevador pode "decidir" para qual andar ele deve se dirigir. Dentre elas, quais irão compor um estado e quais irão compor uma transição? Intuitivamente, o que faz o elevador mudar de um estado para outro é a primeira informação acima, o conjunto dos andares para os quais o elevador está sendo solicitado; e um estado é composto das outras duas informações, já que, pela condição (3) acima, não há necessidade de conferir compatibilidade de chamadas. A seguir, cada estado será representado por $i\theta$, onde $i$ é o andar em que o elevador está, e $\theta$ é $\uparrow$ se o elevador está indo para cima,

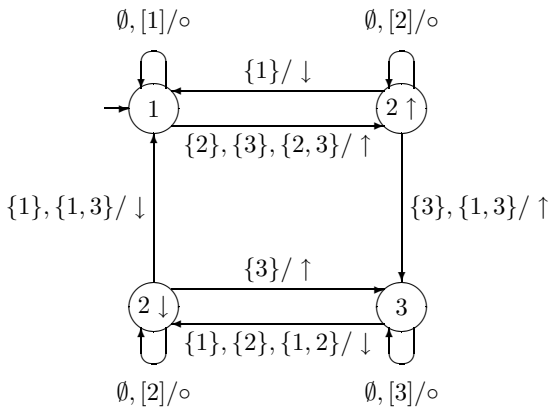



Figura 2.3: Diagrama de estados para um elevador.

é ↓ se está indo para baixo, e é nulo se o andar é o primeiro andar (neste caso, o elevador só pode se movimentar para cima) ou é o último andar (neste caso, o elevador só pode se movimentar para baixo). Com isto, a máquina para modelar o elevador que atende a três andares terá quatro estados: 1, 2 ↑, 2 ↓ e 3. Por este raciocínio, para modelar um elevador para $n$ andares, seriam necessários, obviamente, $2+2(n-2) = 2(n-1)$ estados. Para cada transição, além de especificar o conjunto de chamadas que a ocasionou, será especificada também uma *saída*: a ação do elevador, representada por um dos 3 símbolos: ↑, ↓ ou ∘, conforme o elevador deva se mover para cima, para baixo, ou não se mover.

Um diagrama de estados que modela o problema está mostrado na Figura 2.3. Observe que existem várias transições para o mesmo par de estados. Para simplicar a figura, ao invés de colocar uma seta para cada uma de tais transições, optou-se por colocar todo o conjunto de transições para cada par de estados como uma lista rotulando uma única seta; e como, neste exemplo, a saída é a mesma para cada membro da lista, ela é especificada apenas uma vez, separada da lista por "/". Além disso, para simplificar a figura, é usada a notação $[i]$ significando "a lista de todos os subconjuntos de $\{1, 2, 3\}$ que contêm $i$". Observe que:

- se o conjunto de chamadas contém o andar em que está o elevador, este permanece no mesmo andar; apenas quando a chamada é desligada, manualmente ou não, o elevador pode partir para outro andar (e a máquina para outro estado);

- o elevador muda de sentido quando não há chamada no sentido em que está se movendo, caso haja chamada no outro sentido.

É interessante notar que, neste exemplo, para qualquer palavra de entrada tem-se uma palavra de saída. Não é utilizado o conceito de estado final.

Aqui é útil estender a configuração instantânea para conter a saída computada até o momento, e redefinir $\vdash$. Uma configuração instantânea será uma tripla $[e, x, y]$, onde $e$ e $x$ são como vistos na seção anterior, e $y$ é uma palavra representando a seqüência das saídas emitidas até o momento. Exemplo:

$$[1, \{2,3\}\{1,3\}\{1\}, \lambda] \vdash [2\uparrow, \{1,3\}\{1\}, \uparrow] \vdash [3, \{1\}, \uparrow\uparrow] \vdash [2\downarrow, \lambda, \uparrow\uparrow\downarrow].$$

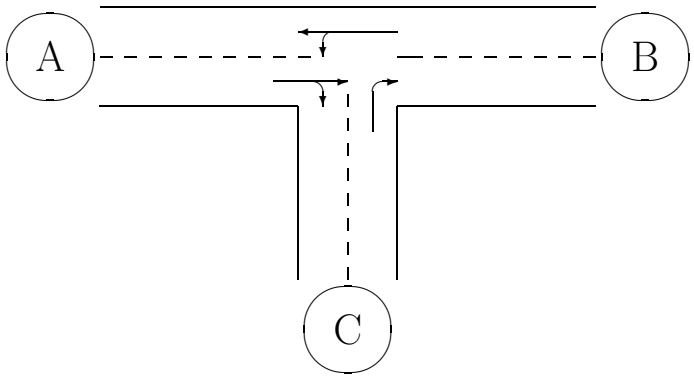



Figura 2.4: Um cruzamento em T.

Assim, à entrada $\{2,3\}\{1,3\}\{1\}$ corresponde a saída $\uparrow\uparrow\downarrow$. Observe que, apesar do último conjunto de chamadas ser $\{1\}$, o elevador não vai até o primeiro andar, porque "alguém" desligou manualmente o chamado, quando o elevador chegou ao segundo andar.

Este tipo de máquina é denominado *Máquina de Mealy*, e será abordado na Seção 2.5.

# Exercícios

1. Faça um diagrama de estados, similar àquele da Figura 2.1, página 58, para o problema dos missionários e canibais:

   > Três missionários e três canibais devem atravessar um rio. Para isto dispõem de uma canoa que pode transportar no máximo 2 pessoas de cada vez. Durante a travessia, se o número de canibais ficar maior do que o de missionários em qualquer uma das margens, os canibais comem os missionários. Determinar um plano para travessia em que nenhum missionário seja devorado.

2. Faça um diagrama de estados, usando a idéia apresentada na Seção 2.1.2, para uma máquina que determine se uma seqüência ternária (com dígitos 0, 1 e 2) é divisível por 4.

3. Suponha um cruzamento em T, como ilustrado na Figura 2.4, com os sentidos possíveis de tráfego lá mostrados: um veículo vindo de $B$ pode ir em frente ou virar para $C$, um vindo de $C$ só pode virar para $B$, etc. O cruzamento deve ser controlado por semáforos que só exibem as cores verde e vermelho; não há amarelo. Suponha que há sensores que determinam o sentido dos veículos. Por exemplo, há um sensor que determina se há veículo se dirigindo de $B$ para $C$, há um que determina se há veículo se dirigindo de $A$ para $B$, etc. Modele o funcionamento dos semáforos por meio de um diagrama de estados, de forma similar àquela utilizada para a modelagem do problema do elevador na Seção 2.1.3.

## 2.2 Autômatos Finitos Determinísticos

Os exemplos apresentados na seção anterior introduziram informalmente a noção de autômato finito na sua modalidade determinística, bem como a maior parte da terminologia associada. Nesta seção serão apresentadas uma definição precisa de autômato finito determinístico, um método para determinar um autômato mínimo equivalente a outro dado, e algumas propriedades importantes de autômatos finitos determinísticos.

### 2.2.1 O que é autômato finito determinístico

Um autômato finito determinístico é uma estrutura matemática constituída de 3 tipos de entidades: um conjunto de estados, um alfabeto e um conjunto de transições. Dos estados, destaca-se um como o estado inicial, e destaca-se um subconjunto de estados como sendo composto dos estados finais. A definição a seguir apresenta de forma precisa esta caracterização. Em particular, o conjunto de transições é apresentado como uma função.

**Definição 1** *Um autômato finito determinístico (AFD) é uma quíntupla* $(E, \Sigma, \delta, i, F)$*, onde*

- $E$ *é um conjunto finito de um ou mais elementos denominados estados;*
- $\Sigma$ *é um alfabeto;*
- $\delta : E \times \Sigma \rightarrow E$ *é a função de transição, uma função total;*
- $i$*, um estado de* $E$*, é o estado inicial;*
- $F$*, subconjunto de* $E$*, é o conjunto de estados finais.* □

Observe como $E$ precisa apenas ser um conjunto finito, e que nada depende da estrutura de seus elementos. Em verdade, se $E$ tem $n$ elementos, formalmente eles precisam apenas ser representados por $n$ nomes distintos (por exemplo, $1, 2, \ldots, n$). No entanto, na prática costuma-se usar nomes mnemômicos, pelos mesmos motivos que, em programação, se usa nomes mnemônicos para variáveis. Assim, no primeiro exemplo da seção anterior, o nome de um estado indica claramente a situação, explicitando os indivíduos presentes na margem esquerda. Algumas pessoas acham ainda mais mnemômico indicar, além do conteúdo da margem esquerda, o conteúdo da margem direita. No segundo exemplo, o nome de um estado indica o resto da divisão por 6 do número representado pelo prefixo que leva até o estado. No terceiro exemplo, o nome de um estado indica o andar em que o elevador está e se ele está subindo ou descendo. É evidente que a concepção de cada um dos autômatos, e o entendimento de cada um deles, são muito facilitados usando-se tais nomes ao invés de se usar nomes arbitrários.

O fato de que um autômato tem apenas um estado inicial não diminui seu poder computacional. Será mostrado mais adiante que o poder computacional de um autômato finito não determinístico, que é o caso de um autômato com mais de um estado inicial, é o mesmo de um AFD.

A Definição 1 modela as transições de um AFD como uma função que mapeia cada par (*estado*, *símbolo*) para *um estado*. Este fato, que cada par (estado, símbolo) leva a

*um único* estado, é que caracteriza o *determinismo* do AFD: a partir do estado inicial, só é possível atingir um único estado para uma dada palavra de entrada. E o fato de que a função é total garante que para toda palavra de entrada atinge-se um estado consumindo-se toda a palavra. Esta exigência simplifica o modelo, facilitando sua manipulação do ponto de vista teórico, sem impor qualquer limitação quanto ao seu poder computacional.

Para conciliar o primeiro exemplo da Seção 2.1 com a definição de AFD, basta inserir mais um estado, digamos $t$ (de "tragédia"), e fazer $\delta(e, a) = t$ se não existir transição de $e$ para algum estado sob $a$; em particular, $\delta(t, a) = t$ para todo $a \in \{\mathtt{s}, \mathtt{l}, \mathtt{c}, \mathtt{r}\}$. Com isto, o exemplo pode ser modelado mediante o autômato (veja Figura 2.1, página 58)

$$(E, \{\mathtt{s}, \mathtt{l}, \mathtt{c}, \mathtt{r}\}, \delta, \{h, l, c, r\}, \{\{\}\})$$

onde $E$ é o conjunto

$$\{\{h, l, c, r\}, \{l, r\}, \{h, l, r\}, \{l\}, \{r\}, \{h, l, c\}, \{h, c, r\}, \{c\}, \{h, c\}, \{\}, t\}$$

e $\delta$ é dada por[3]:

| $\delta$ | s | l | c | r |
|---|---|---|---|---|
| $\{h, l, c, r\}$ | $t$ | $t$ | $\{l, r\}$ | $t$ |
| $\{l, r\}$ | $\{h, l, r\}$ | $t$ | $\{h, l, c, r\}$ | $t$ |
| $\{h, l, r\}$ | $\{l, r\}$ | $\{r\}$ | $t$ | $\{l\}$ |
| $\{l\}$ | $t$ | $t$ | $\{h, l, c\}$ | $\{h, l, r\}$ |
| $\{r\}$ | $t$ | $\{h, l, r\}$ | $\{h, c, r\}$ | $t$ |
| $\{h, l, c\}$ | $t$ | $\{c\}$ | $\{l\}$ | $t$ |
| $\{h, c, r\}$ | $t$ | $t$ | $\{r\}$ | $\{c\}$ |
| $\{c\}$ | $\{h, c\}$ | $\{h, l, c\}$ | $t$ | $\{h, c, r\}$ |
| $\{h, c\}$ | $\{c\}$ | $t$ | $\{\}$ | $t$ |
| $\{\}$ | $t$ | $t$ | $\{h, c\}$ | $t$ |
| $t$ | $t$ | $t$ | $t$ | $t$ |

Assim, para ficar conforme a definição de AFD, o diagrama de estados da Figura 2.1 deveria ser alterado para conter o estado $t$ e as transições relativas a $t$ acima especificadas. No entanto, é útil assumir que, caso alguma transição de algum estado $e$ sob algum símbolo $a$ não esteja especificada no diagrama de estados, então existe um estado (não mostrado no diagrama) $e'$ tal que:

- existe uma transição de $e$ para $e'$ sob $a$;
- $e'$ não é estado final;
- existe uma transição de $e'$ para $e'$ sob cada símbolo do alfabeto.

Observe que os dois últimos itens implicam que se $e'$ é atingido após consumida parte de uma palavra, então o sufixo que se seguir é irrelevante: ele não irá alterar o fato de que a palavra não é reconhecida. Em muitas aplicações este estado é referido como um estado de *erro*, o que é consistente com o primeiro exemplo da seção anterior, na forma em que

[3]Neste formato tabular, uma função $f$ é representada de forma que $f(i, j)$ é exibido na linha $i$, coluna $j$.

ele foi modelado. Embora nem todas as aplicações contenham este "estado de erro", naquelas em que ele está presente, muitas vezes a sua omissão no diagrama de estados simplifica o diagrama, tornando-o mais legível. É este o caso do primeiro exemplo da seção anterior.

Deste ponto em diante, quando se quiser enfatizar que foram omitidos todos os estados de erro porventura existentes, dir-se-á que o diagrama é um *diagrama de estados simplificado*.

O segundo exemplo da seção anterior, que não tem "estado de erro", pode ser modelado mediante o autômato

$$(\{0,1,2,3,4,5\}, \{\mathtt{0},\mathtt{1}\}, \delta, 0, \{0\})$$

onde $\delta$ é dada por:

| $\delta$ | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 2 | 3 |
| 2 | 4 | 5 |
| 3 | 0 | 1 |
| 4 | 2 | 3 |
| 5 | 4 | 5 |

Como ficou dito acima, para qualquer palavra existe um, e apenas um, estado do autômato que pode ser alcançado a partir do estado inicial (e de qualquer outro estado). Assim, define-se abaixo uma função que, dado um estado $e$ e uma palavra $w$, dá o estado alcançado. Evidentemente, ela deve coincidir com a função de transição para palavras de tamanho 1.

**Definição 2** *Seja um AFD $M = (E, \Sigma, \delta, i, F)$. A* função de transição estendida para *$M$, $\hat{\delta}$, é uma função de $E \times \Sigma^*$ para $E$, definida recursivamente como segue:*

*(a)* $\hat{\delta}(e, \lambda) = e$;

*(b)* $\hat{\delta}(e, ay) = \hat{\delta}(\delta(e, a), y)$, *para todo* $a \in \Sigma$ *e* $y \in \Sigma^*$. □

Observe como $\hat{\delta}(e, a)$ coincide com $\delta(e, a)$:

$$\begin{array}{rcll} \hat{\delta}(e,a) & = & \hat{\delta}(\delta(e,a),\lambda) & \text{por } b\text{, Definição 2} \\ & = & \delta(e,a) & \text{por } a\text{, Definição 2.} \end{array}$$

**Exemplo 54** Seja o AFD do segundo exemplo da seção anterior, cujo diagrama de estados é mostrado na Figura 2.2, página 60. Observe como o estado atingido processando-se a palavra `1010` a partir do estado inicial, `0`, é 4:

$$\begin{array}{rcll} \hat{\delta}(0,\mathtt{1010}) & = & \hat{\delta}(\delta(0,\mathtt{1}),\mathtt{010}) & \text{por } b\text{, Definição 2} \\ & = & \hat{\delta}(1,\mathtt{010}) & \text{pois } \delta(0,\mathtt{1}) = 1 \\ & = & \hat{\delta}(\delta(1,\mathtt{0}),\mathtt{10}) & \text{por } b\text{, Definição 2} \\ & = & \hat{\delta}(2,\mathtt{10}) & \text{pois } \delta(1,\mathtt{0}) = 2 \\ & = & \hat{\delta}(\delta(2,\mathtt{1}),\mathtt{0}) & \text{por } b\text{, Definição 2} \\ & = & \hat{\delta}(5,\mathtt{0}) & \text{pois } \delta(2,\mathtt{1}) = 5 \\ & = & \hat{\delta}(\delta(5,\mathtt{0}),\lambda) & \text{por } b\text{, Definição 2} \\ & = & \hat{\delta}(4,\lambda) & \text{pois } \delta(5,\mathtt{0}) = 4 \\ & = & 4 & \text{por } a\text{, Definição 2.} \end{array}$$

□

Utilizando-se $\hat{\delta}$, pode-se definir a *linguagem reconhecida, ou aceita*, por um AFD.

**Definição 3** *A* linguagem reconhecida (aceita) *por um AFD* $M = (E, \Sigma, \delta, i, F)$ *é o conjunto* $L(M) = \{w \in \Sigma^* \mid \hat{\delta}(i, w) \in F\}$. *Uma determinada palavra* $w \in \Sigma^*$ *é dita ser* reconhecida, ou aceita*, por* $M$ *se, e somente se,* $\hat{\delta}(i, w) \in F$. □

Assim, a linguagem aceita pelo AFD do primeiro exemplo é o conjunto de todas as palavras que representam seqüências de movimentos que propiciam o transporte seguro dos elementos citados. E a linguagem aceita pelo AFD do segundo exemplo é o conjunto de toda palavra binária que represente um número divisível por 6.

No primeiro exemplo, o objetivo era encontrar uma palavra $w$ tal que $\hat{\delta}(i, w) \in F$. E no segundo, o objetivo era ter uma máquina abstrata para determinar, para qualquer palavra $w$, se $\hat{\delta}(i, w) \in F$. No primeiro caso, basta usar um algoritmo para determinar um caminho entre dois vértices em um grafo. E no segundo, a resposta é o próprio AFD obtido. Assim, embora o objetivo original em cada exemplo seja diferente, em ambos foram obtidos *reconhecedores* de linguagens.

Em princípio, pode-se pensar em implementar um dado AFD, visto como reconhecedor, nas mais diversas tecnologias, inclusive via um procedimento. Neste último caso, pode-se usar qualquer paradigma de programação, como o procedural, o funcional ou o lógico. Tendo em vista que os maiores usuários deste livro são estudantes de informática, e que, pelo menos por enquanto, o paradigma procedural é mais comum, será exemplificado como implementar um AFD neste último paradigma. O procedimento usa as seguintes variáveis para representar o AFD:

- $i$: estado inicial;
- $F$: conjunto dos estados finais;
- $D$, uma matriz de leitura-apenas, contendo a função de transição, de forma que $D[e, a] = \delta(e, a)$ para todo estado $e$ e símbolo $a$.

Assume-se a existência de um procedimento do tipo função, *prox*, que retorna o próximo símbolo de entrada, quando houver, e *fs* (fim de seqüência), quando não houver. O algoritmo está mostrado na Figura 2.5. Observe que este procedimento é geral, podendo ser aplicado a qualquer AFD recebido como entrada. Uma abordagem que resulta em maior eficiência, mas que leva à obtenção de um procedimento específico para um AFD, é codificar cada transição como se fosse uma "instrução" (veja exercício 7 proposto no final da Seção 2.2, na página 82). De qualquer forma, em ambos os casos, o tempo de execução é diretamente proporcional ao tamanho da palavra de entrada.

**Exemplo 55** Uma das aplicações de AFD's é a análise léxica de compiladores de linguagens de programação, como Pascal, C e Java. Nesta fase, são reconhecidas determinadas entidades sintáticas, como identificadores, constantes inteiras, constantes reais, palavras-chave, etc. Na Figura 2.6 está mostrado o diagrama de estados de um AFD para reconhecimento de constantes reais típicas de uma linguagem de programação de alto nível. Cada transição sob $d$ é, na realidade, uma abreviação para 10 transições: uma para cada um dos dígitos de 0 a 9. Matematicamente, o AFD é uma quíntupla $(E, \Sigma, \delta, e0, F)$, onde:

Entrada: (1) o AFD, dado por $i$, $F$ e $D$, e
(2) a palavra de entrada, dada por *prox*.
Saída: *sim* ou *não*.
$e \leftarrow i$; $s \leftarrow prox()$;
**enquanto** $s \neq fs$ **faça**
$e \leftarrow D[e, s]$; $s \leftarrow prox()$;
**fimenquanto**;
**se** $e \in F$ **então**
retorne *sim*
**senão**
retorne *não*
**fimse**

Figura 2.5: Algoritmo para simular AFD's.

- $E = \{e0, e1, e2, e3, e4, e5, e6, e7, ee\}$ ($ee$ é o "estado de erro")
- $\Sigma = \{\texttt{0}, \texttt{1}, \texttt{2}, \texttt{3}, \texttt{4}, \texttt{5}, \texttt{6}, \texttt{7}, \texttt{8}, \texttt{9}, ., +, -, \texttt{E}\}$
- $F = \{e3, e7\}$
- $\delta$ é dada por (novamente, $d$ representa cada um dos dígitos de `0` a `9`):

| $\delta$ | $d$ | . | + | - | E |
|---|---|---|---|---|---|
| $e0$ | $e2$ | $e4$ | $e1$ | $e1$ | $ee$ |
| $e1$ | $e2$ | $e4$ | $ee$ | $ee$ | $ee$ |
| $e2$ | $e2$ | $e3$ | $ee$ | $ee$ | $e5$ |
| $e3$ | $e3$ | $ee$ | $ee$ | $ee$ | $e5$ |
| $e4$ | $e3$ | $ee$ | $ee$ | $ee$ | $ee$ |
| $e5$ | $e7$ | $ee$ | $e6$ | $e6$ | $ee$ |
| $e6$ | $e7$ | $ee$ | $ee$ | $ee$ | $ee$ |
| $e7$ | $e7$ | $ee$ | $ee$ | $ee$ | $ee$ |
| $ee$ | $ee$ | $ee$ | $ee$ | $ee$ | $ee$ |

□

Antes de apresentar algumas propriedades dos AFD's e seu relacionamento com outros formalismos importantes, serão apresentados mais alguns exemplos. Daqui para frente, será dada ênfase a autômatos finitos como reconhecedores. Tenha em mente, entretanto, que o conceito pode ser usado em vários contextos, não necessariamente como reconhecedores, como mostra o primeiro exemplo da seção anterior.

**Exemplo 56** Será apresentado um AFD $M$ tal que:

$$L(M) = \{w \in \{\texttt{0}, \texttt{1}\}^* \mid w \text{ tem um número par de símbolos}\}.$$

Após lido um prefixo $x$ de uma palavra, o que se deve saber para prosseguir no reconhecimento da mesma? Ora, não é difícil perceber que se deve saber apenas se $x$ tem um número par ou ímpar de símbolos. Assim, o autômato terá apenas dois estados: *par* e

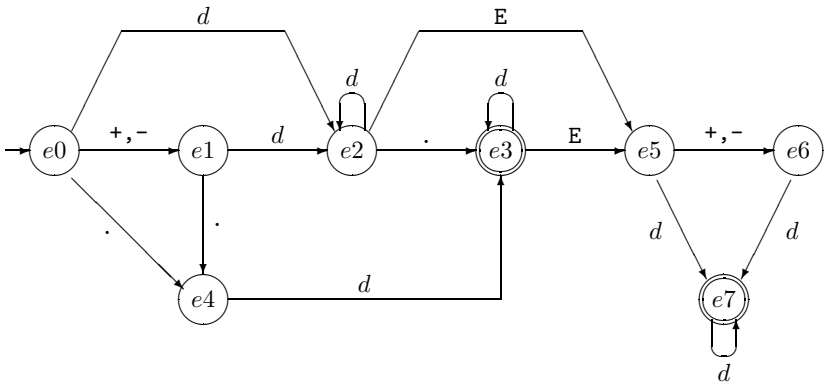



Figura 2.6: Reconhecendo constantes reais.

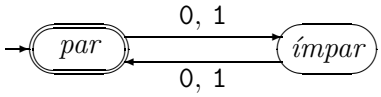



Figura 2.7: Reconhecendo número par de símbolos.

*ímpar*. O diagrama de estados está mostrado na Figura 2.7. Em linguagem matemática, o autômato seria uma quíntupla

$$M = (\{par, ímpar\}, \{0, 1\}, \delta, par, \{par\})$$

onde $\delta$ é dada por:

| $\delta$ | 0 | 1 |
|---|---|---|
| *par* | *ímpar* | *ímpar* |
| *ímpar* | *par* | *par* |

□

**Exemplo 57** Agora apresenta-se um AFD $N$ tal que $L(N) = \{w \in \{0, 1\}^* \mid w$ tem um número par de 0's e um número par de 1's$\}$.

Neste caso, após um prefixo deve-se saber se o número de 0's é par ou ímpar e se o número de 1's é par ou ímpar, para que se tenha controle da paridade de ambos os dígitos todo tempo. O diagrama de estados está mostrado na Figura 2.8. O estado *pp* representa a situação em que o número de 0's e de 1's é par, *pi* representa a situação em que o número de 0's é par e o de 1's é ímpar, e assim por diante. Em linguagem matemática, o autômato seria uma quíntupla

$$N = (\{pp, pi, ip, ii\}, \{0, 1\}, \delta, pp, \{pp\})$$

onde $\delta$ é dada por:

| $\delta$ | 0 | 1 |
|---|---|---|
| *pp* | *ip* | *pi* |
| *pi* | *ii* | *pp* |
| *ip* | *pp* | *ii* |
| *ii* | *pi* | *ip* |

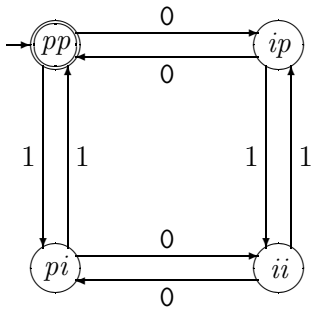



Figura 2.8: Reconhecendo números pares de 0's e de 1's.

□

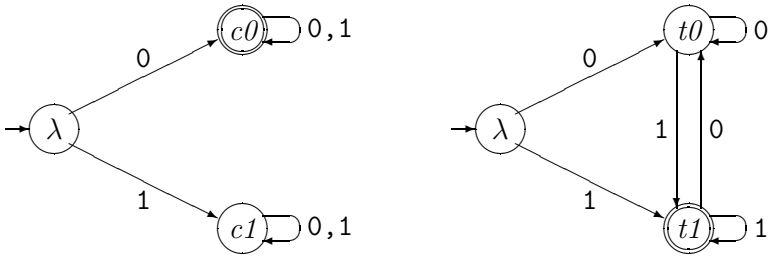
Se o autômato $N$ do Exemplo 57 tivesse como estados finais os estados $pp$ e $ii$, ele aceitaria a mesma linguagem que o AFD do Exemplo 56. É fácil perceber que se uma linguagem pode ser reconhecida por AFD's, então ela pode ser reconhecida por inúmeros AFD's.

**Definição 4** *Dois AFD's, $M_1$ e $M_2$, são ditos* equivalentes *se, e somente se, $L(M_1) = L(M_2)$.* □

Dado que podem existir vários AFD's equivalentes para uma linguagem, tem sentido indagar se existiria um AFD "mínimo" para uma linguagem. Se for assumido um alfabeto mínimo (sem símbolos inúteis), existem duas entidades que influenciam o tamanho de um AFD: os números de estados e de transições. Como a função de transição é total, o número de transições é função apenas do número de estados: quanto menos estados, menos transições. Ou seja, tem sentido dizer que um AFD para reconhecer uma linguagem é *mínimo* se nenhum outro que reconheça a mesma linguagem tem um número de estados menor. Na próxima seção vai ser apresentado um algoritmo que, dado um AFD qualquer, determina um AFD mínimo equivalente. Antes disso, porém, será apresentado mais um exemplo.

**Exemplo 58** Na Figura 2.9 apresenta-se os diagramas de estado de dois AFD's, um que reconhece $A_0 = \{\mathtt{0}y \mid y \in \{\mathtt{0},\mathtt{1}\}^*\}$, e outro que reconhece $A_1 = \{x\mathtt{1} \mid x \in \{\mathtt{0},\mathtt{1}\}^*\}$. O estado inicial de ambos é denominado $\lambda$, o estado $ci$ é atingido para palavras que começam com $i$, e o estado $ti$ é atingido para palavras que terminam com $i$, para $i = \mathtt{0}$ ou $\mathtt{1}$. □

Suponha que se queira um AFD que reconheça a linguagem de todas as palavras de $\{0,1\}^*$ que começam com 0 ou terminam com 1 e que tenham um número par de 0's e um número par de 1's, ou seja, a linguagem $L = (A_0 \cup A_1) \cap L(N)$, onde $A_0$ e $A_1$ são as linguagens do Exemplo 58 e $N$ é o AFD do Exemplo 57. Seria possível obter um AFD para $L$ a partir de $N$ e dos AFD's cujos diagramas de estado estão apresentados na Figura 2.9? A resposta é sim; algoritmos para isto serão esboçados na Seção 2.2.3.




(a) Reconhecendo $\{0\}\{0,1\}^*$. (b) Reconhecendo $\{0,1\}^*\{1\}$.

Figura 2.9: Reconhecendo $\{0\}\{0,1\}^*$ e $\{0,1\}^*\{1\}$.

### 2.2.2 Minimização de AFD's

Como já foi dito, sempre existe um AFD mínimo equivalente a um AFD. Mínimo, no sentido de que não existe nenhum outro com menor número de estados.

**Definição 5** *Um AFD $M$ é dito ser um* AFD mínimo *para a linguagem $L(M)$ se nenhum AFD para $L(M)$ contém menor número de estados que $M$.* □

Pela Definição 5, nenhum AFD que contenha estados não alcançáveis a partir do estado inicial pode ser mínimo. Assim, um primeiro passo para obter um AFD mínimo é a eliminação pura e simples destes estados. Em seguida, deve-se determinar grupos de estados *equivalentes*, no sentido ditado pela Definição 6 abaixo, e substituir cada grupo por um único estado.

**Definição 6** *Seja um AFD $M = (E, \Sigma, \delta, i, F)$. Dois estados $e, e' \in E$ são ditos* equivalentes*, $e \approx e'$, se, e somente se:*

$$\textit{para todo } y \in \Sigma^*,\ \hat{\delta}(e, y) \in F \textit{ se, e somente se, } \hat{\delta}(e', y) \in F.$$

□

Pode-se mostrar que a relação "$\approx$", definida acima, é de fato uma relação de equivalência, ou seja, uma relação reflexiva, simétrica e transitiva.

Dada a Definição 6, durante o processamento de uma palavra, tanto faz atingir $e$ como $e'$: um sufixo $y$ será reconhecido passando-se por $e$ se, e somente se, ele for reconhecido passando-se por $e'$. O que justifica eliminar um dos estados e substituir toda referência a ele por referências ao outro. Ou seja, se $[e] = \{e_1, e_2, \ldots, e_n\}$ é a classe de equivalência de $e$ na partição induzida por $\approx$, os estados $e_1, e_2, \ldots, e_n$ podem ser substituídos por um único estado.

Por outro lado, se os estados $e$ e $e'$ *não* são equivalentes no sentido da Definição 6, ou seja, se $e \not\approx e'$ (equivalentemente, $[e] \neq [e']$), então é porque existe uma palavra $y \in \Sigma^*$ para a qual:

$$\hat{\delta}(e, y) \in F \ \text{ e } \ \hat{\delta}(e', y) \notin F, \text{ ou vice-versa.}$$

Neste caso, atingir $e$ pode significar o reconhecimento de uma palavra e atingir $e'$ pode significar o não reconhecimento da palavra: basta que o sufixo restante da palavra seja

$y$. Assim, se tanto $e$ quanto $e'$ são alcançáveis a partir do estado inicial, os dois estados não podem ser reduzidos a um único.

Na definição a seguir, mostra-se como construir um autômato mínimo equivalente a um AFD $M$, dadas as classes de equivalência induzidas pela relação $\approx$. Não será mostrado formalmente que um autômato reduzido, como está lá definido, é um autômato mínimo. Para os propósitos deste texto são suficientes as evidências apontadas na argumentação acima mais o resultado do Teorema 2, apresentado abaixo.

**Definição 7** *Seja um AFD $M = (E, \Sigma, \delta, i, F)$. Um* autômato reduzido *correspondente a $M$ é o AFD $M' = (E', \Sigma, \delta', i', F')$, onde:*

- $E' = \{[e] \mid e \in E\}$;
- $\delta'([e], a) = [\delta(e, a)]$ *para todo* $e \in E$ *e* $a \in \Sigma$;
- $i' = [i]$;
- $F' = \{[e] \mid e \in F\}$. □

Note que $\delta'$ é bem definida, pois se $[e] = [e']$, então $[\delta(e, a)] = [\delta(e', a)]$ para qualquer $a \in \Sigma^*$. Para provar isto, suponha que $[e] = [e']$, ou seja, $e \approx e'$, e sejam $a \in \Sigma$ e $y \in \Sigma^*$ arbitrários. Então:

$$\begin{array}{lll} \hat{\delta}(\delta(e,a),y) \in F \leftrightarrow \hat{\delta}(e,ay) \in F & & \text{pela definição de } \hat{\delta} \text{ (Definição 2)} \\ \leftrightarrow \hat{\delta}(e',ay) \in F & & \text{pela Definição 6, pois } e \approx e' \\ \leftrightarrow \hat{\delta}(\delta(e',a),y) \in F & & \text{pela definição de } \hat{\delta}. \end{array}$$

Pela Definição 6, conclui-se que $[\delta(e, a)] = [\delta(e', a)]$.

**Teorema 2** *Um autômato reduzido correspondente a um AFD $M$ é equivalente a $M$.*

*Prova*

Sejam AFD's $M$ e $M'$ como na Definição 7. Primeiramente, mostra-se que

$$\hat{\delta}'([e], w) = [\hat{\delta}(e, w)] \text{ para todo } w \in \Sigma^* \tag{2.1}$$

por indução sobre $|w|$. Se $|w| = 0$, tem-se:

$$\begin{array}{lll} \hat{\delta}'([e],\lambda) = [e] & & \text{pela definição de } \hat{\delta} \text{ (Definição 2)} \\ = [\hat{\delta}(e,\lambda)] & & \text{pela definição de } \hat{\delta}. \end{array}$$

Seja $y \in \Sigma^*$ arbitrário, e suponha, como hipótese de indução, que $\hat{\delta}'([e], y) = [\hat{\delta}(e, y)]$. Basta, então, provar que $\hat{\delta}'([e], ay) = [\hat{\delta}(e, ay)]$ para um $a \in \Sigma$ arbitrário. De fato:

$$\begin{array}{lll} \hat{\delta}'([e],ay) = \hat{\delta}'(\delta'([e],a),y) & & \text{pela definição de } \hat{\delta} \\ = \hat{\delta}'([\delta(e,a)],y) & & \text{pela definição de } \delta' \\ = [\hat{\delta}(\delta(e,a),y)] & & \text{pela hipótese de indução} \\ = [\hat{\delta}(e,ay)] & & \text{pela definição de } \hat{\delta}. \end{array}$$

Agora, para provar que $L(M) = L(M')$, prova-se que para todo $w \in \Sigma^*$, $\hat{\delta}'(i', w) \in F' \leftrightarrow \hat{\delta}(i, w) \in F$:

$$\begin{aligned}
\hat{\delta}'(i', w) \in F' &\leftrightarrow \hat{\delta}'([i], w) \in F' && \text{pela definição de } i' \\
&\leftrightarrow [\hat{\delta}(i, w)] \in F' && \text{por 2.1} \\
&\leftrightarrow \hat{\delta}(i, w) \in F && \text{pela definição de } F'.
\end{aligned}$$

□

Assim, o problema básico do algoritmo de minimização é encontrar as classes de equivalência induzidas pela relação $\approx$. A definição indutiva a seguir será a base para a obtenção das partições passo a passo, mediante aplicações simples da função de transição $\delta$. Define-se uma seqüência de relações de equivalência $\approx_0, \approx_1, \approx_2 \ldots$, de forma que $\approx_{n+1}$ é um refinamento de $\approx_n$, ou seja, cada classe de equivalência de $\approx_{n+1}$ está contida em uma classe de equivalência de $\approx_n$.

**Definição 8** *Seja um AFD $M = (E, \Sigma, \delta, i, F)$. Segue uma definição de $\approx_i$ (lê-se "é i-equivalente"), $i \geq 0$:*

*(a) $e \approx_0 e'$ se, e somente se, ($e, e' \in F$ ou $e, e' \in E - F$);*

*(b) para $n \geq 0$: $e \approx_{n+1} e'$ se, e somente se, $e \approx_n e'$ e $\delta(e, a) \approx_n \delta(e', a)$ para todo $a \in \Sigma$.* □

A notação $\approx_n$ é justificada pelo lema a seguir.

**Lema 1** *Seja um AFD $M = (E, \Sigma, \delta, i, F)$, e dois estados $e, e' \in E$. Então $e \approx_n e'$ se, e somente se, para todo $w \in \Sigma^*$ tal que $|w| \leq n$, $\hat{\delta}(e, w) \in F \leftrightarrow \hat{\delta}(e', w) \in F$.*

*Prova*

A demonstração será feita por indução sobre $n$. Para $n = 0$, tem-se:

$$\begin{aligned}
e \approx_0 e' &\leftrightarrow [e, e' \in F \text{ ou } e, e' \in E - F] && \text{pela Definição 8} \\
&\leftrightarrow [\hat{\delta}(e, \lambda) \in F \leftrightarrow \hat{\delta}(e', \lambda) \in F] && \text{pela definição de } \hat{\delta}.
\end{aligned}$$

Supondo que o resultado vale para um certo $n \geq 0$, deve-se provar que $e \approx_{n+1} e'$ se, e somente se, para todo $w \in \Sigma^*$ tal que $|w| \leq n + 1$, $\hat{\delta}(e, w) \in F \leftrightarrow \hat{\delta}(e', w) \in F$. De fato:

$$\begin{aligned}
e \approx_{n+1} e' &\leftrightarrow [e \approx_n e' \text{ e para todo } a \in \Sigma,\ \delta(e, a) \approx_n \delta(e', a)] && \text{pela Definição 8} \\
&\leftrightarrow [\text{para todo } x \in \Sigma^* \text{ tal que } |x| \leq n,\ \hat{\delta}(e, x) \in F \leftrightarrow \hat{\delta}(e', x) \in F, \\
&\quad \text{e para todo } a \in \Sigma \text{ e todo } y \in \Sigma^* \text{ tal que } |y| \leq n, \\
&\qquad \hat{\delta}(\delta(e, a), y) \in F \leftrightarrow \hat{\delta}(\delta(e', a), y) \in F] \\
&\qquad && \text{pela hipótese de indução} \\
&\leftrightarrow [\text{para todo } x \in \Sigma^* \text{ tal que } |x| \leq n,\ \hat{\delta}(e, x) \in F \leftrightarrow \hat{\delta}(e', x) \in F, \\
&\quad \text{e para todo } a \in \Sigma \text{ e todo } y \in \Sigma^* \text{ tal que } |y| \leq n,\ \hat{\delta}(e, ay) \in F \leftrightarrow \hat{\delta}(e', ay) \in F] \\
&\qquad && \text{pela definição de } \hat{\delta} \\
&\leftrightarrow [\text{para todo } x \in \Sigma^* \text{ tal que } |x| \leq n,\ \hat{\delta}(e, x) \in F \leftrightarrow \hat{\delta}(e', x) \in F, \\
&\quad \text{e para todo } z \in \Sigma^* \text{ tal que } 1 \leq |z| \leq n + 1,\ \hat{\delta}(e, z) \in F \leftrightarrow \hat{\delta}(e', z) \in F] \\
&\leftrightarrow [\text{para todo } w \in \Sigma^* \text{ tal que } 0 \leq |w| \leq n + 1,\ \hat{\delta}(e, w) \in F \leftrightarrow \hat{\delta}(e', w) \in F].
\end{aligned}$$

□

Finalmente, a ligação entre as Definições 6 e 8 é feita pelo teorema a seguir.

**Teorema 3** *$e \approx e'$ se, e somente se, $e \approx_n e'$ para todo $n \geq 0$.*

*Prova*

$$\begin{aligned} e \approx e' &\leftrightarrow \text{para todo } w \in \Sigma^*,\ \hat{\delta}(e,w) \in F \leftrightarrow \hat{\delta}(e',w) \in F \quad \text{pela Definição 6} \\ &\leftrightarrow \text{para todo } n \geq 0 \text{ e todo } w \in \Sigma^* \text{ tal que } |w| \leq n, \\ &\qquad\qquad \hat{\delta}(e,w) \in F \leftrightarrow \hat{\delta}(e',w) \in F \\ &\leftrightarrow \text{para todo } n \geq 0,\ e \approx_n e' \qquad\qquad \text{pelo Lema 1.} \end{aligned}$$

□

A seguinte reformulação da Definição 8 em termos das partições sucessivas induzidas pelas relações $\approx_n$, serve de base para a formulação do algoritmo a ser apresentado em seguida. Será usada a notação $[e]_n$ para denotar a classe de equivalência a que pertence o estado $e$ na partição induzida por $\approx_n$. Seja um AFD $M = (E, \Sigma, \delta, i, F)$ e um estado $e \in E$. Segue uma definição de $[e]_i$, $i \geq 0$:

(a) $[e]_0 = \begin{cases} F & \text{se } e \in F \\ E - F & \text{se } e \in E - F \end{cases}$

(b) para $n \geq 0$, $[e]_{n+1} = \{e' \in [e]_n \mid [\delta(e',a)]_n = [\delta(e,a)]_n \text{ para todo } a \in \Sigma\}$.

O algoritmo de minimização, baseado nesta última definição, está mostrado na Figura 2.10. No algoritmo, as partições sucessivas do conjunto de estados $E$ são denotadas $S_0, S_1, \ldots, S_n$. Assim, tem-se que $S_i = \{[e]_i \mid e \in E\}$. Ao se atingir o estágio em que $S_n = S_{n-1}$, tem-se que $S_n$ é o conjunto de estados do AFD mínimo.

**Exemplo 59** No Exemplo 58 foram apresentados dois AFD's simples, um para a linguagem $A_0 = \{0y \mid y \in \{0,1\}^*\}$, e outro para $A_1 = \{x1 \mid x \in \{0,1\}^*\}$. Seus diagramas de estado estão explicitados na Figura 2.9, página 70. Aplicando-se o algoritmo de minimização da Figura 2.10 para o primeiro AFD, as partições do conjunto de estados evoluem assim:

$S_0$: $\{\lambda, c1\}, \{c0\}$ (inicializado pelo algoritmo)
$S_1$: $\{\lambda\}, \{c1\}, \{c0\}$ (pois $\delta(\lambda, 0) \in \{c0\}$ e $\delta(c1, 0) \in \{\lambda, c1\}$)
$S_2$: $\{\lambda\}, \{c1\}, \{c0\}$

e, portanto o AFD não muda. Aplicando-se o mesmo algoritmo para o segundo AFD, obtém-se apenas as partições $S_0$ e $S_1$:

$S_0$: $\{\lambda, t0\}, \{t1\}$ (inicializado pelo algoritmo)
$S_1$: $\{\lambda, t0\}, \{t1\}$.

Portanto, o AFD mínimo é $(\{\{\lambda, t0\}, \{t1\}\}, \{0,1\}, \delta, \{\lambda, t0\}, \{\{t1\}\})$, onde $\delta$ é dada por:

| $\delta$ | 0 | 1 |
|---|---|---|
| $\{\lambda, t0\}$ | $\{\lambda, t0\}$ | $\{t1\}$ |
| $\{t1\}$ | $\{\lambda, t0\}$ | $\{t1\}$ |

Entrada: um AFD $P = (E, \Sigma, \delta, i, F)$.
Saída: um AFD mínimo equivalente a $P$.

elimine de $P$ todo estado não alcançável a partir de $i$;
**se** $F = \emptyset$ **então**
    retorne $(\{i\}, \Sigma, \delta', i, \emptyset)$, onde $\delta'(i, a) = i$ para todo $a \in \Sigma$
**senãose** $E - F = \emptyset$ **então**
    retorne $(\{i\}, \Sigma, \delta', i, \{i\})$, onde $\delta'(i, a) = i$ para todo $a \in \Sigma$
**fimse**;
$S_0 \leftarrow \{E - F, F\}$; $n \leftarrow 0$;
**repita**
    $n \leftarrow n + 1$
    $S_n \leftarrow \emptyset$;
    **para** cada $X \in S_{n-1}$ **faça**
        **repita**
            selecione um estado $e \in X$;
            para cada $a \in \Sigma$:
                seja $[\delta(e, a)]$ o conjunto que contém $\delta(e, a)$ em $S_{n-1}$;
            seja $Y = \{e' \in X \mid \delta(e', a) \in [\delta(e, a)] \text{ para todo } a \in \Sigma\}$;
            $X \leftarrow X - Y$;
            $S_n \leftarrow S_n \cup \{Y\}$
        **até** $X = \emptyset$
    **fimpara**;
**até** $S_n = S_{n-1}$;
$i' \leftarrow$ conjunto em $S_n$ que contém $i$;
$F' \leftarrow \{X \in S_n \mid X \subseteq F\}$;
para cada $X \in S_n$ e $a \in \Sigma$:
    $\delta'(X, a) =$ conjunto em $S_n$ que contém $\delta(e, a)$, para qualquer $e \in X$;
retorne $(S_n, \Sigma, \delta', i', F')$

Figura 2.10: Algoritmo de minimização de AFD's.

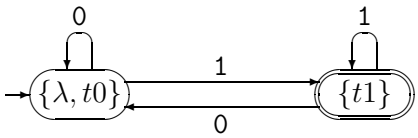



Figura 2.11: AFD mínimo para reconhecer $\{0, 1\}^*\{1\}$.

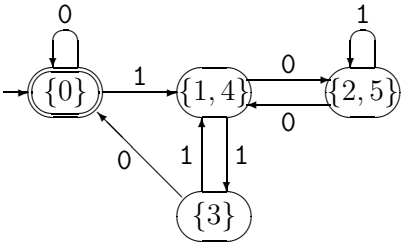



Figura 2.12: AFD mínimo para AFD da Figura 2.2.

Seu diagrama de estados está mostrado na Figura 2.11. □

O seguinte exemplo mostra que às vezes se pode obter um AFD mais conciso em situações não muito óbvias (pelo menos à primeira vista).

**Exemplo 60** Seja o AFD cujo diagrama de estados está mostrado na Figura 2.2, página 60. Aplicando-se o algoritmo de minimização da Figura 2.10, as partições evoluem da seguinte forma:

$S_0$: $\{0\}, \{1, 2, 3, 4, 5\}$
$S_1$: $\{0\}, \{1, 2, 4, 5\}, \{3\}$
$S_2$: $\{0\}, \{1, 4\}, \{2, 5\}, \{3\}$
$S_3$: $\{0\}, \{1, 4\}, \{2, 5\}, \{3\}$.

A Figura 2.12 mostra o diagrama de estados para o AFD mínimo correspondente. □

Pode ser demonstrado também que os AFD's mínimos equivalentes a um AFD são *isomorfos*, ou seja, eles são idênticos a menos dos nomes dos estados.

### 2.2.3 Algumas propriedades dos AFD's

A seguir serão apresentadas algumas propriedades dos AFD's importantes dos pontos de vista prático e teórico.

O teorema a seguir mostra que se existem AFD's para duas linguagens, então existem também AFD's para a união e a interseção delas; mostra também que existe AFD para o complemento da linguagem aceita por qualquer AFD.

Em preparação para o teorema, será mostrado como construir um AFD, $M_3$, que simula o funcionamento "em paralelo" de dois AFD's $M_1 = (E_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$. Para isto, serão colocados como estados de $M_3$ pares de estados de $E_1 \times E_2$, e a função de transição $\delta_3$ será tal que, para todo $e_1 \in E_1$, $e_2 \in E_2$ e $a \in \Sigma$:

$$\delta_3([e_1, e_2], a) = [\delta_1(e_1, a), \delta_2(e_2, a)]. \tag{2.2}$$

O estado inicial será $[i_1, i_2]$. E a composição do conjunto dos estados finais, $F_3$, que será especificada no Teorema 4 abaixo, irá depender do que se quer para $M_3$: que reconheça $L(M_1) \cup L(M_2)$ ou $L(M_1) \cap L(M_2)$.

O lema a seguir será usado na prova do Teorema 4.

**Lema 2** *Sejam dois estados $e_1 \in E_1$ e $e_2 \in E_2$ de um AFD $M_3$ definido como acima. Então,*

$$\hat{\delta_3}([e_1, e_2], w) = [\hat{\delta_1}(e_1, w), \hat{\delta_2}(e_2, w)], \text{ para todo } w \in \Sigma^*.$$

*Prova*

A prova será feita por indução sobre $|w|$. Para $|w| = 0$, tem-se:

$$\begin{aligned} \hat{\delta_3}([e_1, e_2], \lambda) &= [e_1, e_2] && \text{pela definição de } \hat{\delta} \\ &= [\hat{\delta_1}(e_1, \lambda), \hat{\delta_2}(e_2, \lambda)] && \text{pela definição de } \hat{\delta}. \end{aligned}$$

Suponha que $\hat{\delta_3}([e_1, e_2], w) = [\hat{\delta_1}(e_1, w), \hat{\delta_2}(e_2, w)]$, como hipótese de indução, para um determinado $w \in \Sigma^*$. Basta então mostrar que $\hat{\delta_3}([e_1, e_2], aw) = [\hat{\delta_1}(e_1, aw), \hat{\delta_2}(e_2, aw)]$ para qualquer $a \in \Sigma$. De fato:

$$\begin{aligned} \hat{\delta_3}([e_1, e_2], aw) &= \hat{\delta_3}(\delta_3([e_1, e_2], a), w) && \text{pela definição de } \hat{\delta} \\ &= \hat{\delta_3}([\delta_1(e_1, a), \delta_2(e_2, a)], w) && \text{por 2.2} \\ &= [\hat{\delta_1}(\delta_1(e_1, a), w), \hat{\delta_2}(\delta_2(e_2, a), w)] && \text{pela hipótese de indução} \\ &= [\hat{\delta_1}(e_1, aw), \hat{\delta_2}(e_2, aw)] && \text{pela definição de } \hat{\delta}. \end{aligned}$$

□

Segue, então, o Teorema 4.

**Teorema 4** *Sejam dois AFD's $M_1$ e $M_2$. Existem AFD's para as seguintes linguagens:*

*(1)* $\overline{L(M_1)}$*;*

*(2)* $L(M_1) \cap L(M_2)$*; e*

*(3)* $L(M_1) \cup L(M_2)$*.*

*Prova*

Suponha que $M_1 = (E_1, \Sigma, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \delta_2, i_2, F_2)$.

Prova de (1). Um AFD $M_1'$ que aceita $\overline{L(M_1)}$ pode ser obtido a partir de $M_1$ simplesmente colocando-se como estados finais aqueles que não são finais em $M_1$, ou seja, $M_1' = (E_1, \Sigma, \delta_1, i_1, E_1 - F_1)$.

Prova de (2). Seja o AFD $M_3 = (E_1 \times E_2, \Sigma, \delta_3, [i_1, i_2], F_3)$, construído como acima, antes do Lema 2, e onde $F_3 = F_1 \times F_2$. (O caso em que $M_1$ e $M_2$ têm alfabetos diferentes é abordado no exercício 13 no final da seção, página 82).

Abaixo, para provar que $L(M_3) = L(M_1) \cap L(M_2)$, prova-se que $w \in L(M_3) \leftrightarrow w \in L(M_1) \cap L(M_2)$ para $w \in \Sigma^*$ arbitrário:

$$\begin{aligned} w \in L(M_3) &\leftrightarrow \hat{\delta_3}([i_1, i_2], w) \in F_1 \times F_2 && \text{pela Definição 3} \\ &\leftrightarrow [\hat{\delta_1}(i_1, w), \hat{\delta_2}(i_2, w)] \in F_1 \times F_2 && \text{pelo Lema 2} \\ &\leftrightarrow \hat{\delta_1}(i_1, w) \in F_1 \text{ e } \hat{\delta_2}(i_2, w) \in F_2 && \text{pela definição de produto} \\ &\leftrightarrow w \in L(M_1) \text{ e } w \in L(M_2) && \text{pela Definição 3} \\ &\leftrightarrow w \in L(M_1) \cap L(M_2) && \text{pela definição de interseção.} \end{aligned}$$

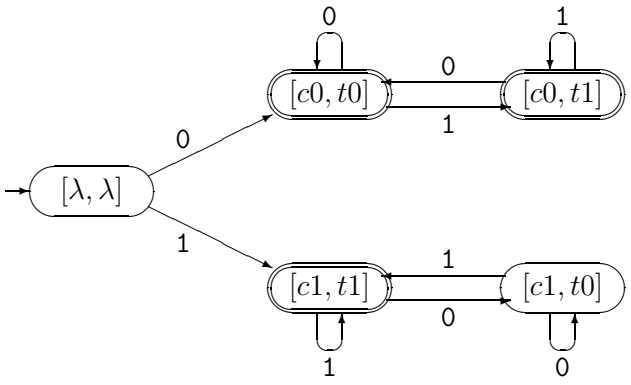



Figura 2.13: Reconhecendo $\{0\}\{0,1\}^* \cup \{0,1\}^*\{1\}$.

Prova de (3). Ora, a existência de AFD para (1) e (2) implica na existência de AFD para (3), pela lei de De Morgan: $L(M_1) \cup L(M_2) = \overline{\overline{L(M_1)} \cap \overline{L(M_2)}}$. E mais, esta lei mais as técnicas acima podem ser usados para construir um AFD para $L(M_1) \cup L(M_2)$: a técnica da complementação seria usada duas vezes, depois a da concatenação, e, novamente, a da complementação. Além disso, de forma análoga ao item (2), pode-se mostrar que $M_3 = (E_1 \times E_2, \Sigma, \delta_3, [i_1, i_2], F_3)$, onde $F_3 = (F_1 \times E_2) \cup (E_1 \times F_2)$, também reconhece $L(M_1) \cup L(M_2)$. □

A prova do Teorema 4 mostra como construir AFD's para $L(M_1) \cup L(M_2)$, $L(M_1) \cap L(M_2)$ e $\overline{L(M_1)}$, a partir de dois AFD's quaisquer $M_1$ e $M_2$. Evidentemente, tais técnicas podem ser úteis para a obtenção de autômatos passo a passo, a partir de autômatos menores e/ou mais simples. Assim, por exemplo, se você tivesse dificuldades para construir um AFD para o conjunto das palavras binárias divisíveis por 6, você poderia construir um AFD $M_1$ para o conjunto das palavras binárias divisíveis por 2 e um AFD $M_2$ para o conjunto das palavras binárias divisíveis por 3; e usando a técnica do Teorema 4, bastaria obter então um AFD para $L(M_1) \cap L(M_2)$.

**Exemplo 61** Sejam os AFD's cujos diagramas de estado estão mostrados na Figura 2.9, página 70, que reconhecem as linguagens $A_0 = \{0y \mid y \in \{0,1\}^*\}$ e $A_1 = \{x1 \mid x \in \{0,1\}^*\}$. O diagrama de estados de um AFD para reconhecer $A_0 \cup A_1$, construído utilizando a técnica do Teorema 4, está mostrado na Figura 2.13. Nela não estão mostrados estados inatingíveis a partir do estado inicial, nem as transições relativas as eles.

Um AFD para reconhecer $A_0 \cap A_1$ teria como única diferença o conjunto de estados finais, que seria $\{[c0, t1]\}$. □

A beleza conceitual e a aplicabilidade dos autômatos finitos devem-se muito à possibilidade, delineada acima, de composição e decomposição. Mais sobre isto será visto na Seção 2.4.2.

Não é difícil perceber que *para toda linguagem finita existe um AFD que a reconhece*. E mais: dentre os AFD's que reconhecem uma linguagem finita, existem aqueles cujos diagramas de estado simplificados *não contêm* ciclos. O exemplo a seguir ilustra que é sempre possível construir um AFD para uma linguagem finita cujo diagrama de estados

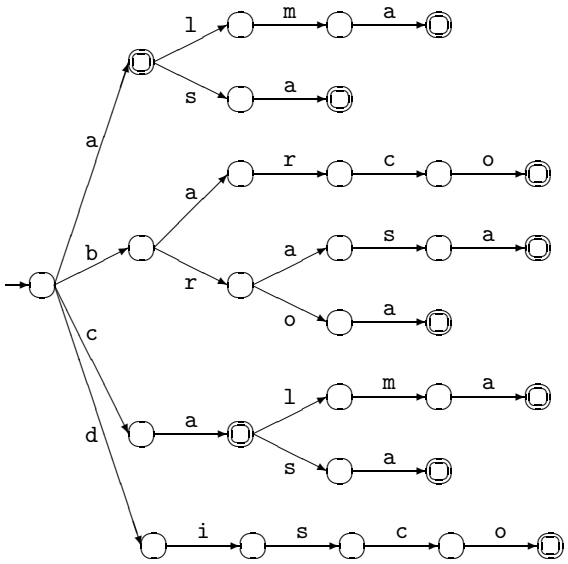



Figura 2.14: Um diagrama de estados do tipo árvore.

simplificado tem um formato de árvore, onde o estado inicial é a raiz. Neste exemplo, não são colocados nomes para os estados para não sobrecarregar o diagrama de estados.

**Exemplo 62** Um exemplo de linguagem finita que contém uma quantidade grande de palavras é o conjunto de todas as palavras do dicionário de uma língua natural. Existem analisadores léxicos e corretores ortográficos baseados em autômatos finitos. Para dar uma idéia de como seria um AFD neste caso, considere o seguinte conjunto, cuja única diferença para com o dicionário completo é o número de palavras:

$$K = \{\texttt{a}, \texttt{alma}, \texttt{asa}, \texttt{barco}, \texttt{brasa}, \texttt{broa}, \texttt{ca}, \texttt{calma}, \texttt{casa}, \texttt{disco}\}.$$

Um diagrama de estados simplificado do tipo árvore pode ser construído facilmente para o reconhecimento de $K$ (não tão facilmente para o dicionário, por causa da grande quantidade de palavras), aproveitando-se a mesma seção do diagrama para cada prefixo, como mostrado na Figura 2.14. Um AFD mais conciso (com menor número de estados) pode ser construído, considerando-se que algumas palavras têm sufixos idênticos, como mostra a Figura 2.15. □

Baseando-se no Exemplo 62, vê-se que é fácil construir um AFD com um diagrama de estados simplificado do tipo árvore para qualquer conjunto finito; a essência é a existência de um único caminho no diagrama para cada prefixo. E mais, é possível construir um algoritmo que, recebendo como entrada as palavras do conjunto, obtém o AFD (ver exercício 14, no final da seção, página 83). Vê-se ainda que, se forem sendo introduzidas mais e mais palavras, a construção do AFD mais conciso vai se tornando mais e mais difícil, correndo-se o risco de se obter um AFD incorreto ou com um número de estados maior do que o necessário. Assim, é oportuno que exista o algoritmo de minimização da

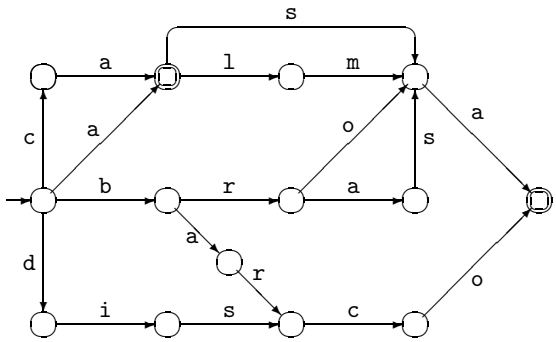



Figura 2.15: AFD mais conciso para o Exemplo 62.

Figura 2.10, página 74: basta aplicar este algoritmo ao AFD do tipo árvore para obter o AFD mais conciso possível.

Assim, se uma linguagem é finita, existe um AFD que a reconhece cujo diagrama de estados simplificado não contém ciclos. Por outro lado, se um diagrama de estados simplificado de um AFD não contém ciclos, então a linguagem que tal AFD reconhece é finita. Assim, tem-se que: *uma linguagem é finita se, e somente se, existe algum AFD que a reconhece cujo diagrama de estados simplificado não tem ciclos*. Dizer isto é equivalente a dizer que *uma linguagem $L$ é infinita se, e somente se*

*(a) não existe AFD que reconhece $L$; ou*

*(b) o diagrama de estados simplificado de qualquer AFD que a reconhece tem ciclo.*

Ora, se um AFD reconhece uma linguagem infinita, é óbvio que seu diagrama de estados simplificado tem que ter ciclo, pois uma linguagem infinita tem palavra de todo tamanho; em particular, tem palavra de tamanho maior ou igual ao número de estados do AFD. E para reconhecer uma palavra de tamanho maior ou igual ao número de estados, deve-se passar por um ciclo.

Seja uma linguagem infinita. Como saber se existe ou não um AFD que a reconhece? Mais especificamente, como mostrar que *não existe* um AFD que reconhece uma linguagem quando todas as tentativas de construir um foram infrutíferas ou, melhor ainda, quando "se desconfia" que a linguagem tem uma estrutura um pouco complexa para ser reconhecível por AFD? Existem várias técnicas para isto, como será visto mais à frente. Uma delas tem como base o arrazoado acima, como mostra o exemplo a seguir.

**Exemplo 63** Seja $L = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$. Como $L$ é infinita, pode existir ou não um AFD para $L$. Suponha que existe um AFD $M$ para $L$. Pelo arrazoado acima, o diagrama de estados simplificado de $M$ contém ciclo. Um ciclo é percorrido necessariamente para a computação correspondente ao reconhecimento de alguma palavra $z$ de tamanho maior ou igual ao número de estados do AFD. Seja $v$ uma subsequência de $z$ consumida ao se percorrer o ciclo (obviamente, $v \neq \lambda$). Neste caso, $z = uvw$ para algum prefixo $u$ e sufixo $w$. Ora, o ciclo pode ser percorrido quantas vezes se queira, inclusive 0, antes de se consumir o sufixo $w$. Assim,

$$uv^iw \in L \text{ para todo } i \geq 0. \qquad (1)$$

Seja $z = \mathtt{a}^k\mathtt{b}^k$ para algum $k$ tal que $|z|$ é maior ou igual ao número de estados de $M$. Então $uv^2w \notin L$, qualquer que seja $v$, pois:

- se $v$ contém apenas $\mathtt{a}$'s, $uv^2w = \mathtt{a}^{k+|v|}\mathtt{b}^k$;
- se $v = \mathtt{a}^i\mathtt{b}^j$ para $1 \leq i, j \leq k$, $uv^2w = \mathtt{a}^{k-i}(\mathtt{a}^i\mathtt{b}^j)^2\mathtt{b}^{k-j} = \mathtt{a}^k\mathtt{b}^j\mathtt{a}^i\mathtt{b}^k$; e
- se $v$ contém apenas $\mathtt{b}$'s, $uv^2w = \mathtt{a}^k\mathtt{b}^{k+|v|}$.

Mas isto contradiz (1). Observe, então, que a existência de ciclo implicaria no reconhecimento de palavras que não pertencem a $L$. Logo, não existe AFD para $L$. □

A técnica utilizada no Exemplo 63 pode ser utilizada para mostrar que várias outras linguagens não são reconhecíveis por AFD's. Se, ao utilizar esta mesma técnica, se utilizar ainda o fato de que, ao percorrer um ciclo de um diagrama de estados de um AFD de $k$ estados, consome-se no máximo $k$ símbolos, pode-se mostrar para uma classe ainda maior de linguagens, que elas não são reconhecíveis por AFD's. Na realidade, esta é a essência do uso do denomimado *lema do bombeamento*, que será apresentado na Seção 2.4.1, para mostrar que uma linguagem não pode ser reconhecida por AFD. Abaixo é apresentado um teorema, cujo formato é muito parecido com o lema do bombeamento, que, no fundo, é uma formulação da técnica utilizada no Exemplo 63.

**Teorema 5** *Seja um AFD $M$ de $k$ estados, e $z \in L(M)$ tal que $|z| \geq k$. Então existem palavras $u$, $v$ e $w$ tais que:*

- $z = uvw$;
- $v \neq \lambda$; *e*
- $uv^iw \in L(M)$ *para todo* $i \geq 0$.

*Prova*

Ora, se $z \in L(M)$ é tal que $|z| \geq k$, então a computação que leva ao reconhecimento de $z$ percorre um ciclo. Basta então tomar $z = uvw$, onde $v$ é uma subpalavra de $z$ consumida ao percorrer o ciclo. □

O teorema seguinte mostra alguns problemas decidíveis no contexto de AFD's. Tais problemas para outros tipos de máquinas podem não ser decidíveis, como será visto posteriormente.

**Teorema 6** *Existem procedimentos de decisão para determinar, para qualquer AFD $M$, se:*

*(1)* $L(M) = \emptyset$; *e*

*(2)* $L(M)$ *é finita.*

*Prova*

Seja $M'$ um AFD mínimo equivalente a $M$, obtido de acordo com o algoritmo da Figura 2.10. Então:

1) $L(M) = \emptyset$ se, e somente se, $M'$ não tem estados finais, ou seja, $M'$ tem um único estado e este não é final;

2) $L(M)$ é finita se, e somente se, o diagrama de estados simplificado de $M'$ não tem ciclos.

No primeiro caso, é trivial verificar se $M'$ tem estados finais. No segundo, é bem sabido que existe algoritmo para verificar se um grafo tem ciclos. □

# Exercícios

1. Construa AFD's para as seguintes linguagens sobre o alfabeto $\{0, 1\}$:
   (a) O conjunto das palavras de tamanho 3.
   (b) O conjunto das palavras de tamanho menor do que 3.
   (c) O conjunto das palavras de tamanho maior do que 3.
   (d) O conjunto das palavras de tamanho múltiplo de 3.
   (e) O conjunto das palavras com no máximo três 1's.
   (f) O conjunto das palavras que contêm um ou dois 1's, cujo tamanho é múltiplo de 3.

2. Construa AFD's para as seguintes linguagens:
   (a) $\{\lambda, 0\}^2$.
   (b) $\{w \in \{0, 1\}^* \mid$ cada 0 de $w$ é imediatamente seguido de, no mínimo, dois 1's$\}$.
   (c) $\{w \in \{0, 1\}^* \mid$ os primeiros 4 símbolos de $w$ contêm, no mínimo, dois 1's$\}$.
   (d) $\{w \in \{0, 1\}^* \mid w$ não contém 000 nem 111$\}$.
   (e) $\{w \in \{0, 1\}^* \mid$ os últimos 3 símbolos de $w$ não são 000$\}$.
   (f) $\{w \in \{0, 1, 2\}^* \mid w$ tem número par de 0's, par de 1's e par de 2's$\}$.

3. Construa AFD's para as linguagens sobre o alfabeto $\{0, 1\}$, a seguir. Considere que o símbolo na posição 1 de uma palavra é o primeiro símbolo desta, o símbolo na posição 2 é o segundo, e assim por diante.
   (a) O conjunto das palavras em que o símbolo na posição $2i$ é diferente do símbolo na posição $2i + 2$, para $i \geq 1$.
   (b) O conjunto das palavras em que o símbolo na posição $2i - 1$ é diferente do símbolo na posição $2i$, para $i \geq 1$.
   (c) O conjunto das palavras em que o símbolo na posição $i$ é diferente do símbolo na posição $i + 2$, para $i \geq 1$.
   (d) O conjunto das palavras com número ímpar de 0's nas posições ímpares e número par de 0's nas posições pares.

(e) O conjunto das palavras de tamanho par com 1's nas posições pares, acrescido das palavras de tamanho ímpar com 1's nas posições ímpares.

4. A função de transição de um AFD foi definida como sendo uma função total. Suponha que tal função fosse parcial, e que a linguagem reconhecida por um AFD $M = (E, \Sigma, \delta, i, F)$ com $\delta$ parcial fosse o conjunto de todas as palavras $w$ tais que há uma computação

$$[i, w] \vdash \cdots \vdash [e, \lambda]$$

em que $e \in F$, ou uma computação da forma

$$[i, w] \vdash \cdots \vdash [e, ay]$$

onde $a \in \Sigma$, $y \in \Sigma^*$, $\delta(e, a)$ é indefinido e $e \in F$. Mostre como obter um AFD com função de transição total, equivalente a este outro tipo de AFD.

5. Prove que $\hat{\delta}(e, xy) = \hat{\delta}(\hat{\delta}(e, x), y)$, onde $\delta$ é a função de transição de um AFD, $e$ é um estado e $x$ e $y$ são palavras.

6. Implemente o algoritmo da Figura 2.5, página 67, em sua linguagem de programação favorita. O AFD pode ser codificado "à mão", ao invés de lido.

7. O algoritmo da Figura 2.5, página 67, mostra como implementar o reconhecimento de palavras via AFD's. Este algoritmo é geral, já que recebe o AFD como entrada. Um outro algoritmo, específico para um AFD, e um pouco mais eficiente, pode ser implementado via um comando de desvio múltiplo (*case* em Pascal, *switch* em C e Java, por exemplo), onde a variável de controle do desvio contém o *estado atual* durante o processamento de uma palavra. Para cada estado, $e$, tem-se um local no código onde se testa qual é o próximo símbolo de entrada, $a$, e se atribui $\delta(e, a)$ à variável de controle do desvio múltiplo. Utilizando esta idéia, implemente o AFD do Exemplo 57 (Figura 2.8, página 69).

8. Mostre que a relação "$\approx$" da Definição 6 é uma relação de equivalência.

9. Seja $\approx_n$ a relação definida na Definição 8. Prove que $[e]_n$ é a classe de equivalência de $e$ na partição induzida por $\approx_n$.

10. Minimize o AFD da Figura 2.13, página 77.

11. Minimize o AFD da Figura 2.14, página 78.

12. Utilizando a técnica do Teorema 4, determine AFD's que reconheçam

    (a) a união e

    (b) a interseção

    das linguagens dos exercícios 1(d) e 2(a).

13. Na prova do Teorema 4 assumiu-se que $M_1$ e $M_2$ tinham o mesmo alfabeto. Ajuste a prova do Teorema 4, e também do Lema 2, para o caso que os alfabetos possam ser distintos.

14. Faça um algoritmo que receba como entrada um conjunto de palavras, e construa um AFD que o reconheça.

15. Utilizando um raciocínio análogo ao utilizado no Exemplo 63 (baseado no Teorema 5), página 79, prove que não existe AFD que reconhece $\{x\mathtt{c}y \mid |x| = |y| \text{ e } x, y \in \{\mathtt{a}, \mathtt{b}\}^*\}$.

16. Sejam as linguagens:

    (a) $\{\mathtt{0}^n\mathtt{1}^n\mathtt{0}^n \mid n \in \mathbf{N}\}$;

    (b) $\{\mathtt{0}^n\mathtt{0}^n\mathtt{0}^n \mid n \in \mathbf{N}\}$.

    Mostre que a primeira não pode ser reconhecida por AFD, e que a segunda pode.

17. Mostre que existem procedimentos de decisão para verificar, para qualquer AFD $M$ e qualquer número natural $n$:

    (a) Se $L(M)$ contém alguma palavra de tamanho $n$.

    (b) Se $L(M)$ contém alguma palavra de tamanho maior que $n$.

    (c) Se $L(M)$ contém alguma palavra que tem $n$ ocorrências do símbolo $a$.

18. Sim ou não? Por que?

    (a) Existe linguagem infinita que pode ser reconhecida por um AFD de apenas um estado.

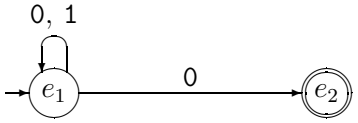
    (b) Existe linguagem finita que pode ser reconhecida por um AFD de, no mínimo, um trilhão de estados.

    (c) Se uma linguagem pode ser reconhecida por um AFD, qualquer subconjunto dela também pode.

    (d) Se uma linguagem não pode ser reconhecida por um AFD e ela é subconjunto de $L$, então $L$ também não pode ser reconhecida por um AFD.

## 2.3 Autômatos Finitos Não Determinísticos

Como ficou dito na Seção 2.2.1, o fato de que para cada par (estado, símbolo) há transição para um único estado confere um caráter determinístico às computações do autômato. Se esta restrição for eliminada, ou seja, se para algum par (estado, símbolo) houver transições para dois ou mais estados, tem-se o que se denomina um autômato finito não determinístico (AFN). Com isto, em um AFN podem existir várias computações possíveis para a mesma palavra. Seja o AFN cujo diagrama de estados é dado abaixo:



O não determinismo se verifica por causa da "indecisão" associada ao estado $e_1$: neste estado existem duas transições possíveis sob o símbolo 0, uma que leva ao próprio $e_1$ e outra que leva a $e_2$. Assim, por exemplo, para a palavra 1010, existiriam as seguintes computações possíveis, dependendo da alternativa tomada neste ponto de indecisão:

$$[e_1, 1010] \vdash [e_1, 010] \vdash \begin{cases} [e_1, 10] \vdash [e_1, 0] \vdash \begin{cases} [e_1, \lambda] \\ [e_2, \lambda] \end{cases} \\ [e_2, 10] \end{cases}$$

Assim, surge a pergunta: em que situações deve ser considerado que um AFN reconhece uma palavra? Para a palavra 1010 do exemplo, existem 3 computações possíveis, sendo que para uma delas a palavra é consumida e a computação termina em estado final. Não é difícil perceber para o exemplo acima que:

(a) se a palavra termina em 1, não existe computação que a consome e termina em estado final; e

(b) se a palavra termina em 0, existe computação que a consome e termina em estado final, embora existam outras computações que não a consomem ou não terminam em estado final.

O critério de reconhecimento para AFN's é justamente

> "*uma palavra é reconhecida se, e somente se, existe uma computação que a consome e termina em estado final*".

Assim, o AFN acima reconhece o conjunto das palavras de $\{0, 1\}^*$ que terminam em 0.

Uma outra maneira, bastante comum, de explicar o processamento de uma máquina não determinística é dizer que, em todo ponto de indecisão, a máquina *adivinha* qual escolha (se houver alguma) leva a uma computação que resulta em sucesso no reconhecimento. Caso nenhuma escolha leve a uma computação que resulte em sucesso no reconhecimento, a máquina escolhe qualquer uma das transições alternativas, já que todas levam à rejeição. Ou seja, a máquina tem tanta "sorte" nas escolhas relativas aos pontos de indecisão que, quando existe uma computação que resulte em aceitação da palavra, suas escolhas levam justamente uma computação com aceitação.

Na próxima seção será apresentada a definição de autômato finito não determinístico, e serão apresentados alguns exemplos ilustrando seu uso. Em seguida, será mostrado como obter um autômato finito determinístico equivalente a um autômato finito não determinístico.

### 2.3.1 O que é autômato finito não determinístico

Os componentes de um AFN são basicamente aqueles de um AFD, exceto que (1) um AFN pode ter mais de um estado inicial, e que (2) a função de transição dá, para cada par (estado, símbolo), um conjunto de estados. Segue a definição de AFN.

**Definição 9** *Um autômato finito não determinístico é uma quíntupla* $(E, \Sigma, \delta, I, F)$*, onde*

- *$E$ é um conjunto finito de um ou mais elementos denominados estados;*
- *$\Sigma$ é um alfabeto;*
- *$I$, subconjunto de $E$, é um conjunto* não vazio *de estados iniciais;*
- *$F$, subconjunto de $E$, é o conjunto de estados finais;*
- *$\delta$, a função de transição, é uma função total de $E \times \Sigma$ para $\mathcal{P}(E)$.* □

Ao contrário da maioria dos textos em que se aborda autômatos finitos não determinísticos, considera-se aqui que um AFN pode ter mais de um estado inicial. Isto não aumenta o poder expressivo, mas é conveniente em alguns contextos.

Observe que $\delta(e, a)$, no presente caso, é um conjunto que especifica os estados para os quais há transições de $e$ sob $a$. Em particular, se este conjunto é vazio, significa que não há transição de $e$ sob $a$ para qualquer estado.

Diagramas de estado para AFN são construídos da forma óbvia. Em particular, se $\delta(e, a) = \{e_1, e_2, \ldots, e_n\}$, haverá uma aresta do vértice $e$ para cada um dos vértices $e_1$, $e_2$, $\ldots$, $e_n$, todas com rótulo $a$. No caso em que $\delta(e, a) = \emptyset$, simplesmente não haverá aresta de $e$ para qualquer vértice com rótulo $a$. Assim, não há necessidade de se ter o conceito de diagrama de estados simplificado para AFN's. Colocar algum "estado de erro" em um AFN e/ou em seu diagrama de estados é desnecessário do ponto de vista teórico.

O diagrama de estados visto no início desta seção seria então um diagrama de estados para o AFN $(\{e_1, e_2\}, \{\mathtt{0}, \mathtt{1}\}, \delta, \{e_1\}, \{e_2\})$, onde $\delta$ é dada por:

| $\delta$ | 0 | 1 |
|---|---|---|
| $e_1$ | $\{e_1, e_2\}$ | $\{e_1\}$ |
| $e_2$ | $\emptyset$ | $\emptyset$ |

Observe que um AFD é um caso particular de AFN em que para todo par $(e, s)$, $|\delta(e, s)| \leq 1$, considerando-se que quando $\delta(e, s) = \emptyset$, há transição para um "estado de erro". Logo, se cada entrada da representação tabular for $\emptyset$ ou conjunto unitário, o AFN pode ser considerado, na realidade, um AFD.

Assim como para AFD's, antes de definir precisamente linguagem reconhecida por AFN, define-se a seguir uma extensão da função $\delta$ que, no caso, dado um conjunto de estados $A$ e uma palavra $w$, dá o conjunto de estados alcançáveis a partir dos estados de $A$, consumindo-se $w$.

**Definição 10** *Seja um AFN $M = (E, \Sigma, \delta, I, F)$. A* função de transição estendida para *$M$, $\hat{\delta}$, é uma função de $\mathcal{P}(E) \times \Sigma^*$ para $\mathcal{P}(E)$, definida recursivamente como segue:*

*(a) $\hat{\delta}(\emptyset, w) = \emptyset$, para todo $w \in \Sigma^*$;*

*(b) $\hat{\delta}(A, \lambda) = A$, para todo $A \subseteq E$;*

*(c) $\hat{\delta}(A, ay) = \hat{\delta}(\bigcup_{e \in A} \delta(e, a), y)$, para $A \subseteq E$, $a \in \Sigma$ e $y \in \Sigma^*$.* □

Em particular, observe que para um símbolo $a$ e $A \neq \emptyset$:

$$\begin{array}{rcll} \hat{\delta}(A, a) & = & \hat{\delta}(\bigcup_{e \in A} \delta(e, a), \lambda) & \text{por } c\text{, Definição 10} \\ & = & \bigcup_{e \in A} \delta(e, a) & \text{por } b\text{, Definição 10.} \end{array}$$

**Exemplo 64** Considere novamente o AFN visto acima, cuja função de transição é reproduzida abaixo:

| $\delta$ | $\mathtt{0}$ | $\mathtt{1}$ |
|---|---|---|
| $e_1$ | $\{e_1, e_2\}$ | $\{e_1\}$ |
| $e_2$ | $\emptyset$ | $\emptyset$ |

Os estados referentes às duas computações que consomem inteiramente a palavra $\mathtt{1010}$, como exemplificado no início da seção, são determinados assim, via $\hat{\delta}$:

$$\begin{array}{rcll}
\hat{\delta}(\{e_1\}, \mathtt{1010}) & = & \hat{\delta}(\delta(e_1, 1), \mathtt{010}) & \text{por } c\text{, Definição 10} \\
& = & \hat{\delta}(\{e_1\}, \mathtt{010}) & \text{por } \delta \\
& = & \hat{\delta}(\delta(e_1, 0), \mathtt{10}) & \text{por } c\text{, Definição 10} \\
& = & \hat{\delta}(\{e_1, e_2\}, \mathtt{10}) & \text{por } \delta \\
& = & \hat{\delta}(\delta(e_1, \mathtt{1}) \cup \delta(e_2, \mathtt{1}), \mathtt{0}) & \text{por } c\text{, Definição 10} \\
& = & \hat{\delta}(\{e_1\} \cup \emptyset, \mathtt{0}) & \text{por } \delta \\
& = & \hat{\delta}(\{e_1\}, \mathtt{0}) & \\
& = & \hat{\delta}(\delta(e_1, 0), \lambda) & \text{por } c\text{, Definição 10} \\
& = & \hat{\delta}(\{e_1, e_2\}, \lambda) & \text{por } \delta \\
& = & \{e_1, e_2\} & \text{por } b\text{, Definição 10.}
\end{array}$$

□

A linguagem reconhecida por um AFN pode, então, ser definida com o auxílio da função $\hat{\delta}$.

**Definição 11** *A* linguagem reconhecida (aceita) *por um AFN $M = (E, \Sigma, \delta, I, F)$ é o conjunto $L(M) = \{w \in \Sigma^* \mid \hat{\delta}(I, w) \cap F \neq \emptyset\}$. Uma determinada palavra $w \in \Sigma^*$ é dita ser* reconhecida (aceita) *por $M$ se, e somente se, $\hat{\delta}(I, w) \cap F \neq \emptyset$.* □

Além do fato de que, em geral, um AFN não pode ser implementado tão diretamente quanto um AFD, para todo AFN existe um AFD equivalente, como é mostrado na próxima seção. Então, pergunta-se: por que o conceito de AFN tem alguma importância? Em primeiro lugar, algumas vezes é mais fácil construir um AFN do que um AFD. Em segundo lugar, além de mais fácil, o AFN pode ser mais claro do que um AFD equivalente, dando mais confiança quanto a sua correção. Assim, justifica-se, pelo menos em alguns casos, construir o AFN e usar um algoritmo para obter um AFD equivalente. Em terceiro lugar, o conceito de não determinismo é importante em outras áreas da Ciência da Computação. Justifica-se, assim, introduzir este conceito em um contexto mais simples, como o de autômatos finitos.

**Exemplo 65** Na Figura 2.16 estão os diagramas de estados de um AFN e de um AFD que aceitam a linguagem $\{\mathtt{0}, \mathtt{1}\}^*\{\mathtt{1010}\}$. Observe como o AFN é bem mais fácil de construir e de entender. □

No Exemplo 65, o AFN e o AFD equivalente têm, coincidentemente, o mesmo número de estados. Isto nem sempre acontece, como demonstra o exemplo a seguir.

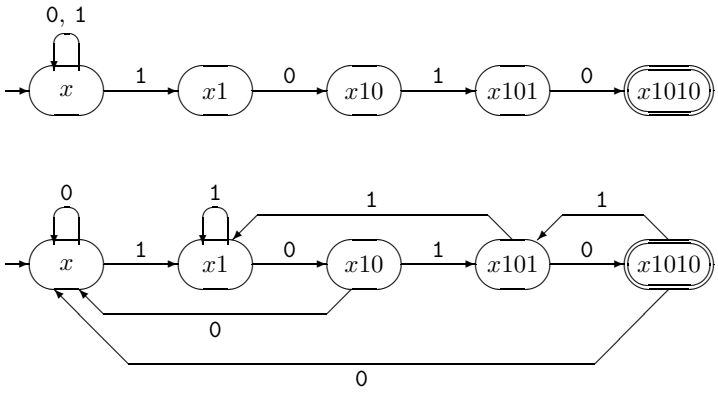



Figura 2.16: AFN e AFD para $\{0,1\}^*\{1010\}$.

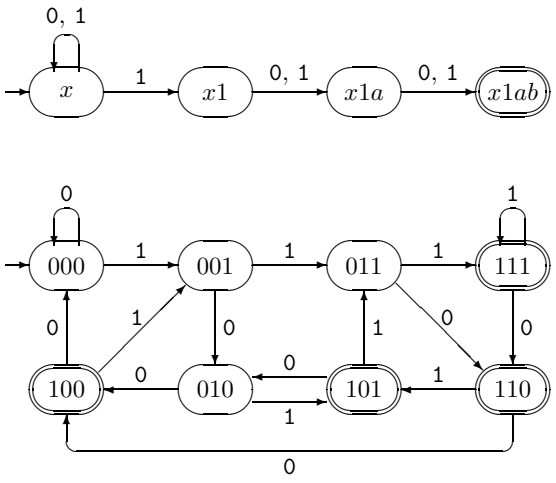



Figura 2.17: AFN e AFD para $\{0,1\}^*\{1\}\{0,1\}\{0,1\}$.

**Exemplo 66** Na Figura 2.17 estão mostrados os diagramas de estado de um AFN e de um AFD para a linguagem $\{w \in \{0,1\}^* \mid |w| \geq 3$ e o terceiro símbolo da direita para a esquerda é $1\}$. Novamente, o AFN é bem mais fácil de construir e de entender. Além disto, o AFD tem mais estados. É fácil perceber que, usando a mesma técnica deste exemplo, o AFN para $\{w \in \{0,1\}^* \mid |w| \geq n$ e o $n$-ésimo símbolo da direita para a esquerda é $1\}$ tem $n+1$ estados, e o AFD tem $2^n$ estados. □

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

Como ficou dito na seção anterior, um AFD é um caso particular de AFN. Assim, para mostrar a equivalência dos dois formalismos basta mostrar que para qualquer AFN pode-se construir um AFD que reconhece a mesma linguagem.

A idéia a ser utilizada para isso é a mesma que foi utilizada no Teorema 4, na página 76, para construir um AFD para $L(M_1) \cup L(M_2)$ a partir dos AFD's $M_1$ e $M_2$: lá tratou-se de simular as computações de $M_1$ e $M_2$ "em paralelo"; aqui, basicamente, vai-se tratar de simular todas as computações possíveis do AFN "em paralelo". Enquanto lá um estado era um par, significando os estados de $M_1$ e $M_2$ atingidos após processar a mesma palavra, aqui um estado será um conjunto, significando todos os estados do AFN atingidos por todas as computações possíveis para a mesma palavra. O AFD equivalente a um AFN $M = (E, \Sigma, \delta, I, F)$, usando esta idéia, seria: $M' = (\mathcal{P}(E), \Sigma, \delta', I, F')$, onde:

- para cada $X \subseteq E$ e $a \in \Sigma$, $\delta'(X,a) = \bigcup_{e \in X} \delta(e,a)$; para cada $a \in \Sigma$, $\delta'(\emptyset, a) = \emptyset$; e
- $F' = \{X \subseteq E \mid X \cap F \neq \emptyset\}$.

O seguinte lema será utilizado em seguida para provar a equivalência de ambos os autômatos.

**Lema 3** *Sejam $M$ e $M'$ como acima. Então para todo $X \subseteq E$ e $w \in \Sigma^*$, $\hat{\delta}'(X,w) = \hat{\delta}(X,w)$.*

*Prova*

A prova será feita por indução sobre $|w|$. Para $w = \lambda$, tem-se:

$$\begin{aligned} \hat{\delta}'(X,\lambda) &= X && \text{pela Definição 2} \\ &= \hat{\delta}(X,\lambda) && \text{pela Definição 10.} \end{aligned}$$

Suponha, como hipótese de indução, que $\hat{\delta}'(X,w) = \hat{\delta}(X,w)$ para certo $w \in \Sigma^*$. Basta, então, provar que $\hat{\delta}'(X,aw) = \hat{\delta}(X,aw)$ para todo $a \in \Sigma$. Devem ser considerados dois casos:

Caso 1: $X = \emptyset$.

$$\begin{aligned} \hat{\delta}'(\emptyset,aw) &= \hat{\delta}'(\delta'(\emptyset,a),w) && \text{pela Definição 2} \\ &= \hat{\delta}'(\emptyset,w) && \text{pela definição de } \delta' \\ &= \hat{\delta}(\emptyset,w) && \text{pela hipótese de indução} \\ &= \emptyset && \text{pela Definição 10} \\ &= \hat{\delta}(\emptyset,aw) && \text{pela Definição 10.} \end{aligned}$$

Caso 2: $X \neq \emptyset$.

$$\begin{array}{lll}
\hat{\delta'}(X, aw) = \hat{\delta'}(\delta'(X, a), w) & & \text{pela Definição 2} \\
= \hat{\delta'}(\bigcup_{e \in X} \delta(e, a), w) & & \text{pela definição de } \delta' \\
= \hat{\delta}(\bigcup_{e \in X} \delta(e, a), w) & & \text{pela hipótese de indução} \\
= \hat{\delta}(X, aw) & & \text{pela Definição 10.}
\end{array}$$

□

Segue o teorema.

**Teorema 7** *Para qualquer AFN existe AFD equivalente.*

*Prova*

Seja um AFN $M = (E, \Sigma, \delta, I, F)$ e um AFD $M' = (\mathcal{P}(E), \Sigma, \delta', I, F')$, onde $\delta'$e $F'$ são definidos como acima. Para provar que $L(M') = L(M)$, basta mostrar que $w \in L(M) \leftrightarrow w \in L(M')$ para todo $w \in \Sigma^*$. Para este fim, seja um $w \in \Sigma^*$ arbitrário. Tem-se:

$$\begin{array}{lll}
w \in L(M') \leftrightarrow \hat{\delta'}(I, w) \in F' & & \text{pela Definição 3} \\
\leftrightarrow \hat{\delta'}(I, w) \cap F \neq \emptyset & & \text{pela definição de } F' \\
\leftrightarrow \hat{\delta}(I, w) \cap F \neq \emptyset & & \text{pelo Lema 3} \\
\leftrightarrow w \in L(M) & & \text{pela Definição 11.}
\end{array}$$

□

Na técnica mostrada acima para construir o AFD $M'$, pode acontecer de certos estados de $M'$ serem "inúteis", no sentido de não serem alcançáveis a partir dos estados em $I$. Ali não houve a preocupação de eliminar tais estados, com o objetivo de manter o texto mais conciso e centrado nos conceitos realmente importantes.

Não é difícil de ver que, em geral, na obtenção manual de um AFD equivalente a um AFN dado, aplicando-se a técnica do Teorema 7, é mais fácil trabalhar com ambas as funções de transição no formato tabular do que com os diagramas de estado. Suponha que se tenha $\delta(e_i, a) = A_i$ para $i = 1, 2, \ldots, n$; em formato tabular:

| $\delta$ | $\cdots$ | $a$ | $\cdots$ |
|---|---|---|---|
| | | $\vdots$ | |
| $e_1$ | $\cdots$ | $A_1$ | $\cdots$ |
| | | $\vdots$ | |
| $e_2$ | $\cdots$ | $A_2$ | $\cdots$ |
| | | $\vdots$ | |
| $e_n$ | $\cdots$ | $A_n$ | $\cdots$ |
| | | $\vdots$ | |

O valor de $\delta'(\{e_1, e_2, \ldots, e_n\}, a)$ é obtido simplesmente fazendo-se a união dos $A_i$'s.

**Exemplo 67** Na Figura 2.18 está mostrado o diagrama de estados de um AFN que reconhece o conjunto das palavras de $\{0, 1\}^*$ com um prefixo de um ou dois 0's e um sufixo com um número par de dois ou mais 1's, e o diagrama de estados para um AFD equivalente, construído utilizando a técnica acima apresentada. Só estão mostrados, nesta última figura, os estados alcançáveis a partir do estado inicial $\{1, 2\}$. □

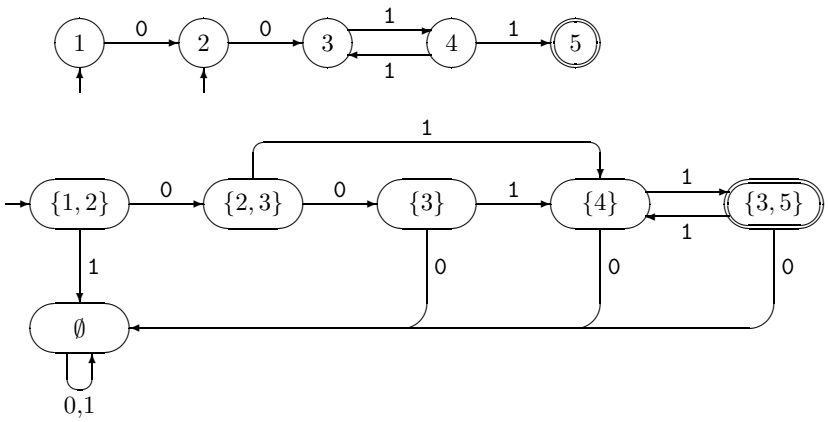



Figura 2.18: AFN e AFD equivalentes.

### 2.3.3 AFN estendido

Um outro conceito que pode ser útil, tanto do ponto de vista teórico quanto prático, é o de autômato finito não determinístico *estendido* (AFNE), embora este não aumente o poder computacional com relação a AFN ou a AFD. A diferença entre autômato finito não determinístico estendido e AFN é que, enquanto neste último as transições são sob símbolos do alfabeto, no primeiro elas são sob palavras.

**Definição 12** *Um* autômato finito não determinístico estendido *é uma quíntupla* $(E, \Sigma, \delta, I, F)$*, onde:*

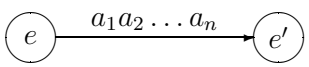
- $E$*,* $\Sigma$*,* $I$ *e* $F$ *são como em AFN's; e*
- $\delta$ *é uma função parcial de* $E \times D$ *para* $\mathcal{P}(E)$*, onde* $D$ *é algum subconjunto finito de* $\Sigma^*$*.* □

Na figura 2.19 mostra-se o diagrama de estados para um AFNE que reconhece a linguagem

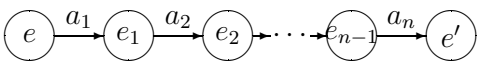
$$\{w \in \{0\}^* \mid |w| \text{ é par}\} \cup \{w \in \{1\}^* \mid |w| \text{ é ímpar}\}.$$

Veja como ele é bem conciso e fácil de entender.

É fácil notar que se $a_1 a_2 \ldots a_n$ é uma palavra de tamanho maior do que 1, ou seja, $n \geq 2$, então uma transição da forma




pode ser substituída por $n$ transições:



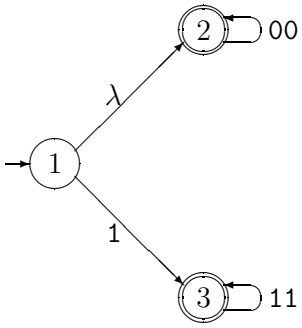



Figura 2.19: Um exemplo de AFNE

onde $e_1$, $e_2$, ..., $e_{n-1}$ são *novos* estados; e a linguagem reconhecida pelo autômato continua a mesma. Assim sendo, uma transição da forma acima pode ser considerada simplesmente como uma abreviação das $n$ equivalentes. Este tipo de transição é utilizado apenas por comodidade.

Por outro lado, transições sob $\lambda$ são mais importantes do que sob palavras de tamanho maior que 1. Embora elas também possam ser eliminadas sem alterar a linguagem reconhecida pelo autômato, elas apresentam maior aplicabilidade, tanto do ponto de vista prático quanto teórico. Assim, justifica-se definir a classe dos autômatos finitos não determinísticos com transições $\lambda$.

**Definição 13** *Um* autômato finito não determinístico com transições $\lambda$ *(AFN$\lambda$) é uma quíntupla* $(E, \Sigma, \delta, I, F)$*, onde:*

- $E$, $\Sigma$, $I$ *e* $F$ *são como em AFN's; e*
- $\delta$ *é uma função total de* $E \times (\Sigma \cup \{\lambda\})$ *para* $\mathcal{P}(E)$*.* □

O conceito de reconhecimento de palavras e de linguagem continua sendo dado pela Definição 11, mas com a função $\hat{\delta}$ alterada para acomodar a nova função $\delta$. Para este efeito, é útil definir uma função denominada *fecho* $\lambda$ que, aplicada a um conjuntos de estados $X$, dá todos os estados alcançáveis a partir dos estados de $X$ utilizando-se apenas transições sob $\lambda$. Observe que uma transição sob $\lambda$ ocorre sem consumo de símbolo algum; assim, a função *fecho* $\lambda$ aplicada a $X$, dá todos os estados alcançáveis a partir dos estados de $X$ sem consumo de símbolos.

**Definição 14** *Seja um autômato finito não determinístico com transições* $\lambda$ $M = (E, \Sigma, \delta, I, F)$*. A função* fecho $\lambda$ *para* $M$*,* $f\lambda$*, é uma função de* $\mathcal{P}(E)$ *para* $\mathcal{P}(E)$*, definida recursivamente como segue:*

*(a)* $X \subseteq f\lambda(X)$*;*

*(b) se* $e \in f\lambda(X)$ *então* $\delta(e, \lambda) \subseteq f\lambda(X)$*.* □

Segue a definição de $\hat{\delta}$ para AFN$\lambda$'s.

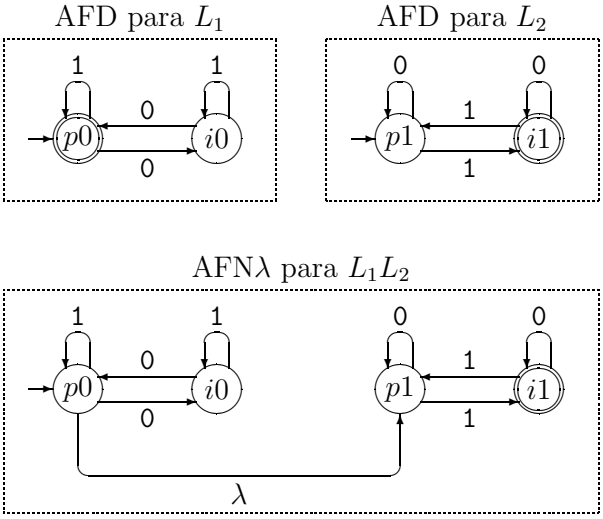



Figura 2.20: AFN$\lambda$ para $L_1L_2$

**Definição 15** *Seja um AFN$\lambda$ $M = (E, \Sigma, \delta, I, F)$. A função de transição estendida, $\hat{\delta}$, é uma função de $\mathcal{P}(E) \times \Sigma^*$ para $\mathcal{P}(E)$, definida recursivamente como segue:*

*(a)* $\hat{\delta}(\emptyset, w) = \emptyset$, *para todo* $w \in \Sigma^*$;

*(b)* $\hat{\delta}(A, \lambda) = f\lambda(A)$, *para* $A \subseteq E$;

*(c)* $\hat{\delta}(A, ay) = \hat{\delta}(\bigcup_{e \in f\lambda(A)} \delta(e, a), y)$, *para* $A \subseteq E$, $a \in \Sigma$ *e* $y \in \Sigma^*$. □

Segue um exemplo de AFN$\lambda$ .

**Exemplo 68** Dados dois AFN's $M_1$ e $M_2$, pode-se construir um AFN$\lambda$ que reconhece $L(M_1)L(M_2)$ simplesmente colocando-se uma transição sob $\lambda$ de cada estado final de $M_1$ para cada estado inicial de $M_2$; os estados iniciais do AFN$\lambda$ seriam os estados iniciais de $M_1$, e os estados finais seriam os estados finais de $M_2$. Seja, por exemplo, o problema de construir um autômato para reconhecer $L = L_1L_2$, onde $L_1 = \{w \in \{0, 1\}^* \mid$ $w$ tem um número par de 0's$\}$ e $L_2 = \{w \in \{0, 1\}^* \mid$ $w$ tem um número ímpar de 1's$\}$. Construir diretamente um AFD (ou mesmo um AFN comum) para $L$ não é trivial. No entanto, é extremamente fácil obter AFD's para $L_1$ e $L_2$ e, em seguida, um AFN$\lambda$ para $L$ a partir destes, como mostra o diagrama de estados da Figura 2.20. □

Para obter um AFN equivalente a um AFN$\lambda$, basta "eliminar" as transições $\lambda$, o que pode ser feito facilmente com o auxílio da função $f\lambda$, como será visto em seguida.

**Teorema 8** *Para qualquer AFN$\lambda$ existe um AFN equivalente.*

*Prova*

Seja um AFN$\lambda$ $M = (E, \Sigma, \delta, I, F)$. Um AFN equivalente a $M$ seria $M' = (E, \Sigma, \delta', I', F)$, onde:

- $I' = f\lambda(I)$; e

- $\delta'(e,a) = f\lambda(\delta(e,a))$, para cada $e \in E$ e $a \in \Sigma$.

Para provar que $L(M') = L(M)$, é suficiente mostrar que

$$\hat{\delta}'(f\lambda(X), w) = \hat{\delta}(X, w) \text{ para todo } X \subseteq E \text{ e } w \in \Sigma^* \tag{2.3}$$

pois, em particular, ter-se-á que:

$$\begin{aligned} \hat{\delta}'(I', w) &= \hat{\delta}'(f\lambda(I), w) && \text{pela definição de } I' \\ &= \hat{\delta}(I, w) && \text{por 2.3.} \end{aligned}$$

Será mostrado, então, que 2.3 é verdadeira, por indução sobre $|w|$. Para $w = \lambda$, tem-se:

$$\begin{aligned} \hat{\delta}'(f\lambda(X), \lambda) &= f\lambda(X) && \text{pela Definição 10} \\ &= \hat{\delta}(X, \lambda) && \text{pela Definição 15.} \end{aligned}$$

Suponha que $\hat{\delta}'(f\lambda(X), y) = \hat{\delta}(X, y)$ para $y \in \Sigma^*$. Basta provar, então, que $\hat{\delta}'(f\lambda(X), ay) = \hat{\delta}(X, ay)$ para $a \in \Sigma$. De fato:

$$\begin{aligned} \hat{\delta}'(f\lambda(X), ay) &= \hat{\delta}'(\textstyle\bigcup_{e \in f\lambda(X)} \delta'(e,a), y) && \text{pela Definição 10} \\ &= \hat{\delta}'(\textstyle\bigcup_{e \in f\lambda(X)} f\lambda(\delta(e,a)), y) && \text{pela Definição de } \delta' \\ &= \hat{\delta}'(f\lambda(\textstyle\bigcup_{e \in f\lambda(X)} \delta(e,a)), y) && \text{pelo exercício 13} \\ &= \hat{\delta}(\textstyle\bigcup_{e \in f\lambda(X)} \delta(e,a), y) && \text{pela hipótese de indução} \\ &= \hat{\delta}(X, ay) && \text{pela Definição 15.} \end{aligned}$$

□

Seguem exemplos de obtenção de AFN's a partir de AFN$\lambda$'s utilizando-se o método apresentado na prova do Teorema 8.

**Exemplo 69** Na Figura 2.21 estão mostrados (a) o AFN$\lambda$ equivalente ao AFNE da Figura 2.19 (página 91), após a eliminação das transições sob `00` e `11`, e (b) o AFN obtido após a eliminação da transição $\lambda$. Note que o AFN tem como estados iniciais $f\lambda(\{\mathtt{1}\}) = \{\mathtt{1}, \mathtt{2}\}$, e que $\delta'(e,a) = \delta(e,a)$ para $a \in \Sigma$, pois, para este exemplo, $f\lambda(\delta(e,a)) = \delta(e,a)$ para todo $(e,a) \in E \times \Sigma$. □

Assim, por exemplo, para se obter um AFD equivalente a um autômato finito não determinístico estendido, basta obter um AFN equivalente, como acima, e depois obter um AFD equivalente ao AFN.

**Exemplo 70** Na Figura 2.22 está mostrado o diagrama de estados de um AFN equivalente ao AFN$\lambda$ do Exemplo 68 (Figura 2.20, página 92). Existem apenas duas transições para as quais $f\lambda(\delta(e,a)) \neq \delta(e,a)$: a de $i0$ para $p0$ sob 0, e a de $p0$ para $p0$ sob 1. Elas originam, então as seguintes transições no AFN resultante (conforme a prova do Teorema 8):

$$\begin{aligned} \delta'(i0, \mathtt{0}) &= f\lambda(\delta(i0, \mathtt{0})) \\ &= f\lambda(\{p0\}) \\ &= \{p0, p1\}. \end{aligned}$$

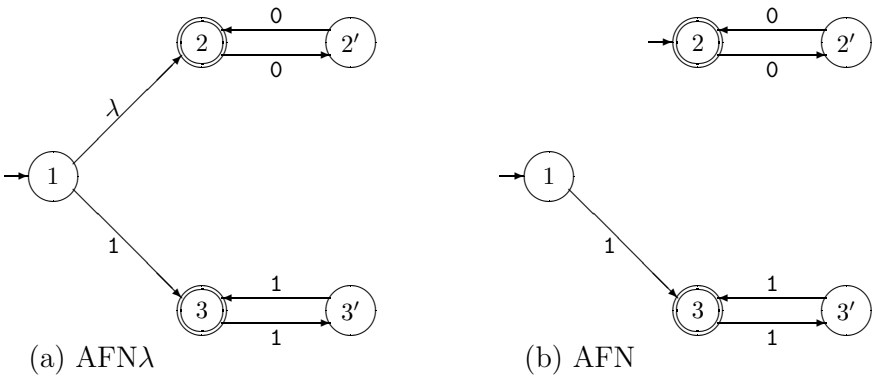



Figura 2.21: AFN equivalente a AFNE da Figura 2.19

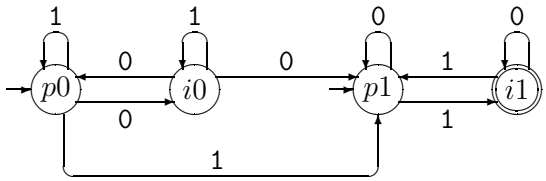



Figura 2.22: AFN para AFNλ do Exemplo 68.

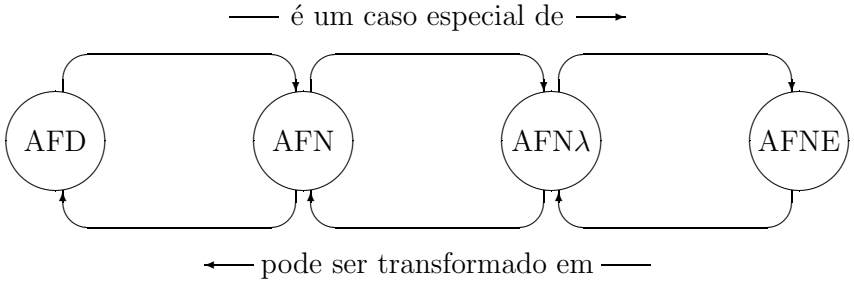



Figura 2.23: Relações entre autômatos finitos

$$\begin{aligned}\delta'(p0, \mathtt{1}) &= f\lambda(\delta(p0, \mathtt{1})) \\ &= f\lambda(\{p0\}) \\ &= \{p0, p1\}.\end{aligned}$$

□

A Figura 2.23 sumariza as relações entre os diversos tipos de autômatos já vistos. Daqui para frente será utilizada a denominação "autômato finito" (AF) para englobar AFD's, AFN's, AFNE's e AFN$\lambda$'s.

**Definição 16** *Uma linguagem é dita ser uma* linguagem regular *(ou* linguagem de estado-finito[4]*, ou ainda,* conjunto regular*) se existe um autômato finito que a reconhece.* □

Que características interessantes dos pontos de vista teórico e prático têm as linguagens regulares? Como identificar se uma linguagem é regular? Estes assuntos, já tocados de leve, são tratados na Seção 2.4.

# Exercícios

1. Construa AFN's para as seguintes linguagens sobre $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$:
   (a) O conjunto das palavras com, no mínimo, 3 ocorrências de `abc`.
   (b) O conjunto das palavras com, no mínimo, 3 ocorrências de `a`'s ou 3 ocorrências de `b`'s ou 3 ocorrências de `c`'s .
   (c) O conjunto das palavras com sufixo `abc` ou `bca`.
   (d) O conjunto das palavras em que existem duas ocorrências de `abc` com um número ímpar de símbolos entre elas.
   (e) O conjunto das palavras em que o último símbolo seja idêntico ao primeiro.
   (f) O conjunto das palavras em que o último símbolo seja diferente do primeiro.
   (g) O conjunto das palavras em que o último símbolo tenha ocorrido antes.
   (h) O conjunto das palavras em que o último símbolo tenha ocorrido antes no máximo uma vez.
   (i) O conjunto das palavras em que o último símbolo não tenha ocorrido antes.

2. Sejam as linguagens da forma $L_n = \{xyx \mid x, y \in \{\mathtt{a}, \mathtt{b}\}^* \text{ e } |x| = n\}$. Determine o menor número de estados para um AFN e para um AFD que reconheçam $L_n$, nos seguintes casos:
   (a) $n = 1$.
   (b) $n = 2$.
   (c) $n$ arbitrário.

[4] *Finite-state language.*

3. Seja um AFN $M = (E, \Sigma, \delta, I, F)$. Sejam $X_1, X_2, \ldots, X_n$ tais que $X_i \neq \emptyset$ para $1 \leq i \leq n$. Prove que, para qualquer $w \in \Sigma^*$, $\hat{\delta}(\bigcup_{i=1}^{n} X_i, w) = \bigcup_{i=1}^{n} \hat{\delta}(X_i, w)$.

4. Mostre que para todo AFN existe um AFN equivalente com um único estado inicial.

5. Seja o AFN $M = (\{1,2,3\}, \{a,b\}, \delta, \{1\}, \{1,2,3\})$, onde $\delta$ é dada por:

| $\delta$ | $a$ | $b$ |
|---|---|---|
| 1 | $\{2\}$ | $\{\}$ |
| 2 | $\{3\}$ | $\{\}$ |
| 3 | $\{\}$ | $\{3\}$ |

   Obtenha um AFN com *um único estado final* equivalente a $M$.

6. Mostre que sim ou não: para todo AFN existe um AFN equivalente com um único estado final.

7. Seja o AFN$\lambda$ $M = (\{0,1,2\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \delta, 0, \{2\})$, sendo $\delta$ dada por:

| $\delta$ | a | b | c | $\lambda$ |
|---|---|---|---|---|
| 0 | $\{0\}$ | $\emptyset$ | $\emptyset$ | $\{1\}$ |
| 1 | $\emptyset$ | $\{1\}$ | $\emptyset$ | $\{2\}$ |
| 2 | $\emptyset$ | $\emptyset$ | $\{2\}$ | $\emptyset$ |

   (a) Determine $f\lambda(e)$ para $e = 0, 1, 2$.

   (b) Determine um AFN $M'$ equivalente a $M$, usando a técnica do Teorema 8.

   (c) Determine um AFD equivalente a $M'$, usando a técnica do Teorema 7 .

8. Obtenha um AFD equivalente ao AFN cujo diagrama de estados está mostrado na Figura 2.21(b), página 94, usando a técnica do Teorema 7.

9. Obtenha um AFD equivalente ao AFN cujo diagrama de estados está mostrado na Figura 2.22, página 94, usando a técnica do Teorema 7.

10. Construa AFNE's para as linguagens do Exercício 1, com um mínimo de transições possível.

11. Defina uma função $\hat{\delta}$ para AFNE, de tal forma que $\hat{\delta}(X, w)$, onde $X$ é um conjunto de estados, seja o conjunto de todos os estados alcançáveis a partir dos estados de $X$ consumindo-se $w$.

12. Na prova do Teorema 4 mostrou-se como construir um AFD para reconhecer a união das linguagens reconhecidas por dois AFD's. Suponha que voce tenha dois AF's $M_1$ e $M_2$. Explique como construir um AF que reconheça $L(M_1) \cup L(M_2)$ usando, além das transições de $M_1$ e $M_2$, apenas algumas transições $\lambda$ adicionais.

13. Seja um AFN$\lambda$ qualquer $M = (E, \Sigma, \delta, I, F)$. Prove que para qualquer $X \subseteq E$ e $a \in \Sigma$, $\bigcup_{e \in X} f\lambda(\delta(e, a)) = f\lambda(\bigcup_{e \in X} \delta(e, a))$.

14. Mostre que para todo AFN$\lambda$ existe um AFN$\lambda$ equivalente com um único estado inicial e um único estado final, sendo que não existem transições entrando no estado inicial, nem transições saindo do estado final.

## 2.4 Linguagens Regulares: Propriedades

Considere a classe de todas as linguagens regulares. Haveria uma ou mais caracterizações adicionais dessa classe de linguagens, além do fato de que elas são reconhecíveis por AF's, de forma que, dada uma linguagem $L$:

- Seja possível determinar se ela pertence ou não à classe (antes de se tentar construir um AF para ela)?
- Seja facilitada a obtenção de um AF para $L$?

A resposta é sim para ambas as perguntas. Na realidade, existem várias caracterizações adicionais que podem servir de auxílio em uma ou ambas as situações. Nesta seção, serão apresentados dois tipos de resultados. O primeiro, o lema do bombeamento [5], explicita uma característica importante de toda linguagem regular, que é útil, por exemplo, para mostrar que uma linguagem não é regular. Em seguida, serão apresentadas algumas propriedades de fechamento[6] da classe das linguagens regulares. Tais propriedades são úteis para as duas aplicações referidas acima. Outras caracterizações virão na Seção 2.6.

### 2.4.1 O Lema do bombeamento

A visão de um AFD como um grafo facilita o raciocínio para obtenção de diversas propriedades da classe das linguagens reconhecidas por esse tipo de máquina. Uma das principais é dada pelo chamado lema do bombeamento. Este lema especifica uma propriedade que qualquer linguagem regular tem. Além destas, outras linguagens (não reconhecidas por AFD's) também têm esta mesma propriedade. Assim, se se mostrar que uma linguagem $L$ satisfaz o lema do bombeamento, isto não implica que $L$ é regular. Mas se se mostrar que $L$ *não* satisfaz o lema do bombeamento, pode-se concluir que $L$ *não* é linguagem regular.

Segue o lema do bombeamento (LB), uma extensão do Teorema 5.

**Lema 4** [7] *Seja $L$ uma linguagem regular. Então existe uma constante $k > 0$ tal que para qualquer palavra $z \in L$ com $|z| \geq k$ existem $u$, $v$ e $w$ que satisfazem as seguintes condições:*

- $z = uvw$;
- $|uv| \leq k$;
- $v \neq \lambda$; *e*
- $uv^iw \in L$ *para todo* $i \geq 0$.

[5] *Pumping lemma.*

[6] *Closure properties.*

[7] Para aqueles com pendor para formalidade, aqui vai um enunciado mais formal:
$L$ é uma linguagem regular $\rightarrow$
$\exists k \in \mathbf{N} \forall z \in L[|z| \geq k \rightarrow \exists u, v, w(z = uvw \wedge |uv| \leq k \wedge |v| \geq 1 \wedge \forall i \in \mathbf{N} uv^iw \in L)]$

*Prova*

Seja um AFD $M$ que reconheça $L$. Será mostrado que o número de estados, $n$, de $M$, pode ser tomado como a constante $k$ referida no enunciado (na verdade, qualquer constante maior ou igual a $n$ serve). Assim, seja uma palavra arbitrária $z \in L$ com $|z| \geq n$ (se não existir nenhum $z$ tal que $|z| \geq n$, o lema vale por vacuidade). Seja $c(z)$ o caminho percorrido no grafo de estados de $M$ para reconhecer $z$ (este caminho existe e é único, pela definição de AFD). Como $|z| \geq n$ e $|c(z)| = |z| + 1$, $c(z)$ repete algum estado. Seja $e$ tal estado (qualquer estado que se repita serve). É óbvio que $u$, $v$ e $w$ do enunciado podem ser tomados da seguinte forma: $u$ é a subpalavra consumida antes da primeira ocorrência de $e$; $v$ é a subpalavra consumida da primeira à segunda ocorrência de $e$; e $w$ é o resto. □

Quando a linguagem é finita, não existe $z \in L$ tal que $|z| \geq n$, onde $n$ é o número de estados do AFD. Por outro lado, se a linguagem é infinita, ela poderá ser reconhecível ou não por AFD's. O Lema 4 mostra que, se tal linguagem for regular, então o grafo de estados simplificado do AFD terá no mínimo um ciclo, e um ciclo será percorrido necessariamente no reconhecimento de uma palavra de tamanho maior ou igual a $n$; ciclos é que tornam possível reconhecer palavras de tamanho maior ou igual ao número de estados do AFD (que, obviamente, sempre existirão, se a linguagem é infinita).

**Exemplo 71** A linguagem $L = \{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbf{N}\}$ não é regular, como mostrado abaixo, por contradição, aplicando-se o lema do bombeamento.

Suponha que $L$ seja uma linguagem regular. Seja $k$ a constante referida no LB, e seja $z = \mathtt{a}^k\mathtt{b}^k$. Como $|z| > k$, o lema diz que que existem $u$, $v$ e $w$ de forma que as seguintes condições se verificam:

- $z = uvw$;
- $|uv| \leq k$;
- $v \neq \lambda$; e
- $uv^iw \in L$ para todo $i \geq 0$.

Neste caso, $v$ só tem a's, pois $z = uvw = \mathtt{a}^k\mathtt{b}^k$ e $|uv| \leq k$, e $v$ tem pelo menos um a, pois $v \neq \lambda$. Isto implica que $uv^2w = \mathtt{a}^{k+|v|}\mathtt{b}^k \notin L$, o que contraria o LB. Logo, a suposição original de que $L$ é linguagem regular não se justifica. Conclui-se que $L$ não é linguagem regular. □

Observe, pois, que o LB é usado para provar que uma linguagem infinita, $L$, não é regular da seguinte forma [8]:

1) Supõe-se que $L$ é linguagem regular.

[8] Mais formalmente, supondo que $L$ é regular, deveria valer, por *modus ponens*:
$\exists k \in \mathbf{N} \forall z \in L[|z| \geq k \rightarrow \exists u, v, w(z = uvw \wedge |uv| \leq k \wedge |v| \geq 1 \wedge \forall i \in \mathbf{N} uv^iw \in L)]$. Assim, para derivar uma contradição, basta provar a negação disto, que é equivalente a:
$\forall k \in \mathbf{N} \exists z \in L[|z| \geq k \wedge \forall u, v, w((z = uvw \wedge |uv| \leq k \wedge |v| \geq 1) \rightarrow \exists i \in \mathbf{N} uv^iw \notin L)]$.

2) Escolhe-se uma palavra $z$, cujo tamanho seja maior que $k$, a constante do LB;

3) Mostra-se que para toda decomposição de $z$ em $u$ $v$ e $w$ existe $i$ tal que $uv^iw \notin L$.

Assim, existem dois aspectos fundamentais: a escolha de $z$ (item 2) e a escolha de $i$ (item 3). A escolha de $z$ deve ser feita de tal forma que facilite mostrar, para algum $i$, que $uv^iw \notin L$.

**Exemplo 72** Seja $L = \{\mathtt{0}^m\mathtt{1}^n \mid m > n\}$. Suponha que $L$ seja uma linguagem regular. Seja $k$ a constante referida no LB, e seja $z = \mathtt{0}^{k+1}\mathtt{1}^k$. Como $|z| > k$, o lema diz que existem $u$, $v$ e $w$ de forma que as seguintes condições se verificam:

- $z = uvw$;
- $|uv| \leq k$;
- $v \neq \lambda$; e
- $uv^iw \in L$ para todo $i \geq 0$.

Como $0 < |v| \leq k$ e $v \neq \lambda$, $v$ só tem 0's e $v$ tem no mínimo um 0. Logo, $uv^0w = \mathtt{0}^{k+1-|v|}\mathtt{1}^k \notin L$, contrariando o LB. Conclui-se que $L$ não é linguagem regular. Observe que, para a palavra $z$ escolhida, o único valor para $i$ que contraria o LB é 0. □

O exemplo a seguir *determina* um $i$ tal que $uv^iw \notin L$ de forma sistemática, a partir das informações disponíveis.

**Exemplo 73** A linguagem $L = \{\mathtt{0}^n \mid n \text{ é primo}\}$ não é regular, como mostrado abaixo, por contradição, aplicando-se o lema do bombeamento (LB).

Suponha que $L$ seja uma linguagem regular. Seja $k$ a constante referida no LB, e seja $z = \mathtt{0}^n$, onde $n$ é um número primo maior que $k$. Como $|z| > k$, o lema diz que existem $u$, $v$ e $w$ de forma que as seguintes condições se verificam:

- $z = uvw$;
- $|uv| \leq k$;
- $v \neq \lambda$; e
- $uv^iw \in L$ para todo $i \geq 0$.

Para provar que $L$ não é linguagem regular, basta então mostrar um $i$ tal que $uv^iw \notin L$ (contrariando o LB). Pelas informações acima, tem-se que $uv^iw = \mathtt{0}^{n+(i-1)|v|}$ (pois $z = \mathtt{0}^n$). Assim, $i$ deve ser tal que $n + (i-1)|v|$ não seja um número primo. Ora, para isto, basta fazer $i = n + 1$, obtendo-se $n + (i-1)|v| = n + n|v| = n(1 + |v|)$, que não é primo (pois $|v| > 0$). Assim, $uv^{n+1}w \notin L$, contradizendo o LB. Logo, $L$ não é linguagem regular. □

É interessante notar que o Teorema 6, que diz que existem procedimentos de decisão para verificar, para qualquer AFD $M$, se (a) $L(M) = \emptyset$ e se (b) $L(M)$ é finita, pode ser demonstrado utilizando-se o LB. Veja o Exercício 24 da Seção 2.9, página 132.

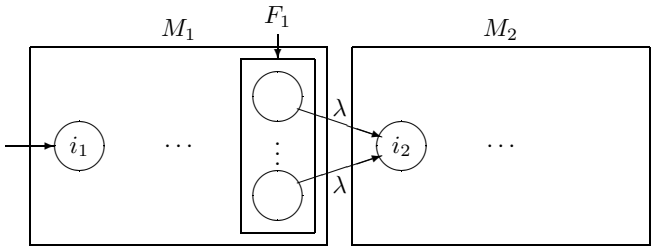



Figura 2.24: Fechamento sob concatenação.

### 2.4.2 Propriedades de fechamento

Seja uma classe de linguagens, $\mathcal{L}$, e uma operação sobre linguagens, $O$. Por exemplo, $\mathcal{L}$ poderia ser o conjunto de todas as linguagens regulares, e $O$ poderia ser a operação de união. Diz-se que $\mathcal{L}$ é *fechada* sob $O$ se a aplicação de $O$ a linguagens de $\mathcal{L}$ resulta sempre em uma linguagem de $\mathcal{L}$.

**Teorema 9** *A classe das linguagens regulares é fechada sob:*

*(1) Complementação.*

*(2) União.*

*(3) Interseção.*

*(4) Concatenação.*

*(5) Fecho de Kleene.*

*Prova*

Os fechamentos sob complementação, união e interseção são corolários do Teorema 4.

*Prova de 4*. Sejam duas linguagens regulares quaisquer $L_1$ e $L_2$. Sejam dois AFD's $M_1 = (E_1, \Sigma_1, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma_2, \delta_2, i_2, F_2)$, tais que $L(M_1) = L_1$ e $L(M_2) = L_2$, e tais que $E_1 \cap E_2 = \emptyset$ (é fácil ver que existem). O seguinte AFN$\lambda$ $M_3$, definido abaixo, reconhece $L(M_1)L(M_2)$ (veja Figura 2.24 para uma representação esquemática de $M_3$):

$$M_3 = (E_1 \cup E_2, \Sigma_1 \cup \Sigma_2, \delta_3, \{i_1\}, F_2)$$

onde $\delta_3$ é dada por:

- $\delta_3(e, a) = \{\delta_1(e, a)\}$ para todo $e \in E_1$ e $a \in \Sigma_1$;
- $\delta_3(e, a) = \{\delta_2(e, a)\}$ para todo $e \in E_2$ e $a \in \Sigma_2$;
- $\delta_3(e, \lambda) = \{i_2\}$ para todo $e \in F_1$, e $\delta_3(e, \lambda) = \emptyset$ para $e \in (E_1 \cup E_2) - F_1$.

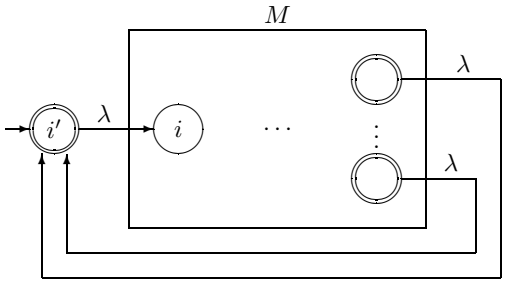



Figura 2.25: Fechamento sob fecho de Kleene.

*Prova de 5*. Seja uma linguagem regular qualquer $L$. Seja um AFD $M = (E, \Sigma, \delta, i, F)$ tal que $L(M) = L$. O seguinte AFN$\lambda$, $M'$, reconhece $L(M)^*$ (veja Figura 2.25 para uma representação esquemática de $M'$):

$$M' = (E \cup \{i'\}, \Sigma, \delta', \{i'\}, F \cup \{i'\})$$

onde $i' \notin E$, e $\delta'$ é dada por:

- $\delta'(i', \lambda) = \{i\}$;
- $\delta'(e, a) = \{\delta(e, a)\}$ para todo $e \in E$ e $a \in \Sigma$;
- $\delta'(e, \lambda) = \{i'\}$ para todo $e \in F$, e $\delta'(e, \lambda) = \emptyset$ para $e \in E - F$. □

Observe como a introdução de transições sob $\lambda$ torna simples a obtenção de AF's para os casos da concatenação e do fecho de Kleene.

Três exemplos de aplicações das propriedades das linguagens regulares: (1) permitem provar que uma linguagem é regular; (2) permitem provar que uma linguagem não é regular; e (3) facilitam a obtenção de AF para uma linguagem regular.

Seguem-se exemplos das 3 aplicações. Primeiramente, uma aplicação do tipo (1).

**Exemplo 74** Seja a linguagem constituída de todas as palavras binárias que representem números divisíveis por 6, com exceção daquelas em que o terceiro dígito da direita para a esquerda seja `1`. Ora, esta linguagem nada mais é do que $L = L_1 - L_2$, onde:

- $L_1 = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w \text{ representa número divisível por } 6\}$; e
- $L_2 = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid \text{ o terceiro dígito de } w, \text{ da direita para a esquerda, é } \mathtt{1}\}$.

Sabe-se que $L_1$ e $L_2$ são linguagens regulares: as Figuras 2.2, página 60, e 2.12, página 75, apresentam AF's para $L_1$, e a Figura 2.17, página 87, apresenta AF's para $L_2$. Como $L = L_1 - L_2 = L_1 \cap \overline{L_2}$, e a classe das linguagens regulares é fechada sob complementação e sob interseção (pelo Teorema 9), segue-se que $L$ é linguagem regular. □

Segue-se um exemplo de aplicação do tipo (3).

**Exemplo 75** Seja a linguagem $L$ do exemplo anterior. Como ressaltado neste exemplo, $L = L_1 - L_2$, onde:

- $L_1 = \{w \in \{\mathtt{0},\mathtt{1}\}^* \mid w$ representa número divisível por 6$\}$; e
- $L_2 = \{w \in \{\mathtt{0},\mathtt{1}\}^* \mid$ o terceiro dígito de $w$, da direita para a esquerda, é $\mathtt{1}\}$.

Na Figura 2.12, página 75, tem-se um AFD para $L_1$ e na Figura 2.17, página 87, tem-se um AFD para $L_2$. Usando-se as técnicas do Teorema 4, pode-se construir facilmente um AFD para $\overline{L_2}$ e, em seguida, um AFD para $L_1 \cap \overline{L_2}$. Este último seria um AFD para $L$. □

Para uma aplicação do tipo (2), provar que uma linguagem $L$ não é regular, raciocina-se por contradição. Primeiramente, supõe-se que $L$ é uma linguagem regular. Depois, aplica-se uma propriedade de fechamento envolvendo $L$ e, eventualmente, outras linguagens, estas comprovadamente linguagens regulares. Obtendo-se uma linguagem que, comprovadamente, *não* é regular, ter-se-á uma contradição já que, pela propriedade de fechamento, deveria ter sido obtida uma linguagem regular. Logo, a suposição de que $L$ seria linguagem regular não se sustentaria. Segue-se um exemplo.

**Exemplo 76** Seja $L = \{\mathtt{a}^k\mathtt{b}^m\mathtt{c}^n \mid k = m + n\}$. Prova-se, a seguir, que $L$ não é regular.

Suponha que $L$ é uma linguagem regular. Como $\{\mathtt{a}\}^*\{\mathtt{b}\}^*$ é linguagem regular e a classe das linguagens regulares é fechada sob interseção, segue-se que $L \cap \{\mathtt{a}\}^*\{\mathtt{b}\}^*$ deve ser uma linguagem regular. Mas, $L \cap \{\mathtt{a}\}^*\{\mathtt{b}\}^* = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$, que não é linguagem regular. Logo, $L$ não é linguagem regular. □

# Exercícios

1. Em ambos os Exemplos 63 e 71, nas páginas 79 e 98, mostrou-se que a linguagem $\{\mathtt{a}^n\mathtt{b}^n\} \mid n \geq 0\}$ não é regular. Em que diferem ambas as provas?

2. Formalmente, o LB seria enunciado assim: $L$ é regular $\rightarrow \exists k \in \mathbf{N}\forall z \in L[|z| \geq k \rightarrow \exists u, v, w(z = uvw \wedge |uv| \leq k \wedge |v| \geq 1 \wedge \forall i \in \mathbf{N} uv^i w \in L)]$. Mostre que a contrapositiva é: $\forall k \in \mathbf{N}\exists z \in L[|z| \geq k \wedge \forall u, v, w((z = uvw \wedge |uv| \leq k \wedge |v| \geq 1) \rightarrow \exists i \in \mathbf{N} uv^i w \notin L)] \rightarrow L$ não é regular.

3. Prove que os seguintes conjuntos não são linguagens regulares, utilizando o LB:
   (a) $\{\mathtt{0}^m\mathtt{1}^n \mid m < n\}$.
   (b) $\{\mathtt{0}^n\mathtt{1}^{2^n} \mid n \geq 0\}$.
   (c) $\{\mathtt{0}^m\mathtt{1}^n\mathtt{0}^m \mid m, n \geq 0\}$.
   (d) $\{x\mathtt{c}x \mid x \in \{\mathtt{a},\mathtt{b}\}^*\}$.
   (e) $\{\mathtt{10}^n\mathtt{1}^n \mid n \geq 0\}$.
   (f) $\{\mathtt{0}^{n^2} \mid n \geq 0\}$.

4. Prove que os seguintes conjuntos não são linguagens regulares, utilizando propriedades de fechamento:

   (a) $\{\mathtt{0},\mathtt{1}\}^* - \{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$.

   (b) $\{\mathtt{0}^m\mathtt{1}^n \mid m < n\} \cup \{\mathtt{0}^m\mathtt{1}^n \mid m > n\}$.

   (c) $\{w \in \{\mathtt{0},\mathtt{1}\}^* \mid$ o número de $\mathtt{0}$'s em $w$ é igual ao número de $\mathtt{1}$'s$\}$.

   (d) $\{w \in \{\mathtt{0},\mathtt{1}\}^* \mid$ o número de $\mathtt{0}$'s em $w$ é igual ao número de $\mathtt{1}$'s$\}$. $-\{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$.

5. Seja $L$ uma linguagem regular sobre o alfabeto $\{\mathtt{a},\mathtt{b},\mathtt{c}\}$. Mostre que cada uma das linguagens seguintes é regular:

   (a) $\{w \in L \mid w$ contém pelo menos um $\mathtt{a}\}$.

   (b) $\{w \mid w \in L$ ou $w$ contém pelo menos um $\mathtt{a}$ (ou ambos)$\}$.

   (c) $\{w \mid$ ou $w \in L$ ou $w$ contém pelo menos um $\mathtt{a}\}$.

   (d) $\{w \notin L \mid w$ não contém $\mathtt{a}$'s$\}$.

6. Complete o Exemplo 75 (página 102): utilizando a receita dada lá, obtenha um AFD que reconheça a linguagem constituída de todas as palavras binárias que representem números divisíveis por 6, com exceção daquelas em que o terceiro dígito da direita para a esquerda seja $\mathtt{1}$.

7. Prove que os seguintes conjuntos são linguagens regulares:

   (a) $\{\mathtt{0}^m\mathtt{1}^n \mid m < n\} \cup \{\mathtt{0}^m\mathtt{1}^n \mid m > n\} \cup \{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$.

   (b) $\{\mathtt{0},\mathtt{1}\}^* - \{\mathtt{01},\mathtt{10}\}^*$.

   (c) $\{\mathtt{0}^n\mathtt{1}^n \mid 0 \leq n \leq 10^{1000}\}$.

8. Seja $L$ uma linguagem regular sobre um alfabeto $\Sigma_1$. Prove que o conjunto de todas as palavras sobre $\Sigma_2$ (que pode ser igual ou não a $\Sigma_1$) que têm como sufixo alguma palavra de $L$ é linguagem regular.

## 2.5 Máquinas de Mealy e de Moore

As máquinas de Mealy e de Moore são autômatos finitos com saída. Uma máquina de Mealy associa um símbolo de saída a cada transição. E uma máquina de Moore associa um símbolo de saída a cada estado. Apesar de um ou outro tipo de máquina ser mais conveniente em aplicações específicas, uma máquina de Moore pode ser convertida em uma máquina de Mealy equivalente, e vice-versa. Neste capítulo, serão apresentados ambos os tipos de máquina de forma sucinta, assim como os procedimentos de conversão de um tipo em outro.

Uma máquina de Moore nada mais é do que um AFD com um símbolo de saída associado a cada estado, como mostra a definição a seguir.

**Definição 17** *Uma* máquina de Moore *é uma sêxtupla* $(E, \Sigma, \Delta, \delta, \sigma, i)$*, onde:*

- *$E$ (o conjunto de estados), $\Sigma$ (o alfabeto de entrada), $\delta$ (a função de transição) e $i$ (o estado inicial) são como em AFD's;*
- *$\Delta$ é o alfabeto de saída; e*
- *$\sigma : E \to \Delta$ é a função de saída, uma função total.* □

Uma máquina de Moore funciona de forma similar a um AFD. A diferença é que, ao invés de uma saída binária do tipo sim ou não, existirá uma *palavra de saída*; esta é inicializada com $\sigma(i)$ e, ao ser efetuada uma transição para um estado $e$, $\sigma(e)$ é concatenado à direita da palavra de saída.

Para uma definição mais precisa da saída computada para uma máquina de Moore $M$, será definida a função $r : E \times \Sigma^* \to \Delta^*$. Dado um estado $e$ e uma palavra $w$, $r(e, w)$ será a palavra de saída correspondente à computação que leva ao estado $\hat{\delta}(e, w)$. A definição de $\hat{\delta}$ para máquina de Moore é igual àquela apresentada na Definição 2 (página 65). Segue a definição de $r$.

**Definição 18** *Seja uma máquina de Moore $M = (E, \Sigma, \Delta, \delta, \sigma, i)$. A* função de saída estendida *para $M$, $r : E \times \Sigma^* \to \Delta^*$ é definida recursivamente como segue:*

*(a) $r(e, \lambda) = \sigma(e)$;*

*(b) $r(e, ay) = \sigma(e)r(\delta(e, a), y)$, para todo $a \in \Sigma$ e $y \in \Sigma^*$.* □

Com isto, pode-se então definir saída de uma máquina de Moore.

**Definição 19** *A* saída computada *por uma máquina de Moore $M = (E, \Sigma, \Delta, \delta, \sigma, i)$ para a palavra $w \in \Sigma^*$ é $r(i, w)$.* □

Dado um AFD $M = (E, \Sigma, \delta, i, F)$ é possível constuir uma máquina de Moore $M' = (E, \Sigma, \Delta, \delta, \sigma, i)$ tal que:

$$w \in L(M) \text{ se, e somente se, } P(r(i, w)),$$

onde $P$ é uma propriedade bem definida. Com isto, pode-se dizer que qualquer AFD pode ser "simulado" por meio de máquina de Moore. Uma possibilidade é fazer:

- $\Delta = \{\mathtt{0}, \mathtt{1}\}$; e
- $\sigma(e) = \mathtt{1}$ se $e \in F$, e $\sigma(e) = \mathtt{0}$ se $e \notin F$.

Tem-se, com isto, que:

$$w \in L(M) \text{ se, e somente se, } r(i, w) \text{ termina em } \mathtt{1}.$$

O diagrama de estados de uma máquina de Moore é similar ao de um AFD. A única diferença é que cada vértice, ao invés de representar um estado, representa um estado e a saída correspondente. Assim, a transição $\delta(e, a) = e'$ é representada assim, juntamente com $\sigma(e)$ e $\sigma(e')$:

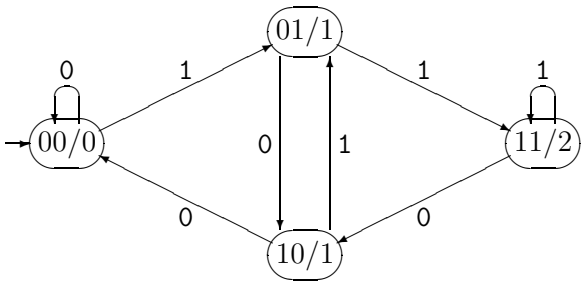
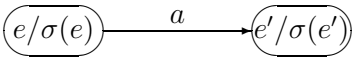



Figura 2.26: Uma Máquina de Moore.




**Exemplo 77** Na Figura 2.26 está mostrado o diagrama de estados para uma máquina de Moore que determina o número de 1's presentes nos dois últimos dígitos de uma palavra de $\{0,1\}^*$. No caso, tal número é dado pelo último símbolo da palavra de saída. Por exemplo:

$$\begin{aligned} r(00,1110) &= 0r(01,110) \\ &= 01r(11,10) \\ &= 012r(11,0) \\ &= 0122r(10,\lambda) \\ &= 01221. \end{aligned}$$

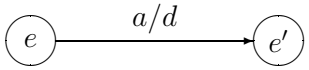
□

Uma máquina de Mealy é um AFD com um símbolo de saída associado a cada transição, como definido a seguir.

**Definição 20** *Uma* máquina de Mealy *é uma sêxtupla* $(E, \Sigma, \Delta, \delta, \sigma, i)$*, onde:*

- $E$ *(o conjunto de estados),* $\Sigma$ *(o alfabeto de entrada),* $\delta$ *(a função de transição) e* $i$ *(o estado inicial) são como em AFD's;*
- $\Delta$ *é o alfabeto de saída;*
- $\sigma : E \times \Sigma \to \Delta$ *é a função de saída, uma função total.* □

Seja uma transição $\delta(e,a) = e'$ e a saída $\sigma(e,a) = d$ ($e, e' \in E$, $a \in \Sigma$, $d \in \Delta$). Tal transição será dita ser uma transição de $e$ para $e'$ sob $a/d$. Ela é representada em um diagrama de estados da seguinte maneira:



como exemplifica a Figura 2.3, na página 61.

Assim como uma máquina de Moore, uma máquina de Mealy é similar a um AFD, a diferença sendo também que, ao invés de aceitação ou rejeição, existirá uma *palavra de saída*. No caso, esta é inicializada com $\lambda$ e, ao ser efetuada uma transição para um estado $e$ sob $a/d$, $d$ é concatenada à direita dela.

A seguir será definida a função $s : E \times \Sigma^* :\to \Delta^*$, onde $s(e, w)$ é a palavra de saída emitida pela máquina de Mealy para a palavra de entrada $w$, quando a máquina é iniciada no estado $e$.

**Definição 21** *Seja uma máquina de Mealy* $M = (E, \Sigma, \Delta, \delta, \sigma, i)$. *A* função de saída estendida *para* $M$, $s : E \times \Sigma^* \to \Delta^*$, *é definida recursivamente como segue:*

*(a)* $s(e, \lambda) = \lambda$;

*(b)* $s(e, ay) = \sigma(e, a)s(\delta(e, a), y)$, *para todo* $a \in \Sigma$ *e* $y \in \Sigma^*$. □

Define-se, com isto, a saída de uma máquina de Mealy.

**Definição 22** *A* saída computada *por uma máquina de Mealy* $M = (E, \Sigma, \Delta, \delta, \sigma, i)$ *para a palavra* $w \in \Sigma^*$ *é* $s(i, w)$. □

A máquina de Mealy apresentada na Seção 2.1.3, cujo diagrama de estados está mostrado na Figura 2.3, página 61, seria então uma sêxtupla

$$(\{1, 2\uparrow, 2\downarrow, 3\}, \mathcal{P}(\{1, 2, 3\}), \{\uparrow, \downarrow, \circ\}, \delta, \sigma, 1)$$

onde $\delta$ e $\sigma$ são dadas por (na linha $e$, coluna $a$, está o par $\delta(e, a)/\sigma(e, a)$):

| $\delta/\sigma$ | $\emptyset$ | $\{1\}$ | $\{2\}$ | $\{3\}$ | $\{1,2\}$ | $\{1,3\}$ | $\{2,3\}$ | $\{1,2,3\}$ |
|---|---|---|---|---|---|---|---|---|
| 1 | $1/\circ$ | $1/\circ$ | $2\uparrow/\uparrow$ | $2\uparrow/\uparrow$ | $1/\circ$ | $1/\circ$ | $2\uparrow/\uparrow$ | $1/\circ$ |
| $2\uparrow$ | $2\uparrow/\circ$ | $1/\downarrow$ | $2\uparrow/\circ$ | $3/\uparrow$ | $2\uparrow/\circ$ | $3/\uparrow$ | $2\uparrow/\circ$ | $2\uparrow/\circ$ |
| $2\downarrow$ | $2\downarrow/\circ$ | $1/\downarrow$ | $2\downarrow/\circ$ | $3/\uparrow$ | $2\downarrow/\circ$ | $1/\downarrow$ | $2\downarrow/\circ$ | $2\downarrow/\circ$ |
| 3 | $3/\circ$ | $2\downarrow/\downarrow$ | $2\downarrow/\downarrow$ | $3/\circ$ | $2\downarrow/\downarrow$ | $3/\circ$ | $3/\circ$ | $3/\circ$ |

**Exemplo 78** Na Figura 2.2, página 60, foi mostrado um diagrama de estados para um AFD que determina se um número em notação binária é divisível por 6. Tal AFD pode ser complementado para se tornar uma máquina de Mealy que determina também o quociente da divisão. Seja $x \in \{\mathtt{0}, \mathtt{1}\}^*$, e sejam $q_1$ e $r_1$ o quociente e o resto da divisão de $\eta(x)$ por 6, onde $\eta(x)$ é o número representado por $x$. Assim, $\eta(x) = 6q_1 + r_1$. Supondo que após $x$ vem outro símbolo $a \in \{\mathtt{0}, \mathtt{1}\}$, sejam $q_2$ e $r_2$ o quociente e o resto da divisão de $\eta(xa)$ por 6. Tem-se dois casos a considerar:

- $a = \mathtt{0}$. Neste caso, tem-se que $\eta(x\mathtt{0}) = 2\eta(x) = 6q_2 + r_2$ e, portanto, $2(6q_1 + r_1) = 6q_2 + r_2$. Assim, $q_2 = 2q_1 + (2r_1 - r_2)/6$.

- $a = \mathtt{1}$. Neste caso, $\eta(x\mathtt{1}) = 2\eta(x)+1 = 6q_2+r_2$ e, portanto, $2(6q_1+r_1)+1 = 6q_2+r_2$. Assim, $q_2 = 2q_1 + (2r_1 + 1 - r_2)/6$.

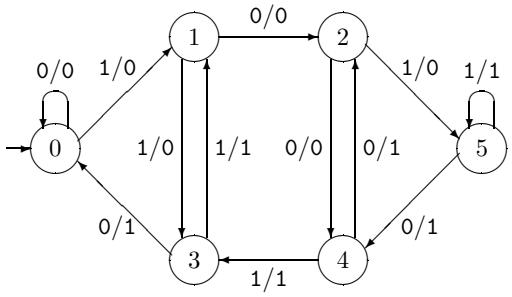



Figura 2.27: Máquina de Mealy para Binário Módulo 6.

Veja, então, que se o próximo dígito da palavra de entrada é `0`, o próximo dígito do quociente é dado por $(2r_1 - r_2)/6$, e se o próximo dígito da palavra de entrada é `1`, o próximo dígito do quociente é dado por $(2r_1 - r_2 + 1)/6$. Com isto, obtém-se a máquina mostrada na Figura 2.27.

Note que esta máquina dá, além do quociente (via as transições), também o resto (via os estados). Pode-se dizer, no caso, que se tem "duas máquinas em uma", uma de Mealy (para o quociente) e outra de Moore (para o resto). □

Estritamente falando, pode não haver uma máquina de Mealy equivalente a uma máquina de Moore: para uma máquina de Moore de estado inicial $i$, $r(i, \lambda) = \sigma(i)$, que pode não ser $\lambda$, enquanto que, para uma máquina de Mealy de estado inicial $i'$, $s(i', \lambda)$ é sempre $\lambda$! Assim, na definição de equivalência abaixo, não se leva em conta a saída da máquina de Moore para o prefixo $\lambda$ da palavra de entrada.

**Definição 23** *Uma máquina de Moore* $(E_1, \Sigma, \Delta, \delta_1, \sigma_1, i_1)$ *e uma máquina de Mealy* $(E_2, \Sigma, \Delta, \delta_2, \sigma_2, i_2)$ *são ditas* equivalentes *se, para todo* $w \in \Sigma^*$, $r(i_1, w) = \sigma_1(i_1)s(i_2, w)$. □

O teorema a seguir mostra como obter uma máquina de Mealy equivalente a uma máquina de Moore.

**Teorema 10** *Para toda máquina de Moore existe uma máquina de Mealy equivalente.*

*Prova*

Seja uma máquina de Moore $M = (E, \Sigma, \Delta, \delta, \sigma, i)$. Uma máquina de Mealy equivalente é $M' = (E, \Sigma, \Delta, \delta, \sigma', i)$, onde para cada $e \in E$ e $a \in \Sigma$, $\sigma'(e, a) = \sigma(\delta(e, a))$. Será mostrado, por indução sobre $|w|$, que $r(e, w) = \sigma(e)s(e, w)$ para $e \in E$ arbitrário. Com isto, conclui-se, em particular, que $r(i, w) = \sigma(i)s(i, w)$.

Para $w = \lambda$ tem-se:

$$\begin{aligned} r(e, \lambda) &= \sigma(e) && \text{pela definição de } r \\ &= \sigma(e)\lambda \\ &= \sigma(e)s(e, \lambda) && \text{pela definição de } s. \end{aligned}$$

Suponha, como hipótese de indução, que $r(e, y) = \sigma(e)s(e, y)$, para $y \in \Sigma^*$ tal que $|y| = n$. Seja uma palavra de tamanho $n + 1$, $w = ay$, $a \in \Sigma$. Então:

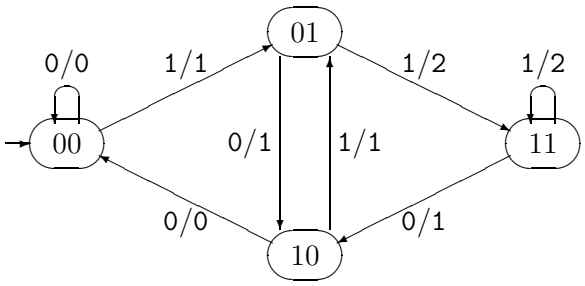



Figura 2.28: Uma Máquina de Mealy obtida de uma Máquina de Moore.

$$\begin{array}{lll} r(e,ay) = \sigma(e)r(\delta(e,a),y) & & \text{pela definição de } r \\ \quad = \sigma(e)\sigma(\delta(e,a))s(\delta(e,a),y) & & \text{pela hipótese de indução} \\ \quad = \sigma(e)\sigma'(e,a)s(\delta(e,a),y) & & \text{pela definição de } \sigma' \\ \quad = \sigma(e)s(e,ay) & & \text{pela definição de } s. \end{array}$$

□

Segue um exemplo de aplicação da técnica descrita na prova do Teorema 10.

**Exemplo 79** Na Figura 2.28 está o diagrama de estados da máquina de Mealy equivalente à máquina de Moore do Exemplo 77, página 105, obtida de acordo com a técnica descrita na prova do Teorema 10. □

A seguir, o Teorema 11 mostra como obter uma máquina de Moore equivalente a uma máquina de Mealy dada.

**Teorema 11** *Para toda máquina de Mealy existe uma máquina de Moore equivalente.*

*Prova*

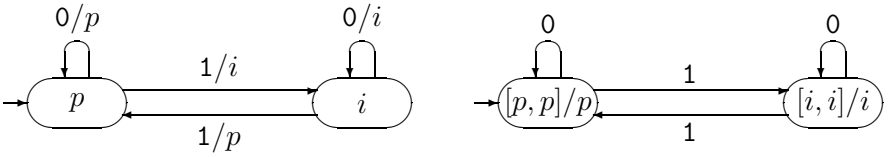
Seja uma máquina de Mealy $M = (E, \Sigma, \Delta, \delta, \sigma, i)$. Uma máquina de Moore equivalente seria $M' = (E', \Sigma, \Delta, \delta', \sigma', i')$, onde:

- $i' = [i, d_0]$ para um certo $d_0 \in \Delta$ (qualquer um serve);
- $E' = \{[\delta(e,a), \sigma(e,a)] \mid e \in E \text{ e } a \in \Sigma\} \cup \{i'\}$;
- $\delta'([e,d], a) = [\delta(e,a), \sigma(e,a)]$ para cada $[e,d] \in E'$ e $a \in \Sigma$;
- $\sigma'([e,d]) = d$ para cada $e \in E$ e $d \in \Delta$.

Para provar a equivalência entre $M$ e $M'$, deve-se mostrar que $r([i,d_0],w) = d_0 s(i,w)$ para todo $w \in \Sigma^*$. Isto segue do resultado mais geral $r([e,d],w) = ds(e,w)$ para todo $[e,d] \in E'$ e $w \in \Sigma^*$, que será provado a seguir, por indução sobre $|w|$.

Seja $[e,d] \in E'$ arbitrário. No caso em que $|w| = 0$, tem-se que:

$$\begin{array}{lll} r([e,d],\lambda) = \sigma'([e,d]) & & \text{pela definição de } r \\ \quad = d & & \text{pela definição de } \sigma' \\ \quad = d\lambda & & \\ \quad = ds(e,\lambda) & & \text{pela definição de } s. \end{array}$$




(a) Máquina de Mealy (b) Máquina de Moore

Figura 2.29: Uma Máquina de Moore obtida de uma Máquina de Mealy.

Suponha, como hipótese de indução, que $r([e,d],z) = ds(e,z)$ para palavras $z \in \Sigma^*$ de tamanho $n$. Seja $w = ay$, onde $a \in \Sigma$, $y \in \Sigma^*$ e $|y| = n$. Tem-se:

$$\begin{aligned}
r([e,d],ay) &= \sigma'([e,d])r(\delta'([e,d],a),y) && \text{pela definição de } r\\
&= dr(\delta'([e,d],a),y) && \text{pela definição de } \sigma'\\
&= dr([\delta(e,a),\sigma(e,a)],y) && \text{pela definição de } \delta'\\
&= d\sigma(e,a)s(\delta(e,a),y) && \text{pela hipótese de indução}\\
&= ds(e,ay) && \text{pela definição de } s.
\end{aligned}$$

□

A técnica descrita na prova do Teorema 11 é exemplificada a seguir.

**Exemplo 80** Na Figura 2.29(b) está o diagrama de estados da máquina de Moore equivalente à máquina de Mealy da Figura 2.29(a), obtida de acordo com a técnica descrita na prova do Teorema 11. Tais máquinas emitem $p$ quando a palavra de entrada tem número par de 1's, e emitem $i$ quando a palavra tem número ímpar de 1's. O estado inicial da máquina de Moore poderia ser $[p,i]$. Neste caso, a máquina teria três estados ao invés de dois. □

# Exercícios

1. Construa uma máquina de Moore que determine o resto da divisão por 3 de um número na representação binária.

2. Construa uma máquina de Moore que determine a quantidade de 1's presentes nos últimos 3 dígitos de palavras sobre $\{0,1\}$.

3. Construa uma máquina de Mealy que determine o quociente da divisão por 3 de um número na representação binária.

4. Construa uma máquina de Mealy que some dois números na base binária. Os números devem ser supridos por meio do alfabeto $\{[0,0],[0,1],[1,0],[1,1]\}$, dígitos menos significativos em primeiro lugar. Por exemplo, para somar os números 13 e 20, pode-se suprir a palavra $[0,1][1,0][1,1][0,0][1,0]$, onde os primeiros dígitos, 01101, codificam o número 13, e os segundos, 10100, codificam o número 20; neste caso, a saída deve ser 100001, que codifica o número 33.

5. Determine uma máquina de Mealy equivalente à máquina de Moore do Exercício 1 usando a técnica da prova do Teorema 10.

6. Determine uma máquina de Moore equivalente à máquina de Mealy do Exercício 3 usando a técnica da prova do Teorema 11.

## 2.6 Expressões Regulares

Até o momento foram vistas duas formas de especificar uma linguagem regular: uma mediante o uso da notação usual de teoria de conjuntos, e outra mediante o desenho de um diagrama de estados, este último favorecendo a visão do reconhecedor como um grafo. As duas têm suas aplicações. Por exemplo, geralmente a primeira é mais conveniente no desenvolvimento da teoria, e em geral a segunda é mais conveniente durante um processo de concepção de um reconhecedor específico. As duas têm em comum o fato de que se destinam à especificação de uma linguagem mediante um *reconhecedor* para a mesma.

Nesta seção e na próxima serão apresentadas duas outras formas de especificar linguagens regulares: expressões regulares e gramáticas regulares. A primeira especifica uma linguagem mediante uma expressão que a *denota*, e a segunda mediante um conjunto de regras que a *gera*. A denotação de uma linguagem via uma expressão regular pode ser útil quando se deseja uma maneira concisa de se referir à linguagem como um todo. Tal utilidade se manisfesta tanto no plano da teoria, onde expressões regulares são adequadas para manipulações formais, quanto no plano prático, onde expressões regulares, ou notações derivadas delas, têm sido utilizadas para referência compacta a conjuntos de palavras (por exemplo, em editores de texto e comandos de sistemas operacionais). As gramáticas regulares, de menor aplicabilidade, têm o mérito teórico de prover um lugar para as linguagens regulares na denominada Hierarquia de Chomsky. A partir de uma expressão regular ou de uma gramática regular pode-se construir um AF de forma automática. Assim, os dois formalismos são também mais dois instrumentos alternativos que podem ser utilizados como etapa intermediária para a obtenção de reconhecedores.

A seguir, serão abordadas as expressões regulares, e na Seção 2.7, as gramáticas regulares. Finalmente, será feito um apanhado geral dos principais resultados do capítulo na Seção 2.8.

Dado um AF $M$, seria possível obter uma formulação *linear* para $L(M)$? Ou seja, seria possível obter uma expressão $r$ que denotasse $L(M)$? A resposta é sim: mediante uma *expressão regular*, como mostrado a seguir, após a definição de expressão regular. Além disso, será mostrado também que, dada uma expressão regular, é possível construir um AF que reconhece a linguagem denotada por ela. Logo, ter-se-á mostrado que a família das linguagens que podem ser denotadas por expressões regulares é exatamente a família das linguagens regulares.

Na definição recursiva a seguir, ao mesmo tempo em que se define expressão regular, define-se também o conjunto denotado por ela. Será usada a notação $L(r)$ para significar o conjunto denotado pela expressão regular $r$.

**Definição 24** *Uma* expressão regular *(ER) sobre um alfabeto $\Sigma$ é definida recursivamente como segue:*

*(a)* $\emptyset$, $\lambda$, *e* $a$ *para qualquer* $a \in \Sigma$ *são expressões regulares; tais ER's denotam, respectivamente, os conjuntos* $\emptyset$, $\{\lambda\}$ *e* $\{a\}$*;*

*(b) se* $r$ *e* $s$ *são expressões regulares, então são expressões regulares:* $(r + s)$, $(rs)$, *e* $r^*$*; tais ER's denotam, respectivamente,* $L(r) \cup L(s)$, $L(r)L(s)$ *e* $L(r)^*$. □

Um conjunto que pode ser denotado por uma ER é usualmente denominado um *conjunto regular*. Mais à frente será mostrado que os conjuntos regulares nada mais são que as linguagens regulares. Seguem alguns exemplos de expressões e conjuntos regulares.

**Exemplo 81** Seja o alfabeto $\Sigma = \{\mathtt{0}, \mathtt{1}\}$. Os seguintes exemplos de ER's sobre $\Sigma$ e conjuntos regulares denotados por elas são imediatos a partir da Definição 24:

- $\emptyset$ denota $\emptyset$.
- $\lambda$ denota $\{\lambda\}$.
- $(01)$ denota $\{\mathtt{0}\}\{\mathtt{1}\} = \{\mathtt{01}\}$.
- $(0 + 1)$ denota $\{\mathtt{0}\} \cup \{\mathtt{1}\} = \{\mathtt{0}, \mathtt{1}\}$.
- $((0 + 1)(01))$ denota $\{\mathtt{0}, \mathtt{1}\}\{\mathtt{01}\} = \{\mathtt{001}, \mathtt{101}\}$.
- $0^*$ denota $\{\mathtt{0}\}^* = \{\mathtt{0}^n \mid n \geq 0\}$.
- $(0 + 1)^*$ denota $\{\mathtt{0}, \mathtt{1}\}^* = \Sigma^*$.
- $(((0+1)^*1)(0+1))$ denota $(\{\mathtt{0}, \mathtt{1}\}^*\{\mathtt{1}\})\{\mathtt{0}, \mathtt{1}\} = \{w \in \Sigma^* \mid$ o penúltimo símbolo de $w$ é $\mathtt{1}\}$. □

Quando uma ER possui muitos parênteses, ela costuma ficar difícil de escrever e de entender. Assim, torna-se útil formular algumas regras para omissão de parênteses:

(a) Como a união é associativa, isto é, $A \cup (B \cup C) = (A \cup B) \cup C$, pode-se escrever $(r_1 + r_2 + \cdots + r_n)$, omitindo-se os parênteses internos. Exemplo: $((0 + (11)) + 1)$ e $(0 + ((11) + 1))$ podem, ambas, serem escritas como $(0 + (11) + 1)$.

(b) Idem, para a concatenação. Exemplo: $((01)((00)1))$ pode ser escrita como $(01001)$.

(c) Se a expressão tem parênteses externos, eles podem ser omitidos. Exemplos: os dois exemplos acima poderiam ser escritos assim: $0 + (11) + 1$ e $01001$.

(d) Para omissão de mais parênteses, será assumindo que o fecho de Kleene tem maior prioridade do que união e concatenação, e que concatenação tem maior prioridade do que união. Exemplo: $(0 + (10^*))$ pode ser escrita como $0 + 10^*$.

**Exemplo 82** Existem duas abordagens básicas para se tentar obter diretamente uma ER que denote uma linguagem. Elas serão introduzidas mediante um exemplo.

Seja $L$ o conjunto das palavras sobre $\{\mathtt{a}, \mathtt{b}\}$ com pelo menos um $\mathtt{b}$. A primeira abordagem é tentar visualizar a linguagem como um todo, utilizando não determinismo quando possível. Assim, uma palavra de $L$ tem um $\mathtt{b}$, que é precedido por zero ou mais símbolos

| | | | |
|---|---|---|---|
| (1) | $r + s = s + r$ | (11) | $r^{**} = r^*$ |
| (2) | $r + \emptyset = r$ | (12) | $r^* = (rr)^*(\lambda + r)$ |
| (3) | $r + r = r$ | (13) | $\emptyset^* = \lambda$ |
| (4) | $r\lambda = \lambda r = r$ | (14) | $\lambda^* = \lambda$ |
| (5) | $r\emptyset = \emptyset r = \emptyset$ | (15) | $r^*r^* = r^*$ |
| (6) | $(r + s)t = rt + st$ | (16) | $rr^* = r^*r$ |
| (7) | $r(s + t) = rs + rt$ | (17) | $(r^* + s)^* = (r + s)^*$ |
| (8) | $(r + s)^* = (r^*s)^*r^*$ | (18) | $(r^*s^*)^* = (r + s)^*$ |
| (9) | $(r + s)^* = r^*(sr^*)^*$ | (19) | $r^*(r + s)^* = (r + s)^*$ |
| (10) | $(rs)^* = \lambda + r(sr)^*s$ | (20) | $(r + s)^*r^* = (r + s)^*$ |

Tabela 2.1: Algumas Equivalências de Expressões Regulares

(inclusive b's – veja o "não determinismo") e seguido por zero ou mais símbolos; isto traduzido em termos de ER é: $(a + b)^*b(a + b)^*$.

A segunda abordagem é tentar visualizar como as palavras podem ser construídas, deterministicamente, da esquerda para a direita ou da direita para a esquerda. Analisando-se uma palavra de $L$, da esquerda para a direita, determina-se que a mesma pertence a $L$ quando se encontra um b; isto traduzido em termos de ER é: $a^*b(a+b)^*$. Analisando-se uma palavra de $L$, da direita para a esquerda, determina-se que a mesma pertence a $L$ quando se encontra um b; isto traduzido em termos de ER é: $(a + b)^*ba^*$. □

O mesmo conjunto pode ser denotado por várias ER's, como visto no Exemplo 82. É comum a notação $r = s$, para duas ER's $r$ e $s$, para dizer que $r$ e $s$ denotam a mesma linguagem, ou seja, $L(r) = L(s)$. Às vezes pode ser interessante tentar obter uma ER "mais simples", no sentido de que se possa visualizar com mais facilidade, a partir dela, qual é a linguagem denotada. As propriedades da união, concatenação e fecho de Kleene fornecem subsídios para obtenção de ER's equivalentes, eventualmente mais simples. A Tabela 2.1 mostra algumas equivalências derivadas direta ou indiretamente de tais propriedades.

Pode-se mostrar que qualquer equivalência que não envolva fecho de Kleene pode ser derivada a partir das equivalências 1 a 7 mais as propriedades de associatividade da união e da concatenação, $r + (s + t) = (r + s) + t$ e $r(st) = (rs)t$, que se está assumindo implicitamente. No entanto, quando se introduz o fecho de Kleene, deixa de haver um conjunto finito de equivalências a partir das quais se possa derivar qualquer equivalência. Assim, as equivalências da Tabela 2.1 não são suficientes para derivar qualquer equivalência. Por outro lado, várias são redundantes, no sentido de que podem ser obtidas a partir de outras. Por exemplo, 13 pode ser obtida a partir de 2, 5 e 10:

$\emptyset^*$= $(r\emptyset)^*$, por 5
= $\lambda + r(\emptyset r)^*\emptyset$, por 10
= $\lambda + \emptyset$, por 5
= $\lambda$, por 2.

**Exemplo 83** Abaixo segue uma série de simplificações de uma ER utilizando as equivalências da Tabela 2.1. As sub-expressões simplificadas em cada passo estão sublinhadas.

$$
\begin{aligned}
\underline{(00^* + 10^*)}0^*(1^* + 0)^* &= (0 + 1)\underline{0^*0^*}(1^* + 0)^* && \text{por 6}\\
&= (0 + 1)0^*\underline{(1^* + 0)^*} && \text{por 15}\\
&= (0 + 1)0^*\underline{(1 + 0)^*} && \text{por 17}\\
&= (0 + 1)0^*\underline{(0 + 1)^*} && \text{por 1}\\
&= (0 + 1)(0 + 1)^* && \text{por 19.}
\end{aligned}
$$

□

Existem várias outras equivalências que podem ser úteis para simplificação de ER's, além daquelas exemplificadas na Tabela 2.1. Lembrando que uma ER é apenas uma expressão que denota um *conjunto* sobre um certo alfabeto e que a justaposição significa *concatenação*, "+" significa *união* e "$*$" significa *fecho de Kleene*, e usando-se as propriedades conhecidas de tais entidades, pode-se obter novas equivalências, como mostra o próximo exemplo.

**Exemplo 84** Dados os significados dos operadores, pode-se verificar que $(r + rr + rrr + rrrr)^* = r^*$. Como outro exemplo, considere a ER: $((0(0+1)1+11)0^*(00+11))^*(0+1)^*$. Observando que ela é da forma $r(0 + 1)^*$, onde $r$ denota um conjunto de palavras sobre $\{0, 1\}$ *que contém* $\lambda$, pode-se imediatamente concluir que ela é equivalente a $(0+1)^*$, sem analisar $r$! Utilizando-se raciociocínio análogo, pode-se concluir que $r^*(r + s^*) = r^*s^*$: veja que $r^*(r + s^*) = rr^* + r^*s^*$ e $L(rr^*) \subseteq L(r^*s^*)$. □

Uma notação bastante útil é $r^+$, onde $r$ é uma ER, para significar $(rr^*)$. Assim, por exemplo, a última ER do Exemplo 83 poderia ser escrita assim: $(0 + 1)^+$. Uma outra notação útil é $r^n$, $n \geq 0$; recursivamente:

(a) $r^0 = \lambda$;

(b) $r^n = rr^{n-1}$, para $n \geq 1$.

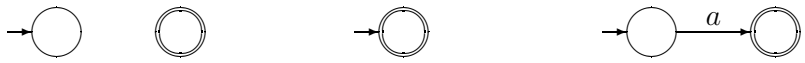
**Exemplo 85** O conjunto de todas as palavras de tamanho 10 sob $\{0, 1\}$ é denotado por $(0 + 1)^{10}$. Como outro exemplo, considere as equivalências da seguinte forma: $r^* = (r^n)^*(\lambda + r + r^2 + \cdots + r^{n-1})$, para $n > 1$ (observe que para $n = 2$, tem-se a identidade 12 da Tabela 2.1). □

A seguir, mostra-se que toda linguagem regular é denotada por uma expressão regular e vice-versa, que toda expressão regular denota uma linguagem regular.

**Teorema 12** *Toda expressão regular denota uma linguagem regular.*

*Prova*

A prova será feita por indução construtiva, ou seja, será mostrado (a) como construir AF's para os elementos primitivos dos conjuntos regulares, quais sejam, $\emptyset$, $\{\lambda\}$ e $\{a\}$ para cada $a \in \Sigma$, e (b) como combinar AF's para simular as operações de união, concatenação e fecho de Kleene. A Figura 2.30 mostra diagramas de estado para AF's que reconhecem $\emptyset$, $\{\lambda\}$ e $\{a\}$. Resta então mostrar que, dados AF's para duas linguagens $L_1$ e $L_2$, é possível construir AF's para $L_1 \cup L_2$, $L_1L_2$ e $L_1^*$. Mas isto já foi mostrado no Teorema 9.




(a) AF para $\emptyset$ (a) AF para $\{\lambda\}$ (a) AF para $\{a\}$

Figura 2.30: AF's para $\emptyset$, $\{\lambda\}$ e $\{a\}$.

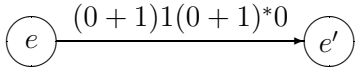
(Observe que os AF's da Figura 2.30 são AFD's. A prova do Teorema 9 constrói AFD para $L_1 \cup L_2$, a partir de quaisquer AFD's para $L_1$ e $L_2$, e AFN$\lambda$'s para $L_1L_2$ e $L_1^*$ a partir de quaisquer AFD's para $L_1$ e $L_2$. E os Teoremas 8 e 7, em seqüência, mostram como construir AFD's a partir de AFN$\lambda$'s. Assim, conclui-se, não apenas que é possível construir AFD's para quaisquer conjuntos regulares, como também tem-se uma forma de construí-los.) □

A construção de um AFD que reconhece a linguagem denotada por uma ER usando os passos apontados no final da prova do Teorema 12, que envolve a aplicação das técnicas dos Teoremas 9, 8 e 7, é bastante trabalhosa, mesmo para ER's relativamente simples. Nos Exercícios 34 e 35, no final do capítulo, são propostos dois outros métodos, um para obtenção de um AFN$\lambda$, e outro para obtenção de um AFN. O primeiro gera AF's ainda maiores e mais redundantes, mas é conceitualmente bastante simples, sendo, inclusive, o método usualmente apresentado em textos de linguagens formais. O segundo, embora ainda possa gerar AFN's grandes e redundantes, pode servir de base para uma implementação real.

A prova do teorema seguinte mostra como construir uma ER que denota a linguagem reconhecida por um AFD. Antes, porém, deve-se introduzir uma extensão do conceito de diagrama de estados de um AF, que será denominado *diagrama ER*.

**Definição 25** *Um* diagrama ER *sobre* $\Sigma$ *é um diagrama de estados cujas arestas, ao invés de serem rotuladas com símbolos do alfabeto* $\Sigma$*, são rotuladas com ER's sobre* $\Sigma$*.* □

Uma transição sob uma ER $r$ pode ser imaginada como representando uma submáquina que lê qualquer palavra do conjunto denotado por $r$. Seja, por exemplo, a transição:




Do estado $e$ há transição para $e'$ sob $w$ se, e somente se, $w \in L((0+1)1(0+1)^*0)$, ou seja, o segundo símbolo de $w$ é `1` e o último é `0`. Assim, por exemplo, as menores palavras para as quais há transição de $e$ para $e'$ são `010` e `110`.

Os exercícios 36 e 37 no final do capítulo permitem formular o conceito de linguagem definida por um diagrama ER. A prova do próximo teorema mostra como obter uma ER a partir de um diagrama ER.

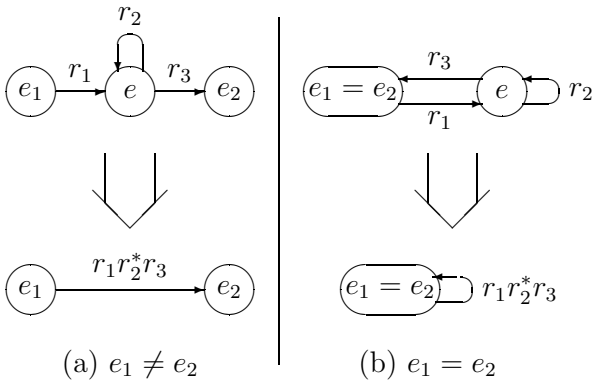



Figura 2.31: Eliminado um estado de um diagrama ER


(a) Diagrama ER para $r^*$ (b) Diagrama ER para $q^*r(s + tq^*r)^*$

Figura 2.32: Diagramas ER básicos

**Teorema 13** *Toda linguagem regular é denotada por alguma expressão regular.*

*Prova*

Seja um AFD $M = (E, \Sigma, \delta, i, \{f_1, f_2, \ldots, f_n\})$. Primeiramente, note que $L(M) = L_1(M) \cup L_2(M) \cup \cdots \cup L_n(M)$, onde $L_k(M) = \{w \in \Sigma^* \mid \hat{\delta}(i, w) = f_k\}$ para $1 \leq k \leq n$. Seja $p_k$ uma expressão regular para $L_k(M)$. Então $L(M)$ seria denotada por meio da expressão regular: $p_1 + p_2 + \cdots + p_n$. Assim, o problema se reduz a encontrar uma ER $p_k$, para cada $L_k(M)$. Abaixo, mostra-se como fazer isto a partir de um diagrama ER obtido a partir de $M$.

Inicialmente, o diagrama ER é como o diagrama de estados de $M$, apenas considerando-se cada transição sob um símbolo $a$ como uma transição sob uma ER $a$. Em seguida, elimina-se um a um os estados do diagrama ER, com exceção de $i$ e $f_k$. Para que um estado $e$ possa ser eliminado, "simula-se" todas as passagens por $e$ possíveis: para cada par de estados $[e_1, e_2]$ tais que há transição de $e_1$ para $e$ e transição de $e$ para $e_2$ ($e_1$ e $e_2$ podem ser o mesmo estado, mas nenhum deles pode ser $e$), produz-se uma transição de $e_1$ para $e_2$, como ilustra a Figura 2.31. Durante todo o processo, é interessante manter apenas uma transição de um vértice para outro. Para isto, se há transições de $e$ para $e'$ sob $s_1$, $s_2$, ..., $s_m$, substitui-se todas elas por uma só transição de $e$ para $e'$ sob $s_1 + s_2 + \cdots + s_m$. Após eliminados todos os estados em $E - \{i, f_k\}$, chega-se a uma das situações básicas mostradas na Figura 2.32. Como está lá mostrado, a ER final para $r_k$ será da forma $r^*$ ou $q^*r(s + tq^*r)^*$. Neste último caso, quaisquer uma das 4 transições do diagrama ER da figura 2.32(b) pode estar ausente; basta, então, substituir a ER correspondente na ER $q^*r(s + tq^*r)^*$ por $\emptyset$ e simplificar; exemplo: se as transições de $i$ para $i$ sob $q$ e de $f_k$ para

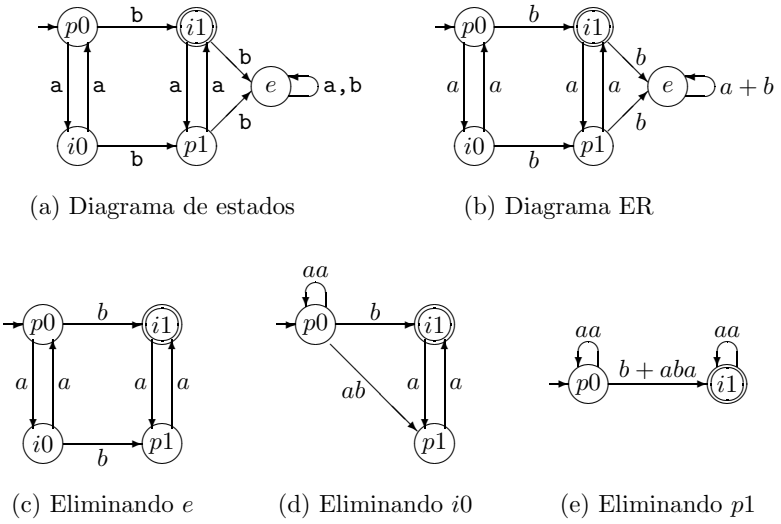
Entrada: um AFD $M = (E, \Sigma, \delta, i, \{f_1, f_2, \ldots, f_n\})$.
Saída: uma ER que denota $L(M)$.

seja o diagrama ER $D$ idêntico ao diagrama de estados de $M$;
    /* Aqui, considere que se não existe $a$ tal que $\delta(e, a) = e'$,
        no diagrama ER há uma transição de $e$ para $e'$ sob $\emptyset$. */
**para** cada par $[e_1, e_2] \in E \times E$ **faça**
    sejam (todas) as transições de $e_1$ para $e_2$ sob $s_1, s_2, \ldots, s_m$ em $D$;
    **se** $m > 1$ **então**
        substitua todas por uma sob $s_1 + s_2 + \cdots + s_m$
    **fimse**
**fimpara**;
**para** $k$ de 1 até $n$ **faça**
    seja o diagrama ER idêntico a $D$, mas com apenas $f_k$ como estado final;
    **para** cada $e \in E - \{i, f_k\}$ **faça**
        **para** cada par $[e_1, e_2] \in E \times E$, $e_1 \neq e$ e $e_2 \neq e$ **faça**
            sejam $s$, $r_1$, $r_2$ e $r_3$ tais que:
                . há transição de $e_1$ para $e_2$ sob $s$,
                . há transição de $e_1$ para $e$ sob $r_1$,
                . há transição de $e$ para $e$ sob $r_2$,
                . há transição de $e$ para $e_2$ sob $r_3$;
            substitua a transição de $e_1$ para $e_2$ sob $s$
                por uma transição de $e_1$ para $e_2$ sob $s + r_1 r_2^* r_3$
        **fimpara**;
        elimine $e$;
    **fimpara**;
    **se** $i = f_k$ **então**
        $p_k \leftarrow r^*$, conforme Figura 2.32(a)
    **senão**
        $p_k \leftarrow q^* r (s + t q^* r)^*$, conforme Figura 2.32(b)
    **fimse**
**fimpara**;
retorne $p_1 + p_2 + \cdots + p_n$

Figura 2.33: Algoritmo AF → ER.

$i$ sob $t$ não existirem, a ER simplificada será: $\emptyset^* r (s + \emptyset q^* r)^* = r s^*$. Veja o Exercício 6 no final desta seção. □

A prova do Teorema 13 é o esboço de um algoritmo para obtenção de uma ER que denota a linguagem reconhecida por um AFD. A seguir, tal esboço é tomado como base para a concretização de um algoritmo. Para propiciar a obtenção de um algoritmo mais conciso e simples, será assumido, para um diagrama ER, que, se não há transição de um estado $e$ para um estado $e'$ (que podem ser o mesmo estado), então "há uma transição de $e$ para $e'$ sob $\emptyset$". É evidente que o efeito, em ambos os casos, é o mesmo. Entretanto, tal suposição serve apenas para simplificar a apresentação do algoritmo; em uma aplicação do mesmo, ou em uma implementação, pode ser mais conveniente trabalhar sem tal suposição. Neste último caso, as alterações requeridas, embora um pouco trabalhosas, são triviais. Na Figura 2.33 está mostrado o algoritmo.




(a) Diagrama de estados (b) Diagrama ER

(c) Eliminando $e$ (d) Eliminando $i0$ (e) Eliminando $p1$

Figura 2.34: Determinando uma ER a partir de AFD.

**Exemplo 86** Seja o diagrama de estados mostrado na Figura 2.34(a), de um AFD que reconhece a linguagem $\{w \in \{\mathtt{a},\mathtt{b}\}^* \mid |w| \text{ é ímpar e } w \text{ contém exatamente um } b\}$. Após o primeiro comando **para** do algoritmo da Figura 2.33, tem-se o diagrama ER mostrado na Figura 2.34(b). No presente caso, será obtido apenas $p_1$, pois há apenas um estado final. Para isto, o algoritmo elimina os estados: $e$, $i0$ e $p1$ (qualquer ordem serve). Supondo que a eliminação ocorre nesta ordem, tem-se:

1) Eliminando $e$. Como não existe transição de $e$ para algum $e_2$ diferente de $e$ (comando **para** mais interno), ele é simplesmente eliminado (após este **para** mais interno). Resultado: Figura 2.34(c).

2) Eliminando $i0$. Existem dois pares $[e_1, e_2]$ que promovem efetivamente mudanças no **para** mais interno: $[p0, p0]$ e $[p0, p1]$. Assim, antes de eliminar $i0$, introduz-se:

   - uma transição de $p0$ para $p0$ sob $\emptyset + a\emptyset^*a = aa$;
   - uma transição de $p0$ para $p1$ sob $\emptyset + a\emptyset^*b = ab$.

   Resultado: Figura 2.34(d).

3) Eliminando $p1$. Existem dois pares $[e_1, e_2]$ que promovem efetivamente mudanças no **para** mais interno: $[p0, i1]$ e $[i1, i1]$. Assim, antes de eliminar $p1$, introduz-se:

   - uma transição de $p0$ para $i1$ sob $b + ab\emptyset^*a = b + aba$ (substituindo a anterior, sob $b$);
   - uma transição de $i1$ para $i1$ sob $\emptyset + a\emptyset^*a = aa$.

   Resultado: Figura 2.34(e).

Tem-se, então a ER determinada: $(aa)^*(b + aba)(aa)^*$. □

No exemplo 86, a ER obtida reflete de forma bastante clara a linguagem que ela denota. Infelizmente, na maior parte dos casos a ER obtida pelo processo do algoritmo da Figura 2.33 é grande e ilegível. Por outro lado, ela é correta e pode ser manipulada, utilizando-se as equivalências da Tabela 2.1 (página 112) e outras, para se chegar a uma ER mais conveniente.

Uma pequena alteração no algoritmo da Figura 2.33 aumenta a eficiência do mesmo quando há mais de um estado final: logo após o primeiro comando **para**, atribuir a $D$ o diagrama ER resultante da eliminação de todos os estados em $E - \{f_1, f_2, \ldots, f_n\} - \{i\}$. Com isto, no comando **para** seguinte, para cada estado final $f_k$, serão eliminados apenas os estados $f_i$ tais que $i \neq k$.

# Exercícios

1. Retire o máximo de parênteses das ER's seguintes, sem alterar seus significados:
   (a) $((0 + ((0 + 1)0)) + (11))$.
   (b) $(((00) + (1(11^*)))^*((01)((01)(0 + 1))))$.

2. Descreva, em português, as linguagens sobre $\{\mathtt{0}, \mathtt{1}\}$ denotadas pelas seguintes ER's:
   (a) $0(0 + 1)^*1$.
   (b) $0^*(0 + 1)1^*$.
   (c) $(0 + 1)^*1(0 + 1)(0 + 1)$.
   (d) $(0 + \lambda)(10 + 1)^*$.

3. Encontre (diretamente) expressões regulares que denotem os seguintes conjuntos:
   (a) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid |w| \geq 3\}$.
   (b) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w$ começa com $\mathtt{a}$ e tem tamanho par$\}$.
   (c) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w$ tem número par de $\mathtt{a}$'s$\}$.
   (d) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w$ contém $\mathtt{bb}\}$.
   (e) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w$ contém exatamente um $\mathtt{bb}\}$.
   (f) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w$ contém apenas um ou dois $\mathtt{b}$'s$\}$.
   (g) $\{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid$ o número de $\mathtt{a}$'s e/ou $\mathtt{b}$'s é par$\}$.
   (h) $\{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid w$ não termina com $\mathtt{cc}\}$.

4. Simplifique as ER's a seguir. Procure usar as equivalências da Tabela 2.1. Se precisar de alguma equivalência não presente na Tabela 2.1, explicite-a e justifique sua validade (mesmo que informalmente).
   (a) $\emptyset^* + \lambda^*$.
   (b) $0^* + 1^* + 0^*1^* + (0 + 1)^*$.
   (c) $(0 + 00)^*(1 + 01)$.

(d) $(0+01)^*(0^*1^*+1)^*$.

(e) $((00+01+10+11)^*(0+1))^*$.

(f) $(0+1)^*0(0+1)^*1(0+1)^*$.

5. Construa um AFD para cada uma das linguagens denotadas pelas ER's:

   (a) $(ab)^*ac$.

   (b) $(ab)^*(ba)^*$.

   (c) $(ab^*a)^*(ba^*b)^*$.

   (d) $(aa+b)^*baab$.

   (e) $((aa+bb)^*cc)^*$.

6. Considere o diagrama ER da Figura 2.32(b), página 115. Simplifique a ER correspondente, $q^*r(s+tq^*r)^*$, para todas as $2^4$ possibilidades de se atribuir $\emptyset$ às expressões básicas $q$, $r$, $s$ e $t$.

7. Construa uma ER que denote a linguagem reconhecida pelo AFD $M$, definido abaixo, utilizando o método do algoritmo da Figura 2.33, página 116:

   $M = (\{0,1,2,3\}, \{\mathtt{a},\mathtt{b}\}, \delta, 0, \{2\})$, sendo $\delta$ dada por:

| $\delta$ | a | b |
|---|---|---|
| 0 | 1 | 3 |
| 1 | 0 | 2 |
| 2 | 3 | 1 |
| 3 | 3 | 3 |

8. Obtenha ER's que denotem as seguintes linguagens sobre $\{\mathtt{0},\mathtt{1}\}$, a partir de um AFD para as mesmas, usando o algoritmo da Figura 2.33, página 116:

   (a) O conjunto das palavras de tamanho maior que 1 tais que os `0`'s (se houver algum) precedem os `1`'s (se houver algum).

   (b) O conjunto das palavras que começam com `1` e terminam com `1`.

   (c) O conjunto das palavras que começam com `1`, terminam com `1` e têm pelo menos um `0`.

   (d) O conjunto das palavras com números ímpar de `0`'s e ímpar de `1`'s.

9. Modifique o algoritmo para obtenção de uma ER a partir de um AFD, mostrado na Figura 2.33, página 116, para a obtenção de uma ER a partir de:

   (a) Um AFN.

   (b) Um AF$\lambda$.

10. Que tipo de linguagem pode ser denotada por uma ER sem ocorrências de fecho de Kleene?

## 2.7 Gramáticas Regulares

Uma gramática regular provê mais uma forma de especificar uma linguagem regular. Enquanto autômatos finitos permitem a especificação de uma linguagem via um reconhecedor para a mesma, e expressões regulares permitem a especificação via uma expressão que a denota, gramáticas regulares permitem a especificação via um *gerador* para mesma. Ou seja, mediante uma gramática regular, mostra-se como gerar todas, e apenas, as palavras de uma linguagem.

**Definição 26** *Uma gramática regular (GR) é uma gramática* $(V, \Sigma, R, P)$*, em que cada regra tem uma das formas:*

- $X \to a$
- $X \to aY$
- $X \to \lambda$

*onde* $X, Y \in V$ *e* $a \in \Sigma$. □

Uma característica interessante das GR's é o formato das formas sentenciais geradas: $wA$, onde $w$ só tem terminais e $A$ é uma variável.

**Exemplo 87** Seja $L = \{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid w \text{ não contém } \mathtt{abc}\}$. Uma GR que gera $L$ seria $(\{A, B, C\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, R, A)$, onde $R$ contém as regras:

$$A \to \mathtt{a}B|\mathtt{b}A|\mathtt{c}A|\lambda$$

$$B \to \mathtt{a}B|\mathtt{b}C|\mathtt{c}A|\lambda$$

$$C \to \mathtt{a}B|\mathtt{b}A|\lambda$$

□

A seguir, mostra-se que as gramáticas regulares geram apenas linguagens regulares e, vice-versa, que toda linguagem regular é gerada por gramática regular. No teorema a seguir, mostra-se como construir um AFN que reconhece a linguagem gerada por uma gramática regular. A cada variável da gramática irá corresponder um estado do AFN, e a cada regra irá corresponder uma transição.

**Teorema 14** *Toda gramática regular gera uma linguagem regular.*

*Prova*

Seja uma GR $G = (V, \Sigma, R, P)$. Será mostrado como construir um AFN $M = (E, \Sigma, \delta, \{P\}, F)$ tal que $L(M) = L(G)$. Deve-se construir $M$ de forma que $P \stackrel{*}{\Rightarrow} w$ se, e somente se, $\hat{\delta}(\{P\}, w) \cap F \neq \emptyset$. Para isto, seja algum $Z \notin V$. Então:

- $E = \begin{cases} V \cup \{Z\} & \text{se } R \text{ contém regra da forma } X \to a \\ V & \text{caso contrário.} \end{cases}$
- Para toda regra da forma $X \to aY$ faça $Y \in \delta(X, a)$, e para toda regra da forma $X \to a$ faça $Z \in \delta(X, a)$.

- $F = \begin{cases} \{X | X \to \lambda \in R\} \cup \{Z\} & \text{se } Z \in E \\ \{X | X \to \lambda \in R\} & \text{caso contrário.} \end{cases}$

Inicialmente, mostra-se por indução sobre $|w|$ que:

$$P \stackrel{*}{\Rightarrow} wX \text{ se, e somente se, } X \in \hat{\delta}(\{P\}, w) \text{ para } w \in \Sigma^* \text{ e } X \in V. \tag{2.4}$$

Pelo formato das regras de $G$, $|w| \geq 1$. Assim, considere $w = a \in \Sigma$. Tem-se:

$$\begin{array}{ll} P \stackrel{*}{\Rightarrow} aX \leftrightarrow P \to aX \in R & \text{pela definição de } \stackrel{*}{\Rightarrow} \text{ e Definição 26} \\ \leftrightarrow X \in \delta(P, a) & \text{pela definição de } \delta \\ \leftrightarrow X \in \hat{\delta}(\{P\}, a) & \text{pela definição de } \hat{\delta} \text{(Definição 10).} \end{array}$$

Suponha que 2.4 vale para um certo $w \in \Sigma^+$ arbitrário. Prova-se que vale para $aw$ para $a \in \Sigma$ arbitrário:

$$\begin{array}{ll} P \stackrel{*}{\Rightarrow} awX \leftrightarrow P \to aY \in R \text{ e } Y \stackrel{*}{\Rightarrow} wX & \text{pela definição de } \stackrel{*}{\Rightarrow} \\ & \text{e Definição 26} \\ \leftrightarrow P \to aY \in R \text{ e } X \in \hat{\delta}(\{Y\}, w) & \text{pela hipótese de indução} \\ \leftrightarrow Y \in \delta(P, a) \text{ e } X \in \hat{\delta}(\{Y\}, w) & \text{pela definição de } \delta \\ \leftrightarrow X \in \hat{\delta}(\{P\}, aw) & \text{pela definição de } \hat{\delta}. \end{array}$$

Finalmente, para provar que $L(M) = L(G)$, será mostrado que $P \stackrel{*}{\Rightarrow} w \leftrightarrow \hat{\delta}(\{P\}, w) \cap F \neq \emptyset$. São considerados dois casos:

*Caso 1.* $w = \lambda$.

$$\begin{array}{ll} P \stackrel{*}{\Rightarrow} \lambda \leftrightarrow P \to \lambda \in R & \text{pela definição de } \stackrel{*}{\Rightarrow} \text{ e Definição 26} \\ \leftrightarrow P \in F \text{ e } P \neq Z & \text{pela definição de } F \\ \leftrightarrow \hat{\delta}(\{P\}, \lambda) \in F & \text{pela definição de } \hat{\delta} \\ \leftrightarrow \hat{\delta}(\{P\}, \lambda) \cap F \neq \emptyset & \text{pois } |\hat{\delta}(\{P\}, \lambda)| = |\{P\}| = 1. \end{array}$$

*Caso 2.* $w = xa$.

$$\begin{array}{ll} P \stackrel{*}{\Rightarrow} xa \leftrightarrow [P \stackrel{*}{\Rightarrow} xaX \text{ e } X \to \lambda \in R] \text{ ou} & \\ \quad [P \stackrel{*}{\Rightarrow} xX \text{ e } X \to a \in R] & \text{pela def. de } \stackrel{*}{\Rightarrow} \text{ e Definição 26} \\ \leftrightarrow [X \in \hat{\delta}(\{P\}, xa) \text{ e } X \to \lambda \in R] \text{ ou} & \\ \quad [P \stackrel{*}{\Rightarrow} xX \text{ e } X \to a \in R] & \text{por 2.4} \\ \leftrightarrow [X \in \hat{\delta}(\{P\}, xa) \text{ e } X \to \lambda \in R] \text{ ou} & \\ \quad [X \in \hat{\delta}(\{P\}, x) \text{ e } X \to a \in R] & \text{por 2.4} \\ \leftrightarrow [X \in \hat{\delta}(\{P\}, xa) \text{ e } X \to \lambda \in R] \text{ ou} & \\ \quad [X \in \hat{\delta}(\{P\}, x) \text{ e } Z \in \delta(X, a)] & \text{pela definição de } \delta \\ \leftrightarrow [X \in \hat{\delta}(\{P\}, xa) \text{ e } X \to \lambda \in R] \text{ ou } Z \in \hat{\delta}(\{P\}, xa) & \\ & \text{pela definição de } \hat{\delta} \\ \leftrightarrow \hat{\delta}(\{P\}, xa) \cap F \neq \emptyset & \text{pela Definição de } F. \end{array}$$

□

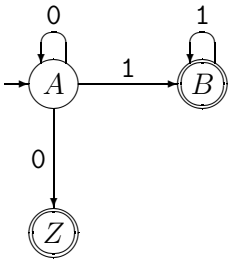



Figura 2.35: AFN a partir de GR.

Pela construção apresentada na prova do Teorema 14, fica evidente que uma GR pode ser vista quase que como uma forma linear de se apresentar um AFN: cada regra da forma $X \to aY$ corresponde a uma transição $Y \in \delta(X, a)$, cada regra da forma $X \to a$ corresponde a uma transição $Z \in \delta(X, a)$, onde $Z$ é estado final no AFN, e cada regra $X \to \lambda$ corresponde a um estado final no AFN.

**Exemplo 88** Seja a GR $G = (\{A, B\}, \{0, 1\}, R, A)$, onde $R$ é dado por:

$A \to 0A|1B|0$

$B \to 1B|\lambda$

Um diagrama de estados para um AFN que reconhece a linguagem $L(G) = 0^*(0 + 1^+)$, construído utilizando o método visto na prova do Teorema 14 está mostrado na Figura 2.35. □

Observe a correpondência entre regras e transições:

| *Regras* | *Transições* | *Observação* |
|---|---|---|
| $A \to 0A$ | $A \in \delta(A, 0)$ | |
| $A \to 1B$ | $B \in \delta(A, 1)$ | |
| $A \to 0$ | $Z \in \delta(A, 0)$ | $Z$ é estado final |
| $B \to 1B$ | $B \in \delta(B, 1)$ | |
| $B \to \lambda$ | | $B$ é estado final. |

A seguir, mostra-se como construir uma GR que gera a linguagem reconhecida por um AFN.

**Teorema 15** *Toda linguagem regular é gerada por gramática regular.*

*Prova*

Seja um AFN $M = (E, \Sigma, \delta, \{i\}, F)$[9]. Uma GR que gera $L(M)$ seria $G = (E, \Sigma, R, i)$, onde:

$$R = \{e \to ae' \mid e' \in \delta(e, a)\} \cup \{e \to \lambda \mid e \in F\}.$$

[9]Basta considerar AFN's com um único estado inicial; afinal, qualquer linguagem regular é reconhecida por um AFD, e um AFD é um AFN com um único estado inicial.

Será utilizado o seguinte lema, que pode ser provado por indução sobre $|w|$ (veja o exercício 5 no final da seção):

$$i \stackrel{*}{\Rightarrow} we \text{ se, e somente se, } e \in \hat{\delta}(\{i\}, w) \text{ para todo } e \in E \text{ e } w \in \Sigma^* \tag{2.5}$$

Para mostrar que $L(M) = L(G)$, será provado que:

$$i \stackrel{*}{\Rightarrow} w \text{ se, e somente se, } \hat{\delta}(\{i\}, w) \cap F \neq \emptyset \text{ para todo } w \in \Sigma^*:$$

$$\begin{aligned}
i \stackrel{*}{\Rightarrow} w &\leftrightarrow i \stackrel{*}{\Rightarrow} we \text{ e } e \to \lambda \in R && \text{pela definição de } R \\
&\leftrightarrow e \in \hat{\delta}(\{i\}, w) \text{ e } e \to \lambda \in R && \text{por 2.5} \\
&\leftrightarrow e \in \hat{\delta}(\{i\}, w) \text{ e } e \in F && \text{pela definição de } R \\
&\leftrightarrow \hat{\delta}(\{i\}, w) \cap F \neq \emptyset && \text{pela definição de interseção.}
\end{aligned}$$

□

A prova do Teorema 15 mostra que pode-se construir uma GR, que reconhece a linguagem reconhecida por um AFN, que só tenha regras das formas $X \to aY$ e $X \to \lambda$. Combinado com o resultado do Teorema 14, isto mostra que para toda linguagem regular existe uma GR equivalente sem regras da forma $X \to a$.

**Exemplo 89** Uma GR, construída de acordo com a prova do Teorema 15, que gera a linguagem aceita pelo AFN cujo diagrama de estados está mostrado na Figura 2.35, é $(\{A, B, Z\}, \{\mathtt{0}, \mathtt{1}\}, R, A)$, onde $R$ é dado por:

$A \to \mathtt{0}A|\mathtt{0}Z|\mathtt{1}B$

$B \to \mathtt{1}B|\lambda$

$Z \to \lambda$ □

# Exercícios

1. Obtenha GR's para as seguintes linguagens sobre {0,1}:
   (a) $\emptyset$.
   (b) $\{\lambda\}$.
   (c) O conjunto das palavras com tamanho múltiplo de 3.
   (d) O conjunto das palavras com um número par de 0's e um número par de 1's.
   (e) O conjunto das palavras em que cada 0 é seguido imediatamente de, no mínimo, dois 1's.
   (f) O conjunto das palavras em que o ante-penúltiplo símbolo é 1.
2. Obtenha um AFN que reconheça a linguagem gerada pela GR do exemplo 87, página 120, utilizando o método da prova do Teorema 14.
3. Seja a GR $G = (\{P, A, B\}, \{\mathtt{a}, \mathtt{b}\}, R, P)$, onde $R$ consta das regras:

$P \to \mathtt{a}P|\mathtt{b}P|\mathtt{a}A$

$A \to \mathtt{a}|\mathtt{b}B$

$B \to \mathtt{b}A$

Construa, a partir de $G$, um AFN que aceite $L(G)$.

4. Seja a linguagem $L = \{w \in \{\mathtt{0},\mathtt{1}\}^* \mid w$ tem números par de 0's e ímpar de 1's$\}$. Obtenha um AFD para $L$. Utilizando o método to Teorema 15, obtenha uma GR que gere $L$.

5. Complete a prova do Teorema 15, provando o lema por indução sobre $|w|$:

$$i \stackrel{*}{\Rightarrow} we \text{ se, e somente se, } e \in \hat{\delta}(\{i\}, w) \text{ para todo } e \in E \text{ e } w \in \Sigma^*.$$

6. Alguns autores definem uma gramática regular como sendo uma gramática em que as regras são da forma $X \to xY$ e $X \to x$, onde $X$ e $Y$ são variáveis e $x$ é uma palavra de terminais. Mostre como obter uma GR equivalente a uma gramática com regras desta forma.

7. Responda às seguintes perguntas, justificando suas respostas:

   (a) Quantas regras, no mínimo, são necessárias para gerar uma linguagem infinita?

   (b) Dada uma expressão regular, é possível obter uma GR que gera a linguagem denotada por ela?

   (c) É possível gerar qualquer linguagem finita por meio de GR?

   (d) Dada uma GR é possível obter uma outra GR equivalente em que a geração de qualquer palavra é determinística (em qualquer passo da derivação existe uma única regra aplicável)?

8. Mostre que os seguintes problemas são decidíveis, onde $G_1$ e $G_2$ são GR's quaisquer:

   (a) $L(G_1) = \emptyset$?

   (b) $L(G_1) \cap L(G_2) = \emptyset$?

## 2.8 Linguagens Regulares: Conclusão

Neste capítulo foram apresentados os principais conceitos associados à classe das linguagens regulares, também chamadas de conjuntos regulares ou linguagens de estado-finito.

Foram analisados vários tipos de reconhecedores de linguagens: AFD's, AFN's, AFN$\lambda$'s e AFNE's. Mostrou-se que cada um deles pode ser mais conveniente que os outros em situações específicas, e também que eles têm o mesmo poder computacional. Ou seja, se um deles reconhece uma linguagem, então existe reconhecedor dos outros tipos para a mesma linguagem; e, mais do que isto, existem algoritmos para obter reconhecedor de um tipo a partir de reconhecedor de qualquer um dos outros tipos. A Figura 2.23, na página 94, sintetiza a situação. Ressalte-se que para certos tipos de máquina a introdução de não determinismo aumenta o poder computacional.

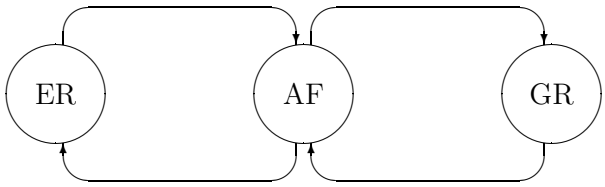



Figura 2.36: Transformações entre formalismos.

Mostrou-se que a classe das expressões regulares denota exatamente a classe das linguagens que podem ser reconhecidas por AF's. Mostrou-se também que existem algoritmos para obter um AF a partir de uma ER, e vice-versa, para obter uma ER a partir de um AF.

Mostrou-se, ainda, que a classe das gramáticas regulares gera exatamente a classe das linguagens que podem ser reconhecidas por AF's. Assim, as gramáticas regulares geram exatamente a classe das linguagens regulares. Mostrou-se também que existem algoritmos para obter um AF a partir de uma GR, e vice-versa, para obter uma GR a partir de um AF.

Combinando os resultados mencionados nestes dois últimos parágrafos, ER's, GR's e AF's são três formalismos alternativos para especificação de linguagens da mesma classe: a classe das linguagens regulares. A Figura 2.36 sintetiza o conjunto das transformações vistas neste capítulo. Combinando-se os algoritmos relativos a tais transformações, obtém-se facilmente algoritmos para obter uma ER que denota a linguagem gerada por uma GR, e para obter uma GR que gera a linguagem denotada por uma ER.

Uma outra característica importante das linguagens regulares, é que elas compõem uma classe fechada com relação a várias operações, como: união, interseção, complemento, concatenação e fecho de Kleene. Nos exercícios ao final do capítulo 1 são abordadas outras operações para as quais a classe das linguagens regulares é fechada, como: reverso, homorfismo, substituição e quociente. Tais propriedades de fechamento constituem um arsenal importante para mostrar que determinadas linguagens são ou não regulares (neste último caso, usando um raciocínio por contradição). Por exemplo, na seção 2.3.3 mostrou-se que AFN$\lambda$'s reconhecem linguagens regulares. Uma alternativa interessante, e mais simples, é utilizar a propriedade de fechamento com relação a homomorfismo (veja Exercício 18, da Seção 2.9, na página 131), como mostra o exemplo a seguir.

**Exemplo 90** Será mostrado que AFN$\lambda$'s reconhecem linguagens regulares. Para isto, seja um AFN$\lambda$ arbitrário $M = (E, \Sigma, \delta, I, F)$. Seja um símbolo $\pi \notin \Sigma$. Seja o seguinte AFN $M'$, idêntico a $M$, com exceção do fato de que cada transição sob $\lambda$ é substituída por uma transição sob $\pi$: $M' = (E, \Sigma \cup \{\pi\}, \delta', I, F)$, onde:

- $\delta'(e, a) = \delta(e, a)$ para todo $e \in E$ e $a \in \Sigma$; e
- $\delta'(e, \pi) = \delta(e, \lambda)$ para todo $e \in E$.

Seja o homomorfismo $h : (\Sigma \cup \{\pi\})^* \to \Sigma^*$, definido por:

- $h(a) = a$ para todo $a \in \Sigma$; e

- $h(\pi) = \lambda$.

Tem-se que $L(M) = h(L(M'))$. Como $L(M')$ é linguagem regular (existe um AFN que a reconhece: $M'$), e o homomorfismo de uma linguagem regular é uma linguagem regular (Exercício 18, Seção 2.9, página 131), segue-se que $L(M)$ é uma linguagem regular. □

As propriedades de fechamento constituem também ferramental adicional que pode ser usado na obtenção de AF's, ER's e/ou GR's para uma linguagem, por meio de decomposição. Por exemplo, às vezes pode ser mais fácil obter um AF para $L(M_1) \cap L(M_2)$ a partir dos AF's $M_1$ e $M_2$, do que obter diretamente um AF para $L(M_1) \cap L(M_2)$.

No Teorema 6, mostrou-se que os problemas de determinar se $L(M)$ é vazia ou se $L(M)$ é finita, para um AFD $M$ quaisquer, são decidíveis. Dadas as transformações ilustradas nas Figuras 2.23 e 2.36, páginas 94 e 125, tem-se que os problemas de determinar se $L(\mathcal{F})$ é vazia ou se $L(\mathcal{F})$ é finita são decidíveis também se $\mathcal{F}$ for AFN, AFN-$\lambda$, AFNE, ER ou GR.

É importante ressaltar também a diversidade de aplicações para as linguagens regulares, considerando os diversos formalismos alternativos, como: análise léxica de linguagens de programação (e outras, como linguagens para interação com o usuário em aplicações específicas), edição de texto, comandos de sistemas operacionais, modelagem de protocolos de comunicação, etc.

No próximo capítulo será abordado um tipo de máquina mais poderoso do que AF's, sendo que o acréscimo de poder computacional é devido a um dispositivo de memória adicional: uma pilha. Sua importância advém do fato de que tem aplicações importantes, principalmente na área de Linguagens de Programação. No entanto, é interessante notar que existem muitos outros tipos de máquinas, menos poderosas do que AF's, com o mesmo poder computacional ou mais poderosas. Para preparar o terreno para os modelos a serem vistos nos próximos capítulos e, ao mesmo tempo, dar uma idéia de possíveis alternativas a AF's apresenta-se a seguir uma visão de AF's como compostos de alguns "dispositivos" similares ao de um computador.

Um AFD pode ser visto como uma máquina (veja Figura 2.37) que opera com uma fita que pode apenas ser lida, e cujo cabeçote de leitura se movimenta apenas para a direita. Tal fita é dividida em células que comportam apenas um símbolo cada uma. Além da fita, a máquina possui um registrador para conter o estado atual, um conjunto de instruções, que nada mais é do que a função de transição do AFD, e uma unidade de controle (estas duas últimas estão representadas na Figura 2.37 juntas). No início, o registrador contém o estado inicial do AFD, a fita contém a palavra de entrada a partir da sua primeira célula, e o cabeçote é posicionado na primeira célula da fita. Enquanto o cabeçote não se posicionar na célula seguinte àquela que contém o último símbolo da palavra de entrada, a unidade de controle repete a seguinte seqüência:

1) Coloca no registrador o estado $\delta(e, a)$, onde:
   - $e$ é o estado atual contido no registrador; e
   - $a$ é o símbolo sob o cabeçote.
2) Avança o cabeçote para a próxima célula.

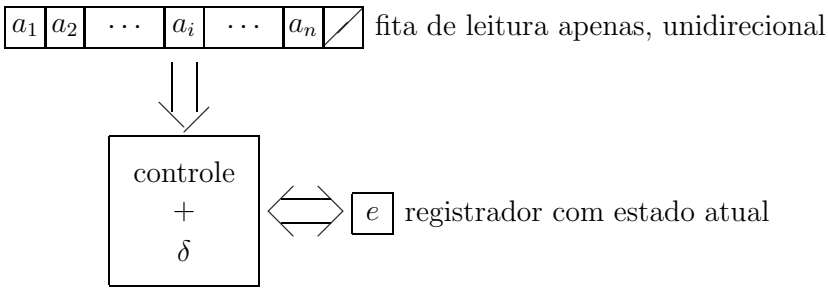



Figura 2.37: Arquitetura de um AFD.

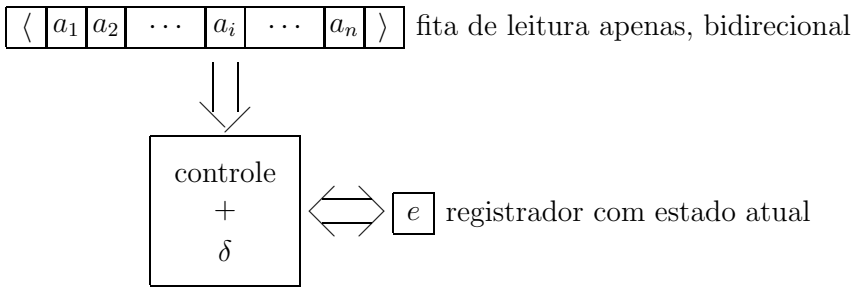



Figura 2.38: AFD com fita bidirecional.

A seguir, na Figura 2.38, descreve-se um AFD $M = (E, \Sigma, \delta, i, F)$ com uma fita bidirecional, isto é, onde o cabeçote pode ser movimentado para a direita e para a esquerda. A função de transição é uma função parcial da forma $\delta : E \times \Sigma \to E \times \{\mathtt{e}, \mathtt{d}\}$. Uma transição $\delta(e, a) = [e', \mathtt{e}]$, por exemplo, significaria: estando no estado $e$, com o cabeçote lendo o símbolo $a$, faz-se a transição para o estado $e'$ e movimenta-se o cabeçote para a esquerda. Para evitar que o cabeçote "caia" da fita, supõe-se que existe uma célula à esquerda e outra à direita da palavra de entrada, com dois símbolos especiais, $\langle, \rangle \notin \Sigma$; para cada estado $e$ só pode haver uma transição de $e$ sob $\langle$, que deve ser da forma $\delta(e, \langle) = [e', \mathtt{d}]$, e uma transição de $e$ sob $\rangle$, que deve ser da forma $\delta(e, \rangle) = [e', \mathtt{e}]$. Observe que neste modelo o AFD não consome a palavra de entrada. A única maneira da máquina parar é atingir um certo estado $e$ com o cabeçote posicionado em um símbolo $a$, para os quais $\delta(e, a)$ é indefinido. A linguagem reconhecida por $M$ seria o conjunto de toda palavra $w \in \Sigma^*$ para a qual $M$ pára em um estado final. Observe, então, que se a máquina não parar para uma certa palavra $w$, por definição $w$ não é aceita.

Pode-se obter um AFD padrão equivalente a um AFD com fita bidirecional, e vice-versa, estabelecendo-se assim que os AFD's com fitas bidirecionais reconhecem exatamente as linguagens regulares. Mesmo que se introduza não determinismo a um autômato com fita bidirecional, o mesmo continua reconhecendo apenas linguagens regulares.

Vários outros "incrementos" a AFD's podem ser imaginados, sendo que alguns deles não aumentam o poder computacional, como os AFD's com fita bidirecional determinísticos ou não, e outros aumentam, como os que serão vistos nos próximos capítulos.

## 2.9 Exercícios

1. Construa AFD's para as linguagens:

   (a) $\{u\mathtt{a}v\mathtt{b}x\mathtt{c}y \mid u, v, x, y \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*\}$.

   (b) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid$ $w$ começa com $\mathtt{a}$ e tem tamanho par$\}$.

   (c) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid$ $w$ nunca tem mais de dois $\mathtt{a}$'s consecutivos$\}$.

   (d) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid$ $w$ tem número ímpar de $\mathtt{ab}$'s$\}$.

   (e) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid$ $|w| \geq 2$ e os $\mathtt{a}$'s (se houver) precedem os $\mathtt{b}$'s (se houver)$\}$.

   (f) $\{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}^* \mid$ os $\mathtt{a}$'s (se houver) precedem os $\mathtt{b}$'s (se houver) e os $\mathtt{c}$'s (se houver) precedem os $\mathtt{d}$'s (se houver)$\}$.

   (g) $\{x\mathtt{ba}^n \mid x \in \{\mathtt{a}, \mathtt{b}\}^*, n \geq 0$ e $x$ tem um número par de $\mathtt{a}$'s$\}$.

   (h) $\{x\mathtt{a}^m\mathtt{ba}^n \mid x \in \{\mathtt{a}, \mathtt{b}\}^*, m + n$ é par e $x$ não termina em $\mathtt{a}\}$.

   (i) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid$ toda subpalavra de $w$ de tamanho 3 tem $\mathtt{a}$'s e $\mathtt{b}$'s$\}$.

   (j) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid$ $w$ tem no máximo uma ocorrência de $\mathtt{aa}$ e no máximo uma ocorrência de $\mathtt{bb}\}$.

2. Construa AFN's para as linguagens:

   (a) $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ $|w| \geq 4$ e o segundo e o penúltimo símbolos são ambos $\mathtt{1}\}$.

   (b) $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ $\mathtt{00}$ não aparece nos 4 últimos símbolos de $w\}$.

   (c) $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ entre dois $\mathtt{1}$'s de $w$ há sempre um número par de $\mathtt{0}$'s, exceto nos 4 últimos símbolos$\}$.

   (d) $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ $w$ tem uma subpalavra constituída de dois $\mathtt{1}$'s separados por um número par de símbolos$\}$.

   (e) $\{x\mathtt{0}^{3n} \mid x \in \{\mathtt{0}, \mathtt{1}\}^*, \eta(x) \bmod 3 = 1$ e $n \geq 0\}$, onde $\eta(x)$ é o número representado por $x$ na base 2.

3. Construa AFD's para as linguagens:

   (a) $L_1 = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ $|w|$ é divisível por 3$\}$.

   (b) $L_2 = \{\mathtt{0}w\mathtt{0} \mid w \in \{\mathtt{0}, \mathtt{1}\}^*\}$.

   Construa um AFD para $L_1 \cap L_2$ utilizando a técnica vista no Teorema 4, página 76.

4. Mostre que o conjunto de todas as mensagens enviadas pela Internet, desde que ela foi estabelecida até o exato momento em que você acabou de ler esta sentença, é uma linguagem regular.

5. Defina um estado $e \in E$ de um AFN $M = (E, \Sigma, \delta, I, F)$ como sendo *inútil* se:

   - $e \notin \hat{\delta}(i, w)$ para todo $i \in I$ e $w \in \Sigma^*$ (ou seja, $e$ não é alcançável a partir de nenhum estado inicial de $M$); ou

- $\hat{\delta}(e, w) \cap F = \emptyset$ para todo $w \in \Sigma^*$ (ou seja, nenhum estado final de $M$ é alcançável a partir de $e$).

Seja $M'$ o AFN resultante da eliminação de todos os estados inúteis de $M$. Mostre que:

(a) $L(M') = L(M)$.

(b) $L(M)$ é finita se, e somente se, o diagrama de estados de $M'$ não tem ciclos.

6. Que linguagens são reconhecidas pelos seguintes AF's?

   (a) AFD $M_1 = (E, \Sigma, \delta, i, E)$.

   (b) AFN $M_2 = (E, \Sigma, \delta, i, E)$ em que $\delta(i, a) = \emptyset$ para todo $a \in \Sigma$.

7. Um conjunto $S \subseteq \mathbf{N}$ é uma progressão aritmética, se existem dois números naturais $n$ e $r$ tais que $S = \{n, n + r, n + 2r, \ldots\}$. Seja $L \subseteq \{\mathtt{a}\}^*$, e seja $T_L = \{|w| \mid w \in L\}$.

   (a) Mostre que se $T_L$ é uma progressão aritmética, $L$ é uma linguagem regular.

   (b) Mostre que se $L$ é uma linguagem regular, $T_L$ é a união de um número finito de progressões aritméticas.

8. Mostre que sim ou que não:

   (a) Para qualquer linguagem $L$ (inclusive aquelas que não são regulares), existem linguagens regulares $R_1$ e $R_2$ tais que $R_1 \subseteq L \subseteq R_2$.

   (b) Todos os subconjuntos de uma linguagem regular são também linguagens regulares.

   (c) Há linguagens regulares que têm como subconjuntos linguagens que não são regulares.

   (d) A união de duas linguagens que não são regulares pode ser ou não uma linguagem regular.

   (e) A interseção de duas linguagens que não são regulares pode ser ou não uma linguagem regular.

   (f) O complemento de uma linguagem que não é regular pode ser ou não uma linguagem regular.

9. Prove que os seguintes conjuntos não são linguagens regulares:

   (a) $\{\mathtt{0}^n\mathtt{1}^{n+10} \mid n \geq 0\}$.

   (b) $\{\mathtt{0}^n y \mid y \in \{\mathtt{0}, \mathtt{1}\}^* \text{ e } |y| \leq n\}$.

   (c) $\{\mathtt{0}^m\mathtt{1}^n \mid m \neq n\}$.

   (d) $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^{m+n} \mid m, n > 0\}$.

   (e) $\{\mathtt{a}^n\mathtt{b}^{n^2} \mid n \geq 0\}$.

   (f) $\{\mathtt{a}^{n^3} \mid n \geq 0\}$.

   (g) $\{\mathtt{a}^m\mathtt{b}^n \mid n \leq m \leq 2n\}$.

(h) $\{xx \mid x \in \{\mathtt{a}, \mathtt{b}\}^*\}$.

(i) $\{x\overline{x} \mid x \in \{\mathtt{0}, \mathtt{1}\}^*\}$, onde $\overline{x}$ é obtido de $x$ substituindo-se $\mathtt{0}$ por $\mathtt{1}$ e $\mathtt{1}$ por $\mathtt{0}$. Exemplo: $\overline{\mathtt{011}} = \mathtt{100}$.

(j) $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w \neq w^R\}$.

(k) $\{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid$ o número de $\mathtt{a}$'s, $\mathtt{b}$'s e $\mathtt{c}$'s, em $w$, é o mesmo$\}$.

(l) $\{w \in \{0, 1\}^* \mid$ o número de $\mathtt{0}$'s em $w$ é um cubo perfeito$\}$.

(m) $\{\mathtt{0}^m\mathtt{1}^n \mid mdc(m, n) = 1\}$.

(n) $\{\mathtt{a}^k\mathtt{b}^m\mathtt{c}^n \mid k \neq m$ ou $m \neq n\}$.

(o) $\{\mathtt{0}^m\mathtt{1}^n\mathtt{0}^n \mid m, n > 0\}$.

10. Seja $\Sigma = \{[0, 0], [0, 1], [1, 0], [1, 1]\}$. Mostre que os seguintes conjuntos são ou não são linguagens regulares:

    (a) O conjunto das palavras $[a_1, b_1][a_2, b_2] \ldots [a_n, b_n]$ em que a seqüência $a_1a_2 \ldots a_n$ é o complemento de $b_1b_2 \ldots b_n$. Exemplo: $[0, 1][0, 1][1, 0]$ pertence à linguagem, pois 001 é o complemento de 110.

    (b) O conjunto das palavras $[a_1, b_1][a_2, b_2] \ldots [a_n, b_n]$ em que o número de $\mathtt{0}$'s de $a_1a_2 \ldots a_n$ é igual ao número de $\mathtt{0}$'s de $b_1b_2 \ldots b_n$.

    (c) O conjunto das palavras $[a_1, b_1][a_2, b_2] \ldots [a_n, b_n]$ tais que $a_1a_2 \ldots a_n = b_2b_3 \ldots b_nb_1$.

    (d) O conjunto das palavras $[a_1, b_1][a_2, b_2] \ldots [a_n, b_n]$ tais que $a_1a_2 \ldots a_n = b_nb_{n-1} \ldots b_1$.

11. Prove que para toda linguagem regular infinita $L$ existem $m, n \in \mathbf{N}$, $n > 0$, tais que para todo $k \geq 0$, $L$ contém alguma palavra $z$ tal que $|z| = m + kn$. Use este resultado para provar que $\{\mathtt{a}^n \mid n$ é primo$\}$ não é regular

12. O lema do bombeamento se baseia no bombeamento de uma subpalavra $v$ de um *prefixo* de tamanho $k$. Generalize o lema do bombeamento de tal forma que o bombeamento não seja necessariamente ancorado a um prefixo. (Isto irá facilitar provar, por exemplo, que $\{\mathtt{0}^m\mathtt{1}^n\mathtt{0}^n \mid m, n > 0\}$ não é regular.)

13. Seja $L$ uma linguagem regular e $M$ um AFD de $k$ estados que reconhece $L$. Se $z \notin L$ e $|z| \geq k$, então será percorrido um ciclo no diagrama de estados de $M$ durante o processamento de $z$... Enuncie um "lema do bombeamento baseado em palavras *não* pertencentes a uma linguagem regular". Para que serviria tal lema? Para provar que uma linguagem é regular? Não regular? Como?

14. Mostre que a classe das linguagens regulares é fechada sob as seguintes operações:

    (a) $pref(L) = \{x \mid xy \in L\}$ (os prefixos das palavras de $L$).

    (b) $suf(L) = \{y \mid xy \in L\}$ (os sufixos das palavras de $L$).

    (c) $rev(L) = \{w^R \mid w \in L\}$ (os reversos das palavras de $L$).

(d) $crev(L) = \{xy^R \mid x, y \in L\}$ (concatenação dos reversos das palavras de $L$ às palavras de $L$).

(e) $mpal(L) = \{w \mid ww^R \in L\}$ (as primeiras metades dos palíndromos de tamanho par de $L$).

(f) $dc(L) = \{xy \mid yx \in L\}$ (deslocamentos circulares das palavras de $L$).

(g) $pm(L) = \{x \mid xy \in L \text{ e } |x| = |y|\}$ (as primeiras metades das palavras de tamanho par).

(h) $\hat{\xi}(L_1, L_2) = \{z \in \xi(x, y) \mid x \in L_1 \text{ e } y \in L_2\}$, onde $\xi$ é definida recursivamente como segue:

(h.1) $\lambda \in \xi(\lambda, \lambda)$;

(h.2) $a\xi(y, w) \in \xi(ay, w)$ e $a\xi(w, y) \in \xi(w, ay)$.

(Todos os "embaralhamentos" das palavras de $L_1$ com as de $L_2$.)

15. Descreva como obter um AFD $M'$ equivalente a um AFD $M$, tal que não exista transição de qualquer estado (inclusive o estado inicial) para o estado inicial de $M'$.

16. Sejam $L_1, L_2, \ldots$ linguagens regulares e seja uma constante $n \geq 2$. Mostre que:

(a) $\bigcup_{2 \leq i \leq n} L_i$ é uma linguagem regular.

(b) $\bigcup_{i \geq 2} L_i$ pode não ser uma linguagem regular.

17. Mostre que $L \cup \{\lambda\}$ é linguagem regular se, e somente se, $L - \{\lambda\}$ é linguagem regular.

18. Um *homomorfismo* de um alfabeto $\Sigma$ para um alfabeto $\Delta$ é uma função de $\Sigma$ para $\Delta^*$ (ou seja, que mapeia cada símbolo de $\Sigma$ para uma palavra de $\Delta^*$). Um homomorfismo $h : \Sigma \to \Delta^*$ é estendido para palavras fazendo-se $h(ay) = h(a)h(y)$, ou seja, a aplicação de um homomorfismo a uma palavra $a_1 a_2 \ldots, a_n$ resulta na concatenação das palavras $h(a_1)$, $h(a_2)$, $\ldots$, $h(a_n)$, nesta ordem. A extensão de um homomorfismo $h$ para uma linguagem $L$ é dada por:

$$h(L) = \{h(w) \mid w \in L\}.$$

A imagem homomórfica inversa de $L$ é dada por:

$$h^{-1}(L) = \{w \in \Sigma^* \mid h(w) \in L\}.$$

Mostre que as linguagens regulares são fechadas sob homomorfismo e sob imagem homomórfica inversa.

19. Seja um homomorfismo $h$ como definido no exercício anterior. Mostre que nem sempre $L \subseteq \Sigma^*$ é uma linguagem regular quando $h(L)$ é uma linguagem regular.

20. Uma *substituição* de um alfabeto $\Sigma$ para um alfabeto $\Delta$ é uma função de $\Sigma$ para $\mathcal{P}(\Delta^*) - \{\emptyset\}$ que mapeia cada símbolo de $\Sigma$ para um conjunto de palavras de $\Delta^*$ que não é vazio e que é uma linguagem regular. Uma substituição $s : \Sigma \to \mathcal{P}(\Delta^*) - \{\emptyset\}$ é estendida para palavras fazendo-se $s(ay) = s(a)s(y)$, ou seja, a aplicação de

uma substituição a uma palavra $a_1 a_2 \ldots, a_n$ resulta na concatenação dos conjuntos $s(a_1)$, $s(a_2)$, $\ldots$, $s(a_n)$, nesta ordem. A extensão de uma substituição $s$ para uma linguagem $L$ é dada por:

$$s(L) = \bigcup_{w \in L} s(w).$$

Mostre que as linguagens regulares são fechadas sob substituição.

21. O *quociente* de duas linguagens $L_1$ e $L_2$, denotado por $L_1/L_2$, é a linguagem

$$\{x \mid xy \in L_1 \text{ para algum } y \in L_2\}.$$

Mostre que as linguagens regulares são fechadas sob quociente.

22. Sejam $L$ uma linguagem não regular, e seja $F$ uma linguagem finita. Mostre que:

    (a) $L \cup F$ não é linguagem regular.

    (b) $L - F$ não é linguagem regular.

    Mostre que ambos os resultados não são verdadeiros se for suposto que $F$ é linguagem regular, mas não finita.

23. Sejam $L_1$ e $L_2$ duas linguagens. Mostre que sim ou que não:

    (a) se $L_1 \cup L_2$ é uma linguagem regular então $L_1$ é uma linguagem regular.

    (b) se $L_1 L_2$ é uma linguagem regular então $L_1$ é uma linguagem regular.

    (c) se $L_1^*$ é uma linguagem regular então $L_1$ é uma linguagem regular.

    (d) se $L_1$ é uma linguagem regular então $\{w \mid w$ é uma subpalavra de $L_1\}$ é uma linguagem regular.

24. Seja um AFD $M$, qualquer, e seja $n$ o número de estados de $M$. Utilizando o lema do bombeamento, mostre que:

    (a) $L(M) = \emptyset$ se, e somente se, $M$ não reconhece palavra de tamanho menor que $n$.

    (b) $L(M)$ é finita se, e somente se, $M$ não reconhece palavra de tamanho maior ou igual a $n$ e menor que $2n$.

    Explique como estes resultados podem ser usados para provar o Teorema 6 (página 80).

25. Mostre que existem procedimentos de decisão para determinar, dadas duas ER's quaisquer $r_1$ e $r_2$, se:

    (a) $L(r_1) = L(r_2)$.

    (b) $L(r_1) \subseteq L(r_2)$.

    (c) $L(r_1) = \Sigma^*$.

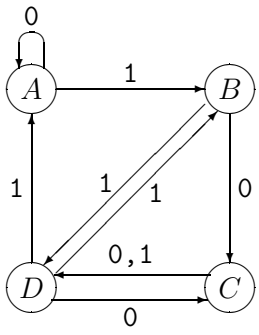



Figura 2.39: Diagrama de estados para Exercício 26.

26. Seja o AFN cujo diagrama de estados está mostrado na Figura 2.39 (não estão marcados o estado inicial, nem os estados finais, por serem desnecessários no presente exercício). Encontre uma palavra $w \in \{0,1\}^*$ e um estado $e \in \{A,B,C,D\}$, tais que:

$$e \in \hat{\delta}(d,w) \text{ para todo } d \in \{A,B,C,D\}.$$

A palavra $w$ deverá ser a *menor* possível.

Construa um algoritmo (em alto nivel, como os deste livro) que, recebendo um AFN $M$ como entrada, determine um par $[w,e]$ tal que $\in \hat{\delta}(d,w)$ para todo estado $d$ de $M$, se houver. Se houver algum par, o algoritmo deverá determinar um em que $w$ seja a menor possível.

27. Modifique a máquina do exemplo da Seção 2.1.3 (Figura 2.3, página 61), de forma que a saída seja a seqüência dos andares percorridos pelo elevador. Por exemplo: 121232, etc.

28. Modifique a máquina do exemplo da Seção 2.1.3 (Figura 2.3, página 61), de forma que a saída seja a seqüência dos andares percorridos pelo elevador, intercalados pela informação de que o elevador está subindo ou descendo. Por exemplo: $1 \uparrow 2 \downarrow 1 \uparrow 2 \uparrow 3 \downarrow 2$, etc.

29. Suponha uma extensão de máquina de Moore que contenha, além das entidades normais, um conjunto de estados finais. Com isto, uma palavra de entrada, além de ter uma saída correspondente, será aceita ou não. Mostre como simular este tipo de máquina utilizando uma máquina de Moore normal.

30. Se a cada transição de uma máquina de Mealy for associada uma palavra ao invés de um símbolo, seu poder computacional diminue ou não? Explique.

31. Determine expressões regulares e gramáticas regulares para as seguintes linguagens sob $\{0,1\}^*$:

    (a) Conjunto das palavras em que 0's só podem ocorrer nas posições pares.

    (b) Conjunto das palavras que não contêm 000.

(c) Conjunto das palavras em que cada subpalavra de tamanho 4 contém pelo menos 3 `1`'s.

(d) Conjunto das palavras que não contêm `00` nos últimos 4 símbolos.

(e) Conjunto das palavras que não contêm `00`, a não ser nos últimos 4 símbolos (se tiver).

32. Determine uma ER para o AFD cujo diagrama de estados está mostrado na Figura 2.1, página 58.

33. Sejam as seguintes ER's:

- $r_1 = (a + b)^*(ab + ba)(a + b)^*$;
- $r_2 = ab^*$;
- $r_3 = a(b^*ab^*a)^*$;
- $r_4 = (aa + bb + (ab + ba)(aa + bb)^*(ab + ba))^*$.

Encontre ER's para:

(a) $\overline{L(r_1)}$.
(b) $\overline{L(r_2)}$.
(c) $\overline{L(r_3)}$.
(d) $\overline{L(r_4)}$.
(e) $L(r_1) \cap L(r_4)$.
(f) $L(r_1) - L(r_4)$.

34. A prova do Teorema 12 mostrou como obter um AFD que reconhece o conjunto denotado por uma ER. Aquela forma de obtenção decorre de teoremas anteriores. Mostre como obter um AFN$\lambda$ que reconheça o conjunto denotado por uma ER usando raciocínio análogo ao utilizado no Teorema 12, ou seja: (a) mostre AFN$\lambda$'s para $\emptyset$, $\{\lambda\}$ e $\{a\}$ para $a \in \Sigma$; e (b) mostre como obter AFN$\lambda$'s para $L(M_1) \cup L(M_2)$, $L(M_1)L(M_2)$ e $L(M_1)^*$ a partir de AFN$\lambda$'s quaisquer $M_1$ e $M_2$. Observe as seguintes restrições:

- todo AFN$\lambda$ deverá ter um único estado inicial e um único estado final, diferentes entre si;
- não deve haver transições de nenhum estado para o estado inicial;
- não deve haver transições do estado final para nenhum outro estado.

Estas restrições são necessárias? Em que elas ajudam ou atrapalham?

35. Mostre como obter um AFN (sem transições $\lambda$) que reconheça o conjunto denotado por uma ER. Preocupe-se em obter um mínimo de estados e transições. Sugestão: comece substituindo cada símbolo do alfabeto que esteja repetido na ER por um símbolo novo, de forma que a ER não tenha símbolos do alfabeto repetidos; em seguida, mostre como obter um AFN para esta nova ER; finalmente, desfaça as substituições, obtendo o AFN final.

36. Considere que as transições especificadas em um diagrama ER sobre $\Sigma$ representam uma função $\delta : E \times R \to \mathcal{P}(E)$, onde $E$ é o conjunto de estados e $R$ é o conjunto (obviamente finito) das ER's sobre $\Sigma$ que ocorrem no diagrama. Defina uma função $\hat{\delta} : \mathcal{P}(E) \times \Sigma^* \to \mathcal{P}(E)$ tal que $\hat{\delta}(X, w)$ seja o conjunto dos estados alcançáveis a partir dos estados de $X$, consumindo-se a palavra $w$. Para facilitar a tarefa, defina antes a função auxiliar $f\lambda : E \to E$ tal que $f\lambda(X)$ seja o conjunto dos estados alcançáveis a partir dos estados de $X$, sem consumo de símbolos. (Use a notação $L(r)$ significando "o conjunto denotado pela ER $r$".)

37. Defina a linguagem especificada por um diagrama ER utilizando a função $\hat{\delta}$ do exercício anterior.

38. Seja uma *expressão regular estendida* sobre $\Sigma$ uma expressão regular sobre $\Sigma$ que:

    - além das expressões básicas para denotar $\{a\}$, $\emptyset$ e $\lambda$, contém expressões básicas para denotar $\Sigma^*$ e para denotar $\Sigma$; e
    - além dos operadores de união, concatenação e fecho de Kleene, tem operadores também para interseção e para complementação.

    Mostre como obter uma ER que denote a mesma linguagem que uma ER estendida.

39. GR's são também denominadas *gramáticas lineares à direita*. Uma gramática é dita *linear à esquerda* se ela só contém regras das formas $X \to Ya$, $X \to a$ e $X \to \lambda$, onde $X$ e $Y$ são variáveis e $a$ é terminal. Mostre que a classe das linguagens geradas por gramáticas lineares à esquerda é a classe das linguagens regulares.

40. Suponha que as gramáticas deste exercício só tenham regras das formas $X \to xY$ e $X \to x$, onde $X$ e $Y$ são variáveis e $x$ é uma palavra (veja Exercício 6 no final da Seção 2.7, página 124). Descreva métodos para:

    (a) Dado um AFNE, $M$, obtenha uma gramática que gere $L(M)$. Procure fazer corresponder a cada transição, exceto transições $\lambda$, uma regra.

    (b) Dada uma gramática, $G$, obtenha um AFNE que reconheça $L(G)$. Procure fazer corresponder a cada regra, exceto regras $\lambda$, uma transição.

41. Seja um tipo de gramática cujas regras sejam das formas $X \to aY$, $X \to Ya$, $X \to a$ e $X \to \lambda$, onde $X$ e $Y$ são variáveis e $a$ é terminal. Com este tipo de gramática é possível gerar linguagem que não seja regular? Justifique sua resposta.

## 2.10 Notas Bibliográficas

O artigo [MP43] é, em geral, considerado como o primeiro a abordar o que hoje se denomina máquina de estado-finito. Mas os autômatos finitos determinísticos, tal como os conhecemos hoje, embora com pequenas variações, surgiram independentemente em três artigos clássicos: [Huf54], [Mea55] e [Moo56]. Nestes dois últimos foram introduzidas as máquinas de Mealy e de Moore. Algoritmos para minimização de AFD's foram mostrados em [Huf54] e [Moo56]. O algoritmo de minimização mais eficiente pode ser encontrado em [Hop71].

A versão não determinística dos autômatos finitos foi primeiramente estudada por Rabin e Scott[RS59], onde eles mostram como obter um AFD equivalente a um AFN usando o método apresentado na Seção 2.3.2, página 88[10].

O lema do bombeamento para linguagens regulares foi descoberto por Bar-Hillel, Perles e Shamir[BPS61]. Alguns autores estenderam o lema para dar também uma condição suficiente para uma linguagem ser regular, como Jaffe[Jaf78], Ehrenfeucht, Parikh e Rozenberg[EPR81] e Stanat e Weiss[SW82].

Além de Rabin e Scott[RS59], vários outros autores apresentaram propriedades de fechamento para as linguagens regulares, como McNaughton e Yamada[MY60], Bar-Hillel et al.[BPS61], Ginsburg e Rose[GR63a] e Ginsburg[Gin66].

As expressões regulares foram criadas por Kleene[Kle56], que também formalizou a noção de autômato finito e demonstrou a equivalência entre os mesmos e expressões regulares.

[10]Denominado *subset construction* na literatura em geral.

# Capítulo 3

# Autômatos com Pilha

Apesar das inúmeras aplicações dos formalismos associados às linguagens regulares, existem aplicações que requerem linguagens mais sofisticadas e que, portanto, envolvem o uso de formalismos mais sofisticados. Por exemplo, são comuns as aplicações em que se deve escrever expressões aritméticas. As linguagens que contêm expressões aritméticas normalmente contêm todas as palavras da forma:

$$(^n t_1 + t_2) + t_3) \cdots + t_{n+1})$$

onde $n \geq 1$, cada $t_i$ é uma sub-expressão, e o número de (’s é igual ao número de )’s. Aplicando-se o lema do bombeamento, vê-se que tais linguagens não são regulares: tomando-se $z = (^k t_1 + t_2) + t_3) \cdots + t_{k+1})$, onde $k$ é a constante referida no lema, vê-se que para quaisquer $u$, $v$ e $w$ tais que $z = uvw$, $|uv| \leq k$ e $v \neq \lambda$,

$$uv^2w = (^{k+|v|} t_1 + t_2) + t_3) \cdots + t_{k+1}),$$

que tem mais (’s do que )’s.

Intuitivamente, um AF não pode reconhecer a linguagem acima porque não tem uma memória poderosa o suficiente para “lembrar” que leu $n$ ocorrências de certo símbolo, para $n$ *arbitrário*. A única maneira de ler uma quantidade arbitrária de determinado símbolo, em um AF, é por meio de um ciclo. E, neste caso, não há como contar o número de símbolos lidos.

Neste capítulo, será apresentada uma extensão dos AF’s, os denominados *autômatos com pilha*[1], de grande importância, visto que constituem uma base para a obtenção de reconhecedores para muitas linguagens que ocorrem na prática. Em particular, alguns compiladores de linguagens de programação utilizam alguma variante de autômato com pilha na fase de análise sintática.

Ao contrário dos AF’s, a versão não determinística deste tipo de autômato tem um poder de reconhecimento maior do que a determinística. Por outro lado, as linguagens que podem ser reconhecidas por autômatos com pilha determinísticos são especialmente importantes, já que admitem reconhecedores eficientes. Algumas construções que ocorrem em linguagens que podem ser reconhecidas por autômatos com pilha não determinísticos, mas que não podem ocorrer em linguagens que possam ser reconhecidas por autômatos

[1] *Pushdown automata.*

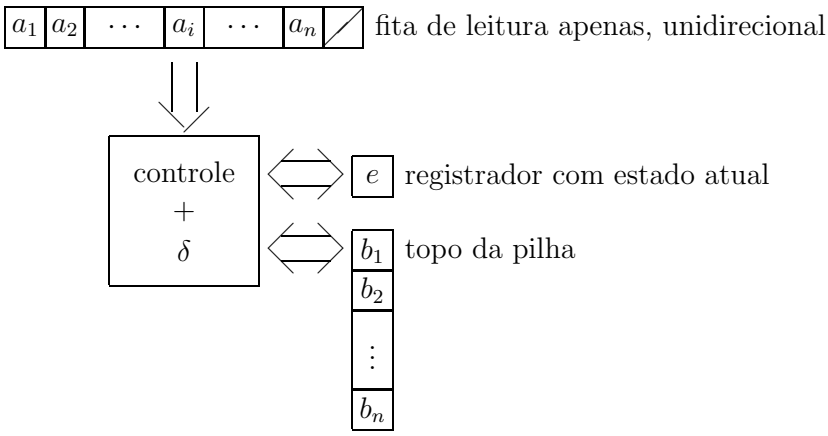



Figura 3.1: Arquitetura de um AP.

com pilha determinísticos, alertam para o fato de que se está transitando do campo da eficiência de reconhecimento para o da ineficiência.

Antes de apresentar as versões determinística e não determinística de autômatos com pilha nas Seções 3.2 e 3.3, será vista uma introdução informal de autômato com pilha na Seção 3.1. Depois, na Seção 3.4, serão estudadas as gramáticas livres do contexto, que são um formalismo de grande utilidade prática para a especificação de linguagens reconhecíveis por autômatos com pilha. Para finalizar, propriedades importantes desta classe de linguagens serão abordadas na Seção 3.5.

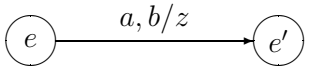
## 3.1 Uma Introdução Informal

Um autômato com pilha (AP) pode ser visto como uma máquina similar àquela ilustrada na Figura 2.37, página 127, que contém adicionalmente uma pilha, como mostrado na Figura 3.1. Como a fita, a pilha é dividida em células que comportam apenas um símbolo cada uma, mas o cabeçote de leitura da pilha só se posiciona na célula do topo da pilha. No início, o registrador contém o estado inicial do AP, a fita contém a palavra de entrada a partir da sua primeira célula, o cabeçote da fita é posicionado na primeira célula da fita, e a pilha está vazia.

Suponha um AP com conjunto de estados $E$, alfabeto de entrada (alfabeto da fita) $\Sigma$ e alfabeto da pilha $\Gamma$. Cada transição do AP será da forma

$$\delta(e, a, b) = [e', z] \text{ } ([e', z] \in \delta(e, a, b) \text{ para AP não determinístico}),$$

onde $e, e' \in E$, $a \in \Sigma \cup \{\lambda\}$, $b \in \Gamma \cup \{\lambda\}$ e $z \in \Gamma^*$. Tal transição será dita ser uma transição de $e$ para $e'$ sob $a$ com $b/z$, sendo representada em um diagrama de estados da seguinte forma:



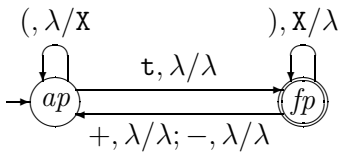



Figura 3.2: Reconhecendo expressões aritméticas simples.

significando, no caso em que $a \neq \lambda$, $b \neq \lambda$ e $z \neq \lambda$, que "estando no estado $e$, se o próximo símbolo de entrada for $a$ e o símbolo no topo da pilha for $b$, há uma transição para o estado $e'$, $b$ é desempilhado e $z$ é empilhado (o símbolo mais à esquerda em $z$, no topo)". Se $a = \lambda$, não é consumido símbolo de entrada, e a transição é dita ser uma *transição* $\lambda$. Se $b = \lambda$, a transição acontece sem consulta à pilha e nada é desempilhado. E se $z = \lambda$, nada é empilhado.

Um AP simples que reconhece um certo tipo de expressão aritmética é apresentado a seguir.

**Exemplo 91** Seja o conjunto EA das expressões aritméticas com parênteses e as operações de soma (`+`) e subtração (`-`), definido recursivamente por:

(a) `t` $\in$ EA;

(b) se $x, y \in$ EA, então $(x) \in$ EA, $x$+$y \in$ EA e $x-y \in$ EA.

O símbolo `t` pode ser imaginado como representando expressões mais básicas, como números inteiros e/ou reais, identificadores de variáveis, etc. O reconhecimento de tais expressões básicas não oferece nenhum problema, podendo ser feito mediante um AF. A Figura 3.2 apresenta um diagrama de estados para um AP que reconhece EA. Observe que o conjunto de estados é $E = \{ap, fp\}$, o alfabeto de entrada é $\Sigma = \{\texttt{t}, (, ), +, -\}$ e o da pilha é $\Gamma = \{\texttt{X}\}$. O estado inicial é $ap$ e o conjunto de estados finais é $\{fp\}$. Existem 5 transições:

1. $\delta(ap, (, \lambda) = [ap, \texttt{X}]$
2. $\delta(ap, \texttt{t}, \lambda) = [fp, \lambda]$
3. $\delta(fp, ), \texttt{X}) = [fp, \lambda]$
4. $\delta(fp, +, \lambda) = [ap, \lambda]$
5. $\delta(fp, -, \lambda) = [ap, \lambda]$.

As transições estão numeradas para referência futura. Os detalhes de funcionamento deste AP serão elucidados do decorrer desta seção, na medida em que os conceitos necessários forem sendo introduzidos. □

Uma pilha de símbolos de um alfabeto $\Gamma$ será representada por meio de uma palavra $w$ de $\Gamma^*$. A convenção adotada é que o símbolo mais a esquerda está no *topo*. Assim, o resultado de empilhar o símbolo $a$ na pilha $y$ é a pilha $ay$. O resultado de desempilhar o elemento do topo da pilha $ay$ é a pilha $y$. A pilha vazia será representada pela palavra $\lambda$.

No capitulo 2, mostrou-se que a *configuração instantânea* de um AF é um par $[e, w]$, onde $e$ é o estado atual do autômato e $w$ é o restante da palavra de entrada. A configuração instantânea consta das informações necessárias para o autômato prosseguir no reconhecimento da palavra de entrada em certo instante. Em um autômato com pilha, ela será uma tripla $[e, w, p]$, onde $e$ é o estado atual, $w$ é o restante da palavra de entrada e $p$ é a pilha. Como explicado na seção 2.1.1, usa-se a notação $ci \vdash ci'$ para dizer que a configuração instantânea $ci'$ é o resultado de uma transição a partir da configuração instantânea $ci$. Com isto, por exemplo, pode-se expressar a seguinte computação para o AP do Exemplo 91, Figura 3.2, quando a palavra de entrada é `(t-(t+t))`:

$$\begin{array}{lll}
[ap, (\mathtt{t} - (\mathtt{t} + \mathtt{t})), \lambda] & & \\
\quad \vdash [ap, \mathtt{t} - (\mathtt{t} + \mathtt{t})), X] & \text{por 1} \\
\quad \vdash [fp, -(\mathtt{t} + \mathtt{t})), X] & \text{por 2} \\
\quad \vdash [ap, (\mathtt{t} + \mathtt{t})), X] & \text{por 5} \\
\quad \vdash [ap, \mathtt{t} + \mathtt{t})), XX] & \text{por 1} \\
\quad \vdash [fp, +\mathtt{t})), XX] & \text{por 2} \\
\quad \vdash [ap, \mathtt{t})), XX] & \text{por 4} \\
\quad \vdash [fp, )), XX] & \text{por 1} \\
\quad \vdash [fp, ), X] & \text{por 3} \\
\quad \vdash [fp, \lambda, \lambda] & \text{por 3.}
\end{array}$$

Não há transição que se aplique a $\vdash [fp, \lambda, \lambda]$.

Um outro exemplo:

$$[ap, \mathtt{t}), \lambda] \vdash [fp, ), \lambda] \qquad \text{por 2.}$$

Não há transição que se aplique a $[fp, ), \lambda]$. Este segundo exemplo mostra que o AP pode não consumir toda a palavra de entrada. Usando a metáfora propiciada pelo esquema da Figura 3.1, onde a cada transição corresponde uma "instrução" da máquina, diz-se que um AP pode *parar* sem consumir toda a palavra de entrada.

A condição para uma palavra ser reconhecida é que ela seja totalmente consumida e que a máquina termine em um estado final com a pilha vazia. Assim, para o AP do exemplo 91, a palavra `(t-(t+t))` é reconhecida, como mostra a primeira computação acima. A palavra `t)` não é reconhecida, pois o AP não consome toda a palavra, como mostra a segunda computação acima. E a palavra `(t` não é reconhecida, pois o AP pára com a pilha não vazia:

$$\begin{array}{lll}
[ap, (\mathtt{t}, \lambda] \vdash [ap, \mathtt{t}, \mathtt{X}] & \text{por 1} \\
\quad \vdash [fp, \lambda, \mathtt{X}] & \text{por 2.}
\end{array}$$

Não há transição que se aplique a $[fp, \lambda, \mathtt{X}]$.

É interessante notar que existem AP's que *não* param para algumas entradas, ou mesmo para todas as entradas, como mostra o próximo exemplo.

**Exemplo 92** Um exemplo conciso de um AP com computações de tamanho ilimitado seria aquele com alfabeto de entrada $\{\mathtt{1}\}$ e com o diagrama de estados mostrado na Figura 3.3. Para toda palavra em $\{1\}^+$, pode-se dizer que o AP não pára, visto que o primeiro símbolo nunca é lido e a única transição existente é sempre aplicável. Em particular:

$$[0, \mathtt{1}, \lambda] \vdash [0, \mathtt{1}, X] \vdash [0, \mathtt{1}, XX] \ldots$$

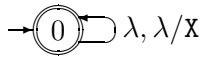



Figura 3.3: Um AP com computações ilimitadas.

Para a palavra $\lambda$, a transição também é aplicável, e tem-se computações de todo tamanho. Pergunta: para a palavra de entrada $\lambda$, deve-se considerar que o AP pára ou não? A palavra $\lambda$ é reconhecida ou não? □

Na próxima seção é definido formalmente o conceito de autômato com pilha determinístico e apresentados alguns exemplos. O problema levantado no Exemplo 92 é resolvido formalizando-se convenientemente a noção de reconhecimento.

## 3.2 Autômatos com Pilha Determinísticos

Os autômatos com pilha determinísticos (APD's) são especialmente importantes, já que lidam com uma classe de linguagens para as quais há reconhecedores eficientes.

A definição de autômato com pilha determinístico, basicamente, acrescenta uma pilha a um AFD. Para que haja *determinismo*, não deverá ser possível duas transições $\delta(e, a, b)$ e $\delta(e, a', b')$ serem definidas para uma mesma configuração instantânea. A definição abaixo captura exatamente as situações em que duas transições podem ocorrer simultaneamente para alguma configuração instantânea. O exercício 1 no final desta seção, página 147, propõe a prova deste fato.

**Definição 27** *Seja uma função de transição* $\delta : E \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \to E \times \Gamma^*$. *Duas transições* $\delta(e, a, b)$ *e* $\delta(e, a', b')$ *são ditas compatíveis se, e somente se:*

$$(a = a' \text{ ou } a = \lambda \text{ ou } a' = \lambda) \text{ e } (b = b' \text{ ou } b = \lambda \text{ ou } b' = \lambda).$$

□

Para algumas pessoas pode parecer mais intuitivo o complemento da expressão apresentada na Definição 27: duas transições $\delta(e, a, b)$ e $\delta(e, a', b')$ são *não* compatíveis se, e somente se:

$$(a \neq a' \text{ e } a \neq \lambda \text{ e } a' \neq \lambda) \text{ ou } (b \neq b' \text{ e } b \neq \lambda \text{ e } b' \neq \lambda).$$

**Definição 28** *Um autômato com pilha determinístico (APD) é uma sêxtupla* $(E, \Sigma, \Gamma, \delta, i, F)$, *onde*

- $E$ *é um conjunto finito de um ou mais elementos denominados estados;*
- $\Sigma$ *é o alfabeto de entrada;*
- $\Gamma$ *é o alfabeto de pilha;*
- $\delta$, *a função de transição, é uma função parcial de* $E \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})$ *para* $E \times \Gamma^*$, *sem transições compatíveis;*

- *$i$, um estado de $E$, é o estado inicial;*
- *$F$, subconjunto de $E$, é o conjunto de estados finais.*

□

As seguintes razões fazem com que não haja como definir uma função similar à função $\hat{\delta}$ do Capítulo 2:

- Na Seção 3.1 ficou claro que pode haver computações que não terminam.
- Além do(s) estado(s) atingido(s) é importante saber o conteúdo da pilha.

Assim, ao invés de uma função $\hat{\delta}$, será usada a relação $\vdash$ definida a seguir.

**Definição 29** *Seja um APD $M = (E, \Sigma, \Gamma, \delta, i, F)$. A relação $\vdash \subseteq (E \times \Sigma^* \times \Gamma^*)^2$, para $M$, é tal que para todo $e \in E$, $a \in \Sigma \cup \{\lambda\}$, $b \in \Gamma \cup \{\lambda\}$ e $x \in \Gamma^*$:*

$$[e, ay, bz] \vdash [e', y, xz] \text{ para todo } y \in \Sigma^* \text{ e } z \in \Gamma^*;$$

$$\text{se, e somente se, } \delta(e, a, b) = [e', x].$$ □

Utilizando a relação $\overset{*}{\vdash}$, que é o fecho transitivo e reflexivo de $\vdash$, define-se a seguir o que é a linguagem reconhecida (aceita) por um APD.

**Definição 30** *Seja um APD $M = (E, \Sigma, \Gamma, \delta, i, F)$. A* linguagem reconhecida *por $M$ é*

$$L(M) = \{w \in \Sigma^* \mid [i, w, \lambda] \overset{*}{\vdash} [e, \lambda, \lambda] \text{ e } e \in F\}.$$

*Uma palavra $w$ tal que $[i, w, \lambda] \overset{*}{\vdash} [e, \lambda, \lambda]$, onde $e \in F$, é dita ser* reconhecida *(*aceita*) por $M$.* □

**Exemplo 93** No exemplo 71, página 98, mostrou-se que o conjunto $\{\mathtt{a}^n\mathtt{b}^n \mid n \in \mathbf{N}\}$ não é uma linguagem regular. Ela é reconhecida pelo autômato com pilha determinístico $(\{a, b\}, \{\mathtt{a}, \mathtt{b}\}, \{\mathtt{X}\}, \delta, a, \{a, b\})$, onde $\delta$ é dada por:

1. $\delta(a, \mathtt{a}, \lambda) = [a, \mathtt{X}]$
2. $\delta(a, \mathtt{b}, \mathtt{X}) = [b, \lambda]$
3. $\delta(b, \mathtt{b}, \mathtt{X}) = [b, \lambda]$.

O diagrama de estados está apresentado na Figura 3.4. □

A seguir, mostra-se um exemplo um pouco mais elaborado.

**Exemplo 94** [2] A Figura 3.5 apresenta o diagrama de estados de um APD $M$ que reconhece a linguagem

$$\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid \text{ o número de } \mathtt{0}\text{'s em } w \text{ é igual ao de } \mathtt{1}\text{'s}\}.$$

Tem-se que $M = (\{\mathit{igual}, \mathit{dif}\}, \{\mathtt{0}, \mathtt{1}\}, \{\mathtt{Z}, \mathtt{U}, \mathtt{F}\}, \delta, \mathit{igual}, \{\mathit{igual}\})$, onde $\delta$ é dada por:

[2] O APD apresentado neste exemplo foi desenvolvido por Jonatan Schröeder, na época (segundo semestre de 2000), aluno da UFPR.

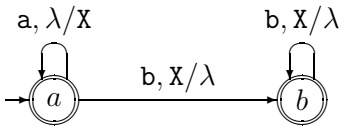



Figura 3.4: APD para $a^n b^n$.

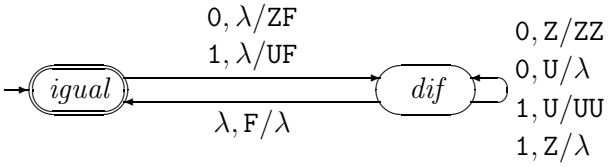



Figura 3.5: APD para número igual de 0's e 1's.

1. $\delta(igual, \mathtt{0}, \lambda) = [dif, \mathtt{ZF}]$
2. $\delta(igual, \mathtt{1}, \lambda) = [dif, \mathtt{UF}]$
3. $\delta(dif, \mathtt{0}, \mathtt{Z}) = [dif, \mathtt{ZZ}]$
4. $\delta(dif, \mathtt{0}, \mathtt{U}) = [dif, \lambda]$
5. $\delta(dif, \mathtt{1}, \mathtt{U}) = [dif, \mathtt{UU}]$
6. $\delta(dif, \mathtt{1}, \mathtt{Z}) = [dif, \lambda]$
7. $\delta(dif, \lambda, \mathtt{F}) = [igual, \lambda]$.

Observe que, como requerido, não há transições compatíveis.

Após lido um prefixo $x$ da palavra de entrada, a pilha é:

- $\mathtt{Z}^n$, se $x$ tem $n$ 0's a mais que 1's; ou
- $\mathtt{U}^n$, se $x$ tem $n$ 1's a mais que 0's.

A seguinte computação mostra que 001110 pertence à linguagem reconhecida por $M$:

$$
\begin{aligned}
[igual, \mathtt{001110}, \lambda] &\vdash [dif, \mathtt{01110}, \mathtt{ZF}] \text{ por 1} \\
&\vdash [dif, \mathtt{1110}, \mathtt{ZZF}] \text{ por 3} \\
&\vdash [dif, \mathtt{110}, \mathtt{ZF}] \text{ por 6} \\
&\vdash [dif, \mathtt{10}, \mathtt{F}] \text{ por 6} \\
&\vdash [igual, \mathtt{10}, \lambda] \text{ por 7} \\
&\vdash [dif, \mathtt{0}, \mathtt{UF}] \text{ por 2} \\
&\vdash [dif, \lambda, \mathtt{F}] \text{ por 4} \\
&\vdash [igual, \lambda, \lambda] \text{ por 7.}
\end{aligned}
$$

□

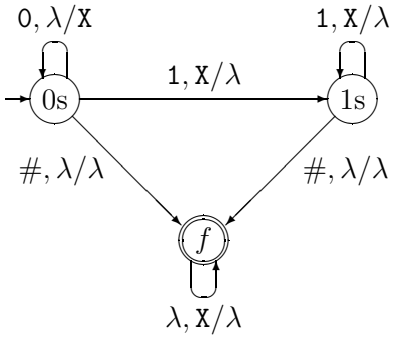



Figura 3.6: APD com símbolo de final de palavra.

O exemplo 94 utiliza uma técnica para verificar se a pilha chegou ao fundo: foi introduzido um símbolo de pilha (F) especificamente para isto. Algumas formalizações de APD's incluem um símbolo especial para tal propósito. No entanto, a introdução de um símbolo especial de fundo de pilha não aumenta o poder de reconhecimento dos APD's, já que um símbolo de pilha comum pode fazer seu papel, como mostra o Exemplo 94.

Existem linguagens que podem ser reconhecidas por autômatos com pilha não determinísticos, mas que não podem ser reconhecidas por APD's, como será visto na próxima seção. Dentre estas, existem algumas que passam a ser reconhecíveis por APD's caso haja um símbolo específico para finalizar a palavra de entrada. Segue um exemplo.

**Exemplo 95** A Figura 3.6 apresenta o diagrama de estados de um APD que reconhece a linguagem

$$\{0^m 1^n \# \mid m \geq n\}.$$

Observe que o símbolo # só é utilizado para finalizar as palavras da linguagem. A linguagem similar, sem tal símbolo,

$$\{0^m 1^n \mid m \geq n\}.$$

não pode ser reconhecida por APD, como pode ser verificado.[3]

Observe que, para cada 0 alguma coisa deve ser empilhada para, no futuro, garantir-se que o número de 1's não ultrapasse o de 0's. Mas, após lido o prefixo de 0's, caso possa ser lido mais algum 1 (conforme indicado pela pilha), pode-se também terminar a palavra. Neste último caso, a pilha deve ser esvaziada sem leitura de mais símbolos. O símbolo para indicar final de palavra propicia, justamente, reconhecer determinísticamente o momento de parar a leitura e esvaziar a pilha. □

A definição de reconhecimento apresentada na Definição 30, página 142, não faz referência à *parada* da máquina. Assim, por exemplo, a linguagem reconhecida pelo APD cujo diagrama de estados está mostrado na Figura 3.3, página 141, é $\{\lambda\}$, não sendo importante se o APD pára ou não para tal entrada.

Será apresentado, a seguir, um "algoritmo" que implementa um APD, de forma similar à utilizada para AFD's apresentada na Figura 2.5, página 67. Na realidade, tal

[3]Na verdade, esta linguagem pode ser reconhecida por APD, mas com um outro critério de reconhecimento, como será visto na Seção 3.3.

Entrada: (1) o APD, dado por $i$, $F$ e $D$, e
(2) a palavra de entrada, dada por *prox*.
Saída: *sim* ou *não*.
$e \leftarrow i$; *empilhe*($\bigtriangledown$); $ps \leftarrow$ *prox*();
**enquanto** $D[e, a, b]$ é definido p/ $a \in \{ps, \lambda\}$ e $b \in \{topo(), \lambda\}$ **faça**
    seja $D[e, a, b] = [e', z]$;
    **se** $a \neq \lambda$ **então** $ps \leftarrow$ *prox*() **fimentão;**
    **se** $b \neq \lambda$ **então** *desempilhe*() **fimentão;**
    *empilhe*($z$);
    $e \leftarrow e'$
**fimenquanto**;
**se** $ps = fs$ e *topo*() $= \bigtriangledown$ e $e \in F$ **então**
    retorne *sim*
**senão**
    retorne *não*
**fimse**

Figura 3.7: Algoritmo para simular APD's.

"algoritmo" pode entrar em *loop*, como já ficou ressaltado acima. Assim, para ser utilizado, deve-se antes obter um APD, equivalente ao APD dado, em que *loops* não ocorram, seja manualmente ou por meio de um algoritmo. Serão usadas as seguintes variáveis para representar um APD $(E, \Sigma, \Gamma, \delta, i, F)$:

- $i$: estado inicial;
- $F$: conjunto dos estados finais;
- $D$, uma matriz de leitura-apenas, contendo a função de transição, de forma que $D[e, a, b] = \delta(e, a, b)$ para todo $e \in E$, $a \in \Sigma \cup \{\lambda\}$ e $b \in \Gamma \cup \{\lambda\}$.

Assume-se a existência de um procedimento do tipo função, *prox*, que retorna o próximo símbolo de entrada, quando houver, e *fs* (fim de seqüência), quando não houver. A pilha é inicializada com $\bigtriangledown$, um símbolo que não pertence a $\Gamma$, e é manipulada mediante os seguintes procedimentos:

- *empilhe*($z$): empilha a palavra $z$; se $z = \lambda$, nada é empilhado;
- *topo*(): retorna o símbolo do topo, sem desempilhar; e
- *desempilhe*(): desempilha o símbolo do topo.

O algoritmo está mostrado na Figura 3.7.

Assim como os AFD's, os APD's podem ser instrumentos úteis na etapa intermediária entre a especificação e a implementação de alguns tipos de aplicações, ou seja, na etapa de modelagem. Segue um exemplo.

**Exemplo 96** No exemplo 91 mostrou-se o diagrama de estados de um APD (Figura 3.2, página 139) para reconhecimento de expressões aritméticas simples, envolvendo apenas os operadores de soma e subtração. Em uma aplicação que envolva expressões aritméticas,

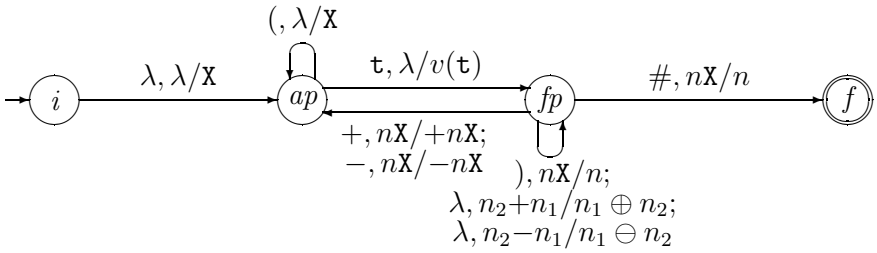



Figura 3.8: Avaliando expressões aritméticas simples.

além de verificar se elas estão sintaticamente corretas, em geral deve-se produzir, durante a verificação, algum outro tipo de saída. Um exemplo de saída seria uma estrutura de dados que representasse adequadamente a expressão para posterior processamento. Um outro, caso a expressão possa ser avaliada durante a verificação sintática, seria o valor resultante da avaliação da expressão. Na Figura 3.8, apresenta-se um diagrama de estados de um APD similar ao da Figura 3.2, "enriquecido" para incluir a avaliação da expressão durante o reconhecimento da mesma.

Assume-se a existência de uma função $v$ que, dado um termo `t` (uma palavra de dígitos), obtém o número correspondente, de tal forma que ele possa ser utilizado como operando em operações aritméticas. As operações de soma e subtração estão designadas pelos símbolos $\oplus$ e $\ominus$; seus operandos são números e o resultado de uma operação é um número. Cada célula da pilha agora pode conter um símbolo de pilha (`X`, `+` ou `-`) ou um número. Para consultar se um número está na pilha, usa-se a notação $n$, $n_1$ ou $n_2$. As transições foram estendidas para consultar e desempilhar um segmento do topo da pilha. Assim, por exemplo, a transição

$$\delta(fp, \lambda, n_2{+}n_1) = [fp, n_1 \oplus n_2]$$

ocorre quando, no estado *fp* a pilha tem o segmento $n_2{+}n_1$ no topo, ou seja, um número $n_2$ no topo, depois o símbolo `+` e depois um número $n_1$; ao ocorrer a transição, tal segmento é desempilhado e é empilhado o número $n_1 \oplus n_2$, ou seja, a soma de $n_1$ e $n_2$.

Supõe-se associação de operações à esquerda. Com isto, por exemplo, tem-se que $3 - 2 + 1 = 1 + 1 = 2$.

Assume-se que a palavra de entrada é terminada com #. O resultado da avaliação da expressão, caso a mesma esteja sintaticamente correta, é colocado na pilha pela transição de *fp* para *f* sob # com $n$`X`$/n$. Segue uma computação para a expressão `5-(2+4)+3#`:

$$\begin{aligned}
[i, \texttt{5} - (\texttt{2} + \texttt{4}) + \texttt{3}\#, \lambda] &\vdash [ap, \texttt{5} - (\texttt{2} + \texttt{4}) + \texttt{3}\#, \texttt{X}] \\
&\vdash [fp, -(\texttt{2} + \texttt{4}) + \texttt{3}\#, \texttt{5X}] \\
&\vdash [ap, (\texttt{2} + \texttt{4}) + \texttt{3}\#, -\texttt{5X}] \\
&\vdash [ap, \texttt{2} + \texttt{4}) + \texttt{3}\#, \texttt{X}-\texttt{5X}] \\
&\vdash [fp, +\texttt{4}) + \texttt{3}\#, \texttt{2X}-\texttt{5X}] \\
&\vdash [ap, \texttt{4}) + \texttt{3}\#, +\texttt{2X}-\texttt{5X}] \\
&\vdash [fp, ) + \texttt{3}\#, \texttt{4}+\texttt{2X}-\texttt{5X}] \\
&\vdash [fp, ) + \texttt{3}\#, \texttt{6X}-\texttt{5X}] \\
&\vdash [fp, +\texttt{3}\#, \texttt{6}-\texttt{5X}] \\
&\vdash [fp, +\texttt{3}\#, -\texttt{1X}]
\end{aligned}$$

$\vdash [ap, 3\#, +-1\mathtt{X}]$
$\vdash [fp, \#, 3+-1\mathtt{X}]$
$\vdash [fp, \#, 2\mathtt{X}]$
$\vdash [f, \lambda, 2]$.

□

Evidentemente, no Exemplo 96 o conceito de APD foi utilizado apenas como base para a modelagem do problema de reconhecimento e avaliação de expressões aritméticas. Detalhes de implementação, alheios à essência da solução, são evitados. Por exemplo, o detalhe de lidar com uma pilha com tipos variados de dados não é abordado. Isto é conveniente, visto que o tratamento adequado deste tipo de detalhe depende muitas vezes da linguagem de programação a ser utilizada.

Ao contrário dos autômatos finitos, os autômatos com pilha têm o seu poder aumentado quando se introduz não determinismo, como será visto na próxima seção. Depois, na Seção 3.4, serão estudadas as gramáticas livres do contexto, que geram exatamente as linguagens reconhecíveis por autômatos com pilha. Isto é importante, já que na maioria das situações que ocorrem na prática, é mais fácil e conveniente obter uma gramática para a linguagem e, a partir da gramática, obter o autômato com pilha (determinístico ou não).

# Exercícios

1. Mostre que duas transições são compatíveis (veja Definição 27, página 141) se, e somente se, elas podem ocorrer simultaneamente para alguma configuração instantânea.

2. No Exemplo 94, página 142, apresentou-se um APD para a linguagem

$$\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid \text{ o número de } \mathtt{0}\text{'s em } w \text{ é igual ao de } \mathtt{1}\text{'s}\}$$

   com dois estados e três símbolos de pilha. Construa um APD para esta mesma linguagem com três estados e dois símbolos de pilha, *sem transição* $\lambda$.

3. Construa APD's para as seguintes linguagens:

   (a) $\{\mathtt{0}^n\mathtt{1}^{2n} \mid n \geq 0\}$.
   (b) $\{\mathtt{0}^{3n}\mathtt{1}^{2n} \mid n \geq 0\}$.
   (c) $\{w\mathtt{0}w^R \mid w \in \{\mathtt{1}, \mathtt{2}\}^*\}$.
   (d) $\{\mathtt{0}^m\mathtt{1}^n \mid m < n\}$.
   (e) $\{\mathtt{0}^m\mathtt{1}^n\# \mid m \neq n\}$.
   (f) $\{w\# \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ o número de $\mathtt{0}$'s em $w$ é maior que o de $\mathtt{1}$'s$\}$.

4. Explique porque não há APD para as seguintes linguagens:

   (a) $\{ww^R \mid w \in \{\mathtt{1}, \mathtt{2}\}^*\}$.

(b) $\{\mathtt{0}^m\mathtt{1}^n \mid m > n\}$.

(c) $\{\mathtt{0}^m\mathtt{1}^n \mid m \neq n\}$.

(d) $\{w \in \{\mathtt{0},\mathtt{1}\}^* \mid$ o número de `0`'s em $w$ é maior que o de `1`'s$\}$.

5. Construa um APD que reconheça toda palavra com parênteses balanceados. Exemplos de palavras da linguagem: $\lambda$, `()`, `(())()`. Exemplos de palavras que não pertencem à linguagem: `(`, `)(`, `())`.

   Generalize para o caso em que existem $n$ tipos de parênteses. Neste caso, considere que cada ocorrência de um abre parênteses, $a_i$, tem de ser seguida à direita por uma ocorrência do fecha parênteses respectivo, $b_i$, sendo que entre $a_i$ e $b_i$ só pode ocorrer uma palavra com parênteses balanceados. Se $i = 2$, $a_1 = ($, $b_1 = )$, $a_2 = [$, $b_2 = ]$, seriam exemplos de palavras da linguagem: $\lambda$, `()`, `[()]()`, `[[()]()]([])`. Exemplos de palavras que não pertencem à linguagem: `(`, `][`, `()]`, `(]`, `([)]`.

6. Construa um APD que reconheça as expressões aritméticas na forma prefixada, EAPre, definidas recursivamente como segue:

   (a) `t` é uma EAPre;

   (b) se $x$ e $y$ são EAPre's, então $+xy$ e $-xy$ são EAPre's.

   *Sugestão*: sempre que ler `+` ou `-`, empilhe dois `X`'s, para "lembrar" de ler duas subexpressões à frente.

7. Construa um APD que reconheça as expressões aritméticas na forma posfixada, EAPos, definidas recursivamente como segue:

   (a) `t` é uma EAPos;

   (b) se $x$ e $y$ são EAPos's, então $xy+$ e $xy-$ são EAPos's.

   *Sugestão*: use os seguintes fatos, fáceis de mostrar por indução: o número de `t`'s é um a mais que o de operadores, e qualquer palavra que tenha mais `t`'s que operadores é prefixo de EAPos.

## 3.3 Autômatos com Pilha Não Determinísticos

A diferença entre um autômato com pilha determinístico e um não determinístico é que este último pode conter transições compatíveis, como visto na definição a seguir.

**Definição 31** *Um autômato com pilha não determinístico (APN) é uma sêxtupla* $(E, \Sigma, \Gamma, \delta, I, F)$*, onde*

- $E$*,* $\Sigma$*,* $\Gamma$ *e* $F$ *são como em APD's;*
- $\delta$*, a função de transição, é uma função parcial de* $E \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})$ *para* $D$*, onde* $D$ *é constituído dos subconjuntos finitos de* $E \times \Gamma^*$*;*
- $I$*, um subconjunto de* $E$*, é o conjunto de estados iniciais.*

□

A relação $\overset{*}{\vdash}$ da Definição 29, página 142, será utilizada para definir reconhecimento para APN's, de forma similar ao reconhecimento para APD's apresentado na Definição 30.

**Definição 32** *Seja um APN* $M = (E, \Sigma, \Gamma, \delta, I, F)$. *A* linguagem reconhecida *por* $M$ *é*

$$L(M) = \{w \in \Sigma^* \mid [i, w, \lambda] \overset{*}{\vdash} [e, \lambda, \lambda] \text{ e } e \in F \text{ para algum } i \in I\}.$$

*Uma palavra* $w$ *tal que* $[i, w, \lambda] \overset{*}{\vdash} [e, \lambda, \lambda]$, *onde* $i \in I$ *e* $e \in F$, *é dita ser* reconhecida *(*aceita*) por* $M$. □

Segue um exemplo que mostra a evolução de um APD para um APN equivalente "mais conciso".

**Exemplo 97** No Exemplo 94, página 142, foi visto um APD para a linguagem

$$\{w \mid w \in \{\mathtt{0}, \mathtt{1}\}^* \text{ e o número de } \mathtt{0}\text{'s em } w \text{ é igual ao de } \mathtt{1}\text{'s}\}.$$

Neste APD é usado o símbolo `F` para marcar o fundo da pilha, de forma que, quando os números de `0`'s e de `1`'s se tornam idênticos (mesmo que a palavra não tenha sido toda processada ainda), seja ativada a transição para o estado final *igual*. Ora, um autômato *não determinístico* pode "adivinhar" quando a pilha se torna vazia e fazer a transição citada. Assim, um APN equivalente ao APD do Exemplo 94 seria:

$$N = (\{igual, dif\}, \{\mathtt{0}, \mathtt{1}\}, \{\mathtt{Z}, \mathtt{U}\}, \delta, \{igual\}, \{igual\}),$$

onde $\delta$ é dada por:

1. $\delta(igual, \mathtt{0}, \lambda) = \{[dif, \mathtt{Z}]\}$
2. $\delta(igual, \mathtt{1}, \lambda) = \{[dif, \mathtt{U}]\}$
3. $\delta(dif, \mathtt{0}, \mathtt{Z}) = \{[dif, \mathtt{ZZ}]\}$
4. $\delta(dif, \mathtt{0}, \mathtt{U}) = \{[dif, \lambda]\}$
5. $\delta(dif, \mathtt{1}, \mathtt{U}) = \{[dif, \mathtt{UU}]\}$
6. $\delta(dif, \mathtt{1}, \mathtt{Z}) = \{[dif, \lambda]\}$
7. $\delta(dif, \lambda, \lambda) = \{[igual, \lambda]\}$.

Veja o diagrama de estados de $N$ na Figura 3.9. Note que a única diferença com relação ao APD do Exemplo 94 é que o símbolo `F` que lá ocorre foi substituído por $\lambda$.

Observe que a transição 7 é compatível com as transições 3 a 6, mas de uma forma restrita: partindo-se do estado *dif*, quando a pilha está vazia, apenas a transição 7 é aplicável; somente quando a pilha não está vazia, uma das transições 3 a 6 é aplicável, além da transição 7. Assim, se for dada *prioridade* sempre para as transições 3 a 6, o comportamento do APN é análogo ao do APD do Exemplo 94. Isto evidencia que este

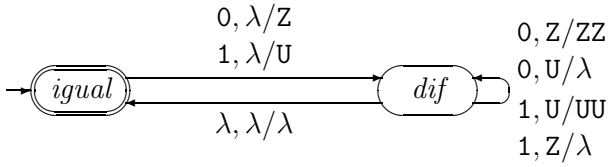
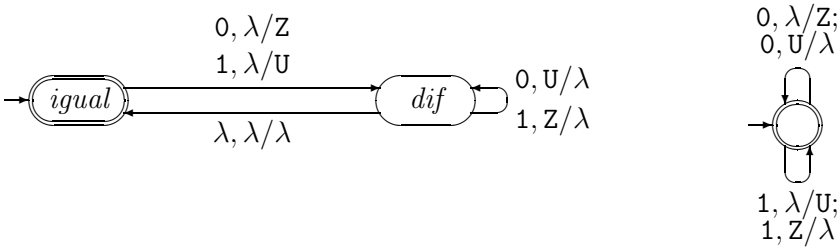



Figura 3.9: APN para número igual de 0's e 1's.




(a) O segundo. (b) O terceiro.

Figura 3.10: Mais dois APN's para número igual de 0's e 1's.

APN reconhece toda palavra que o referido APD reconhece. Por outro lado, tal APN continua não podendo reconhecer palavras com números diferentes de 0's e 1's, como pode ser notado verificando-se o padrão de empilhamentos e desempilhamentos.

Na realidade, o APN $N$ é menos conciso do que poderia ser. Observe que as transições 3 e 5 são desnecessárias. O mesmo efeito da transição 3 pode ser conseguido aplicando-se, em seqüência, as transições 7 (compatível com a 3) e 1, e o mesmo efeito da transição 5 pode ser conseguido aplicando-se, em seqüência, as transições 7 (compatível com a 5) e 2. Obtém-se, com isto, o APN cujo diagrama de estados está mostrado na Figura 3.10(a). Analisando-se este último diagrama de estados, levando-se em conta que o reconhecimento se dá quando a pilha fica vazia, chega-se ao APN equivalente cujo diagrama de estados está na Figura 3.10(b). □

A seguir, é apresentado um APN para uma linguagem que não pode ser reconhecida por APD's.

**Exemplo 98** Na Figura 3.11 está mostrado o diagrama de estados para um APN que reconhece a linguagem dos palíndromos sobre o alfabeto $\{0, 1\}$, ou seja, $\{w \in \{0, 1\}^* \mid w = w^R\}$.

Caso uma palavra $w$ seja palíndromo, existirá uma computação para $w$ em que $w$ é consumida e a pilha fica vazia; para tal computação, uma das 3 transições de 1 para 2 é percorrida:

- se $|w|$ for par, é percorrida a transição de 1 para 2 sob $\lambda$;
- se $|w|$ for ímpar e o símbolo do meio for 0, é percorrida a transição de 1 para 2 sob 0;

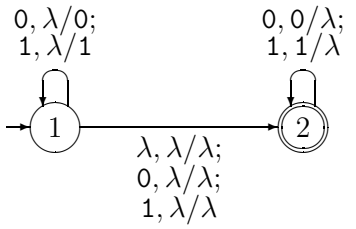



Figura 3.11: APN para palíndromos sobre $\{0,1\}^*$.

- se $|w|$ for ímpar e o símbolo do meio for 1, é percorrida a transição de 1 para 2 sob 1.

Ao processar uma palavra da esquerda para a direita, quando é atingido o meio da palavra, não há como o autômato reconhecer tal fato, para, a partir daí, comparar a segunta metade com a primeira. Assim sendo, não há como construir um APD para a linguagem dos palíndromos. □

Pela Definicao 32, a linguagem reconhecida por um APN $M = (E, \Sigma, \Gamma, \delta, I, F)$ é

$$L(M) = \{w \in \Sigma^* \mid [i, w, \lambda] \overset{*}{\vdash} [e, \lambda, \lambda] \text{ e } e \in F \text{ para algum } i \in I\}.$$

Uma definição alternativa, que levaria a uma concepção diferente de APN's, é aquela em que o reconhecimento de uma palavra se dá ao ser atingido um estado final, após consumida a palavra de entrada, esteja a pilha vazia ou não. Usando o índice $F$ em $L_F(M)$ para significar *reconhecimento por estado final*, segue tal definição alternativa, mais formalmente.

**Definição 33** *Seja um APN $M = (E, \Sigma, \Gamma, \delta, I, F)$. A* linguagem reconhecida *por $M$* por estado final *é*

$$L_F(M) = \{w \in \Sigma^* \mid [i, w, \lambda] \overset{*}{\vdash} [e, \lambda, y], \\ \text{onde } y \in \Gamma^*, \text{ e } e \in F \text{ para algum } i \in I\}.$$

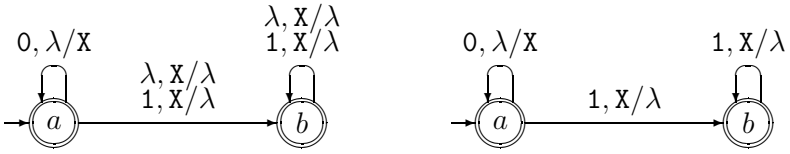
*Uma palavra $w$ tal que $[i, w, \lambda] \overset{*}{\vdash} [e, \lambda, y]$, onde $y \in \Gamma^*$, $i \in I$ e $e \in F$, é dita ser* reconhecida *(*aceita*) por $M$* por estado final. □

O reconhecimento segundo a Definição 32 será denominado a seguir de *reconhecimento por pilha vazia e estado final*.

Pode-se mostrar que uma linguagem pode ser reconhecida por pilha vazia e estado final se, e somente se, pode ser reconhecida por estado final, como será visto no Teorema 16 no final desta seção.

O exemplo a seguir mostra dois autômatos com pilha que reconhecem a mesma linguagem, um deles usando reconhecimento por pilha vazia e estado final, e o outro usando reconhecimento por estado final.

**Exemplo 99** Seja o problema de determinar um APN que reconheça a linguagem $L = \{0^m 1^n \mid m \geq n\}$.




(a) Rec. por estado final e pilha vazia. (b) Reconhecimento por estado final.

Figura 3.12: APN's para $\{0^m1^n \mid m \geq n\}$.

A Figura 3.12(a) mostra o diagrama de estados de um APN $M$ tal que $L(M) = L$, onde $M$ reconhece por estado final e pilha vazia, enquanto que a Figura 3.12(b) mostra o diagrama de estados de um APN $M'$ tal que $L_F(M') = L$, onde $M'$ reconhece por estado final. Veja que, por coincidência, o diagrama de estados da Figura 3.12(b) é idêntico ao da Figura 3.4, página 143, que reconhece a linguagem $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$ por estado final e pilha vazia (substituindo-se `0` por `a` e `1` por `b`); assim, $L(M') = \{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$. Observe também que $L_F(M) = L$: coincidentemente, o APN cujo diagrama de estados está exibido na Figura 3.12(a) reconhece a mesma linguagem para os dois métodos de reconhecimento. □

Uma outra definição alternativa é aquela em que o reconhecimento de uma palavra se dá quando a pilha fica vazia, após consumida a palavra de entrada. Neste caso, não há o conceito de estado final. Usando o índice $V$ em $L_V(M)$ para significar *reconhecimento por pilha vazia*, segue tal definição alternativa, observando a ausência do conjunto de estados finais.

**Definição 34** *Seja um APN $M = (E, \Sigma, \Gamma, \delta, I)$. A* linguagem reconhecida *por $M$* por pilha vazia *é*

$$L_V(M) = \{w \in \Sigma^* \mid [i, w, \lambda] \overset{*}{\vdash} [e, \lambda, \lambda] \text{ para algum } i \in I\}.$$

*Uma palavra $w$ tal que $[i, w, \lambda] \overset{*}{\vdash} [e, \lambda, \lambda]$, onde $i \in I$, é dita ser* reconhecida *(*aceita*) por $M$* por pilha vazia. □

Note que, por esta definição, $\lambda$ é sempre reconhecida, já que a pilha começa vazia. Será mostrado também, no Teorema 16, que uma linguagem com a palavra $\lambda$ pode ser reconhecida por pilha vazia e estado final se, e somente se, pode ser reconhecida por pilha vazia.

O APN cujo diagrama de estados está mostrado na Figura 3.12(a) reconhece $\{\mathtt{0}^m\mathtt{1}^n \mid m \geq n\}$ também por pilha vazia, visto que *todos* os seus estados são estados finais. Aliás, se um APN $M = (E, \Sigma, \Gamma, \delta, I)$ reconhece $L_V(M)$, então o APN $M' = (E, \Sigma, \Gamma, \delta, I, E)$ (observe que todos os estados são finais em $M'$) reconhece $L_V(M)$ por estado final e pilha vazia.

**Exemplo 100** Seja o APN cujo diagrama de estados está mostrado na Figura 3.13. Tal APN reconhece a linguagem $\{\mathtt{0}^m\mathtt{1}^n \mid m \leq n\}$ por pilha vazia. Considerando todos os seus estados como sendo estados finais, ele reconhece a mesma linguagem por estado final e pilha vazia. □

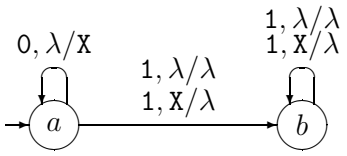



Figura 3.13: APN para $\{0^m1^n \mid m \leq n\}$.

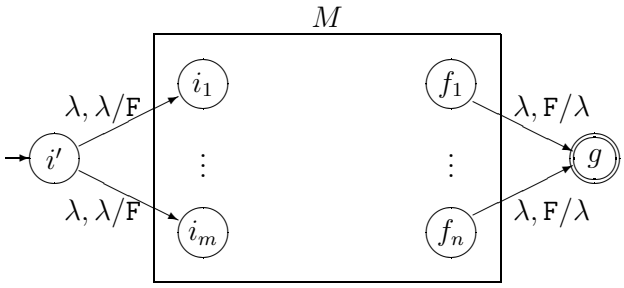



Figura 3.14: Obtenção de APN, parte (a) $\rightarrow$ (b).

O teorema abaixo mostra a equivalência dos três métodos de reconhecimento.

**Teorema 16** *Seja $L$ uma linguagem. As seguinte afirmativas são equivalentes:*

*(a) $L$ pode ser reconhecida por estado final e pilha vazia.*

*(b) $L$ pode ser reconhecida por estado final.*

*(c) $L \cup \{\lambda\}$ pode ser reconhecida por pilha vazia.*

*Prova*

(a) $\rightarrow$ (b).

Seja um APN $M = (E, \Sigma, \Gamma, \delta, I, F)$. É possível obter, a partir de $M$, um APN $M'$ tal que $L_F(M') = L(M)$. A idéia central é utilizar um símbolo de pilha novo para marcar o fundo da pilha, de forma que $M'$ possa reconhecer quando $M$ estaria com a pilha vazia.

Serão usados, além dos estados em $E$, mais dois estados $i', g \notin E$, e, além dos símbolos de $\Gamma$, mais um símbolo de pilha $\mathtt{F} \notin \Gamma$. Basta fazer $M' = (E \cup \{i', g\}, \Sigma, \Gamma \cup \{\mathtt{F}\}, \delta', \{i'\}, \{g\})$ (veja a Figura 3.14 para uma representação esquemática de $M'$), onde $\delta'$ inclui $\delta$ mais as seguintes transições:

- para cada $i_k \in I$, $\delta'(i', \lambda, \lambda) = \{[i_k, \mathtt{F}]\}$;
- para cada $f_j \in F$, $\delta'(f_j, \lambda, \mathtt{F}) = \{[g, \lambda]\}$.

(b) $\rightarrow$ (c).

Seja um APN $M = (E, \Sigma, \Gamma, \delta, I, F)$. Um APN $M'$ tal que $L_V(M') = L_F(M) \cup \{\lambda\}$ seria $M' = (E \cup \{i', g, h\}, \Sigma, \Gamma \cup \{\mathtt{F}\}, \delta', \{i'\})$ (veja a Figura 3.15 para uma representação esquemática de $M'$), onde $i', g, h \notin E$, $\mathtt{F} \notin \Gamma$ e $\delta'$ inclui $\delta$ mais as seguintes transições:

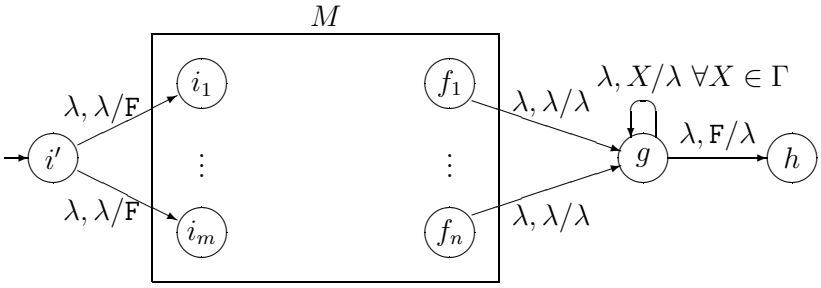



Figura 3.15: Obtenção de APN, parte (b) → (c).

- para cada $i_k \in I$, $\delta'(i', \lambda, \lambda) = \{[i_k, \mathtt{F}]\}$;
- para cada $f_j \in F$, $\delta'(f_j, \lambda, \lambda) = \{[g, \lambda]\}$;
- para cada $X \in \Gamma$, $\delta(g, \lambda, X) = \{[g, \lambda]\}$;
- $\delta(g, \lambda, \mathtt{F}) = \{[h, \lambda]\}$.

O símbolo de pilha `F` é utilizado aqui para evitar que a pilha fique vazia, exceto quando a palavra deva ser reconhecida. A pilha fica vazia se, e somente se, é atingido o estado $h$.

(c) → (a).

Como já ficou ressaltado, um APN que reconhece por pilha vazia é um APN que reconhece por pilha vazia e estado final, bastando considerar todos os seus estados como estados finais. Ou seja, se $M = (E, \Sigma, \Gamma, \delta, I)$, um APN $M'$ tal $L(M') = L_V(M)$ seria, então, $M' = (E, \Sigma, \Gamma, \delta, I, E)$. Observe que, como $L_V(M)$ contém $\lambda$, $L(M')$ também contém. □

Daqui para frente, será usada também a expressão AP para designar autômato com pilha (não determinístico).

# Exercícios

1. Seja o AP $M = (\{i, f\}, \{\mathtt{a}, \mathtt{b}\}, \{\mathtt{B}, \mathtt{C}\}, \delta, \{i\}, \{f\})$, onde $\delta$ é dada por:

   $\delta(i, \mathtt{a}, \lambda) = [i, \mathtt{B}]$
   $\delta(i, \lambda, \lambda) = [f, \lambda]$
   $\delta(f, \mathtt{b}, \mathtt{B}) = [f, \mathtt{C}]$
   $\delta(f, \mathtt{c}, \mathtt{C}) = [f, \lambda]$.

   (a) Exiba as computações para as palavras `aa`, `bb`, `aabcc` e `aabcbc`. Quais destas palavras são reconhecidas por $M$?

   (b) Que linguagem é reconhecida por $M$?

2. Construa um AP com um alfabeto de pilha contendo apenas dois símbolos, que reconheça $\{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \mathtt{d}\}^* \mid w = w^R\}$.

3. Construa APN's que reconheçam as linguagens seguintes por estado final e pilha vazia:

   (a) $\{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\} \cup \{\mathtt{0}^n\mathtt{1}^{2n} \mid n \geq 0\}$.

   (b) $\{\mathtt{0}^n\mathtt{1}^k \mid n \leq k \leq 2n\}$.

   (c) $\{\mathtt{0}^n\mathtt{1}^n\mathtt{0}^k \mid n, k \geq 0\}$.

   (d) $\{\mathtt{0}^m\mathtt{1}^n \mid m > n\}$.

4. Construa APD's que reconheçam $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$:

   (a) por estado final;

   (b) por pilha vazia.

5. Construa APN's que reconheçam as linguagens do Exercício 3:

   (a) por estado final;

   (b) por pilha vazia.

6. Mostre que um APN em que é empilhado no máximo um símbolo por transição tem o mesmo poder que um APN normal.

7. Mostre como simular um AFNE por meio de um APN.

8. Obtenha um APD que reconheça *por estado final* a linguagem

   $$L = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid \text{o número de } \mathtt{0}\text{'s em } w \text{ difere do de } \mathtt{1}\text{'s}\}.$$

   A partir dele, obtenha um APD que reconheça $L\{\#\}$ *por estado final e pilha vazia*.

9. Mostre que toda linguagem regular pode ser reconhecida por algum APD sob qualquer um dos três critérios de reconhecimento. Para isto, mostre como obter, a partir de qualquer AFD, APD's equivalentes para cada um dos critérios de reconhecimento.

## 3.4 Gramáticas Livres do Contexto

O seguinte trecho é um pedaço de uma gramática livre do contexto, na notação BNF[4], que define uma parte da sintaxe de uma linguagem de programação similar àquela que é utilizada para a apresentação dos algoritmos deste texto:

[4] *Backus-Naur Form.*

$$
\begin{array}{rcl}
\langle \text{programa} \rangle & \rightarrow & \langle \text{declarações} \rangle \ ; \ \langle \text{lista-de-cmds} \rangle \ . \\
& \vdots & \\
\langle \text{lista-de-cmds} \rangle & \rightarrow & \langle \text{comando} \rangle \ ; \ \langle \text{lista-de-cmds} \rangle \ | \\
& & \langle \text{comando} \rangle \\
\langle \text{comando} \rangle & \rightarrow & \langle \text{cmd-enquanto} \rangle \ | \\
& & \langle \text{cmd-se} \rangle \ | \\
& & \langle \text{cmd-atribuição} \rangle \ | \\
& & \langle \text{cmd-nulo} \rangle \ | \\
& & \ldots \\
\langle \text{cmd-enquanto} \rangle & \rightarrow & \textbf{enquanto} \ \langle \text{exp-lógica} \rangle \ \textbf{faça} \\
& & \langle \text{lista-de-cmds} \rangle \ \textbf{fimenquanto} \\
\langle \text{cmd-se} \rangle & \rightarrow & \textbf{se} \ \langle \text{exp-lógica} \rangle \ \textbf{então} \\
& & \langle \text{lista-de-cmds} \rangle \ \langle \text{senaoses} \rangle \ \langle \text{senao} \rangle \ \textbf{fimse} \\
\langle \text{senaoses} \rangle & \rightarrow & \textbf{senãose} \ \langle \text{exp-lógica} \rangle \ \textbf{então} \\
& & \langle \text{lista-de-cmds} \rangle \ \langle \text{senaoses} \rangle \ | \\
& & \lambda \\
\langle \text{senao} \rangle & \rightarrow & \textbf{senão} \ \langle \text{lista-de-cmds} \rangle \ | \\
& & \lambda \\
\langle \text{cmd-atribuição} \rangle & \rightarrow & \langle \text{variável} \rangle \leftarrow \langle \text{expressão} \rangle \\
\langle \text{cmd-nulo} \rangle & \rightarrow & \lambda
\end{array}
$$

Nesta notação, as variáveis figuram entre "⟨" e "⟩". Os outros símbolos são terminais; por ordem de ocorrência: ";", **enquanto**, **faça**, **fimenquanto**, **se**, **então**, **fimse**, **senãose**, "←".

Cada regra de uma gramática livre do contexto tem no lado esquerdo apenas uma variável. No lado direito pode ser colocada uma palavra qualquer constituída de variáveis e/ou terminais.

Existem programas que aceitam uma gramática livre do contexto no formato BNF e produzem um analisador sintático para a mesma. Apesar da notação BNF ser comumente mais adequada para a descrição de linguagens que ocorrem na prática, como linguagens de programação, a notação formal a ser introduzida na próxima seção é mais adequada para o estudo de gramáticas livres do contexto em geral. Após isto, na Seção 3.4.2, serão apresentados os conceitos de árvore de derivação e ambiguidade de gramáticas, muito importantes por terem grande repercussão em aplicações que envolvem o uso de gramáticas como base no processamento de linguagens. Depois, na Seção 3.4.3 será abordado o problema de manipular as regras de uma gramática com o objetivo de que a gramática resultante tenha certas características. Para finalizar, na Seção 3.4.4 será mostrada a equivalência dos formalismos de gramáticas livres do contexto e autômatos com pilha.

### 3.4.1 Definição e exemplos.

Segue a definição de gramática livre do contexto.

**Definição 35** *Uma* gramática livre do contexto *(GLC) é uma gramática* $(V, \Sigma, R, P)$*, em que cada regra tem a forma* $X \rightarrow w$*, onde* $X \in V$ *e* $w \in (V \cup \Sigma)^*$*.* □

Para uma GLC, em cada passo de uma derivação deve-se escolher, na forma sentencial, a variável $A$ a ser substituída pelo lado direito de uma regra com $A$ do lado esquerdo. Ao se fazer tal substituição, diz-se que $A$ é *expandida*.

Observe que uma gramática regular é uma gramática livre do contexto especial em que uma única variável pode ser expandida em qualquer forma sentencial. Por outro lado, existem linguagens que não são regulares, e que portanto não podem ser geradas por GR's, mas que podem ser geradas por GLC's. A seguir são vistos alguns exemplos.

**Exemplo 101** A linguagem não regular $\{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbf{N}\}$ é gerada pela GLC $G = (\{P\}, \{\mathtt{0}, \mathtt{1}\}, R, P)$, onde $R$ consta das duas regras:

$$P \to \mathtt{0}P\mathtt{1} \mid \lambda$$

As únicas palavras geradas por tal gramática são aquelas que podem ser geradas por $n$ aplicações da regra $P \to \mathtt{0}P\mathtt{1}$, $n \geq 0$, seguidas de uma aplicação da regra $P \to \lambda$. Esquematicamente: $P \stackrel{n}{\Rightarrow} \mathtt{0}^n P \mathtt{1}^n \Rightarrow \mathtt{0}^n\mathtt{1}^n$. Logo, $L(G) = \{\mathtt{0}^n\mathtt{1}^n \mid n \in \mathbf{N}\}$. □

**Exemplo 102** A gramática $G$ a seguir gera os palíndromos sobre $\{\mathtt{0}, \mathtt{1}\}$, ou seja, $L(G) = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w = w^R\}$. $G = (\{P\}, \{\mathtt{0}, \mathtt{1}\}, R, P)$, onde $R$ consta das cinco regras:

$$P \to \mathtt{0}P\mathtt{0} \mid \mathtt{1}P\mathtt{1} \mid \mathtt{0} \mid \mathtt{1} \mid \lambda$$

Aplicando-se as duas primeiras regras, gera-se qualquer forma sentencial do tipo $wPw^R$, para $w \in \{\mathtt{0}, \mathtt{1}\}^*$. Por fim, para gerar uma palavra, aplica-se uma das 3 últimas regras; a última, quando a palavra tem tamanho par, e uma das outras duas, quando ela tem tamanho ímpar. □

**Exemplo 103** A linguagem $L = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w$ tem um número igual de 0's e 1's$\}$ é gerada pela gramática $G = (\{P\}, \{\mathtt{0}, \mathtt{1}\}, R, P)$, onde $R$ consta das três regras:

$$P \to \mathtt{0}P\mathtt{1}P \mid \mathtt{1}P\mathtt{0}P \mid \lambda$$

Como o lado direito de cada uma das 3 regras tem número igual de 0's e 1's, $G$ só gera palavras de $L$. O fato de que $G$ gera *todas* as palavras de $L$, vem do fato de que para toda palavra $w$ de $L$, tem-se um dos três casos:

(a) $w = \lambda$; neste caso, basta aplicar a regra $P \to \lambda$;

(b) $w = \mathtt{0}y$ para algum $y \in \{\mathtt{0}, \mathtt{1}\}^*$ e $y$ tem um 1 a mais que 0's; neste caso, $y$ é da forma $x\mathtt{1}z$, onde $x$ tem número igual de 0's e 1's e $z$ também tem número igual de 0's e 1's; assim, basta iniciar a derivação de $w$ com a primeira regra;

(c) $w = \mathtt{1}y$ para algum $y \in \{\mathtt{0}, \mathtt{1}\}^*$ e $y$ tem um 0 a mais que 1's; por motivo análogo ao segundo caso, basta aplicar a segunda regra.

Nos casos (b) e (c), tem-se novamente (recursivamente) os três casos aplicados para as subpalavras $x$ e $z$. Com isto, tem-se um método para construir uma derivação de qualquer palavra de $L$. A seguinte derivação de 01010110 ilustra a aplicação do método subjacente, onde a variável expandida é sempre a mais à esquerda:

$$
\begin{array}{lll}
P \Rightarrow & \mathtt{0}P\mathtt{1}P & P \to \mathtt{0}P\mathtt{1}P \ (x = \mathtt{10};\ y = \mathtt{0110}) \\
\Rightarrow & \mathtt{01}P\mathtt{0}P\mathtt{1}P & P \to \mathtt{1}P\mathtt{0}P \ (x = \lambda;\ y = \lambda) \\
\Rightarrow & \mathtt{010}P\mathtt{1}P & P \to \lambda \\
\Rightarrow & \mathtt{0101}P & P \to \lambda \\
\Rightarrow & \mathtt{01010}P\mathtt{1}P & P \to \mathtt{0}P\mathtt{1}P \ (x = \lambda;\ y = \mathtt{10}) \\
\Rightarrow & \mathtt{010101}P & P \to \lambda \\
\Rightarrow & \mathtt{0101011}P\mathtt{0}P & P \to \mathtt{1}P\mathtt{0}P \ (x = \lambda;\ y = \lambda) \\
\Rightarrow & \mathtt{01010110}P & P \to \lambda \\
\Rightarrow & \mathtt{01010110} & P \to \lambda.
\end{array}
$$

□

O exemplo a seguir ilustra uma gramática que contém a essência da especificação da sintaxe das expressões aritméticas das linguagens de programação usuais.

**Exemplo 104** Seja a GLC $(\{E, T, F\}, \{\mathtt{t}, +, *, (, )\}, R, E)$, para expressões aritméticas, onde $R$ consta das regras:

$$E \to E{+}T \mid T$$

$$T \to T{*}F \mid F$$

$$F \to (E) \mid \mathtt{t}$$

As duas primeiras regras dizem que uma expressão aritmética ($E$) é constituída de um ou mais termos ($T$'s) somados. As duas seguintes dizem que um termo é constituído de um ou mais fatores ($F$'s) multiplicados. E as duas últimas dizem que um fator é terminal `t` ou, recursivamente, uma expressão aritmética entre parênteses. Esta gramática será utilizada em vários exemplos daqui para frente. □

Mais à frente, na Seção 3.4.4, será mostrado que as linguagens geradas por gramáticas livres do contexto são exatamente as reconhecidas por autômatos com pilha. A definição a seguir dá um nome à classe formada por tais linguagens.

**Definição 36** *Uma linguagem é dita ser uma* linguagem livre do contexto *se existe uma gramática livre do contexto que a gera.* □

Em geral, existem várias derivações de uma mesma palavra da linguagem gerada por uma gramática. Note que no Exemplo 103, página 157, inicia-se uma derivação de `01010110` por

$$P \Rightarrow \mathtt{0}P\mathtt{1}P \quad (\text{regra } P \to \mathtt{0}P\mathtt{1}P).$$

E após isto, a variável expandida é sempre a *mais à esquerda*. Pode-se ver que a mesma palavra pode ser derivada expandindo-se sempre a variável *mais à direita* ao invés da variável mais à esquerda. E mais: a mesma palavra pode ser derivada expandindo-se variáveis em ordem aleatória. Isto mostra que existem várias derivações distintas para a palavra `01010110`. O que tais derivações têm em comum? Este assunto será abordado na próxima seção.

### 3.4.2 Derivações e ambigüidade

Um conceito bastante útil, base para muitas implementações de compiladores de linguagens de programação, é o de *árvore de derivação* (AD). Uma AD captura a essência de uma derivação, a história da obtenção de uma forma sentencial que não depende da ordem de aplicação das regras da GLC. A cada derivação irá corresponder uma única AD, mas a uma AD irá corresponder, quase sempre, uma quantidade muito grande de derivações. Assim, pode-se dizer que as AD's particionam o conjunto de todas as derivações de uma GLC em "derivações equivalentes": duas derivações seriam equivalentes se, e somente se, correspondessem à mesma AD.

**Definição 37** *Seja uma GLC* $G = (V, \Sigma, R, P)$*. Uma árvore de derivação (AD) de uma forma sentencial de* $G$ *é uma árvore ordenada construída recursivamente como segue:*

*(a) uma árvore sem arestas cujo único vértice tem rótulo* $P$ *é uma AD de* $P$*;*

*(b) se* $X \in V$ *é rótulo de uma folha* $f$ *de uma AD* $A$*, então:*

*(b.1) se* $X \to \lambda \in R$*, então a árvore obtida acrescentando-se a* $A$ *mais um vértice* $v$ *com rótulo* $\lambda$ *e uma aresta* $\{f, v\}$ *é uma AD;*

*(b.2) se* $X \to x_1 x_2 \ldots x_n \in R$*, onde* $x_1, x_2, \ldots, x_n \in V \cup \Sigma$*, então a árvore obtida acrescentando-se a* $A$ *mais* $n$ *vértices* $v_1$*,* $v_2$*, ...,* $v_n$ *com rótulos* $x_1$*,* $x_2$*, ...,* $x_n$*,* nesta ordem*, e* $n$ *arestas* $\{f, v_1\}$*,* $\{f, v_2\}$*, ...,* $\{f, v_n\}$*, é uma AD.*

*Se a seqüência dos rótulos da fronteira da AD é a forma sentencial* $w$*, diz-se que a AD é uma árvore de derivação de* $w$*.* □

**Exemplo 105** Seja a gramática do exemplo 104, cujas regras são reproduzidas abaixo:

$E \to E{+}T \mid T$

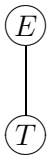
$T \to T{*}F \mid F$

$F \to (E) \mid \texttt{t}$

Na Figura 3.16 mostra-se uma AD de `t`$*$(`t`+`t`). Para a construção de tal árvore, tomou-se como ponto de partida a derivação:

$E \Rightarrow\ T$ (regra $E \to T$)

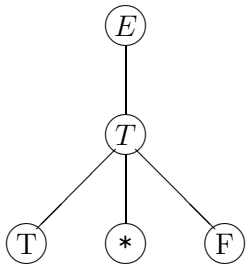
produzindo-se:




Em seguida, a derivação evoluiu para:

$E \Rightarrow\ T$ (regra $E \to T$)
$\Rightarrow\ T{*}F$ (regra $T \to T{*}F$)

e a árvore correspondente para:



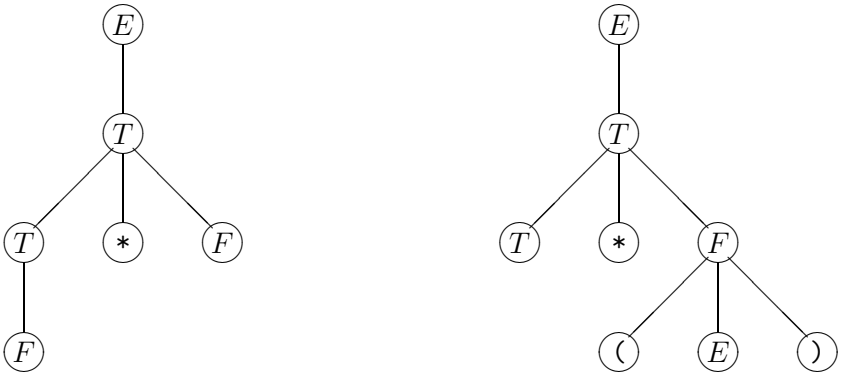

Neste instante, tinha-se duas opções para continuar a derivação:

$$\begin{array}{lll} E \Rightarrow & T & (\text{regra } E \to T) \\ \Rightarrow & T*F & (\text{regra } T \to T*F) \\ \Rightarrow & F*F & (\text{regra } T \to F) \end{array}$$

ou então:

$$\begin{array}{lll} E \Rightarrow & T & (\text{regra } E \to T) \\ \Rightarrow & T*F & (\text{regra } T \to T*F) \\ \Rightarrow & T*(E) & (\text{regra } F \to (E)). \end{array}$$

À esquerda, mostra-se a AD correspondente à primeira derivação, e à direita, a AD correspondente à segunda derivação::




Prosseguindo-se por qualquer uma destas alternativas, chega-se, após uma derivação de 11 passos, à AD mostrada na Figura 3.16. □

Observe que o número de passos de qualquer derivação que leva a uma AD $X$ é o número de vértices internos de $X$, já que a cada vértice interno corresponde a aplicação de uma regra (e vice-versa).

Muitas vezes, a estrutura da árvore de derivação é utilizada para associar significado para as sentenças de uma linguagem, de forma similar ao que se faz em análise sintática de sentenças na língua portuguesa (onde se indentifica sujeito, verbo, predicado, etc.). Em português, se a mesma sentença pode ser desmembrada de mais de uma forma durante a análise, então ela tem vários significados, e diz-se que ela é ambígüa. De forma análoga, se existir mais de uma AD de uma mesma palavra, provavelmente ela terá mais de um significado. Isto inspira a definição a seguir.

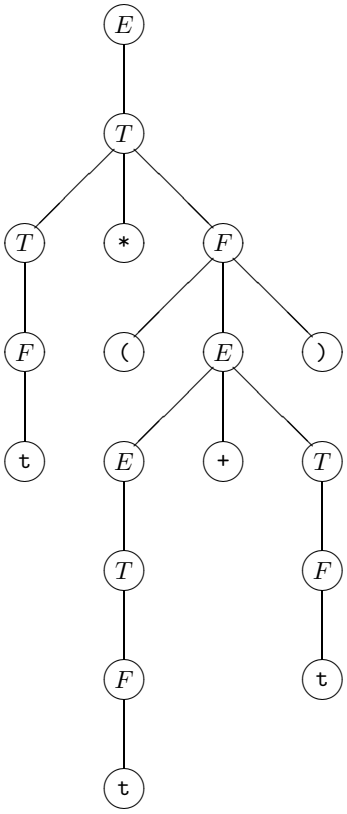



Figura 3.16: Uma árvore de derivação.

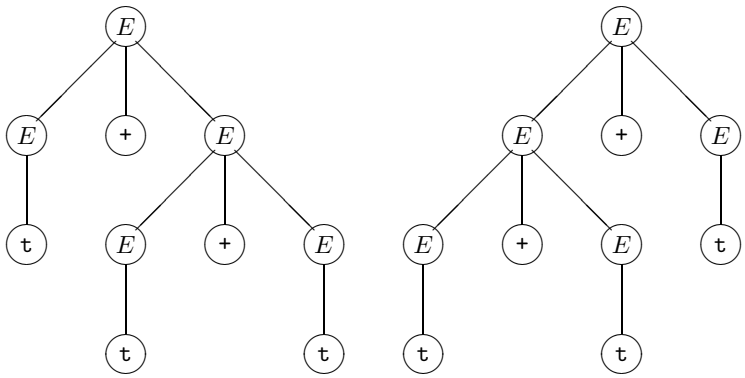



Figura 3.17: Duas árvores de derivação para $t + t + t$.

**Definição 38** *Uma GLC é dita* ambígüa *quando existe mais de uma AD para alguma sentença que ela gera.* □

Observe que é a gramática que é dita ambígua. Não é a linguagem que ela gera, nem as sentenças para as quais haja mais de uma AD. Afinal, podem haver outras GLC's equivalentes a uma GLC ambígüa que não sejam ambígüas.

Exemplos de gramáticas não ambígüas são todas aquelas gramáticas dos Exemplos 101 a 104 da Seção 3.4.1, exceto a do Exemplo 103.

O próximo exemplo apresenta uma gramática ambígüa que gera a linguagem de expressões aritméticas que é gerada também pela gramática do Exemplo 104.

**Exemplo 106** Seja a gramática $G = (\{E\}, \{\mathtt{t}, +, *, (, )\}, R, E)$, para expressões aritméticas, onde $R$ consta das regras:

$$E \to E{+}E \mid E{*}E \mid (E) \mid \mathtt{t}$$

Tal gramática é ambígüa, já que existem duas AD's da palavra `t+t+t`, as quais estão mostradas na Figura 3.17. À árvore da esquerda corresponde, dentre outras, a derivação:

$$\begin{aligned}
E &\Rightarrow E{+}E && \text{(regra } E \to E{+}E\text{)}\\
&\Rightarrow \mathtt{t}{+}E && \text{(regra } E \to \mathtt{t}\text{)}\\
&\Rightarrow \mathtt{t}{+}E{+}E && \text{(regra } E \to E{+}E\text{)}\\
&\Rightarrow \mathtt{t}{+}\mathtt{t}{+}E && \text{(regra } E \to \mathtt{t}\text{)}\\
&\Rightarrow \mathtt{t}{+}\mathtt{t}{+}\mathtt{t} && \text{(regra } E \to \mathtt{t}\text{).}
\end{aligned}$$

À árvore da direita corresponde a seguinte derivação, dentre outras:

$$\begin{aligned}
E &\Rightarrow E{+}E && \text{(regra } E \to E{+}E\text{)}\\
&\Rightarrow E{+}E{+}E && \text{(regra } E \to E{+}E\text{)}\\
&\Rightarrow \mathtt{t}{+}E{+}E && \text{(regra } E \to \mathtt{t}\text{)}\\
&\Rightarrow \mathtt{t}{+}\mathtt{t}{+}E && \text{(regra } E \to \mathtt{t}\text{)}\\
&\Rightarrow \mathtt{t}{+}\mathtt{t}{+}\mathtt{t} && \text{(regra } E \to \mathtt{t}\text{).}
\end{aligned}$$

Como já foi dito acima, normalmente o significado é associado a uma palavra de acordo com a AD obtida. Por exemplo, a AD do lado esquerdo da Figura 3.17 leva à interpretação de `t+t+t` como sendo a soma de um elemento com a soma de dois elementos, isto é, $t+(t+t)$, enquanto que a AD do lado direito da mesma figura leva à interpretação de `t+t+t` como sendo a soma da soma de dois elementos com um elemento, isto é, $(t+t)+t$. □

Dentre as derivações correspondentes a uma AD, existem duas de particular interesse: as *derivações mais à esquerda* e as *derivações mais à direita*.

**Definição 39** *Uma derivação é dita mais à esquerda (DME) se em cada passo é expandida a variável mais à esquerda. E é dita mais à direita (DMD) se em cada passo é expandida a variável mais à direita. Para enfatizar que uma derivação é mais à esquerda, pode-se usar o símbolo $\Rightarrow_E$ ao invés de $\Rightarrow$, e para uma derivação mais à direita pode-se usar $\Rightarrow_D$.* □

Existe uma única DME e uma única DMD correspondente a uma AD: para obter a DME a partir de uma AD, basta ir gerando os passos de derivação à medida em que se percorre a AD visitando primeiro as subárvores à esquerda, antes de visitar as subárvores à direita; para obter a DMD, visita-se primeiro as subárvores à direita. Assim sendo, pode-se dizer que:

> uma GLC é ambígüa se, e somente se existe mais de uma DME para alguma sentença que ela gera.

e também que:

> uma GLC é ambígüa se, e somente se existe mais de uma DMD para alguma sentença que ela gera.

**Exemplo 107** No Exemplo 106 foram mostradas as duas DME's que correspondem às AD's da Figura 3.17. As duas DMD's que correspondem às mesmas AD's são, para a primeira AD:

$$\begin{array}{lll} E \Rightarrow_D & E+E & (\text{regra } E \rightarrow E+E) \\ \Rightarrow_D & E+E+E & (\text{regra } E \rightarrow E+E) \\ \Rightarrow_D & E+E+\texttt{t} & (\text{regra } E \rightarrow \texttt{t}) \\ \Rightarrow_D & E+\texttt{t}+\texttt{t} & (\text{regra } E \rightarrow \texttt{t}) \\ \Rightarrow_D & \texttt{t}+\texttt{t}+\texttt{t} & (\text{regra } E \rightarrow \texttt{t}) \end{array}$$

e para a segunda AD:

$$\begin{array}{lll} E \Rightarrow_D & E+E & (\text{regra } E \rightarrow E+E) \\ \Rightarrow_D & E+\texttt{t} & (\text{regra } E \rightarrow \texttt{t}) \\ \Rightarrow_D & E+E+\texttt{t} & (\text{regra } E \rightarrow E+E) \\ \Rightarrow_D & E+\texttt{t}+\texttt{t} & (\text{regra } E \rightarrow \texttt{t}) \\ \Rightarrow_D & \texttt{t}+\texttt{t}+\texttt{t} & (\text{regra } E \rightarrow \texttt{t}). \end{array}$$

□

Existem linguagens livres do contexto (LLC's) para as quais existem apenas gramáticas ambígüas. Tais linguagens são denominadas *linguagens inerentemente ambígüas*. Um exemplo de linguagem inerentemente ambígüa é $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^k \mid m = n \text{ ou } n = k\}$. Pode-se mostrar que qualquer GLC que gere tal linguagem terá mais de uma AD para palavras da forma $\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n$.

A detecção e remoção de ambigüidade em GLC's é muito importante, por exemplo, como um passo prévio ao uso de uma gramática para geração de um compilador para uma linguagem de programação. Na próxima seção serão vistas algumas técnicas de modificação de GLC's, que não alteram a linguagem gerada. No entanto, infelizmente, o problema de determinar se uma GLC arbitrária é ambígüa é indecidível, como será mostrado no Capítulo 5.

Existem dois tipos básicos de analisadores sintáticos gerados[5] a partir de GLC's: o *bottom-up* e o *top-down*. Um analisador *bottom-up* parte do programa, lendo-o da esquerda para a direita, e aplica as regras de forma invertida, construindo a AD da fronteira para a raiz, ou seja, *bottom-up*; a derivação considerada durante o processo é uma DMD (obtida de trás para frente). Por outro lado, um analisador *top-down* parte do símbolo de partida da GLC e constrói a AD da raiz em direção à fronteira, ou seja, *top-down*; a derivação considerada é uma DME. Detalhes estariam fora do escopo deste texto, e podem ser encontrados em qualquer livro texto sobre construção de compiladores. De qualquer forma fica evidenciada a importância dos três conceitos do ponto de vista prático: AD, DME e DMD.

A mesma linguagem pode ser gerada por inúmeras gramáticas. Algumas gramáticas podem ser mais adequadas do que outras, dependendo do contexto para o qual elas foram projetadas. Assim, é importante saber algumas técnicas de manipulação de GLC's de forma a obter GLC's equivalentes, mas com a presença ou ausência de certa(s) característica(s) relevantes para certa aplicação. Em particular, existem algumas *formas normais* de GLC's que são apropriadas em diversas situações, como, por exemplo, quando se pretende mostrar que uma certa propriedade vale para todas as linguagens geradas por GLC's. Na próxima seção, serão vistas algumas técnicas de manipulação de GLC's, assim como duas formas normais importantes.

### 3.4.3 Manipulação de gramáticas e formas normais

A detecção de variáveis que nunca participam de derivações de palavras da linguagem gerada por uma GLC, as chamadas *variáveis inúteis*, é importante por vários motivos. Por exemplo, em gramáticas grandes, como gramáticas de linguagens de programação, pode acontecer de se esquecer de definir as regras relativas a uma variável; ou então, uma variável, apesar de ter suas regras já definidas, pode não ter sido utilizada ainda na formação de novas regras. Ambos os tipos de variáveis inúteis podem ser detectados. Após a detecção das variáveis inúteis, pode-se acrescentar novas regras para prever a definição ou uso das variáveis. Ou então, caso uma variável seja efetivamente inútil, deve-se eliminar todas as regras que possuem alguma ocorrência da mesma.

Segue uma definição precisa de variável útil, assim como algoritmos para detecção de variáveis inúteis e um método para eliminar todas as variáveis inúteis de uma GLC.

[5]Um analisador sintático é um programa cujo objetivo principal é determinar se um programa está sintaticamente correto.

**Definição 40** *Seja uma GLC $G = (V, \Sigma, R, P)$. Uma variável $X \in V$ é dita ser uma* variável útil *se, e somente se, existem $u, v \in (V \cup \Sigma)^*$ tais que:*

$$P \stackrel{*}{\Rightarrow} uXv \text{ e existe } w \in \Sigma^* \text{ tal que } uXv \stackrel{*}{\Rightarrow} w.$$

□

Observe que, pela Definição 40, para a variável $X$ ser útil é necessário, não apenas que existam $u$ e $v$ tais que $P \stackrel{*}{\Rightarrow} uXv$, mas também que $uXv \stackrel{*}{\Rightarrow} w$ para algum $u$ e algum $v$ tais que $P \stackrel{*}{\Rightarrow} uXv$ e $w \in \Sigma^*$.

**Exemplo 108** Seja a gramática $(\{P, A, B, C\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, R, P)$, onde $R$ contém as regras:

$P \rightarrow AB \mid \mathtt{a}$

$B \rightarrow \mathtt{b}$

$C \rightarrow \mathtt{c}$

- $C$ é inútil: não existem $u$ e $v$ tais que $P \stackrel{*}{\Rightarrow} uCv$;
- $A$ é inútil: não existe $w \in \Sigma^*$ tal que $A \stackrel{*}{\Rightarrow} w$;
- $B$ é inútil: $P \stackrel{*}{\Rightarrow} uBv$ apenas para $u = A$ e $v = \lambda$, e não existe $w \in \Sigma^*$ tal que $AB \stackrel{*}{\Rightarrow} w$.

Eliminando-se estas variáveis e todas as regras que as referenciam, além dos terminais `b` e `c` que não ocorrem em nenhuma regra retida[6], tem-se a gramática equivalente $(\{P\}, \{\mathtt{a}\}, \{P \rightarrow \mathtt{a}\}, P)$. □

A notação $\Rightarrow_G$, sendo $G$ uma gramática, será usada para informar que a derivação está sendo tomada com relação à gramática $G$.

**Teorema 17** *Seja uma GLC $G$ tal que $L(G) \neq \emptyset$. Existe uma GLC, equivalente a $G$, sem variáveis inúteis.*

*Prova*

Seja $G = (V, \Sigma, R, P)$ tal que $L(G) \neq \emptyset$. Uma GLC $G''$ equivalente a $G$, sem variáveis inúteis, pode ser construída em dois passos:

(a) obtenha $G' = (V', \Sigma, R', P)$, onde:

- $V' = \{X \in V \mid X \stackrel{*}{\Rightarrow}_G w \text{ e } w \in \Sigma^*\}$, e
- $R' = \{r \in R \mid r \text{ não contém símbolo de } V - V'\}$;

(b) obtenha $G'' = (V'', \Sigma, R'', P)$, onde:

- $V'' = \{X \in V' \mid P \stackrel{*}{\Rightarrow}_{G'} uXv\}$, e
- $R'' = \{r \in R' \mid r \text{ não contém símbolo de } V' - V''\}$;

Entrada: uma GLC $G = (V, \Sigma, R, P)$.
Saída: $\mathcal{I}_1 = \{X \in V \mid X \stackrel{*}{\Rightarrow} w \text{ e } w \in \Sigma^*\}$.
$\mathcal{I}_1 \leftarrow \emptyset$;
**repita**
    $\mathcal{N} \leftarrow \{Y \notin \mathcal{I}_1 \mid Y \rightarrow z \in R \text{ e } z \in (\mathcal{I}_1 \cup \Sigma)^*\}$;
    $\mathcal{I}_1 \leftarrow \mathcal{I}_1 \cup \mathcal{N}$
**até** $\mathcal{N} = \emptyset$;
retorne $\mathcal{I}_1$.

(a)

Entrada: uma GLC $G = (V, \Sigma, R, P)$.
Saída: $\mathcal{I}_2 = \{X \in V \mid P \stackrel{*}{\Rightarrow} uXv\}$.
$\mathcal{I}_2 \leftarrow \emptyset$; $\mathcal{N} \leftarrow \{P\}$;
**repita**
    $\mathcal{I}_2 \leftarrow \mathcal{I}_2 \cup \mathcal{N}$;
    $\mathcal{N} \leftarrow \{Y \notin \mathcal{I}_2 \mid X \rightarrow uYv \text{ para algum } X \in \mathcal{N}\}$
**até** $\mathcal{N} = \emptyset$;
retorne $\mathcal{I}_2$.

(b)

Figura 3.18: Algoritmos para achar variáveis úteis.

Alternativamente a $\Sigma$, pode-se considerar como alfabeto de $G''$ o conjunto daqueles terminais que ocorrem em regras de $R''$.

Os algoritmos das Figuras 3.18(a) e 3.18(b) determinam, repectivamente, os conjuntos $\{X \in V \mid X \stackrel{*}{\Rightarrow} w \text{ e } w \in \Sigma^*\}$ e $\{X \in V \mid P \stackrel{*}{\Rightarrow} uXv\}$.

Inicialmente, veja que $L(G') = L(G)$, pois apenas as regras de $G$ cujas variáveis $X$ são tais que $X \stackrel{*}{\Rightarrow} w$, para algum $w \in \Sigma^*$, podem contribuir para a geração de alguma palavra de $L(G)$; e $G'$ contém exatamente tais regras. Analogamente, $L(G'') = L(G')$. Resta então mostrar que $G''$ não tem variáveis inúteis, ou seja, que todas as suas variáveis são úteis. Para isto, seja uma variável arbitrária $X \in V''$. Em primeiro lugar, tem-se que $P \stackrel{*}{\Rightarrow}_{G''} uXv$, por construção de $R''$. E para qualquer $uXv$ tem-se que $uXv \stackrel{*}{\Rightarrow}_{G''} w$ e $w \in \Sigma^*$, pois todas as variáveis $Y$ da forma sentencial $uXv$ são tais que $Y \stackrel{*}{\Rightarrow} y$ e $y \in \Sigma^*$; isto porque as variáveis de $R'$ têm esta propriedade por construção, e ela é preservada na construção de $R''$. □

Segue um exemplo de aplicação do método de eliminação de variáveis inúteis delineado na prova do Teorema 17. Deve-se notar que o algoritmo 3.18(a) deve ser aplicado *antes* do algoritmo 3.18(b).

**Exemplo 109** Seja a gramática $G = (\{A, B, C, D, E, F\}, \{\mathtt{0}, \mathtt{1}\}, R, A\})$, onde $R$ contém as regras:

$$A \rightarrow ABC \mid AEF \mid BD$$

$$B \rightarrow B\mathtt{0} \mid \mathtt{0}$$

[6]Pode-se dizer que tais terminais são *inúteis*, visto que não são usados para formar palavras da linguagem gerada.

$C \to \mathtt{0}C \mid EB$

$D \to \mathtt{1}D \mid \mathtt{1}$

$E \to BE$

$F \to \mathtt{1}F\mathtt{1} \mid \mathtt{1}$

Aplicando-se o algoritmo 3.18(a), determina-se $V' = \{B, D, F, A\}$. Portanto, de acordo com o método do Teorema 17, $R'$ é formado pelas regras que contêm apenas tais variáveis:

$A \to BD$

$B \to B\mathtt{0} \mid \mathtt{0}$

$D \to \mathtt{1}D \mid \mathtt{1}$

$F \to \mathtt{1}F\mathtt{1} \mid \mathtt{1}$

Aplicando-se o algoritmo 3.18(b), determina-se $V'' = \{A, B, D\}$. Pelo método do Teorema 17, $R''$ contém apenas as regras que contêm tais variáveis:

$A \to BD$

$B \to B\mathtt{0} \mid \mathtt{0}$

$D \to \mathtt{1}D \mid \mathtt{1}$ □

Durante a concepção de uma gramática, o projetista pode deparar-se com a necessidade de modificar uma ou mais regras, sem alterar a linguagem gerada. Inicialmente, será mostrado como eliminar uma regra $X \to w$, onde $X$ *não é a variável de partida*, usando-se o expediente de "simular" a aplicação da mesma em todos os contextos possíveis: para cada ocorrência de $X$ do lado direito de cada regra, prevê-se o caso em que $X$ é substituída por $w$ e o caso em que não é. Com este expediente, consegue-se produzir algumas derivações mais curtas, às custas do aumento do número de regras da gramática, como será exemplificado mais abaixo, após o teorema a seguir.

**Teorema 18** *Seja uma GLC* $G = (V, \Sigma, R, P)$. *Seja* $X \to w \in R$, $\underline{X \neq P}$. *Seja a GLC* $G' = (V, \Sigma, R', P)$ *onde* $R'$ *é obtido assim:*

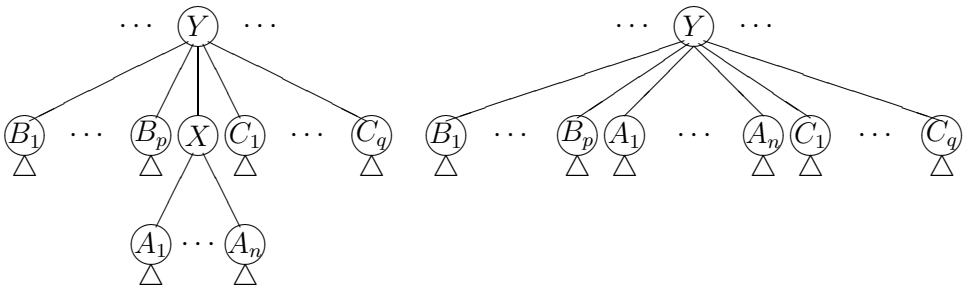
*(1) para cada regra* $Y \to z \in R$, *para cada forma de escrever* $z$ *como* $x_1 X x_2 \ldots X x_{n+1}$, *onde* $n \geq 0$ *e* $x_i \in (V \cup \Sigma)^*$, *coloque a regra* $Y \to x_1 w x_2 \ldots w x_{n+1}$ *em* $R'$;

*(2) retire* $X \to w$ *de* $R'$.

$G'$ *é equivalente a* $G$.

*Prova*

Não será feita uma demonstração rigorosa deste teorema, mas uma argumentação relativamente precisa e clara, utilizando o conceito de árvore de derivação (AD) desenvolvido na Seção 3.4.2. Uma GLC $G$ gera uma palavra $w$ se, e somente se, existe uma AD




(a) Antes. (b) Depois.

Figura 3.19: Transformação entre AD's induzida por remoção de regra.

Entrada: (1) uma GLC $G = (V, \Sigma, R, P)$, e
(2) uma regra $X \to w \in R$, $X \neq P$.
Saída: uma GLC $G'$ equivalente a $G$, sem a regra $X \to w$.
$R' \leftarrow \emptyset$;
**para** cada regra $Y \to z \in R$ **faça**
**para** cada forma de escrever $z$ como $x_1 X x_2 \dots X x_{n+1}$ **faça**
$R' \leftarrow R' \cup \{Y \to x_1 w x_2 \dots w x_{n+1}\}$
**fimpara**;
**fimpara**;
retorne $G' = (V, \Sigma, R' - \{X \to w\}, P)$.

Figura 3.20: Algoritmo para eliminar uma regra.

de $w$. Será mostrado, então, como transformar uma AD de $w$ em $G$ em uma AD de $w$ em $G'$, e vice-versa. Seja, então a regra $X \to w$ eliminada de $G$, com $w = A_1 A_2 \dots A_n$, onde $A_i \in V \cup \Sigma$, e seja uma regra da forma $Y \to B_1 \dots B_p X C_1 \dots C_q$, onde $B_i, C_j \in V \cup \Sigma$ (note que cada $B_i$ e $C_j$ pode ser ou não $X$). Tendo em vista como $G'$ é obtida, uma AD de $w$ em $G$ pode ser transformada em uma AD de $w$ em $G'$ substituindo-se toda subárvore da forma mostrada na Figura 3.19(a) pela subárvore mostrada na Figura 3.19(b). Em palavras: para todo vértice $v_X$ de rótulo $X$, filho de $v_Y$, de rótulo $Y$, e cujos filhos sejam (nesta ordem) $v_{A_1}$, $v_{A_2}$, ..., $v_{A_n}$, com rótulos $A_1$, $A_2$, ..., $A_n$,

(1) eliminar o vértice $v_X$; e

(2) colocar os vértices $v_{A_1}$, $v_{A_2}$, ..., $v_{A_n}$ (nesta ordem) como filhos de $v_Y$, entre os vértices $v_{B_p}$ e $v_{C_1}$.

Tal transformação, assim como a inversa, é possível, visto que $X \neq P$. Nela, a subárvore à esquerda é substituída pela subárvore à direita (ou vice-versa). □

Um algoritmo correspondente à formulação do Teorema 18 está formulado na Figura 3.20.

O método delineado no enunciado do Teorema 18, reformulado pelo algoritmo da Figura 3.20, será exemplificado abaixo. Observe que para se eliminar uma regra $X \to w$,

cada regra com $n$ ocorrências de $X$ no seu lado direito dá origem a até $2^n$ regras: para cada ocorrência, considera-se o caso em que ela é substituída por $w$ (aplicação da regra $X \to w$) e o caso em que não é (prevendo os casos de aplicações de outras regras $X$).

**Exemplo 110** Seja a gramática $G = (\{P, A, B\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, R, P)$, onde $R$ contém as regras:

$$P \to ABA$$

$$A \to \mathtt{a}A \mid \mathtt{a}$$

$$B \to \mathtt{b}B\mathtt{c} \mid \lambda$$

que gera a linguagem $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^n\mathtt{a}^k \mid m, k \geq 1 \text{ e } n \geq 0\}$. Seja o problema de eliminar a regra $A \to \mathtt{a}$. A regra $P \to ABA$ dá origem a quatro regras:

$$P \to ABA \mid AB\mathtt{a} \mid \mathtt{a}BA \mid \mathtt{a}B\mathtt{a}$$

e a regra $A \to \mathtt{a}A$ dá origem a duas regras:

$$A \to \mathtt{a}A \mid \mathtt{aa}$$

Assim, a gramática resultante é $G' = (\{P, A, B\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, R', P)$, onde $R'$ contém as regras:

$$P \to ABA \mid AB\mathtt{a} \mid \mathtt{a}BA \mid \mathtt{a}B\mathtt{a}$$

$$A \to \mathtt{a}A \mid \mathtt{aa}$$

$$B \to \mathtt{b}B\mathtt{c} \mid \lambda$$

Note que o número de regras aumentou, mas as derivações propiciadas são mais curtas. Por exemplo, $\mathtt{aa}$ tem a seguinte derivação em $G$:

$$\begin{array}{rll} P \Rightarrow & ABA & (\text{regra } P \to ABA) \\ \Rightarrow & \mathtt{a}BA & (\text{regra } A \to \mathtt{a}) \\ \Rightarrow & \mathtt{a}A & (\text{regra } B \to \lambda) \\ \Rightarrow & \mathtt{aa} & (\text{regra } A \to \mathtt{a}). \end{array}$$

E a mesma palavra tem a seguinte derivação em $G'$:

$$\begin{array}{rll} P \Rightarrow & \mathtt{a}B\mathtt{a} & (\text{regra } P \to \mathtt{a}B\mathtt{a}) \\ \Rightarrow & \mathtt{aa} & (\text{regra } B \to \lambda). \end{array}$$

□

Existem duas formas normais especialmente importantes para GLC's: as formas normais de Chomsky e de Greibach, que serão definidas mais abaixo. Um passo comum para a obtenção de uma gramática em uma destas formas, que seja equivalente a um GLC dada, $G$, é a obtenção de uma gramática $G'$, equivalente a $G$, que só tenha regras das formas:

Entrada: uma GLC $G = (V, \Sigma, R, P)$;
Saída: $\{X \in V \mid X \stackrel{*}{\Rightarrow} \lambda\}$.
$\mathcal{A} \leftarrow \emptyset$;
**repita**
    $\mathcal{N} \leftarrow \{Y \notin \mathcal{A} \mid Y \rightarrow z \in R \text{ e } z \in \mathcal{A}^*\}$;
    $\mathcal{A} \leftarrow \mathcal{A} \cup \mathcal{N}$
**até** $\mathcal{N} = \emptyset$;
retorne $\mathcal{A}$.

Figura 3.21: Algoritmo para determinar variáveis anuláveis.

- $P \rightarrow \lambda$, somente no caso em que $\lambda \in L(G)$ e $P$ é o símbolo de partida de $G'$;
- $X \rightarrow w$, $w \neq \lambda$, somente nos casos em que $w \in \Sigma$ ou $|w| > 1$.

Em outras palavras, $G'$ não pode ter regras $\lambda$, exceto se $\lambda \in L(G)$ (e neste caso tem a regra $P \rightarrow \lambda$), e não pode ter regras unitárias[7], que são regras da forma $X \rightarrow Y$, onde $X$ e $Y$ são variáveis. A seguir será mostrado, primeiramente, como eliminar as regras $\lambda$ de uma GLC sem alterar a linguagem gerada. Em seguida, será mostrado como eliminar as regras unitárias.

Um corolário do Teorema 18 é o fato de que qualquer regra $\lambda$ pode ser eliminada, com exceção da regra $P \rightarrow \lambda$, onde $P$ é o símbolo de partida. Mas, ao se eliminar uma regra $\lambda$ usando-se o método do Teorema 18, pode-se criar *outras* regras $\lambda$. O teorema apresentado a seguir mostra uma técnica para eliminar todas as regras $\lambda$, preservando-se ou criando-se a regra $P \rightarrow \lambda$ no caso em que $\lambda \in L(G)$.

O método a ser introduzido no teorema a seguir utiliza o conceito de *variável anulável*. Uma variável $X$ é dita ser anulável em uma GLC $G$ se, e somente se, $X \stackrel{*}{\Rightarrow}_G \lambda$. O algoritmo da Figura 3.21 determina o conjunto das variáveis anuláveis de uma GLC.

**Teorema 19** *Para qualquer GLC existe uma GLC equivalente cuja única regra $\lambda$, se houver, é $P \rightarrow \lambda$, onde $P$ é o símbolo de partida.*

*Prova*

Seja uma GLC $G = (V, \Sigma, R, P)$. Seja a GLC $G' = (V, \Sigma, R', P)$ onde $R'$ é obtido assim:

(1) para cada regra $Y \rightarrow z \in R$, para cada forma de escrever $z$ como $x_1 X_1 x_2 \ldots X_n x_{n+1}$, onde $n \geq 0$, $x_i \in (V \cup \Sigma)^*$, $x_1 x_2 \ldots x_{n+1} \neq \lambda$ e $X_i$ é variável anulável, coloque a regra $Y \rightarrow x_1 x_2 \ldots x_{n+1}$ em $R'$;

(2) se $P$ for anulável, coloque $P \rightarrow \lambda$ em $R'$.

Analisando-se estas duas etapas da construção de $G'$, vê-se que $G'$ não contém regra $\lambda$, exceto no caso em que $P$ é anulável, ou seja, $P \stackrel{*}{\Rightarrow}_G \lambda$, ou ainda, $\lambda \in L(G)$. Assim, para provar o teorema, basta provar que $P \stackrel{*}{\Rightarrow}_G w$ se, e somente $P \stackrel{*}{\Rightarrow}_{G'} w$ para todo $w \in \Sigma^*$ tal que $\underline{w \neq \lambda}$. Mas isto segue do fato de que para todo $X \in V$, $X \stackrel{*}{\Rightarrow}_G w$ se, e somente $X \stackrel{*}{\Rightarrow}_{G'} w$ para todo $w \in \Sigma^*$ tal que $\underline{w \neq \lambda}$, resultado este que será provado a seguir.

[7] *Unit rules* ou *chain rules*.

($\rightarrow$)

Será mostrado, inicialmente, por indução sobre $n$, que para todo $n \geq 1$, para todo $X \in V$ e todo $w \in \Sigma^*$ tal que $w \neq \lambda$, se $X \stackrel{n}{\Rightarrow}_G w$ então $X \stackrel{*}{\Rightarrow}_{G'} w$.

Sejam $X \in V$ e $w \in \Sigma^*$ tal que $w \neq \lambda$, arbitrários, e suponha que $X \stackrel{1}{\Rightarrow}_G w$. Tem-se, então, que $X \rightarrow w \in R$. Como $w \neq \lambda$, pela definição de $R'$ $X \rightarrow w \in R'$ e, portanto, $X \stackrel{1}{\Rightarrow}_{G'} w$. Seja $n \geq 1$ arbitrário, e suponha, como hipótese de indução, que se $X \stackrel{n}{\Rightarrow}_G w$ então $X \stackrel{*}{\Rightarrow}_{G'} w$ para todo $X \in V$ e todo $w \in \Sigma^*$ tal que $w \neq \lambda$. Suponha agora que $X \stackrel{n+1}{\Rightarrow}_G w$ para $X \in V$ e $w \in \Sigma^*$ tal que $w \neq \lambda$, arbitrários. Neste caso,

$$X \stackrel{1}{\Rightarrow}_G Y_1 Y_2 \ldots Y_k \stackrel{n}{\Rightarrow}_G w = x_1 x_2 \ldots x_k,$$

onde $Y_i \stackrel{p_i}{\Rightarrow}_G x_i$, para cada $1 \leq i \leq k$, sendo $p_i \leq n$. Para cada $x_i$: se $x_i = \lambda$ e $Y_i$ é uma variável, então $Y_i$ é uma variável anulável; e se $x_i \neq \lambda$, então, pela hipótese de indução, tem-se que $Y_i \stackrel{*}{\Rightarrow}_{G'} x_i$. E como $G$ tem a regra $X \rightarrow Y_1 Y_2 \ldots Y_k$, $G'$ tem a regra $X \rightarrow Z_1 Z_2 \ldots Z_k$, onde $Z_i = Y_i$ se $x_i \neq \lambda$, e $Z_i = \lambda$ se $x_i = \lambda$. Assim sendo, tem-se, finalmente, que:

$$X \stackrel{1}{\Rightarrow}_{G'} Z_1 Z_2 \ldots Z_k \stackrel{*}{\Rightarrow}_{G'} x_1 Z_2 \ldots Z_k \stackrel{*}{\Rightarrow}_{G'} x_1 x_2 \ldots Z_k \stackrel{*}{\Rightarrow}_{G'} x_1 x_2 \ldots x_k.$$

($\leftarrow$)

Será mostrado, também por indução sobre $n$, que para todo $n \geq 1$, para todo $X \in V$ e todo $w \in \Sigma^*$ tal que $w \neq \lambda$, se $X \stackrel{n}{\Rightarrow}_{G'} w$ então $X \stackrel{*}{\Rightarrow}_G w$.

Sejam $X \in V$ e $w \in \Sigma^*$ tal que $w \neq \lambda$, arbitrários, e suponha que $X \stackrel{1}{\Rightarrow}_{G'} w$. Tem-se, então, que $X \rightarrow w \in R'$. Como $w \neq \lambda$, pela definição de $R'$ existe $y$ tal que $|y| > 0$ e $X \rightarrow y \in R'$ e $w$ é o resultado de substituir zero ou mais ocorrências de variáveis anuláveis de $y$ por $\lambda$. Assim sendo, tem-se que $X \stackrel{1}{\Rightarrow}_G y \stackrel{*}{\Rightarrow}_G w$. Seja $n \geq 1$ arbitrário, e suponha, como hipótese de indução, que se $X \stackrel{n}{\Rightarrow}_{G'} w$ então $X \stackrel{*}{\Rightarrow}_G w$ para todo $X \in V$ e todo $w \in \Sigma^*$ tal que $w \neq \lambda$. Suponha agora que $X \stackrel{n+1}{\Rightarrow}_{G'} w$ para $X \in V$ e $w \in \Sigma^*$ tal que $w \neq \lambda$, arbitrários. Neste caso,

$$X \stackrel{1}{\Rightarrow}_{G'} Z_1 Z_2 \ldots Z_k \stackrel{n}{\Rightarrow}_{G'} w = x_1 x_2 \ldots x_k,$$

onde $Z_i \stackrel{p_i}{\Rightarrow}_G x_i$, para cada $1 \leq i \leq k$, sendo $p_i \leq n$. Como $X \rightarrow Z_1 Z_2 \ldots Z_k \in R'$, segue-se que há em $R$ uma regra $X \rightarrow y$ onde $Z_1 Z_2 \ldots Z_k$ é obtida de $y$ pela eliminação de zero ou mais variáveis anuláveis. Portanto, $X \stackrel{*}{\Rightarrow}_G Z_1 Z_2 \ldots Z_k$. Se $Z_i$ é uma variável, pela hipótese de indução tem-se que $Z_i \stackrel{*}{\Rightarrow}_G x_i$. Logo, $Z_1 Z_2 \ldots Z_k \stackrel{*}{\Rightarrow}_G x_1 x_2 \ldots x_k$. Concluindo:

$$X \stackrel{*}{\Rightarrow}_G Z_1 Z_2 \ldots Z_k \stackrel{*}{\Rightarrow}_G w = x_1 x_2 \ldots x_k.$$

□

Na Figura 3.22 está apresentado um algoritmo para eliminação de regras $\lambda$ abstraído do método apresentado no Teorema 19. Tal algoritmo utiliza o algoritmo definido na Figura 3.21, para determinar o conjunto das variáveis anuláveis. Segue um exemplo.

**Exemplo 111** Seja a gramática $G = (\{P, A, B, C\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, R, P)$, onde $R$ contém as regras:

Entrada: uma GLC $G = (V, \Sigma, R, P)$;
Saída: uma GLC $G'$ equivalente a $G$, sem regras $\lambda$, exceto $P \to \lambda$.
$\mathcal{A} \leftarrow$ variáveis anuláveis de $G$;
$R' \leftarrow \emptyset$;
**para** cada regra $X \to w \in R$ **faça**
    **para** cada forma de escrever $w$ como $x_1 Y_1 x_2 \ldots Y_n x_{n+1}$, com $Y_1 \ldots Y_n \in \mathcal{A}$ **faça**
        $R' \leftarrow R' \cup \{X \to x_1 x_2 \ldots x_{n+1}\}$
    **fimpara**;
**fimpara**;
retorne $G' = (V, \Sigma, R' - \{X \to \lambda \mid X \neq P\}, P)$.

Figura 3.22: Algoritmo para eliminar regras $\lambda$.

$$P \to APB \mid C$$

$$A \to A\mathtt{aa}A \mid \lambda$$

$$B \to BB\mathtt{b} \mid C$$

$$C \to \mathtt{c}C \mid \lambda$$

Aplicando-se o algoritmo da Figura 3.21, obtém-se o conjunto das variáveis anuláveis de $G$: no presente caso, é o conjunto de todas as variáveis de $G$. Em seguida, faz-se como mostrado no algoritmo da Figura 3.22, obtendo-se as seguintes regras:

$$P \to APB \mid AP \mid AB \mid PB \mid A \mid B \mid C \mid \lambda$$

$$A \to A\mathtt{aa}A \mid \mathtt{aa}A \mid A\mathtt{aa} \mid \mathtt{aa}$$

$$B \to BB\mathtt{b} \mid B\mathtt{b} \mid \mathtt{b} \mid C$$

$$C \to \mathtt{c}C \mid \mathtt{c}$$

A regra $P \to P$, obtida a partir da regra $P \to APB$, foi descartada por motivos óbvios. □

Antes da apresentação das formas normais já citadas, resta apresentar um método para eliminação de regras unitárias. O método fará uso do conceito de *variáveis encadeadas*. Seja uma gramática $G = (V, \Sigma, R, P)$. Diz-se que uma variável $Z \in V$ é encadeada a uma variável $X \in V$ se $Z = X$ ou se existe uma seqüência de regras $X \to Y_1$, $Y_1 \to Y_2$, ..., $Y_n \to Z$ em $R$, $n \geq 0$; no caso em que $n = 0$, tem-se apenas a regra $X \to Z$. Ao conjunto de todas as variáveis encadeadas a $X$ é dado o nome de $enc(X)$. Note que $X \in enc(X)$. O algoritmo da Figura 3.23 calcula tal conjunto.

**Teorema 20** *Seja uma GLC $G$. Existe uma GLC, equivalente a $G$, sem regras unitárias.*

*Prova*

Uma GLC equivalente a $G = (V, \Sigma, R, P)$ seria $G' = (V, \Sigma, R', P)$, onde

$$R' = \{X \to w \mid Y \in enc(X), Y \to w \in R \text{ e } w \notin V\}.$$

Entrada: (1) uma GLC $G = (V, \Sigma, R, P)$, e
(2) uma variável $X \in V$.
Saída: $enc(X)$.
$\mathcal{U} \leftarrow \emptyset$; $\mathcal{N} \leftarrow \{X\}$;
**repita**
$\mathcal{U} \leftarrow \mathcal{U} \cup \mathcal{N}$;
$\mathcal{N} \leftarrow \{Y \notin \mathcal{U} \mid Z \rightarrow Y \in R \text{ para algum } Z \in \mathcal{N}\}$
**até** $\mathcal{N} = \emptyset$
retorne $\mathcal{U}$.

Figura 3.23: Algoritmo para variáveis encadeadas.

Para provar a equivalência entre $G$ e $G'$, será mostrado que para todo $X \in V$ e todo $w \in \Sigma^*$ $X \stackrel{*}{\Rightarrow}_G w$ se, e somente se $X \stackrel{*}{\Rightarrow}_{G'} w$.

($\rightarrow$)

Será mostrado, por indução sobre $n$, que para todo $n \geq 0$, para todo $X \in V$ e todo $w \in \Sigma^*$ se $X \stackrel{n}{\Rightarrow}_G w$ então $X \stackrel{*}{\Rightarrow}_{G'} w$.

Para $n = 0$, sendo $X \in V$, não existe $w \in \Sigma^*$ tal que $X \stackrel{0}{\Rightarrow}_G w$; logo, a afirmativa vale por vacuidade. Seja $n \geq 0$ arbitrário, e suponha, como hipótese de indução, que para todo $X \in V$ e todo $w \in \Sigma^*$ se $X \stackrel{n}{\Rightarrow}_G w$ então $X \stackrel{*}{\Rightarrow}_{G'} w$. Sejam $X \in V$ e $w \in \Sigma^*$ arbitrários e suponha que $X \stackrel{n+1}{\Rightarrow}_G w$. Neste caso,

$$X \Rightarrow_G Y_1 \Rightarrow_G Y_2 \ldots \Rightarrow_G Y_k \stackrel{n+1-k}{\Rightarrow}_G w,$$

onde $Y_1, Y_2, \ldots, Y_{k-1} \in V$ e $Y_k \notin V$, $k \geq 1$. Pela definição de $G'$, $X \rightarrow Y_k \in R'$. Logo, $X \Rightarrow_{G'} Y_k$. Supondo que $Y_K = Z_1 Z_2 \ldots Z_m$, $Z_i \in V \cup \Sigma$, e que $w = u_1 u_2 \ldots u_m$, onde $Z_i \stackrel{p_i}{\Rightarrow}_G u_i$, como $p_i \leq n$, pela hipótese de indução, se $Z_i \in V$ então $Z_i \stackrel{*}{\Rightarrow}_{G'} u_i$ e, portanto,

$$X \Rightarrow_{G'} Z_1 Z_2 \ldots Z_m \stackrel{*}{\Rightarrow}_{G'} u_1 Z_2 \ldots Z_m \stackrel{*}{\Rightarrow}_{G'} u_1 u_2 \ldots Z_m \stackrel{*}{\Rightarrow}_{G'} u_1 u_2 \ldots u_m = w.$$

($\leftarrow$)

Segue diretamente da definição das regras de $G'$ que se $X \stackrel{*}{\Rightarrow}_{G'} w$ então $X \stackrel{*}{\Rightarrow}_G w$. □

Um algoritmo correspondente ao método esboçado no Teorema 20 está apresentado na Figura 3.24. Lembre-se que $enc(X)$ pode ser determinado através do algoritmo mostrado na Figura 3.23. Segue um exemplo.

**Exemplo 112** Seja novamente a GLC para expressões aritméticas, cujas regras são reproduzidas abaixo:

$E \rightarrow E{+}T \mid T$

$T \rightarrow T{*}F \mid F$

$F \rightarrow (E) \mid \mathtt{t}$

Entrada: uma GLC $G = (V, \Sigma, R, P)$;
Saída: uma GLC $G'$ equivalente a $G$, sem regras unitárias.
$R' \leftarrow \emptyset$;
**para** cada variável $X \in V$ **faça**
    $R' \leftarrow R' \cup \{X \rightarrow w \mid$ existe $Y \in enc(X)$ tal que $Y \rightarrow w \in R$ e $w \notin V\}$
**fimpara**;
retorne $G' = (V, \Sigma, R', P)$.

Figura 3.24: Algoritmo para eliminar regras unitárias.

Os conjuntos $enc(X)$ para cada variável $X$ são:

- $enc(E) = \{E, T, F\}$;
- $enc(T) = \{T, F\}$;
- $enc(F) = \{F\}$.

A GLC equivalente, sem regras unitárias, obtida de acordo com o método do Teorema 20 é então $(\{E, T, F\}, \{\mathtt{t}, +, *, (, )\}, R', E)$, onde $R'$ tem as regras:

$$E \rightarrow E{+}T \mid T{*}F \mid (E) \mid \mathtt{t}$$

$$T \rightarrow T{*}F \mid (E) \mid \mathtt{t}$$

$$F \rightarrow (E) \mid \mathtt{t}$$

□

Até agora foi visto como eliminar uma regra qualquer (que não tenha o símbolo de partida do lado esquerdo), como eliminar todas as variáveis inúteis, como eliminar todas as regras $\lambda$ (com exceção, eventualmente, de regra da forma $P \rightarrow \lambda$) e como eliminar todas as regras unitárias. Algumas eliminações podem ocasionar o aparecimento de novas regras. Ao se aplicar vários tipos de eliminações em seqüência, certo tipo de regra, já eliminado, pode reaparecer. Seguem alguns exemplos:

(a) Ao se eliminar regras $\lambda$ podem aparecer regras unitárias. Exemplo: GLC com as regras $A \rightarrow BC$ e $B \rightarrow \lambda$.

(b) Ao se eliminar regras unitárias podem aparecer regras $\lambda$. Exemplo: GLC com $P \rightarrow \lambda$, onde $P$ é o símbolo de partida, e a regra $A \rightarrow P$.

(c) Ao se eliminar regras $\lambda$ podem aparecer variáveis inúteis. Exemplo: o acima, no item (a), caso $B \rightarrow \lambda$ seja a única regra $B$.

(d) Ao se eliminar regras unitárias podem aparecer variáveis inúteis. Exemplo: GLC que contém $A \rightarrow B$ e $B$ não aparece do lado direito de nenhuma outra regra ($B$ torna-se inútil).

Evidentemente, ao se eliminar variáveis inúteis, não podem aparecer novas regras, inclusive regras $\lambda$ ou unitárias. Assim, dados (c) e (d), variáveis inúteis devem ser eliminadas por último. O exemplo apresentado em (b) apresenta o único caso que propicia o aparecimento de regra $\lambda$ ao se eliminar regras unitárias: quando ocorre da GLC conter, simultaneamente os dois tipos de regras:

- $P \to \lambda$, e
- $X \to P$,

onde $P$ é o símbolo de partida. Assim, pode-se garantir que a seguinte seqüência de eliminações é consistente, caso a GLC inicial não contenha um dos dois tipos de regra:

(1) eliminar regras $\lambda$;

(2) eliminar regras unitárias;

(3) eliminar símbolos inúteis.

Uma maneira simples de garantir a consistência da seqüência de eliminações acima é antecedê-la pela mudança da variável de partida para um novo símbolo $P'$, e pelo acréscimo à GLC de uma nova regra $P' \to P$. Neste caso, a eliminação de regras $\lambda$ leva ao aparecimento de uma regra $P' \to \lambda$ quando $\lambda$ pertence à linguagem gerada, mas nunca ao aparecimento de uma regra do tipo $X \to P'$ pois, sendo $P'$ uma variável que não aparece no lado direito de nenhuma regra, ela continua a não aparecer após a eliminação de regras $\lambda$.

A discussão dos dois últimos parágrafos permite enunciar o teorema a seguir.

**Teorema 21** *Para qualquer GLC $G = (V, \Sigma, R, P)$, existe uma GLC equivalente cujas regras são das formas:*

- $P \to \lambda$ *se* $\lambda \in L(G)$*;*
- $X \to a$ *para* $a \in \Sigma$*;*
- $X \to w$ *para* $|w| \geq 2$.

*Prova*

Este resultado segue da discussão acima. □

A seguir serão apresentadas as formas normais de Chomsky e de Greibach, ainda mais restritas que a do Teorema 21, mas que não diminuem o poder expressivo das GLC's e que encontram várias aplicações.

**Definição 41** *Uma GLC $G = (V, \Sigma, R, P)$ é dita estar na* forma normal de Chomsky *(FNC) se todas as suas regras estão nas formas:*

- $P \to \lambda$ *se* $\lambda \in L(G)$*;*
- $X \to YZ$ *para* $Y, Z \in V$*;*
- $X \to a$ *para* $a \in \Sigma$. □

**Teorema 22** *Seja uma GLC $G$. Existe uma GLC na FNC equivalente a $G$.*

*Prova*

Pelo Teorema 21, pode-se obter uma GLC equivalente a $G$ tal que:

- $P \to \lambda$ se $\lambda \in L(G)$;
- $X \to a$ para $a \in \Sigma$;
- $X \to w$ para $|w| \geq 2$.

Assim, basta mostrar como obter um conjunto de regras cujo efeito na derivação de uma palavra seja equivalente ao efeito de uma regra do tipo $X \to w$ para $|w| \geq 2$, mas que tenha apenas regras das formas permitidas na FNC. Isto pode ser feito em dois passos:

(1) Modificar cada regra $X \to w$, $|w| \geq 2$, se necessário, de forma que ela fique contendo apenas variáveis. Para isto, substituir cada ocorrência de cada $a \in \Sigma$ que ocorra em $w$ por uma variável, da seguinte forma: se existe uma regra da forma $Y \to a$ e esta é a única regra $Y$, substituir as ocorrências de $a$ por $Y$ em $w$; caso contrário, criar uma regra $Y \to a$, onde $Y$ é uma variável *nova*, e substituir as ocorrências de $a$ por $Y$ em $w$.

(2) Substituir cada regra $X \to Y_1 Y_2 \ldots Y_n$, $n \geq 3$, onde cada $Y_i$ é uma variável, pelo conjunto das regras: $X \to Y_1 Z_1$, $Z_1 \to Y_2 Z_2$, $\ldots$, $Z_{n-2} \to Y_{n-1} Y_n$, onde $Z_1$, $Z_2$, $\ldots$, $Z_{n-2}$ são variáveis *novas*. □

A prova do Teorema 22 leva ao algoritmo da Figura 3.25 para construção de uma GLC na FNC equivalente a uma GLC dada. Segue um exemplo de obtenção de gramática na forma normal de Chomsky.

**Exemplo 113** Seja a GLC $G = (\{L, S, E\}, \{\mathtt{a}, (, )\}, R, L)$, onde $R$ consta das regras:

$L \to (S)$

$S \to SE \mid \lambda$

$E \to \mathtt{a} \mid L$

Apesar de $L$ aparecer do lado direito de uma regra, esta GLC claramente não gera $\lambda$. Assim, não é preciso acrescentar uma regra $L' \to L$, embora o algoritmo da Figura 3.25 o faça. Observando-se que $S$ é a única variável anulável, tem-se as seguintes regras após a eliminação de regras $\lambda$:

$L \to (S) \mid ()$

$S \to SE \mid E$

$E \to \mathtt{a} \mid L$

Observando-se que $enc(L) = \{L\}$, $enc(S) = \{S, E, L\}$ e $enc(E) = \{E, L\}$, são obtidas as seguintes regras após a eliminação de regras unitárias:

$L \to (S) \mid ()$

$S \to SE \mid \mathtt{a} \mid (S) \mid ()$

$E \to \mathtt{a} \mid (S) \mid ()$

Entrada: (1) uma GLC $G = (V, \Sigma, R, P)$.
Saída: uma GLC $G'$ equivalente a $G$, na FNC.
$P' \leftarrow$ uma variável que não pertence a $V$;
**se** $P$ ocorre do lado direito de alguma regra **então**
    $V' \leftarrow V \cup \{P'\}$; $R' \leftarrow R \cup \{P' \rightarrow P\}$;
**senão**
    $V' \leftarrow V$; $R' \leftarrow R$; $P' \leftarrow P$
**fimse**;
elimine regras $\lambda$ (algoritmo da Figura 3.22);
elimine regras unitárias (algoritmo da Figura 3.24);
/* aqui pode-se eliminar variáveis inúteis */
**para** cada conjunto de regras $S \subseteq R'$ com $w$ do lado direito, $|w| \geq 2$ **faça**
    **para** cada $a \in \Sigma$ em $w$ **faça**
        **se** existe uma regra $Y \rightarrow a \in R'$ **então**
            substitua $a$ por $Y$ nas regras de $S$
        **senão**
            $Y \leftarrow$ uma variável que não pertence a $V'$;
            $V' \leftarrow V' \cup \{Y\}$;
            $R' \leftarrow R' \cup \{Y \rightarrow a\}$;
            substitua $a$ por $Y$ nas regras de $S$
        **fimse**
    **fimpara**;
    **para** cada regra $X \rightarrow Y_1 Y_2 \ldots Y_n \in S$, $n > 2$ **faça**
        substituí-la pelas regras $X \rightarrow Y_1 Z_1$, $Z_1 \rightarrow Y_2 Z_2$, $\ldots$, $Z_{n-2} \rightarrow Y_{n-1} Y_n$
            onde $Z_1, Z_2, \ldots, Z_{n-2} \notin V'$;
        $V' \leftarrow V' \cup \{Z_1, Z_2, \ldots, Z_{n-2}\}$
    **fimpara**
**fimpara**;
retorne $G' = (V', \Sigma, R', P')$.

Figura 3.25: Algoritmo para FNC.

Finalmente, após a substituição de terminais por variáveis nas regras cujo lado direito é maior ou igual a 2, e a "quebra" daquelas regras com lado direito maior que 2, tem-se as regras:

$$L \rightarrow AX \mid AB$$

$$S \rightarrow SE \mid \mathtt{a} \mid AX \mid AB$$

$$E \rightarrow \mathtt{a} \mid AX \mid AB$$

$$X \rightarrow SB$$

$$A \rightarrow ($$

$$B \rightarrow )$$

□

Antes de apresentar a forma normal de Greibach, serão vistos dois métodos de manipulação, e teoremas respectivos, que são úteis, não apenas como passos intermediários para obtenção de GLC's nesta forma normal, mas também em outros contextos. O primeiro diz respeito à eliminação de *regras recursivas à esquerda*, isto é regras da forma

$A \rightarrow Ay$. É interessante observar que em analisadores sintáticos *top-down* gerados a partir de GLC's, as GLC's não podem conter regras recursivas à esquerda. A seguir, mostra-se como eliminar este tipo de regra.

**Teorema 23** *Para qualquer GLC existe uma GLC equivalente sem regras recursivas à esquerda.*

*Prova*

Sejam as seguintes <u>todas</u> as regras $X$ de uma GLC $G$:

$$X \rightarrow Xy_1 \mid Xy_2 \mid \ \ldots \ \mid Xy_n \mid w_1 \mid w_2 \mid \ \ldots \ \mid w_k$$

onde nenhum $w_i$ começa com $X$. Se $k = 0$, obviamente $X$ é uma variável inútil e as regras $X$ podem ser simplesmente eliminadas. Caso contrário, em uma DME as regras $X$ são utilizadas da seguinte forma:

$$X \stackrel{p}{\Rightarrow} Xy_{i_p}y_{i_{p-1}} \ldots y_{i_1} \Rightarrow w_j y_{i_p} y_{i_{p-1}} \ldots y_{i_1} \ (p \geq 0)$$

onde $1 \leq i_q \leq n$ para $1 \leq q \leq p$, e $1 \leq j \leq k$. Ora, a forma sentencial $w_j y_{i_p} y_{i_{p-1}} \ldots y_{i_1}$ pode ser obtida utilizando recursão à direita, ao invés de recursão à esquerda, por meio das regras:

$$X \rightarrow w_1 \mid w_2 \mid \ \ldots \ \mid w_k \mid w_1 Z \mid w_2 Z \mid \ \ldots \ \mid w_k Z$$

$$Z \rightarrow y_1 \mid y_2 \mid \ \ldots \ \mid y_n \mid y_1 Z \mid y_2 Z \mid \ \ldots \ \mid y_n Z$$

onde $Z$ é uma variável *nova*. □

Segue um exemplo de eliminação de regras recursivas à esquerda.

**Exemplo 114** Seja a gramática $G = (\{E\}, \{\mathtt{t}, +, *, (, )\}, R, E)$, sendo $R$ dado por:

$$E \rightarrow E{+}E \mid E{*}E \mid (E) \mid \mathtt{t}$$

Utilizando o raciocínio do Teorema 23, obtém-se:

$$E \rightarrow (E) \mid \mathtt{t} \mid (E)Z \mid \mathtt{t}Z$$

$$Z \rightarrow {+}E \mid {*}E \mid {+}EZ \mid {*}EZ$$

Observe que, como efeito colateral, a ambigüidade da gramática foi removida. □

O segundo método referido acima, é o utilizado para eliminar uma variável que aparece do lado direito de uma regra, método este apresentado no enunciado do teorema a seguir.

**Teorema 24** *Seja uma GLC $G = (V, \Sigma, R, P)$ tal que $X \rightarrow uYv \in R$, onde $Y \in V$ e $Y \neq X$. Sejam $Y \rightarrow w_1 \mid w_2 \mid \ \ldots \ \mid w_n$ <u>todas</u> as regras $Y$ em $R$. Seja $G' = (V, \Sigma, R', P)$ onde*

$$R' = (R - \{X \rightarrow uYv\}) \cup \{X \rightarrow uw_1v \mid uw_2v \mid \ \ldots \ \mid uw_nv\}.$$

*Então* $L(G') = L(G)$.

*Prova*

Suponha que $P \stackrel{*}{\Rightarrow}_G w$. Se a regra $X \to uYv$ não for usada na derivação, então ela mesma é uma derivação de $w$ em $G'$. Por outro lado, se tal regra for utilizada, a subderivação $X \Rightarrow_G uYv \Rightarrow_G uw_iv$ pode ser substituída por $X \Rightarrow_{G'} uw_iv$ (aplicando-se a regra $X \to uw_iv$). Logo, $P \stackrel{*}{\Rightarrow}_{G'} w$ e, portanto, $L(G) \subseteq L(G')$.

Suponha, por outro lado, que $P \stackrel{*}{\Rightarrow}_{G'} w$. Caso as regras $X \to uw_1v \mid uw_2v \mid \ldots \mid uw_nv$ não sejam usadas na derivação de $w$ em $G'$, então ela é também uma derivação de $w$ em $G$. Por outro lado, se para regra $X \to uw_iv$ usada na derivação de $w$ em $G'$, a subderivação $X \Rightarrow_{G'} uw_iv$ for substituída por $X \Rightarrow_G uYv \Rightarrow_G uw_iv$, obtém-se uma derivação de $w$ em $G$ e, portanto, $L(G') \subseteq L(G)$. □

**Definição 42** *Uma GLC* $G = (V, \Sigma, R, P)$ *é dita estar na* forma normal de Greibach *(FNG) se todas as suas regras são das formas:*

- $P \to \lambda$ *se* $\lambda \in L(G)$*;*
- $X \to ay$ *para* $a \in \Sigma$ *e* $y \in V^*$*.* □

Veja que uma forma sentencial de uma gramática na forma normal de Greibach, com exceção de $\lambda$, é sempre da forma $xy$, onde $x \in \Sigma^+$ e $y \in V^*$. Note ainda que a cada passo de uma derivação, concatena-se um terminal a mais ao prefixo de terminais $x$, a menos que a regra utilizada seja $P \to \lambda$. Supondo que $P$ não apareça do lado direito de nenhuma regra, o tamanho de uma derivação de uma palavra $w \in \Sigma^*$ é sempre $|w|$, exceto quando $w = \lambda$; neste caso, o tamanho é 1, correspondendo à derivação $P \Rightarrow \lambda$.

**Teorema 25** *Seja uma GLC* $G$*. Existe uma GLC na FNG equivalente a* $G$*.*

*Prova*

Como já foi visto, pode-se obter uma GLC equivalente a $G$ tal que:

- $P \to \lambda$ se $\lambda \in L(G)$;
- $X \to a$ para $a \in \Sigma$;
- $X \to w$ para $|w| \geq 2$.

Usando-se o mesmo artifício que no item (1) do método descrito na prova do Teorema 22, pode-se substituir por variáveis todos os terminais de $w$, a partir de seu segundo símbolo, nas regras da forma $X \to w$ para $|w| \geq 2$. Com isto obtém-se regras da forma $X \to Yy$, onde $Y \in V \cup \Sigma$ e $y \in V^+$. Assim, basta mostrar como as regras desta forma podem ser substituídas por outras das formas permitidas na FNG, de forma que a mesma linguagem seja gerada. A chave para isto é utilizar repetidamente o Teorema 24 para substituir $Y$ se $Y \neq X$, ou o Teorema 23 se $Y = X$. Mas isto deve ser feito de forma a garantir que o processo termine. O processo começa pela numeração sequencial, a partir de 1, das variáveis; embora a complexidade da GLC resultante dependa da numeração, qualquer numeração em que $P$ receba o número 1 serve. Em seguida, para cada $A \in V$, começando com $P$, na ordem dada pela numeração escolhida, faz-se o seguinte até não poder mais:

1. Se existe uma regra $A \to By$, para $|y| \geq 1$, tal que o número de $B$ é *menor* que o de $A$, aplica-se o método do Teorema 24 para substituir $B$. (Observe que isto não é feito para as variáveis novas introduzidas pelo uso do Teorema 23.)

2. Se existe uma regra $A \to Ay$, para $|y| \geq 1$, aplica-se o método do Teorema 23 para eliminar a recursão à esquerda.

Com isto, obtém-se uma GLC em que as regras da forma $X \to Yy$, para $|y| \geq 1$, são tais que o número de $X$ é menor que o de $Y$. (No Exercício 21 da Seção 3.6, página 198, pede-se para provar que este processo termina.)

Terminado o processo descrito acima, sendo $A$ a variável de maior número, as regras $A$ são da forma $A \to ay$, $y \in V^*$. Sendo $B$ a variável de número anterior ao maior número, as regras $B$ da forma $B \to Cy$ são tais que $C = A$. Assim, basta aplicar o método do Teorema 24 para obter regras da forma pretendida. Tal processo é repetido, sucessivamente, para as variáveis de número menor, até ser atingida a variável $P$. Ao final, para cada $A \in V$ tem-se apenas regras $A$ na forma pretendida e, eventualmente, regras da forma $Z \to w$, onde $Z$ é variável *nova* criada eliminando-se recursão à esquerda. Neste último caso, regras da forma $Z \to Ay$ podem ser eliminadas aplicando-se o método do Teorema 24. Após isto, todas as regras estarão no formato permitido pela FNG. □

**Exemplo 115** Seja a GLC $G = (\{A, B, C, D\}, \{\mathtt{b}, \mathtt{c}, \mathtt{d}\}, R, A)$, onde $R$ consta das regras:

$A \to CB$

$B \to BBD \mid \mathtt{b}$

$C \to BBC \mid D\mathtt{c}$

$D \to AD \mid \mathtt{d}$

Inicialmente, elimina-se a regra $C \to D\mathtt{c}$ e introduz-se as regras $C \to DE$ e $E \to \mathtt{c}$.

Como esta GLC não tem regras $\lambda$ e regras unitárias, começa-se numerando-se as variáveis. Seja a numeração em que o número de $A$ é 1, de $B$ é 2, de $C$ é 3, de $D$ é 4 e de $E$ é 5. A regra $A \to CB$ fica como está, pois o número de $A$ é menor que o de $C$. Nas regras $B$ tem-se recursão à esquerda; aplicando-se o método do Teorema 23, as regras $B$ são substituídas por: $B \to \mathtt{b} \mid \mathtt{b}Z_1$, mais as regras $Z_1 \to BD \mid BDZ_1$. A GLC resultante, até agora é (as regras *novas* são marcadas com "$*$"):

$A \to CB$

$*$ $B \to \mathtt{b} \mid \mathtt{b}Z_1$

$C \to BBC \mid DE$

$D \to AD \mid \mathtt{d}$

$E \to \mathtt{c}$

$*$ $Z_1 \to BD \mid BDZ_1$

A regra $C \to BBC$ deve ser substituída, pois o número de $C$ é maior que o de $B$. Obtém-se:

$A \rightarrow CB$

$B \rightarrow \texttt{b} \mid \texttt{b}Z_1$

* $C \rightarrow \texttt{b}BC \mid \texttt{b}Z_1BC$

$C \rightarrow DE$

$D \rightarrow AD \mid \texttt{d}$

$E \rightarrow \texttt{c}$

$Z_1 \rightarrow BD \mid BDZ_1$

Em seguida, substitui-se a regra $D \rightarrow AD$, pois o número de $D$ é maior que o de $A$, obtendo-se:

$A \rightarrow CB$

$B \rightarrow \texttt{b} \mid \texttt{b}Z_1$

$C \rightarrow \texttt{b}BC \mid \texttt{b}Z_1BC \mid DE$

* $D \rightarrow CBD$

$D \rightarrow \texttt{d}$

$E \rightarrow \texttt{c}$

$Z_1 \rightarrow BD \mid BDZ_1$

A regra $D \rightarrow CBD$ deve ser substituída, pois o número de $D$ é maior que o de $C$. Obtém-se:

$A \rightarrow CB$

$B \rightarrow \texttt{b} \mid \texttt{b}Z_1$

$C \rightarrow \texttt{b}BC \mid \texttt{b}Z_1BC \mid DE$

* $D \rightarrow \texttt{b}BCBD \mid \texttt{b}Z_1BCBD \mid DEBD$

$D \rightarrow \texttt{d}$

$E \rightarrow \texttt{c}$

$Z_1 \rightarrow BD \mid BDZ_1$

Eliminando-se a recursão à esquerda para as regras $D$, obtém-se:

$A \rightarrow CB$

$B \rightarrow \texttt{b} \mid \texttt{b}Z_1$

$C \rightarrow \texttt{b}BC \mid \texttt{b}Z_1BC \mid DE$

* $D \rightarrow \texttt{b}BCBD \mid \texttt{b}Z_1BCBD \mid \texttt{d} \mid \texttt{b}BCBDZ_2 \mid \texttt{b}Z_1BCBDZ_2 \mid \texttt{d}Z_2$

$E \rightarrow \texttt{c}$

$Z_1 \rightarrow BD \mid BDZ_1$

* $Z_2 \rightarrow EBD \mid EBDZ_2$

Agora, analisando-se as regras $E$, $D$, $C$, $B$ e $A$, nesta ordem, vê-se que apenas as regras $C \rightarrow DE$ e $A \rightarrow CB$ devem ser substituídas. Substituindo-se $C \rightarrow DE$, obtém-se:

$A \rightarrow CB$

$B \rightarrow \texttt{b} \mid \texttt{b}Z_1$

$C \rightarrow \texttt{b}BC \mid \texttt{b}Z_1BC$

* $C \rightarrow \texttt{b}BCBDE \mid \texttt{b}Z_1BCBDE \mid \texttt{d}E \mid \texttt{b}BCBDZ_2E$

$\quad \mid \texttt{b}Z_1BCBDZ_2E \mid \texttt{d}Z_2E$

$D \rightarrow \texttt{b}BCBD \mid \texttt{b}Z_1BCBD \mid \texttt{d} \mid \texttt{b}BCBDZ_2 \mid \texttt{b}Z_1BCBDZ_2 \mid \texttt{d}Z_2$

$E \rightarrow \texttt{c}$

$Z_1 \rightarrow BD \mid BDZ_1$

$Z_2 \rightarrow EBD \mid EBDZ_2$

Substituindo-se $A \rightarrow CB$, obtém-se:

* $A \rightarrow \texttt{b}BCB \mid \texttt{b}Z_1BCB \mid \texttt{b}BCBDEB \mid \texttt{b}Z_1BCBDEB \mid \texttt{d}EB$

$\quad \mid \texttt{b}BCBDZ_2EB \mid \texttt{b}Z_1BCBDZ_2EB \mid \texttt{d}Z_2EB$

$B \rightarrow \texttt{b} \mid \texttt{b}Z_1$

$C \rightarrow \texttt{b}BC \mid \texttt{b}Z_1BC \mid \texttt{b}BCBDE \mid \texttt{b}Z_1BCBDE$

$\quad \mid \texttt{d}E \mid \texttt{b}BCBDZ_2E \mid \texttt{b}Z_1BCBDZ_2E \mid \texttt{d}Z_2E$

$D \rightarrow \texttt{b}BCBD \mid \texttt{b}Z_1BCBD \mid \texttt{d} \mid \texttt{b}BCBDZ_2 \mid \texttt{b}Z_1BCBDZ_2 \mid \texttt{d}Z_2$

$E \rightarrow \texttt{c}$

$Z_1 \rightarrow BD \mid BDZ_1$

$Z_2 \rightarrow EBD \mid EBDZ_2$

Finalmente, substitui-se as regras introduzidas por eliminação de recursão à esquerda:

$A \rightarrow \texttt{b}BCB \mid \texttt{b}Z_1BCB \mid \texttt{b}BCBDEB \mid \texttt{b}Z_1BCBDEB \mid \texttt{d}EB$

$\quad \mid \texttt{b}BCBDZ_2EB \mid \texttt{b}Z_1BCBDZ_2EB \mid \texttt{d}Z_2EB$

$B \to \mathtt{b} \mid \mathtt{b}Z_1$

$C \to \mathtt{b}BC \mid \mathtt{b}Z_1BC \mid \mathtt{b}BCBDE \mid \mathtt{b}Z_1BCBDE$

$\quad \mid \mathtt{d}E \mid \mathtt{b}BCBDZ_2E \mid \mathtt{b}Z_1BCBDZ_2E \mid \mathtt{d}Z_2E$

$D \to \mathtt{b}BCBD \mid \mathtt{b}Z_1BCBD \mid \mathtt{d} \mid \mathtt{b}BCBDZ_2 \mid \mathtt{b}Z_1BCBDZ_2 \mid \mathtt{d}Z_2$

$E \to \mathtt{c}$

* $Z_1 \to \mathtt{b}D \mid \mathtt{b}Z_1D \mid \mathtt{b}DZ_1 \mid \mathtt{b}Z_1DZ_1$

* $Z_2 \to \mathtt{c}BD \mid \mathtt{c}BDZ_2$ □

### 3.4.4 GLC's e autômatos com pilha

Nesta seção, mostra-se que GLC's e AP's reconhecem a mesma classe de linguagens, qual seja a classe das linguagens livres do contexto.

No teorema a seguir, mostra-se que para qualquer GLC $G$ existe um AP, de apenas dois estados, que reconhece $L(G)$.

**Teorema 26** *Para qualquer GLC $G$ existe um AP que reconhece $L(G)$.*

*Prova*

Seja $G' = (V, \Sigma, R, P)$ uma GLC na FNG equivalente a $G$. Um APN que aceita $L(G')$ é $M = (\{i, f\}, \Sigma, V, \delta, \{i\}, \{f\})$, onde $\delta$ consta das transições:

- $\delta(i, \lambda, \lambda) = \{[f, P]\}$;
- se $P \to \lambda \in R$, $\delta(f, \lambda, P) = \{[f, \lambda]\}$; e
- para cada $a \in \Sigma$ e cada $X \in V$, $\delta(f, a, X) = \{[f, y] \mid y \in V^* \text{ e } X \to ay \in R\}$.

Para provar que $L(M) = L(G')$, basta mostrar que para todo $x \in \Sigma^*$ e $y \in V^*$, $P \stackrel{*}{\Rightarrow} xy$ se, e somente se, $[i, x, \lambda] \stackrel{*}{\vdash} [f, \lambda, y]$, pois quando $y = \lambda$, seguir-se-á que para todo $x \in \Sigma^*$, $P \stackrel{*}{\Rightarrow} x$ se, e somente se, $[i, x, \lambda] \stackrel{*}{\vdash} [f, \lambda, \lambda]$.

$(\rightarrow)$

Será provado, por indução sobre $n$, que para todo $n \geq 1$, todo $x \in \Sigma^*$ e $y \in V^*$, se $P \stackrel{n}{\Rightarrow} xy$ então $[i, x, \lambda] \stackrel{*}{\vdash} [f, \lambda, y]$. Para $n = 1$, tem-se dois casos: $P \stackrel{1}{\Rightarrow} \lambda$ $(x = y = \lambda)$ e, portanto, $P \to \lambda \in R$, ou $P \stackrel{1}{\Rightarrow} ay$ $(x = a)$ e, portanto, $P \to ay \in R$. Para o primeiro caso tem-se, pela definição de $\delta$, que $[i, \lambda, \lambda] \vdash [f, \lambda, P] \vdash [f, \lambda, \lambda]$. E para o segundo caso, tem-se que $[i, a, \lambda] \vdash [f, a, P] \vdash [f, \lambda, y]$.

Seja um $n \geq 1$ arbitrário, e suponha, como hipótese de indução, que para todo $x \in \Sigma^*$ e todo $y \in V^*$, se $P \stackrel{n}{\Rightarrow} xy$ então $[i, x, \lambda] \stackrel{*}{\vdash} [f, \lambda, y]$. Suponha então que $P \stackrel{n+1}{\Rightarrow} xay$ para $y \in V^*$. Tem-se, então, que ou $P \stackrel{n}{\Rightarrow} xaPy \Rightarrow xay$ com aplicação da regra $P \to \lambda$ no último passo, ou então $P \stackrel{n}{\Rightarrow} xXu \Rightarrow xazu$, onde $X \to az \in R$ e $y = zu$. No primeiro caso, tem-se que $[i, xa, \lambda] \stackrel{*}{\vdash} [f, \lambda, Py]$ pela hipótese de indução; e, pela definição de $\delta$,

tem-se que $[f, \lambda, Py] \vdash [f, \lambda, y]$; logo, $[i, xa, \lambda] \overset{*}{\vdash} [f, \lambda, y]$, como requerido. No segundo caso, pela hipótese de indução, $[i, x, \lambda] \overset{*}{\vdash} [f, \lambda, Xu]$, e, portanto, $[i, xa, \lambda] \overset{*}{\vdash} [f, a, Xu]$; como $X \to az \in R$, $[f, a, Xu] \vdash [f, \lambda, zu]$; e como $y = zu$, segue-se finalmente que $[i, xa, \lambda] \overset{*}{\vdash} [f, \lambda, y]$.

($\leftarrow$)

Esta parte pode ser provada de forma análoga à anterior. □

A seguir, apresenta-se um exemplo de aplicação do método apresentado na prova do Teorema 26, de obtenção de AP's a partir de GLC's.

**Exemplo 116** Seja a GLC do Exemplo 103, página 157, $G = (\{P\}, \{\mathtt{0}, \mathtt{1}\}, R, P)$, onde $R$ consta das regras:

$$P \to \mathtt{0}P\mathtt{1}P \mid \mathtt{1}P\mathtt{0}P \mid \lambda$$

Uma GLC equivalente na FNG seria aquela com as regras:

$$P \to \mathtt{0}PUP \mid \mathtt{1}PZP \mid \lambda$$

$$Z \to \mathtt{0}$$

$$U \to \mathtt{1}$$

Um AP que reconhece $L(G)$ seria então $(\{i, f\}, \{\mathtt{0}, \mathtt{1}\}, \{P, Z, U\}, \delta, i, \{f\})$, onde $\delta$ é dada por:

$$\delta(i, \lambda, \lambda) = \{[f, P]\}$$

$$\delta(f, \lambda, P) = \{[f, \lambda]\}$$

$$\delta(f, \mathtt{0}, P) = \{[f, PUP]\}$$

$$\delta(f, \mathtt{1}, P) = \{[f, PZP]\}$$

$$\delta(f, \mathtt{0}, Z) = \{[f, \lambda]\}$$

$$\delta(f, \mathtt{1}, U) = \{[f, \lambda]\}.$$ □

O Exemplo 116 apresenta mais uma solução alternativa para o mesmo problema que já teve soluções exibidas nas Figuras 3.5 (página 143), 3.9 (página 150) e 3.10 (página 150).

Para completar, mostra-se a seguir que é sempre possível obter uma GLC que gera a linguagem reconhecida por um AP. Antes, porém, será apresentada a idéia central relativa ao processo de obter a GLC a partir do AP.

Seja um APN $M = (E, \Sigma, \Gamma, \delta, I, F)$. Dados dois estados $e, e' \in E$ e $A \in \Gamma \cup \{\lambda\}$, considere o conjunto $C(e, A, e')$ de todas as palavras $w \in \Sigma^*$ tais que o APN $M$, começando em $e$, com a pilha contendo $A$, termina no estado $e'$ com a pilha vazia, após consumir $w$, ou seja,

$$C(e, A, e') = \{w \in \Sigma^* \mid [e, w, A] \overset{*}{\vdash} [e', \lambda, \lambda]\}.$$

Observe que

$$L(M) = \bigcup_{(i,f)\in I\times F} C(i,\lambda,f).$$

Suponha que seja possível gerar o conjunto $C(e,A,e')$ por meio de uma GLC cuja variável de partida seja $[e,A,e']$. Então $L(M)$ pode ser gerada por uma GLC constituída de todas as regras $[e,A,e']$, mais as regras da forma $P \to [i,\lambda,f]$, para cada $i \in I$ e $f \in F$. A prova do teorema a seguir apresenta os detalhes.

**Teorema 27** *Para qualquer APN $M$ existe uma GLC que gera $L(M)$.*

*Prova*

Seja um APN $M = (E,\Sigma,\Gamma,\delta,I,F)$. Como discutido acima, será mostrado como construir uma GLC com variáveis da forma $[e,A,e']$, para $e,e' \in E$ e $A \in \Gamma \cup \{\lambda\}$, de modo que para todo $w \in \Sigma^*$,

$$[e,A,e'] \stackrel{*}{\Rightarrow} w \text{ se, e somente se, } [e,w,A] \stackrel{*}{\vdash} [e',\lambda,\lambda].$$

A gramática $G$ tal que $L(G) = L(M)$ será $(V,\Sigma,R,P)$, onde $V = \{P\} \cup E \times (\Gamma \cup \{\lambda\}) \times E$ e $R$ contém as regras:

- para cada $i \in I$ e cada $f \in F$, $P \to [i,\lambda,f]$;
- para cada $e \in E$, $[e,\lambda,e] \to \lambda$;

e também, para cada transição $[e',z] \in \delta(e,a,A)$, onde $a \in \Sigma \cup \{\lambda\}$, $z = B_1B_2\ldots B_n$ ($B_i \in \Gamma$) e $A \in \Gamma \cup \{\lambda\}$, as seguintes regras (tipo 1):

- se $z = \lambda$, $[e,A,d] \to a[e',\lambda,d]$ para cada $d \in E$;
- se $z \neq \lambda$, $[e,A,d_n] \to a[e',B_1,d_1]\ldots[d_{n-1},B_n,d_n]$ para cada $d_1,d_2,\ldots,d_n \in E$.

Ainda, se $A = \lambda$, tem-se, adicionalmente (tipo 2):

- se $z = \lambda$, $[e,C,d] \to a[e',C,d]$ para cada $C \in \Gamma$ e cada $d \in E$;
- se $z \neq \lambda$, $[e,C,d_{n+1}] \to a[e',B_1,d_1]\ldots[d_{n-1},B_n,d_n][d_n,C,d_{n+1}]$ para cada $C \in \Gamma$ e cada $d_1,d_2,\ldots,d_n,d_{n+1} \in E$.

Dada a discussão anterior, para concluir a prova basta mostrar que

$$[e,A,e'] \stackrel{*}{\Rightarrow} w \text{ se, e somente se, } [e,w,A] \stackrel{*}{\vdash} [e',\lambda,\lambda]$$

para todo $e,e' \in E$, $A \in \Gamma \cup \{\lambda\}$ e $w \in \Sigma^*$, o que é deixado como exercício. □

**Exemplo 117** Seja o APD cujo diagrama de estados está mostrado na Figura 3.26, que reconhece $\{\mathtt{a}^n\mathtt{cb}^n \mid n \geq 0\} \cup \{\lambda\}$. Utilizando o método delineado no Teorema 27, obtém-se inicialmente as regras:

- $P \to [0,\lambda,0] \mid [0,\lambda,1]$

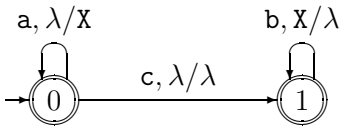



Figura 3.26: APD para $\{a^n cb^n \mid n \geq 0\} \cup \{\lambda\}$.

- $[0, \lambda, 0] \rightarrow \lambda$
  $[1, \lambda, 1] \rightarrow \lambda$

A seguir, vem as regras restantes, um grupo delas para cada transição do APD:

- para a transição $[0, \mathtt{X}] \in \delta(0, \mathtt{a}, \lambda)$:

  Tipo 1:

  $[0, \lambda, 0] \rightarrow \mathtt{a}[0, \mathtt{X}, 0]$

  $[0, \lambda, 1] \rightarrow \mathtt{a}[0, \mathtt{X}, 1]$

  Tipo 2:

  $[0, \mathtt{X}, 0] \rightarrow \mathtt{a}[0, \mathtt{X}, 0][0, \mathtt{X}, 0] \mid \mathtt{a}[0, \mathtt{X}, 1][1, \mathtt{X}, 0]$

  $[0, \mathtt{X}, 1] \rightarrow \mathtt{a}[0, \mathtt{X}, 0][0, \mathtt{X}, 1] \mid \mathtt{a}[0, \mathtt{X}, 1][1, \mathtt{X}, 1]$

- para a transição $[1, \lambda] \in \delta(0, \mathtt{c}, \lambda)$:

  Tipo 1:

  $[0, \lambda, 0] \rightarrow \mathtt{c}[1, \lambda, 0]$

  $[0, \lambda, 1] \rightarrow \mathtt{c}[1, \lambda, 1]$

  Tipo 2:

  $[0, \mathtt{X}, 0] \rightarrow \mathtt{c}[1, \mathtt{X}, 0]$

  $[0, \mathtt{X}, 1] \rightarrow \mathtt{c}[1, \mathtt{X}, 1]$

- para a transição $[1, \lambda] \in \delta(1, \mathtt{b}, \mathtt{X})$:

  Tipo 1:

  $[1, \mathtt{X}, 0] \rightarrow \mathtt{b}[1, \lambda, 0]$

  $[1, \mathtt{X}, 1] \rightarrow \mathtt{b}[1, \lambda, 1]$

Eliminando-se as variáveis inúteis, obtém-se as seguintes regras:

$P \rightarrow [0, \lambda, 0] \mid [0, \lambda, 1]$

$[0, \lambda, 1] \rightarrow \mathtt{a}[0, \mathtt{X}, 1] \mid \mathtt{c}[1, \lambda, 1]$

$[0, \mathtt{X}, 1] \rightarrow \mathtt{a}[0, \mathtt{X}, 1][1, \mathtt{X}, 1] \mid \mathtt{c}[1, \mathtt{X}, 1]$

$[1, \mathtt{X}, 1] \rightarrow \mathtt{b}[1, \lambda, 1]$

$[0, \lambda, 0] \to \lambda$

$[1, \lambda, 1] \to \lambda$ □

O método descrito na prova do Teorema 27 pode gerar muitas variáveis (e, portanto, regras) inúteis, como evidenciado no Exemplo 117. Isto pode ser, pelo menos em grande parte, evitado observando-se que para uma variável $[e, A, e']$ ser útil é necessário que exista um caminho de $e$ para $e'$ no diagrama de estados do AP, e que seja possível desempilhar o que for empilhado em alguma computação iniciando em $e$ e terminando em $e'$. Observando-se o diagrama de estados da Figura 3.26, vê-se claramente que são inúteis variáveis como:

- $[0, \mathtt{X}, 0]$: não é possível desempilhar `X` em um caminho que comece e termine em 0;
- $[1, \mathtt{X}, 0]$: não há caminho de 1 para 0.

# Exercícios

1. Construa GLC's para as linguagens do Exercício 3 do final da Seção 3.3, página 155.
2. Construa GLC's para as linguagens:
   (a) $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^{3m+2n+1} \mid m, n \geq 0\}$.
   (b) $\{\mathtt{a}^n\mathtt{b}^{2n+k}\mathtt{c}^{3k} \mid n, k \geq 0\}$.
   (c) $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^k \mid n > m + k\}$.
3. Construa GLC's para:
   (a) $L_1 = \{\mathtt{0}^n\mathtt{1}^k \mid 2n \leq k \leq 3n\}$.
   (b) $L_2 = \{\mathtt{a}^n\mathtt{b}^k\mathtt{c}^m \mid k = 2n + m\}$.
   (c) $(L_1 \cup L_2)^2$.
4. Seja $G$ a gramática:

   $$\begin{aligned} P &\to AB \\ A &\to \mathtt{a}A\mathtt{b} \mid \mathtt{c} \\ B &\to \mathtt{b}B\mathtt{c} \mid \mathtt{a} \end{aligned}$$

   (a) Construa uma derivação mais a esquerda de `acbbbacc`.
   (b) Construa a árvore de derivação para a derivação construída em (a).
   (c) Defina $L(G)$ utilizando notação de conjunto.
5. Seja a gramática $G$:

   $$P \to \mathtt{a}P\mathtt{b} \mid \mathtt{aa}P\mathtt{b} \mid \lambda$$

(a) Mostre que $G$ é ambígua.

(b) Construa uma gramática não ambígua equivalente a $G$.

6. Existe GLC ambígüa, sem variáveis inúteis, que gere $\{\lambda\}$? e $\{\mathtt{0}\}$?

7. Seja $G$ uma GLC que contenha, dentre outras, as regras:

$$\langle \text{cmd} \rangle \rightarrow \textbf{se } \langle \text{exp-rel} \rangle \textbf{ então } \langle \text{cmd} \rangle$$
$$\langle \text{cmd} \rangle \rightarrow \textbf{se } \langle \text{exp-rel} \rangle \textbf{ então } \langle \text{cmd} \rangle \textbf{ senão } \langle \text{cmd} \rangle$$

onde $\langle \text{cmd} \rangle$ e $\langle \text{exp-rel} \rangle$ são variáveis úteis e **se**, **então** e **senão** são terminais. Mostre que $G$ é ambígüa. Como eliminar a ambigüidade causada por tais regras?

8. Construa uma gramática sem regras $\lambda$ equivalente à seguinte gramática:

$$P \rightarrow BPA \mid A$$
$$A \rightarrow \mathtt{a}A \mid \lambda$$
$$B \rightarrow B\mathtt{ba} \mid \lambda$$

9. Seja a gramática $G$:

$$P \rightarrow A \mid BC$$
$$A \rightarrow B \mid C$$
$$B \rightarrow \mathtt{b}B \mid \mathtt{b}$$
$$C \rightarrow \mathtt{c}C \mid \mathtt{c}$$

(a) Construa uma gramática equivalente a $G$, sem regras de cadeias.

(b) Mostre que a gramática construída contém símbolos inúteis.

10. Seja $G$ uma gramática e $w \in L(G)$, de tamanho $n$. Para os casos em que $G$ está na forma normal de Chomsky e em que está na forma normal de Greibach, determine:

(a) O tamanho de uma derivação de $w$.

(b) A profundidade máxima de uma AD de $w$.

(c) A profundidade mínima de uma AD de $w$.

11. Seja a GLC $G$:

$$E \rightarrow E{+}E \mid E{*}E \mid \mathtt{a}$$

Construa uma gramática na FNG equivalente a $G$.

12. Construa um algoritmo que, dada uma GLC $G$ arbitrária, obtenha uma GLC equivalente a $G$ na FNG.

13. Obtenha um AP que reconheça a linguagem do Exercício 2(c), a partir da GLC de tal linguagem, usando o método apresentado na prova do Teorema 26.

14. O Teorema 26 apresenta um método para obter um AP a partir de uma gramática na FNG. Altere este método de forma que seja obtido um AP diretamente da GLC original.

15. Seja o autômato com pilha $P = (\{i\}, \{(,)\}, \{\mathtt{0}\}, \delta, \{i\}, \{i\})$ tal que $\delta$ é dada por:

    $\delta(i, (, \lambda) = [i, \mathtt{0}]$
    $\delta(i, ), \mathtt{0}) = [i, \lambda]$.

    Construa uma gramática livre do contexto que gere $L(P)$, utilizando o método do Teorema 27.

16. Seja o autômato com pilha $P = (\{e_0, e_1\}, \{\mathtt{0}, \mathtt{1}\}, \{\mathtt{A}\}, \delta, \{e_0\}, \{e_0, e_1\})$ tal que $\delta$ é dada por:

    $\delta(e_0, \mathtt{0}, \lambda) = [e_0, \mathtt{A}]$
    $\delta(e_0, \mathtt{1}, \mathtt{A}) = [e_1, \lambda]$
    $\delta(e_1, \mathtt{1}, \mathtt{A}) = [e_1, \lambda]$.

    Construa uma gramática livre do contexto que gere $L(P)$, utilizando o método do Teorema 27.

17. Construa um algoritmo que, dado um AP $M$, determine uma GLC que gere $L(M)$. Para isto, ao invés de seguir de forma literal o método apresentado no Teorema 27, as regras deverão ir sendo geradas com base nas variáveis que forem surgindo. Assim, deve-se gerar, inicialmente, as regras $P$. Em seguida, deve-se gerar todas as regras para uma das variáveis que ocorrem do lado direito de uma regra $P$. E assim por diante, as regras com certa variável $X$ do lado esquerdo devem ser geradas, todas elas, somente quando $X$ já tiver aparecido do lado direito de alguma regra já gerada.

18. Seja o APD $M = (\{0, 1\}, \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}, \{A, B\}, \delta, \{1\})$, onde $\delta$ é dada por:

    $\delta(0, \mathtt{a}, \lambda) = \{[0, A]\}$
    $\delta(0, \mathtt{b}, A) = \{[1, B]\}$
    $\delta(1, \mathtt{b}, A) = \{[1, B]\}$
    $\delta(1, \mathtt{c}, B) = \{[1, \lambda]\}$.

    Utilize o algoritmo do exercício 17 ou o método apresentado no Teorema 27 para obter uma GLC que gere $L(M)$. Para cada variável inútil que aparecer, ou, se for o caso, que for descartada pelo algoritmo, explique porque ela é inútil.

19. Complete a prova do Teorema 27 mostrando que

    $$[e, A, e'] \stackrel{*}{\Rightarrow} w \text{ se, e somente se, } [e, w, A] \stackrel{*}{\vdash} [e', \lambda, \lambda]$$

    para todo $e, e' \in E$, $A \in \Gamma \cup \{\lambda\}$ e $w \in \Sigma^*$.

## 3.5 Linguagens Livres do Contexto: Propriedades

Nesta seção serão apresentadas algumas propriedades das LLC's, com propósitos análogos àqueles da Seção 2.4. Inicialmente, será apresentado o lema do bombeamento (LB) para LLC's, cuja aplicação principal é a demonstração de que uma linguagem não é livre de contexto. Em seguida, serão apresentadas algumas propriedades de fechamento para esta classe de linguagens.

Na Seção 2.4, o LB para linguagens regulares foi obtido raciocinando-se a partir dos autômatos que reconhecem tais linguagens. Já o LB para linguagens livres de contexto é mais facilmente obtido raciocinando-se a partir das gramáticas que geram tais linguagens. Mais especificamente, o LB será obtido a partir da estrutura das árvores de derivação associadas a GLC's.

**Lema 5** [8] *Seja $L$ uma linguagem livre do contexto. Então existe uma constante $k > 0$ tal que para qualquer palavra $z \in L$ com $|z| \geq k$ existem $u$, $v$, $w$, $x$ e $y$ que satisfazem as seguintes condições:*

- $z = uvwxy$;
- $|vwx| \leq k$;
- $vx \neq \lambda$; *e*
- $uv^iwx^iy \in L$ *para todo* $i \geq 0$.

*Prova*

Se a LLC $L$ é finita, o lema vale por vacuidade. Assim, seja $G = (V, \Sigma, R, P)$ uma GLC na FNC que gere uma LLC $L$ infinita. Como $L$ é infinita, existe palavra em $L$ de todo tamanho. No entanto, $V$ e $R$ são finitos. Isto nos leva a concluir que existe um número $k > 0$ tal que qualquer palavra $z \in L$ com $|z| \geq k$ terá uma AD da forma mostrada na Figura 3.27, onde $X$ é uma variável que é um rótulo que se repete em algum caminho simples que se inicia na raiz. Nesta figura, $u, v, w, x, y \in \Sigma^*$ e qualquer uma destas subpalavras pode ser $\lambda$. Mas, estando $G$ na forma normal de Chomsky, pelo menos um dentre $v$ e $x$ é diferente de $\lambda$[9]. Logo, tem-se que $|vwx| \leq k$ e $vx \neq \lambda$. Pela estrutura da AD, vê-se que:

- $P \stackrel{*}{\Rightarrow} uXy$;
- $X \stackrel{*}{\Rightarrow} vXx$; e
- $X \stackrel{*}{\Rightarrow} w$.

Tem-se, então, que $P \stackrel{*}{\Rightarrow} uXy \stackrel{*}{\Rightarrow} uv^iXx^iy$, $i \geq 0$, e, portanto, $P \stackrel{*}{\Rightarrow} uv^iwx^iy$, $i \geq 0$. Assim, $uv^iwx^iy \in L$ para todo $i \geq 0$. □

[8] Para aqueles com pendor para formalidade, aqui vai um enunciado mais formal: $L$ é LLC $\rightarrow$ $\exists k \in \mathbf{N} \forall z \in L[|z| \geq k \rightarrow \exists u, v, w, x, y(z = uvwxy \wedge |vwx| \leq k \wedge |vx| \geq 1 \wedge \forall i \in \mathbf{N} uv^iwx^iy \in L)]$

[9] Mesmo que $G$ não esteja na FNC, pode-se ter $k$ grande o suficiente para que hajam $u, v, w, x, y$ em que $v \neq \lambda$ ou $x \neq \lambda$.

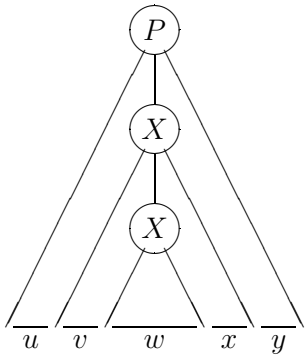



Figura 3.27: Esquema de AD para palavra "grande".

O exemplo a seguir ilustra como utilizar o LB para mostrar que uma linguagem não é livre do contexto.

**Exemplo 118** A linguagem $L = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbf{N}\}$ não é livre do contexto, como mostrado abaixo, por contradição, aplicando-se o lema do bombeamento.

Suponha que $L$ seja uma LLC. Seja $k$ a constante referida no LB, e seja $z = \mathtt{a}^k\mathtt{b}^k\mathtt{c}^k$. Como $|z| > k$, o lema diz que existem $u$, $v$, $w$, $x$ e $y$ de forma que as seguintes condições se verificam:

- $z = uvwxy$;
- $|vwx| \leq k$;
- $vx \neq \lambda$; e
- $uv^iwx^iy \in L$ para todo $i \geq 0$.

Suponha, então, que $\mathtt{a}^k\mathtt{b}^k\mathtt{c}^k = uvwxy$, $|vwx| \leq k$ e $vx \neq \lambda$. Considera-se dois casos:

- $vx$ contém algum $\mathtt{a}$. Como $|vwx| \leq k$, $vx$ não contém $\mathtt{c}$'s. Portanto, $uv^2wx^2y$ contém mais $\mathtt{a}$'s do que $\mathtt{c}$'s. Assim, $uv^2wx^2y \notin L$.
- $vx$ não contém $\mathtt{a}$. Como $vx \neq \lambda$, $uv^2wx^2y$ contém menos $\mathtt{a}$'s do que $\mathtt{b}$'s e/ou $\mathtt{c}$'s. Assim, $uv^2wx^2y \notin L$.

Logo, em qualquer caso $uv^2wx^2y \notin L$, contrariando o LB. Portanto, a suposição original de que $L$ é livre do contexto não se justifica. Conclui-se que $L$ não é LLC. □

**Exemplo 119** A linguagem $L = \{\mathtt{0}^n \mid n \text{ é primo}\}$ não é livre do contexto, como mostrado abaixo.

Suponha que $L$ seja uma LLC. Seja $k$ a constante referida no LB, e seja $z = \mathtt{0}^n$, onde $n$ é um número primo maior que $k$. A existência de $n$ é garantida pelo teorema provado no Exemplo 7 da seção 1.2, página 9. Como $|z| > k$, o lema diz que existem $u$, $v$, $w$, $x$ e $y$ tais que:

- $z = uvwxy$;

- $|vwx| \leq k$;
- $vx \neq \lambda$; e
- $uv^iwx^iy \in L$ para todo $i \geq 0$.

Para provar que $L$ não é livre do contexto, basta então mostrar um $i$ tal que $uv^iwx^iy \notin L$ (contrariando o LB). Pelas informações acima, tem-se que $uv^iwx^iy = \mathtt{0}^{n+(i-1)(|vx|)}$ (pois $z = \mathtt{0}^n$). Assim, $i$ deve ser tal que $n + (i-1)|vx|$ não seja um número primo. Ora, para isto, basta fazer $i = n + 1$, obtendo-se $n + (i-1)|vx| = n + n|vx| = n(1 + |vx|)$, que não é primo (pois $|vx| > 0$). Assim, $uv^{n+1}wx^{n+1}y \notin L$, contradizendo o LB. Logo, $L$ não é LLC. □

O fato de que as LLC's são fechadas sob as operações de união, concatenação e fecho de Kleene pode ser demostrado trivialmente utilizando gramáticas, como mostrado no próximo teorema.

**Teorema 28** *A classe das LLC's é fechada sob união, concatenação e fecho de Kleene.*

*Prova*

Sejas duas LLC's $L_1$ e $L_2$ com gramáticas $G_1 = (V_1, \Sigma_1, R_1, P1)$ e $G_2 = (V_2, \Sigma_2, R_2, P2)$, com $V_1 \cap V_2 = \emptyset$. Uma gramática para $L_1 \cup L_2$ seria $(V_3, \Sigma_3, R_3, P_3)$, onde:

- $V_3 = V_1 \cup V_2 \cup \{P_3\}$;
- $\Sigma_3 = \Sigma_1 \cup \Sigma_2$;
- $R_3 = R_1 \cup R_2 \cup \{P_3 \rightarrow P_1, P_3 \rightarrow P_2\}$; e
- $P_3 \notin V_1 \cup V_2$.

Uma gramática para $L_1L_2$ seria $(V_3, \Sigma_3, R_3, P_3)$, onde:

- $V_3 = V_1 \cup V_2 \cup \{P_3\}$;
- $\Sigma_3 = \Sigma_1 \cup \Sigma_2$;
- $R_3 = R_1 \cup R_2 \cup \{P_3 \rightarrow P_1P_2\}$; e
- $P_3 \notin V_1 \cup V_2$.

Uma gramática para $L_1^*$ seria $(V_3, \Sigma_3, R_3, P_3)$, onde:

- $V_3 = V_1 \cup \{P_3\}$;
- $\Sigma_3 = \Sigma_1$;
- $R_3 = R_1 \cup \{P_3 \rightarrow P_1P_3, P_3 \rightarrow \lambda\}$; e
- $P_3 \notin V_1$. □

O seguinte teorema mostra que, ao contrário das linguagens regulares, as LLC's não são fechadas sob as operações de interseção e complementação.

**Teorema 29** *A classe das LLC's não é fechada sob interseção, nem sob complementação.*

*Prova*

Sejam as linguagens livres de contexto $L_1 = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^k \mid n, k \geq 0\}$ e $L2 = \{\mathtt{a}^n\mathtt{b}^k\mathtt{c}^k \mid n, k \geq 0\}$. Tem-se que $L_1 \cap L_2 = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$, que, como mostrado no Exemplo 118, não é LLC. Dado este contraexemplo, conclui-se que as LLC's não são fechadas sob interseção. De Morgan diz que $L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$. Logo, como as LLC's são fechadas sob união, se elas fossem fechadas sob complementação, seriam fechadas também sob interseção. Assim, as LLC's não são fechadas sob complementação. □

Apesar da classe das LLC's não ser fechada sob interseção, a interseção de uma LLC com uma linguagem regular é sempre uma LLC, como mostra o teorema abaixo.

**Teorema 30** *Seja $L$ uma LLC e $R$ uma linguagem regular. Então $L \cap R$ é uma LLC.*

*Prova*

A idéia, similar à utilizada para construir um AFD para $L_1 \cap L_2$ a partir de AFD's para $L_1$ e $L_2$, é simular o funcionamento "em paralelo" de um APN que reconhece $L$ e de um AFD que reconhece $R$. Evidentemente, a pilha do APN resultante é a pilha do APN que reconhece $L$. Sejam $M_1 = (E_1, \Sigma_1, \Gamma, \delta_1, I, F_1)$ um APN para $L$ e $M_2 = (E_2, \Sigma_2, \delta_2, i, F_2)$ um AFD para $R$. Um APN que reconhece $L \cap R$ é $M_3 = (E_3, \Sigma_3, \Gamma, \delta_3, I', F_3)$, onde:

- $E_3 = E_1 \times E_2$;
- $\Sigma_3 = \Sigma_1 \cap \Sigma_2$;
- para cada par de transições $[e_1', z] \in \delta_1(e_1, a, A)$ e $\delta(e_2, a) = e_2'$, onde $e_1, e_1' \in E_1$, $e_2, e_2' \in E_2$, $a \in \Sigma_3$, $A \in \Gamma \cup \{\lambda\}$, e $z \in \Gamma^*$, há uma transição $[(e_1', e_2'), z] \in \delta_3((e_1, e_2), a, A)$; e para cada transição $\lambda$, $[e_1', z] \in \delta_1(e_1, \lambda, A)$, há transições $[(e_1', e_2), z] \in \delta_3((e_1, e_2), \lambda, A)$ para cada $e_2 \in E_2$.
- $I' = I \times \{i\}$; e
- $F_3 = F_1 \times F_2$.

Pode-se mostrar, por indução sobre $n$, que para todo $i_1 \in I$, $w \in \Sigma^*$, $e_1 \in E_1$ e $e_2 \in E_2$, que $[[i_1, i], w, \lambda] \overset{n}{\vdash}_{M_3} [[e_1, e_2], \lambda, \lambda]$ se, e somente se, $[i_1, w, \lambda] \overset{n}{\vdash}_{M_1} [e_1, \lambda, \lambda]$ e $\hat{\delta}(i, w) = e_2$. □

O exemplo a seguir mostra um uso do Teorema 30 para mostrar que uma linguagem não é LLC.

**Exemplo 120** A linguagem

$$L = \{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid w \text{ tem o mesmo número de } \mathtt{a}\text{'s, } \mathtt{b}\text{'s e } \mathtt{c}\text{'s}\}$$

não é livre do contexto, como mostrado abaixo.

Suponha que $L$ é uma LLC. Então, como $R = L(a^*b^*c^*)$ é uma linguagem regular, pelo Teorema 30 $L \cap R$ é LLC. Mas, $L \cap R = \{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \in \mathbf{N}\}$, que não é LLC, como mostrado no Exemplo 118. Logo, $L$ não é LLC. □

No Capítulo 2 mostrou-se que o problema de determinar se $L(\mathcal{F})$ é vazia, onde $\mathcal{F}$ é um AF, ER ou GR qualquer, é decidível. Este mesmo problema é decidível, caso $\mathcal{F}$ seja um AP ou uma GLC. Já que um AP pode ser tranformado em uma GLC equivalente, e vice-versa, basta usar um dos dois formalismos, AP ou GLC. Seja uma GLC $G = (V, \Sigma, R, P)$ qualquer. Aplicando-se o algoritmo da Figura 3.18(a), página 166, obtém-se o conjunto $\mathcal{I}_1 = \{X \in V \mid X \stackrel{*}{\Rightarrow} w \text{ e } w \in \Sigma^*\}$. Ora, segue-se que:

$$P \in \mathcal{I}_1 \text{ se, e somente se, } L(G) \text{ não é vazia.}$$

Vários problemas decidíveis para linguagens regulares não são decidíveis para linguagens livres do contexto. Alguns exemplos são (estão sendo usadas GLC's, mas poderiam ser AP's):

- Determinar se $L(G) = \Sigma^*$, para qualquer GLC $G$.
- Determinar se $L(G_1) \cap L(G_2) = \emptyset$, para quaisquer GLC's $G_1$ e $G_2$.
- Determinar se $L(G_1) \subseteq L(G_2)$, para quaisquer GLC's $G_1$ e $G_2$.
- Determinar se $L(G_1) = L(G_2)$, para quaisquer GLC's $G_1$ e $G_2$.

A prova da indecidibilidade de alguns destes problemas, assim como de alguns outros, será vista no Capítulo 5.

# Exercícios

1. Mostre que as seguintes linguagens satisfazem o lema do bombeamento para LLC's:
   (a) $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w \text{ tem número par de } \mathtt{0}\text{'s}\}$.
   (b) $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w \text{ tem número igual de } \mathtt{0}\text{'s e } \mathtt{1}\text{'s}\}$.
2. Use o lema do bombeamento para mostrar que as seguintes linguagens não são livres do contexto:
   (a) $\{\mathtt{a}^{n^2} \mid n \geq 0\}$.
   (b) $\{\mathtt{a}^n\mathtt{b}^{2n}\mathtt{a}^n \mid n \geq 0\}$.
   (c) $\{\mathtt{a}^n\mathtt{b}^k\mathtt{c}^n\mathtt{d}^k \mid k, n > 0\}$.
3. Sejam
   $$L_1 = \{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\} \text{ e } L_2 = \{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid |w| \text{ é múltiplo de } 5\}.$$
   Mostre, para cada linguagem abaixo, que ela é ou não uma LLC:
   (a) $\overline{L_1}$.
   (b) $L_1 \cap L_2$.
   (c) $L_1 \cap \overline{L_2}$.
4. Se $X \subseteq L$ e $L$ é uma LLC, $X$ também é LLC?

5. Seja $n_a(w)$ a quantidade de símbolos $a$ na palavra $w$. Assim, por exemplo, $n_0(0010) = 3$ e $n_1(0010) = 1$. Para cada linguagem abaixo, mostre que ela é ou não é LLC:

   (a) $\{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid n_{\mathtt{a}}(w) = n_{\mathtt{b}}(w)\}$.

   (b) $\{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid n_{\mathtt{a}}(w) = n_{\mathtt{b}}(w) = n_{\mathtt{c}}(w)\}$.

   (c) $\{w \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* \mid n_{\mathtt{c}}(w)$ é quadrado perfeito$\}$.

6. Mostre que as LLC's são fechadas sob reverso.

7. Mostre que se $L$ é uma LLC e $R$ uma linguagem regular, então $L - R$ é LLC.

8. Mostre que as LLC's não são fechadas sob diferença.

9. Sejam as definições de homomorfismo e de substituição apresentadas nos Exercícios 18 e 20 da Seção 2.9, páginas 131 e 131. Mostre que as LLC's são fechadas sob substituição e sob homomorfismo. (*Sugestão:* use gramáticas para mostrar o fechamento sob substituição.)

10. Construa um AP para $L_1 = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ $w$ tem mais $\mathtt{1}$'s que $\mathtt{0}$'s$\}$. Construa um AFD para $L_2 = \{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid$ $|w| \geq 2$ e o penúltimo símbolo de $w$ é $\mathtt{1}\}$. Usando o método da prova do Teorema 30 construa um AP para $L_1 \cap L_2$.

11. Mostre que o problema de determinar se $L(M)$ é finita, onde $M$ é um AP arbitrário, é decidível. *Sugestão*: use o lema do bombeamento.

## 3.6 Exercícios

1. Para cada linguagem abaixo, construa um APD, se possível. Se não for possível, construa um APN e explique porque não há APD.

   (a) $\{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\} \times \{\mathtt{0}^n\mathtt{1}^n \mid n \geq 0\}$.

   (b) $\{\mathtt{0}^n\mathtt{1}^n\mathtt{2}^k \mid n, k \geq 0\}$.

   (c) $\{\mathtt{0}^n\mathtt{1}^n\mathtt{0}^k \mid n, k \geq 0\}$.

   (d) $\{\mathtt{0}^n\mathtt{1}^n\mathtt{0}^k \mid n \geq 1$ e $k \geq 0\}$.

   (e) $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w$ tem algum prefixo com mais $\mathtt{1}$'s que $\mathtt{0}$'s$\}$.

   (f) $\{w \in \{\mathtt{0}, \mathtt{1}\}^* \mid w$ tem algum sufixo com mais $\mathtt{1}$'s que $\mathtt{0}$'s$\}$.

2. Seja o conjunto das expressões *booleanas* (EB's), definido recursivamente como segue:

   (a) V e F são EB's;

   (b) se $\alpha$ é uma EB, então ( $\alpha$) é uma EB;

   (c) se $\alpha$ e $\beta$ são EB's, então ($\alpha$&$\beta$), ($\alpha$|$\beta$), ($\alpha$->$\beta$) e ($\alpha$<->$\beta$) são EB's.

Os símbolos V e F significam *verdadeiro* e *falso*. Parênteses são usados para delimitar com precisão o escopo de cada operador (observe que não são permitidos parênteses a mais nem a menos). Os operadores, cujos significados são dados pelas tabelas da Figura 1.2, página 4, são representados por: (negação), `&` (conjunção), `|` (disjunção), `->` (condicional) e `<->` (bicondicional).

   (a) Construa um APD que reconheça as EB's.

   (b) Projete um avaliador de EB's, baseado nas tabelas da Figura 1.2, no estilo daquele do Exemplo 96, página 145.

3. Sejam $M_1$ e $M_2$ APN's. Mostre como construir APN's para:

   (a) $L(M_1) \cup L(M_2)$.

   (b) $L(M_1)L(M_2)$.

   (c) $L(M_1)^*$.

4. Seja um AP cuja pilha pode conter, no máximo, $n$ símbolos. Que limitações terá tal tipo de AP? Justifique sua resposta.

5. Mostre como obter um APN $P$ a partir de um AFN$\lambda$ $M$, tal que $L(P) = L(M)$, sendo que $P$ deve ter apenas dois estados e a pilha deve conter no máximo um símbolo. *Sugestão*: faça o alfabeto de pilha igual ao conjunto de estados do AFN$\lambda$.

6. Considere a classe das linguagens reconhecidas por APD's por estado final. Mostre que ela é ou não fechada sob cada uma das seguintes operações:

   (a) União.

   (b) Interseção.

   (c) Complementação.

7. Mostre que uma linguagem é reconhecida por APD's por estado final e pilha vazia se, e somente se, é reconhecida por pilha vazia.

8. Mostre que se uma linguagem é reconhecida por APD's por estado final e pilha vazia, então ela é reconhecida por estado final.

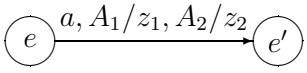
9. Mostre que, para qualquer APD $M$, $L(M)$ é aceita por um APD sem transições $\lambda$.

10. Sejam $L$ uma LLC reconhecida por um APD por estado final, e seja # um símbolo que não pertença ao alfabeto de $L$. Mostre que existe um APD que aceita $L\{\#\}$ por pilha vazia. Use o resultado do Exercício 9.

11. Um autômato com duas pilhas é uma sêxtupla $M = (E, \Sigma, \Gamma, \delta, I, F)$, onde:

    - $E$, $\Sigma$, $\Gamma$,$I$ e $F$ são como em APN's; e
    - $\delta$ é uma função de $E \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\})$ para $D$, onde $D$ é constituído dos subconjuntos finitos de $E \times \Gamma^* \times \Gamma^*$.

Uma transição neste tipo de autômato manipula duas pilhas simultaneamente. Assim, a transição $[e', z_1, z_2] \in \delta(e, a, A_1, A_2)$, que pode ser representada em um diagrama de estados por:




significa, no caso em que nenhuma das entidades envolvidas é $\lambda$, que "estando no estado $e$, se o próximo símbolo de entrada for $a$, o símbolo no topo da pilha 1 for $A_1$ e o símbolo no topo da pilha 2 for $A_2$, há uma transição para o estado $e'$, $A_1$ é desempilhado e $z_1$ é empilhado na pilha 1, e $A_2$ é desempilhado e $z_2$ é empilhado na pilha 2. Faça autômatos com duas pilhas que reconheçam:

(a) $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$

(b) $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n \mid n \geq 0\}$

12. Na Figura 2.38, página 127, está ilustrado um AFD com fita bidirecional, isto é, um AFD cujo cabeçote de leitura pode se movimentar para a esquerda ou para a direita após a leitura de um símbolo. Seja um AP com este mesmo tipo de fita bidirecional. Faça uma formalização deste conceito. Em seguida, construa, se possível, AP's com fitas bidirecionais que reconheçam:

    (a) $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$.

    (b) $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n\mathtt{d}^n \mid n \geq 0\}$.

13. Construa gramáticas livres do contexto para as linguagens:

    (a) $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^{2(m+n)} \mid m, n \geq 0\}$.

    (b) $\{w \in \{a, b\}^* \mid$ o número de $a$'s em $w$ é o dobro do número de $b$'s$\}$.

    (c) $\{w \in \{a, b\}^* \mid$ o número de $a$'s em $w$ é diferente do número de $b$'s$\}$.

    (d) $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^k \mid n > m$ ou $n > k\}$.

    (e) $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^i \mid m + n > i\}$.

    (f) $\{a^m b^n c^p d^q \mid m + n \geq p + q\}$.

    (g) $\{w \in \{a, b\}^* \mid$ $w$ não é da forma $xx\}$.

14. Construa uma GLC que gere todas as expressões regulares sobre o alfabeto $\{\mathtt{0}, \mathtt{1}\}$.

15. Seja $G$ a gramática $(\{P, A, B\}, \{\mathtt{a}, \mathtt{b}\}, R, P)$, onde $R$ consta de:

$$\begin{array}{lcl} P & \rightarrow & APB \mid \lambda \\ A & \rightarrow & \mathtt{a}A\mathtt{b} \mid \lambda \\ B & \rightarrow & \mathtt{b}B\mathtt{a} \mid \mathtt{ba} \end{array}$$

    (a) Construa uma derivação mais à esquerda de `aabbba`.

    (b) Construa a árvore de derivação para a derivação construída em (a).

(c) Defina $L(G)$ utilizando notação de conjunto.

16. Quantas derivações levam à AD da Figura 3.16, página 161? Encontre uma forma de calcular o número de derivações que levam à uma AD qualquer.

17. Seja $G$ a gramática

$$\begin{aligned} P &\rightarrow \mathtt{a}P\mathtt{a} \mid \mathtt{b}P\mathtt{b} \mid \mathtt{a}A\mathtt{b} \\ A &\rightarrow \mathtt{a}A \mid A\mathtt{b} \mid \lambda \end{aligned}$$

(a) Que linguagem é gerada por $G$?

(b) Mostre que $G$ é ambígüa.

(c) Construa uma gramática não ambígüa equivalente a $G$.

18. Seja $G$ a gramática

$$P \rightarrow \mathtt{a}P \mid \mathtt{a}P\mathtt{b}P \mid \lambda$$

Prove que $L(G) = \{x \in \{\mathtt{a},\mathtt{b}\}^* \mid$ todo prefixo de $x$ tem no mínimo tantos a's quantos b's$\}$. Construa um AP que reconheça $L(G)$.

19. Prove que toda linguagem livre do contexto pode ser reconhecida por um AP sem transições $\lambda$. Para isto, mostre como obter um AP a partir de uma GLC na forma normal de Greibach, de forma similar àquela do Teorema 26, porém sem as transições $\lambda$ lá explicitadas.

20. Prove que toda linguagem livre do contexto é gerada por uma gramática na qual cada uma das regras é de uma das formas:

(i) $P \rightarrow \lambda$

(ii) $A \rightarrow a$

(iii) $A \rightarrow aB$

(iv) $A \rightarrow aBC$

onde $P$ é a variável de partida e $A, B, C \in V$ e $a \in \Sigma$. (*Sugestão*: parta de uma gramática na forma normal de Greibach.)

21. Prove que termina o processo utilizado no Teorema 25 para obter uma GLC em que regras da forma $A \rightarrow By$ são tais que o número de $A$ é menor que o de $B$.

22. Prove que existe procedimento de decisão para determinar se uma LLC é:

(a) Vazia.

(b) Finita.

23. Prove que existe procedimento de decisão para determinar, para uma palavra $w$ e uma GLC $G$ arbitrárias, se $w \in L(G)$.

24. Seja $L = \{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^k \mid m \neq n \text{ ou } n \neq k\}$. Mostre que:

    (a) $L$ é uma linguagem livre do contexto.

    (b) $\overline{L}$ não é uma linguagem livre do contexto.

25. Prove que as seguintes linguagens não são linguagens livres do contexto:

    (a) $\{\mathtt{0}^m\mathtt{1}^n\mathtt{2}^k \mid m < n < k\}$.

    (b) $\{\mathtt{0}^n\mathtt{1}^{n^2} \mid n \geq 0\}$.

    (c) $\{\mathtt{0}^n\mathtt{1}^n\mathtt{2}^k \mid n \leq k \leq 2n\}$.

    (d) $\{ww \mid w \in \{\mathtt{0},\mathtt{1}\}^*\}$.

    (e) $\{ww^Rw \mid w \in \{\mathtt{0},\mathtt{1}\}^*\}$.

26. Construa uma GLC para o complemento de $\{ww \mid w \in \{\mathtt{0},\mathtt{1}\}^*\}$.

27. Prove que as seguintes afirmativas são ou não verdadeiras:

    (a) Se $L$ é uma linguagem livre do contexto e $F$ é finita, então $L - F$ é linguagem livre do contexto.

    (b) Se $L$ é uma linguagem livre do contexto e $R$ é regular, então $L - R$ é linguagem livre do contexto.

    (c) Se $L$ não é uma linguagem livre do contexto e $F$ é finita, então $L - F$ não é linguagem livre do contexto.

    (d) Se $L$ não é uma linguagem livre do contexto e $R$ é regular, então $L - R$ não é linguagem livre do contexto.

    (e) Se $L$ não é uma linguagem livre do contexto e $F$ é finita, então $L \cup F$ não é linguagem livre do contexto.

    (f) Se $L$ não é uma linguagem livre do contexto e $R$ é regular, então $L \cup R$ não é linguagem livre do contexto.

## 3.7 Notas Bibliográficas

Os autômatos de pilha foram propostos por Oettinger em [Oet61]. A equivalência dos mesmos e gramáticas livres do contexto foi mostrada por Chomsky[Cho62], Schützenberger[Sch63] e Evey[Eve63]. Os autômatos com pilha determinísticos foram estudados por Fischer[Fis63], Schützenberger[Sch63], Haines[Hai91] e Ginsburg e Greiback[GG66].

Bar-Hillel, Perles e Shamir[BPS61] foram os primeiros a explicitar o lema do bombeamento para linguagens livres do contexto. Uma versão mais forte do lema, que não foi abordada neste livro, mas também muito importante, é a de Ogden[Ogd68].

Algumas propriedades de fechamento para linguagens livres do contexto foram explicitadas por Scheinberg[Sch60], Bar-Hillel, Perles e Shamir[BPS61], Ginsburg e Rose[GR63a][GR66], e Ginsburg e Spanier[GS63].

As gramáticas livres do contexto foram propostas por Noam Chomsky[Cho56], [Cho59]. A notação BNF, concebida para a especificação da sintaxe de linguagens de programação,

é obra de Backus[Bac59] e Naur[Nau63]. Bar-Hillel, Perles e Shamir[BPS61], já citados, mostraram também como eliminar regras $\lambda$ e regras unitárias. A formal normal de Chomsky foi proposta em [Cho59], e a forma normal de Greibach em [Gre65].

# Capítulo 4

# Máquinas de Turing

> So many ideas and technological advances converged to create the modern computer that it is foolhardy to give one person the credit for inventing it. But the fact remains that every one who taps at a keyboard, opening a spreadsheet or a word-processing program, is working on an incarnation of a Turing machine.
>
> *Time Magazine* (March 29, 1999) *apud* M. Davis[Dav00]

Nos capítulos anteriores foram estudados dois tipos básicos de máquinas: os autômatos finitos e os autômatos com pilha. Apesar da importância destes dois tipos de máquinas, tanto do ponto de vista prático quanto teórico, eles têm limitações importantes que devem ser sobrepujadas caso se queira aumentar a classe das linguagens que podem ser reconhecidas. Por exemplo, linguagens relativamente simples, como $\{xx \mid x \in \{\mathtt{a},\mathtt{b}\}^*\}$, $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$ e $\{\mathtt{a}^n\mathtt{b}^k\mathtt{c}^n\mathtt{d}^k \mid n,k \geq 0\}$ não podem ser reconhecidas por AF's nem por AP's.

A seguir serão apresentadas as máquinas de Turing, uma classe de máquinas proposta por volta de 1930 pelo matemático inglês Alan Turing, tão poderosa que até hoje não se conseguiu nenhum outro tipo de máquina que tenha maior poder computacional. Em particular, se forem considerados os computadores hoje existentes, nenhum deles tem poder computacional maior do que o das máquinas de Turing.

Inicialmente, na Seção 4.1, é apresentado o conceito de máquina de Turing, assim como as duas classes de linguagens importantes para o que virá no próximo capítulo, as linguagens recursivas e recursivamente enumeráveis. Em seguida, na Seção 4.2, são apresentadas algumas variações de máquinas de Turing que, apesar de não aumentar o poder computacional das mesmas, facilita o tratamento dos assuntos de seções posteriores. Na Seção 4.3, é visto o relacionamento entre gramáticas e máquinas de Turing, culminando com a apresentação de uma hierarquia gramatical, denominada *hierarquia de Chomsky*, e da hierarquia completa de todas as classes de linguagens tratadas neste texto. Finalmente, na Seção 4.4, são apresentadas algumas propriedades das linguagens recursivas e recursivamente enumeráveis.

## 4.1 O que é Máquina de Turing

Uma máquina de Turing (MT) pode ser vista como uma máquina que opera com uma fita na qual, ao contrário dos autômatos finitos e dos autômatos com pilha, pode-se também

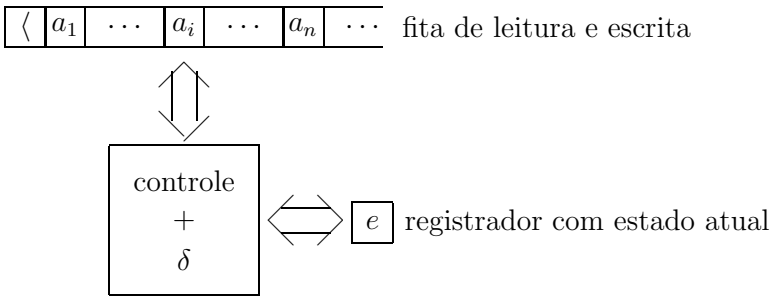



Figura 4.1: Arquitetura de uma Máquina de Turing.

escrever, além de ler (veja Figura 4.1). O cabeçote de leitura pode se movimentar para a direita e para a esquerda. A fita é dividida em células que comportam apenas um símbolo cada uma, e é ilimitada à direita. Além da fita, a máquina possui um registrador para conter o estado atual, um conjunto de instruções, que nada mais é do que a função de transição da máquina, e uma unidade de controle (estas duas últimas estão representadas na Figura 4.1 juntas).

Como o cabeçote de leitura pode se movimentar para a esquerda e a fita não é ilimitada à esquerda, existe na primeira célula da fita um símbolo especial, $\langle$, com o propósito de evitar movimentação do cabeçote para a esquerda de tal célula. Tal símbolo não pode ocorrer em nenhuma outra célula da fita.[1]

No início, o registrador da máquina contém o estado inicial e a fita contém a palavra de entrada a partir da sua segunda célula; o restante da fita, com exceção da primeira célula, que contém $\langle$, contém somente o símbolo $\sqcup$, o qual denota *branco*, ou "célula vazia".[2] O cabeçote é posicionado no início da palavra de entrada, ou seja, na segunda célula. A função de transição, uma função parcial, dá, para cada par $(e, a)$, onde $e$ é um estado e $a$ é um símbolo, uma tripla $[e', b, d]$, onde:

- $e'$ é o próximo estado;
- $b$ é o símbolo a substituir $a$; e
- $d$ é a direção, esquerda (E) ou direita (D) , em que o cabeçote deve se mover.

No caso em que $a = \langle$, obrigatoriamente $b = \langle$ e $d =$ D ou $\delta(e, \langle)$ é indefinido, pois não é permitido movimentar o cabeçote para a esquerda da primeira posição da fita. Observe que o símbolo $\langle$, além de não poder ser escrito em qualquer outra célula da fita, não pode ser apagado da primeira célula da fita.

A unidade de controle de uma MT repete a seguinte seqüência, enquanto $\delta(e, a)$ é definido, onde $e$ é o estado no registrador da máquina, $a$ é o símbolo sob o cabeçote e $\delta(e, a) = [e', b, d]$:

1) coloca no registrador o estado $e'$;

[1]Na verdade, o primeiro símbolo da fita não precisa ser $\langle$, mas é importante que ele não ocorra em nenhuma outra célula da fita.

[2]Aqui também pode-se usar outro símbolo ao invés de $\sqcup$, desde que apenas para este propósito.

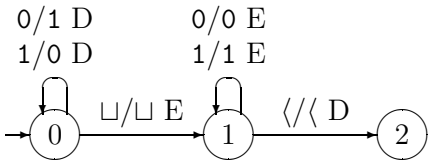
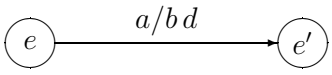



Figura 4.2: Uma MT para complementação da entrada.

2) substitui $a$ por $b$ na posição sob o cabeçote; e

3) avança o cabeçote para a célula da esquerda, se $d = \mathrm{E}$, ou para a da direita, se $d = \mathrm{D}$.

Observe que uma MT é *determinística*: para cada estado $e$ e símbolo $a$ há, no máximo, uma transição especificada pela função de transição. Uma transição $\delta(e, a) = [e', b, d]$, onde $d \in \{\mathrm{E}, \mathrm{D}\}$, será representada assim em um diagrama de estados:




Uma MT pode ser usada como reconhecedora de linguagens e também como transdutora. Neste último caso, a MT, recebendo na fita como entrada uma palavra $w$, produz na própria fita a saída respectiva. Apesar do enfoque deste texto privilegiar o uso de MT's como reconhecedoras de linguagens, o primeiro exemplo, apresentado a seguir, trata de uma MT do tipo transdutora.

**Exemplo 121** A Figura 4.2 apresenta o diagrama de estados de uma MT que, recebendo como entrada uma palavra de $\{\mathtt{0}, \mathtt{1}\}^*$, produz o complemento da mesma, isto é, substitui os `0`'s por `1`'s e os `1`'s por `0`'s. Após fazer isto, a MT retorna o cabeçote para o início da saída produzida que, no caso, substitui a palavra de entrada. Esta técnica de substituir a entrada pela saída e posicionar o cabeçote no início da palavra de saída pode ser útil quando se compõe várias MT's para a obtenção de uma outra. □

Segue uma definição de máquina de Turing, já introduzindo o conceito de estado final, de modo a propiciar, posteriormente, a definição do conceito de reconhecimento.

**Definição 43** *Uma* máquina de Turing *é uma óctupla* $(E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$*, onde:*

*(a)* $E$ *é um conjunto finito de* estados*;*

*(b)* $\Sigma \subseteq \Gamma$ *é o* alfabeto de entrada*;*

*(c)* $\Gamma$ *é o* alfabeto da fita*, que contém todos os símbolos que podem aparecer na fita;*

*(d)* $\langle$ *é o primeiro símbolo da fita (*$\langle \in \Gamma - \Sigma$*);*

*(e)* $\sqcup$ *é o branco (*$\sqcup \in \Gamma - \Sigma$*,* $\sqcup \neq \langle$*);*

*(f)* $\delta : E \times \Gamma \to E \times \Gamma \times \{E, D\}$ *é a* função de transição*, uma função parcial;*

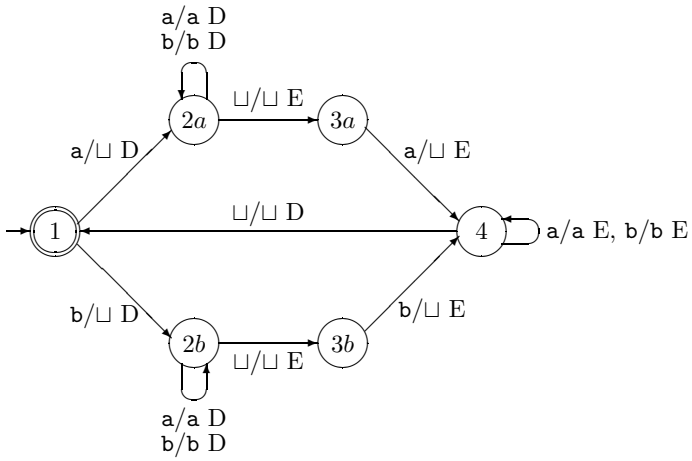



Figura 4.3: MT para palíndromos pares.

*(g) $i$ é o estado inicial;*

*(h) $F \subseteq E$ é um conjunto de estados finais.* □

Para distingui-la das outras versões a serem apresentadas na Seção 4.2, uma MT como definida acima será chamada de MT *padrão*.

Segue um exemplo de MT utilizada como reconhecedora de linguagem.

**Exemplo 122** Na Figura 4.3 está mostrado um diagrama de estados para uma MT que reconhece a linguagem dos palíndromos de tamanho par no alfabeto $\{\mathtt{a}, \mathtt{b}\}$. Formalmente, tal MT é a óctupla $(\{1, 2a, 2b, 3a, 3b, 4\}, \{\mathtt{a}, \mathtt{b}\}, \{\langle, \sqcup, \mathtt{a}, \mathtt{b}\}, \langle, \sqcup, \delta, 1, \{1\})$, onde $\delta$ consta das transições:

$$\delta(1, \mathtt{a}) = [2a, \sqcup, \mathrm{D}],\ \delta(1, \mathtt{b}) = [2b, \sqcup, \mathrm{D}],\ \delta(2a, \mathtt{a}) = [2a, \mathtt{a}, \mathrm{D}],\ \text{etc.}$$

Em resumo, a máquina verifica qual é o primeiro símbolo da palavra de entrada, apaga-o, percorre o resto da palavra até o final, verifica se o último símbolo é idêntico ao primeiro (já apagado), apaga-o, volta o cabeçote para o início, e repete o processo. Pode-se verificar que a máquina pára no estado 1 se, e somente se, a palavra de entrada é da forma $xx^R$ para $x \in \{\mathtt{a}, \mathtt{b}\}^*$. □

Seja uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$. Uma *configuração instantânea* de $M$ é um par $[e, x\underline{a}y]$, onde:

- $e \in E$ é o estado atual;
- $x \in \Gamma^*$ é a palavra situada à esquerda do cabeçote de leitura;
- $a \in \Gamma$ é o símbolo sob o cabeçote; e
- $y \in \Gamma^*$ é a palavra à direita do cabeçote até o último símbolo diferente de $\sqcup$; se não existir símbolo diferente de $\sqcup$, $y = \lambda$.

A configuração inicial, por exemplo, é $[i, \langle \underline{a_1} a_2 \dots a_n]$, caso a palavra de entrada seja $a_1 a_2 \dots a_n$. Caso a palavra de entrada seja $\lambda$, a configuração inicial é $[i, \langle \underline{\sqcup}]$.

Como uma MT pode entrar em *loop*, não será definida uma função que retorne o estado alcançado a partir de uma certa configuração instantânea. Ao invés disso, será definida a relação $\vdash$ mais à frente. Para facilitar a definição, será utilizada a função $\pi : \Gamma^* \to \Gamma^*$, definida a seguir. Informalmente, $\pi(w)$ elimina de $w$ os brancos à direita do último símbolo diferente de branco.

$$\pi(w) = \begin{cases} \lambda & \text{se } w \in \{\sqcup\}^* \\ xa & \text{se } w = xay, a \in \Gamma - \{\sqcup\} \text{ e } y \in \{\sqcup\}^*. \end{cases}$$

**Definição 44** *Seja uma máquina de Turing $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$. A relação $\vdash \subseteq (E \times \Gamma^+)^2$, para $M$, é tal que para todo $e \in E$ e todo $a \in \Gamma$:*

*(a) se $\delta(e, a) = [e', b, D]$, então $[e, x\underline{a}cy] \vdash [e', xb\underline{c}y]$ para $c \in \Gamma$, e $[e, x\underline{a}] \vdash [e', xb\underline{\sqcup}]$;*

*(b) se $\delta(e, a) = [e', b, E]$, então $[e, xc\underline{a}y] \vdash [e', x\underline{c}\pi(by)]$ para $c \in \Gamma$;*

*(c) se $\delta(e, a)$ é indefinido, então não existe configuração $f$ tal que $[e, x\underline{a}y] \vdash f$.* □

Note, no item (b), que não é possível uma transição para a esquerda se o cabeçote se encontrar posicionado na primeira célula (o fato de que $c \in \Gamma$ garante isto, pois, neste caso, $xc \neq \lambda$).

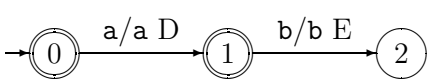
Como usual, $\overset{*}{\vdash}$ será usado para denotar o fecho reflexivo e transitivo de $\vdash$, e $f \overset{n}{\vdash} f'$, para $n \geq 0$, será usado para significar que a configuração instantânea $f'$ é obtida a partir de $f$ percorrendo-se $n$ transições.

**Exemplo 123** Pode-se verificar que, para a máquina do Exemplo 122 (veja a Figura 4.3):

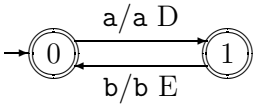
- $[1, \langle \underline{\sqcup}] \overset{0}{\vdash} [1, \langle \underline{\sqcup}]$ e $\delta(1, \sqcup)$ é indefinido.
- $[1, \langle \underline{\mathtt{a}}\mathtt{ab}] \overset{4}{\vdash} [3a, \langle \sqcup \mathtt{a}\underline{\mathtt{b}}]$ e $\delta(3a, \mathtt{b})$ é indefinido.
- $[1, \langle \underline{\mathtt{a}}\mathtt{bba}] \overset{14}{\vdash} [1, \langle \sqcup \sqcup \underline{\sqcup}]$ e $\delta(1, \sqcup)$ é indefinido.

Como 1 é estado final e $3a$ não é, segue-se que $\lambda$ e `abba` são aceitas e que `aab` não é aceita. □

Como uma MT é determinística, se a máquina parar existirá um único estado para o qual isto acontece. Seja $[e, x\underline{a}y]$ a configuração instantânea no momento em que uma MT $M$ pára. Existe uma única situação que provoca parada em tal configuração, correspondente à situação prevista no item (c) da Definição 44: quando $\delta(e, a)$ é indefinido. Partindo-se da configuração inicial, se a máquina parar e $e$ for um estado final, a palavra de entrada é aceita. Se a máquina parar e $e$ não for estado final, a palavra de entrada não é aceita. Por outro lado, se a máquina não parar, a palavra de entrada não é aceita. Daí, a definição abaixo. Daqui para frente, a expressão $[i, \langle \underline{w}]$ será usada para denotar a configuração instantânea inicial. Assim, se $w = \lambda$, $[i, \langle \underline{w}]$ significa o mesmo que $[i, \langle \underline{\sqcup}]$; e se $w = ay$, onde $a$ é um símbolo, $[i, \langle \underline{w}]$ é o mesmo que $[i, \langle \underline{a}y]$.




(a) Máquina que pára sempre.




(b) Máquina que pára se aceita.

Figura 4.4: Duas MT's para $\{a,b,c\}^* - (\{ab\}\{a,b,c\}^*)$

**Definição 45** *Seja uma MT* $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$. *A* linguagem reconhecida *por* $M$ *é*

$$L(M) = \{w \in \Sigma^* \mid [i, \langle \underline{w}] \overset{*}{\vdash} [e, x\underline{a}y], \delta(e,a) \text{ é indefinido e } e \in F\}.$$

*A palavra* $w \in \Sigma^*$ *tal que* $[i, \langle \underline{w}] \overset{*}{\vdash} [e, x\underline{a}y], \delta(e,a)$ *é indefinido e* $e \in F$ *é dita ser* aceita *(*reconhecida*) pela máquina.* □

**Exemplo 124** Na Figura 4.4 estão mostrados os diagramas de estado de duas MT's que aceitam a linguagem das palavras no alfabeto $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ que não têm `ab` como prefixo, ou seja, $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^* - (\{\mathtt{ab}\}\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*)$.

Para a máquina da Figura 4.4(a), se a palavra de entrada não começar com `a`, ocorre uma parada no estado final 0. E se a palavra de entrada começar com `a` e tal `a` não for seguido de `b`, ocorre uma parada no estado final 1. Apenas se a palavra de entrada começar com `ab`, o estado 2 é atingido. Neste último caso, como 2 não é estado final, a palavra de entrada não é aceita. Observe que a máquina pára para toda palavra.

Para a máquina da Figura 4.4(b), se a palavra de entrada não começar com `a`, ocorre uma parada no estado final 0, de forma análoga ao que acontece para a máquina da Figura 4.4(a). Também de forma análoga ao que acontece para esta última, se a palavra de entrada começar com `a` e tal `a` não for seguido de `b`, ocorre uma parada no estado final 1. A diferença ocorre quando a palavra de entrada começa com `ab`: na máquina da Figura 4.4(a), ocorre parada no estado 2, e na máquina da Figura 4.4(b), ocorre uma computação ilimitada, com o cabeçote se movendo da primeira posição, que contém `a` para a segunda, que contém `b`, e vice-versa, sem parar. Assim, a máquina da Figura 4.4(b) também não reconhece as palavras que começam com `ab`. Observe que, como todos os estados desta máquina são estados finais, ela aceita uma palavra $w$ se, e somente se, ela parar quando acionada com $w$. □

O exemplo anterior ilustra, além do papel que a "não parada" tem com relação ao reconhecimento, uma faceta diferente das MT's com relação aos AF's e AP's: não é necessário que uma palavra de entrada seja toda lida para que ela possa ser aceita ou rejeitada.

Um AFD pode ser simulado, de forma trivial, por uma MT cujos movimentos estão restritos à direção D. Detalhes são deixados para um exercício do final desta seção. Um AP também pode ser simulado por meio de uma MT, embora tal simulação dê um pouco mais de trabalho: basta acomodar a pilha na fita após a palavra de entrada. A simulação de AP's se torna mais fácil após introduzidos alguns incrementos na próxima seção. Assim, as linguagens livres do contexto podem ser reconhecidas por MT's.

A definição a seguir dá um nome para a classe das linguagens que podem ser reconhecidas por MT's.

**Definição 46** *Uma linguagem é dita ser uma* linguagem recursivamente enumerável *se existe uma MT que a reconhece.* □

Existem fortes evidências de que não há outro modelo de máquina que tenha maior poder computacional do que MT. Os outros modelos, até agora propostos, nunca foram além das LRE's: o poder computacional de tais modelos alternativos, na melhor das hipóteses, se revelou o mesmo que o das MT's.

Como já foi visto, algumas MT's, como, por exemplo, a da Figura 4.4(b), não param para algumas palavras. E tais palavras não são reconhecidas, conforme estabelecido pela Definição 45. Assim, surge a questão: se existe uma MT que reconhece uma linguagem $L$, necessariamente existe uma MT que *sempre pára* e que reconhece $L$? A resposta é não: na verdade, existem LRE's para as quais não existem MT's que parem para todas as palavras que *não* pertençam à linguagem. Daí, a definição a seguir.

**Definição 47** *Uma linguagem é dita ser uma* linguagem recursiva *se existe uma MT que a reconhece e que pare para todas as palavras do alfabeto de entrada.* □

No resto desta seção, serão apresentados mais dois modelos alternativos de reconhecimento para MT's, os quais são úteis em certos contextos. Uma MT cujo reconhecimento se dá como fixado na Definição 45 será dita uma MT que reconhece por *parada em estado final*.

No primeiro modelo alternativo de reconhecimento, ao invés de uma palavra ser reconhecida quando a máquina parar em um estado final, ela será reconhecida quando a máquina atingir um estado final, simplesmente. Utilizando o índice $F$, para enfatizar o *reconhecimento por estado final*, segue uma definição.

**Definição 48** *Seja uma MT* $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$. *A* linguagem reconhecida *por* $M$ por estado final *é*

$$L_F(M) = \{w \in \Sigma^* \mid [i, \langle \underline{w}] \overset{*}{\vdash} [e, x\underline{a}y], a \in \Gamma \ e \ e \in F\}.$$

*A palavra* $w \in \Sigma^*$ *tal que* $[i, \langle \underline{w}] \overset{*}{\vdash} [e, x\underline{a}y]$ $e$ $e \in F$ *é dita ser* aceita *(*reconhecida*) por* $M$ por estado final. □

Pela definição acima, ao ser atingida uma configuração $[e, x\underline{a}y]$ tal que $e$ é estado final, a palavra de entrada é reconhecida, mesmo que $\delta(e, a)$ seja definido; em particular, mesmo se a máquina entrar em *loop* a partir de tal configuração. Assim, é claro que toda transição que emane de um estado final é inócua, podendo ser eliminada sem alterar a linguagem reconhecida pela máquina. Em outras palavras, se $M$ reconhece uma linguagem por estado final, pode-se considerar que $\delta(e, a)$ é indefinido para todo estado final $e$ e todo símbolo $a$ do alfabeto da fita. Ora, neste caso, os estados finais são todos equivalentes, e podem ser reduzidos a *um só*!

**Exemplo 125** Na Figura 4.5 está o diagrama de estados de uma MT equivalente àquelas cujos diagramas de estados estão mostrados na Figura 4.4, mas que reconhece por estado final. □

Observe também que a máquina da Figura 4.5 reconhece a mesma linguagem por parada em estado final, assim como a máquina da Figura 4.4(a), visto que não há transições emanando do estado $f$, condição esta que garante a parada da máquina ao ser atingido este único estado final.

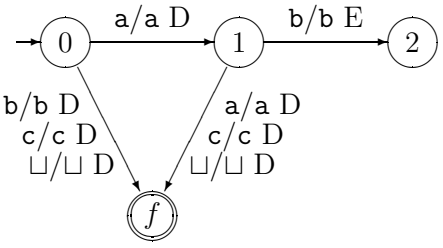



Figura 4.5: Máquina que reconhece por estado final.

No segundo modelo alternativo de reconhecimento, uma palavra é reconhecida quando a máquina pára (em qualquer estado). Utilizando o índice $P$ para enfatizar o *reconhecimento por parada*, segue uma definição.

**Definição 49** *Seja uma MT* $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i)$. *A* linguagem reconhecida *por* $M$ por parada *é*

$$L_P(M) = \{w \in \Sigma^* \mid [i, \langle \underline{w}] \overset{*}{\vdash} [e, x\underline{a}y], a \in \Gamma \text{ e } \delta(e, a) \text{ é indefinido}\}.$$

*A palavra* $w \in \Sigma^*$ *tal que* $[i, \langle \underline{w}] \overset{*}{\vdash} [e, x\underline{a}y]$, $a \in \Gamma$ *e* $\delta(e, a)$ *é indefinido é dita ser* aceita *(*reconhecida*) por* $M$ por parada. □

A MT cujo diagrama de estados está mostrado na Figura 4.4(b) aceita a mesma linguagem por parada em estado final e por parada simplesmente, já que todos os seus estados são finais na modalidade de reconhecimento por parada em estado final.

O teorema a seguir mostra que as três modalidades de reconhecimento são equivalentes.

**Teorema 31** *Seja* $L$ *uma linguagem. As seguintes afirmativas são equivalentes:*

*(a)* $L$ *é uma LRE.*

*(b)* $L$ *pode ser reconhecida por uma MT por estado final.*

*(c)* $L$ *pode ser reconhecida por uma MT por parada.*

*Prova*

(a)→(b) Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ uma MT padrão. Uma máquina equivalente a $M$, que reconhece por estado final, seria $M' = (E \cup \{f\}, \Sigma, \Gamma, \langle, \sqcup, \delta', i, \{f\})$, $f \notin F$, onde:

(a) para todo $(e, a) \in E \times \Gamma$, se $\delta(e, a)$ é definido, então $\delta'(e, a) = \delta(e, a)$;

(b) para todo $(e, a) \in F \times \Gamma$, se $\delta(e, a)$ é indefinido, $\delta'(e, a) = [f, a, \mathrm{D}]$;

(c) para todo $(e, a) \in (E - F) \times \Gamma$, se $\delta(e, a)$ é indefinido, $\delta'(e, a)$ é indefinido;

(d) para todo $a \in \Gamma$, $\delta'(f, a)$ é indefinido.

Observe que o cabeçote se move para a direita na situação $(b)$, prevendo-se o caso em que $a = \langle$.

(b)→(c) Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ uma MT que reconhece por estado final. Uma máquina equivalente a $M$, que reconhece por parada, seria $M' = (E \cup \{l\}, \Sigma, \Gamma, \langle, \sqcup, \delta', i)$, $l \notin E$, onde:

(a) para todo $(e, a) \in (E - F) \times \Gamma$: se $\delta(e, a)$ é definido, então $\delta'(e, a) = \delta(e, a)$, senão $\delta'(e, a) = [l, a, \mathrm{D}]$;

(b) para todo $a \in \Gamma$, $\delta'(l, a) = [l, a, \mathrm{D}]$; e

(c) para todo $(e, a) \in F \times \Gamma$, $\delta'(e, a)$ é indefinido.

Observe que o cabeçote é movido indefinidamente para a direita ao ser atingido o estado $l$, conforme especifica o item (b).

(c)→(a) Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i)$ uma MT que reconhece por parada. Uma MT normal equivalente a $M$ é obtida simplesmente tornando-se todos os estados de $M$ estados finais: $M' = (E, \Sigma, \Gamma, \langle, \sqcup, \delta', i, E)$. □

A partir deste momento, quando não se disser o contrário, é assumido que o reconhecimento se dá por parada em estado final.

# Exercícios

1. Construa uma MT que, recebendo como entrada um número na notação binária, some 1 ao mesmo e retorne o cabeçote para a posição inicial. Se a palavra de entrada for $\lambda$, a MT deverá escrever `0`.

2. Construa uma MT com alfabeto de entrada $\{\mathtt{a}\}$ que, recebendo como entrada uma palavra $w$, concatena $w$ imediatamente à sua direita e retorna o cabeçote para o início. Por exemplo, se a configuração inicial for $[i, \langle \underline{\mathtt{a}}\mathtt{aa}]$, a configuração final deve ser $[i, \langle \underline{\mathtt{a}}\mathtt{aaaaa}]$.

3. Construa uma MT com alfabeto de entrada $\{\mathtt{a}\}$ que pare se, e somente se, a palavra de entrada for da forma $\mathtt{a}^{2n}$ para $n \geq 0$.

4. Construa uma MT que reconheça a linguagem denotada pela ER $a(a + b)^*$, assumindo que o alfabeto é $\{\mathtt{a}, \mathtt{b}\}$, de forma que ela tenha:

   (a) Um número mínimo de estados.

   (b) Um número mínimo de transições.

5. Altere a MT do Exemplo 122, cujo diagrama de estados está mostrado na Figura 4.3, página 204, para que sejam reconhecidos *todos* os palíndromos.

6. Construa MT's para as seguintes linguagens:

   (a) $\{\mathtt{a}^{2n} \mid n \geq 0\}$.

(b) $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$.

(c) $\{\mathtt{a}^m\mathtt{b}^n \mid m \neq n\}$.

(d) $\{w \in \{\mathtt{a}, \mathtt{b}\} \mid$ o número de $\mathtt{a}$'s em $w$ é igual ao de $\mathtt{b}$'s$\}$.

(e) $\{\mathtt{a}^n\mathtt{b}^k\mathtt{c}^n\mathtt{d}^k \mid n, k \geq 0\}$.

(f) $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$.

(g) $\{xx \mid x \in \{\mathtt{a}, \mathtt{b}\}^*\}$.

7. Mostre como construir uma MT para uma linguagem da forma:

$$\{\mathtt{a}^{in+j} \mid n \geq 0\},$$

sendo $i$ e $j$ duas constantes maiores ou iguais a zero.

8. Mostre como construir MT's para linguagens das formas:

(a) $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}^k$;

(b) $\{x^k \mid x \in \{\mathtt{a}, \mathtt{b}\}^*\}$;

sendo $k$ uma constante maior que zero.

9. Mostre em detalhes como simular um AFD por meio de uma MT.

10. Seja a linguagem dos parênteses balanceados, que é gerada pela gramática $(\{P\}, \{(, )\}, R, P)$, onde $R$ consta das regras:

$$P \rightarrow \lambda \mid (P) \mid PP$$

Construa uma MT que reconheça tal linguagem.

11. Mostre como construir:

(a) Uma MT que reconhece por parada em estado final equivalente a uma que reconhece por estado final.

(b) Uma MT que reconhece por estado final equivalente a uma que reconhece por parada.

(c) Uma MT que reconhece por parada equivalente a uma MT que reconhece por parada em estado final.

## 4.2 Algumas Variações de MT's

Na seção anterior, foi definido um modelo padrão para MT (com parada em estado final), com duas versões alternativas de reconhecimento: por estado final e por parada. Nesta seção serão apresentadas algumas variações de MT's, sempre com reconhecimento por parada em estado final. Apesar de não ser mostrado explicitamente neste texto, também tais variações não perdem nem ganham em poder de reconhecimento, quando se considera versões de reconhecimento por estado final ou por parada. As variações a

serem apresentadas não aumentam o poder de reconhecimento das máquinas, mas podem ser mais cômodas de usar em determinados contextos.

Cada variação terá um "incremento" com relação à MT padrão definida na seção anterior. Os incrementos das várias máquinas podem ser combinados entre si, sem, ainda assim, aumentar o poder de reconhecimento. Fica ressaltado que a lista de variações a serem apresentadas a seguir não esgota as possibilidades, mas é suficiente para ilustrar o poder de reconhecimento das MT's e para simplificar o tratamento do assunto a ser apresentado nas próximas seções.

### 4.2.1 Máquina com cabeçote imóvel

Nesta variação permite-se que o cabeçote possa ficar imóvel em uma transição. Assim, a máquina é uma óctupla $(E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$, onde $E$, $\Sigma$, $\Gamma$, $\langle$, $\sqcup$, $i$ e $F$ são como em MT's padrão, e $\delta$ é uma função de $E \times \Gamma$ para $E \times \Gamma \times \{\mathtt{D}, \mathtt{E}, \mathtt{I}\}$. A única diferença é que pode haver transição do tipo $\delta(e, a) = [e', b, \mathrm{I}]$, onde I indica que o cabeçote deve ficar imóvel.

Evidentemente, uma transição do tipo $\delta(e, a) = [e', b, \mathrm{I}]$ pode ser simulada por transições das formas a seguir, sendo $d$ um *novo* estado:

- $\delta(e, a) = [d, b, \mathrm{D}]$;
- $\delta(d, c) = [e', c, \mathrm{E}]$ para cada $c \in \Gamma - \{\langle\}$.

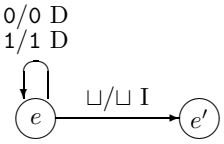
Isto basta para mostrar que o poder de reconhecimento de um máquina cujo cabeçote pode ficar imóvel não é maior do que o de uma MT padrão.

**Exemplo 126** A Figura 4.6(a) apresenta um trecho do diagrama de estados de uma máquina com alfabeto de fita igual a $\{\mathtt{0}, \mathtt{1}, \sqcup, \langle\}$, que move o cabeçote para a célula seguinte a uma palavra de $\{\mathtt{0}, \mathtt{1}\}^*$, supondo que após tal palavra deva seguir necessariamente o símbolo $\sqcup$, que após este $\sqcup$ pode ocorrer qualquer símbolo de $\{\mathtt{0}, \mathtt{1}, \sqcup\}$, e que antes da referida palavra pode ocorrer qualquer símbolo de $\{\mathtt{0}, \mathtt{1}, \sqcup, \langle\}$. As Figuras 4.6(b) e (c) mostram duas alternativas para simular tal trecho em uma máquina padrão, sendo que a máquina da Figura 4.6(c) está em conformidade com a técnica de simulação apresentada no parágrafo anterior. □

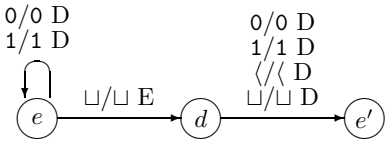
### 4.2.2 Máquina com múltiplas trilhas

Em uma MT com múltiplas trilhas, a fita é composta de múltiplas trilhas, isto é, cada célula da fita, ao invés de receber um símbolo, recebe uma $k$-upla de símbolos, como mostra a Figura 4.7. Assume-se que no início a trilha 1 contém $\langle$ na primeira posição, a palavra de entrada está na trilha 1 a partir da segunda posição, e o restante da trilha 1 e todas as outras trilhas contêm $\sqcup$.

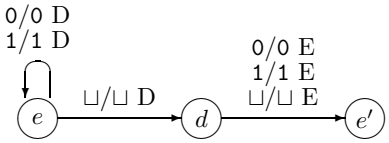
Uma *máquina de Turing com k trilhas* é uma óctupla $(E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$, onde $E$, $\Sigma$, $\Gamma$, $\langle$, $\sqcup$, $i$ e $F$ são como em MT's padrão, e $\delta$ é uma função de $E \times \Gamma^k$ para $E \times \Gamma^k \times \{\mathtt{D}, \mathtt{E}\}$. Assim, uma transição tem a forma $\delta(e, a_1, a_2, \ldots, a_k) = [e', b_1, b_2, \ldots, b_k, d]$, indicando que cada $a_i$ deve ser substituído por $b_i$, para $i = 1, 2, \ldots, k$. Deve-se notar também que se $a_1 = \langle$, $d \neq E$. No entanto, apesar do símbolo $\langle$ não poder ser manipulado nas posições restantes na primeira trilha, ele pode ser manipulado sem restrições nas outras trilhas!




(a) Trecho de MT com imobilidade.




(b) Trecho de MT padrão I.




(c) Trecho de MT padrão II.

Figura 4.6: Posicionando o cabeçote após uma palavra.

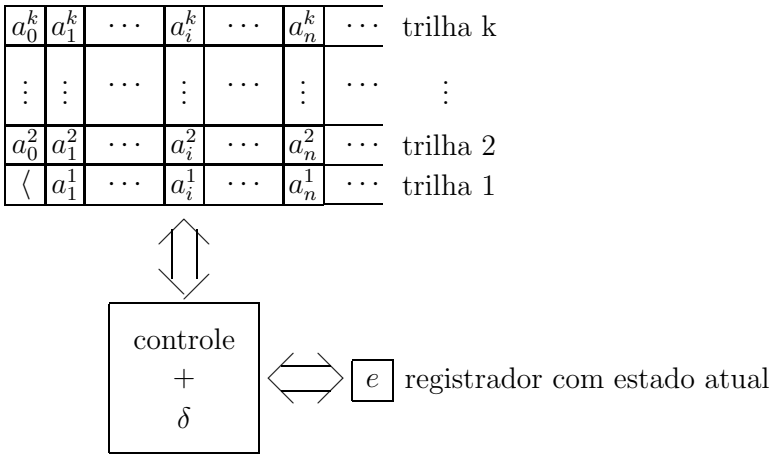



Figura 4.7: Máquina de Turing com múltiplas trilhas.

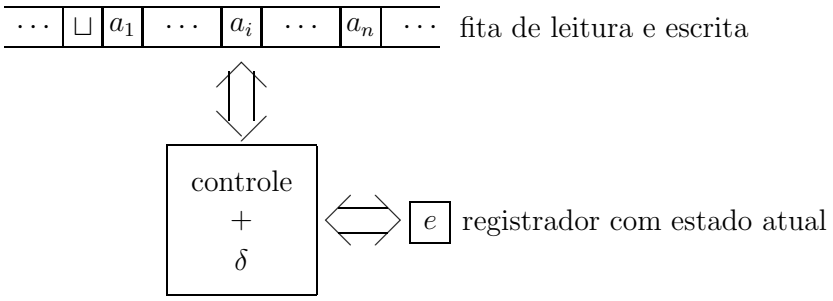



Figura 4.8: Máquina de Turing com fita bidirecional.

Uma configuração instantânea de uma MT com $k$ trilhas tem a forma

$$[e, x_1\underline{a_1}y_1, x_2\underline{a_2}y_2, \ldots, x_k\underline{a_k}y_k]$$

onde $|x_i| = |x_j|$ para $i \neq j$. O conteúdo da trilha $i$ é $x_i a_i y_i$. Com isto, pode-se definir a linguagem aceita por uma MT de $k$ trilhas, $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$, como o conjunto de toda palavra $w \in \Sigma^*$ tal que

$$[i, \langle\underline{w}, \sqcup\underline{\sqcup}, \ldots, \sqcup\underline{\sqcup}] \overset{*}{\vdash} [e, x_1\underline{a_1}y_1, x_2\underline{a_2}y_2, \ldots, x_k\underline{a_k}y_k]$$

onde $e \in F$ e $\delta(e, a_1, a_2, \ldots, a_k)$ é indefinido.

Uma máquina padrão $(E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ pode ser simulada por uma com $k$ trilhas $(E, \Sigma, \Gamma, \langle, \sqcup, \delta', i, F)$, onde se $\delta(e, a) = [e', b, d]$, tem-se que $\delta'(e, a, \sqcup, \ldots, \sqcup) = [e', b, \sqcup, \ldots, \sqcup, d]$. Por outro lado, dada uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ com $k$ trilhas, pode-se obter uma MT padrão equivalente $M' = (E, \Sigma \times \{\sqcup\}^{k-1}, \Gamma^k, \langle, \sqcup, \delta', i, F)$, de tal forma que se $\delta(e, a_1, a_2, \ldots, a_k) = [e', b_1, b_2, \ldots, b_k, d]$, então $M'$ tem a transição $\delta'(e, [a_1, a_2, \ldots, a_k]) = [e', [b_1, b_2, \ldots, b_k], d]$. [3]

Veja então que, como os símbolos de um alfabeto podem ser $k$-uplas, uma máquina com múltiplas trilhas nada mais é do que uma MT padrão "disfarçada", onde o alfabeto de fita, ao invés de ser um conjunto $\Gamma$ constituído de símbolos "indivisíveis", é o produto cartesiano $\Gamma^k$.

### 4.2.3 Máquina com fita ilimitada em ambas as direções

Uma MT com fita ilimitada em ambas as direções difere de uma MT padrão apenas no fato de que a fita não é limitada à esquerda, como mostra a Figura 4.8. No início o cabeçote de leitura/escrita está posicionado no primeiro símbolo da palavra de entrada, se esta não for $\lambda$. Como a fita é ilimitada à esquerda, não há necessidade do símbolo "$\langle$".

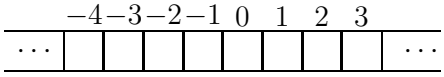
Seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ uma MT padrão, e sejam $i', j \notin E$. Uma MT com fita ilimitada em ambas as direções, equivalente a $M$, seria $M' = (E \cup \{i', j\}, \Sigma, \Gamma, \sqcup, \delta', i', F)$[4], onde $\delta'$ consta das mesmas transições que $M$ acrescidas das seguintes duas:

[3] Para que os símbolos de entrada não sejam alterados, ou seja, para que o alfabeto de entrada de $M'$ seja também $\Sigma$, basta substituir cada símbolo $[a, \sqcup, \ldots, \sqcup]$ de $\Sigma \times \{\sqcup\}^{k-1}$ por $a$ e fazer o alfabeto de fita igual a $(\Gamma^k - \Sigma \times \{\sqcup\}^{k-1}) \cup \Sigma$.

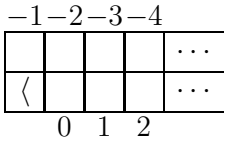
[4] Note que não é necessário o símbolo marcador de início, $\langle$.

- $\delta'(i', a) = [j, a, \mathrm{E}]$ para cada $a \in \Gamma$;
- $\delta'(j, \sqcup) = [i, \langle, \mathrm{D}]$.

Seja $M = (E, \Sigma, \Gamma, \sqcup, \delta, i, F)$ uma máquina com fita ilimitada em ambas as direções. Pode-se obter uma máquina de duas trilhas, $M' = (E', \Sigma, \Gamma', \langle, \sqcup, \delta', i', F')$, que simula $M$, como segue. A idéia é ter na primeira trilha, além do símbolo $\langle \notin \Gamma$, o conteúdo da fita bidirecional que começa na posição inicial do cabeçote e se estende para a direita, e ter na segunda trilha o conteúdo da fita bidirecional que começa na posição anterior à posição inicial do cabeçote e se estende para a esquerda. Assim, considerando a fita bidirecional com as células assim indexadas:




obtém-se a fita de duas trilhas:




O conjunto de estados é $E' = E \times \{1, 2\}$. O primeiro elemento do par $[e, k] \in E'$ é o estado que seria atingido por $M$, e o segundo elemento é a trilha que está sendo processada por $M'$. O estado inicial é $i' = [i, 1]$, e os estados finais são $F' = F \times \{1, 2\}$. O alfabeto de fita é $\Gamma' = \Gamma \cup \{\langle\}$, sendo que $\langle \notin \Gamma$. A função $\delta'$ é obtida de $\delta$ da seguinte forma:

- para cada transição $\delta(e, a) = [e', b, \mathrm{D}]$, deve-se ter:
  - $\delta'([e, 1], a, c) = [[e', 1], b, c, \mathrm{D}]$ para cada $c \in \Gamma$;
  - $\delta'([e, 2], c, a) = [[e', 2], c, b, \mathrm{E}]$ para cada $c \in \Gamma$;
  - $\delta'([e, 1], \langle, a) = \delta'([e, 2], \langle, a) = [[e', 1], \langle, b, \mathrm{D}]$;
- para cada transição $\delta(e, a) = [e', b, \mathrm{E}]$, deve-se ter:
  - $\delta'([e, 1], a, c) = [[e', 1], b, c, \mathrm{E}]$ para cada $c \in \Gamma$;
  - $\delta'([e, 2], c, a) = [[e', 2], c, b, \mathrm{D}]$ para cada $c \in \Gamma$;
  - $\delta'([e, 1], \langle, a) = \delta'([e, 2], \langle, a) = [[e', 2], \langle, b, \mathrm{D}]$.

Conclui-se, então, que uma MT com fita ilimitada à esquerda e à direita tem o mesmo poder de reconhecimento que uma MT padrão.

### 4.2.4 Máquinas com múltiplas fitas

Em uma máquina com múltiplas fitas, cada fita tem seu cabeçote de leitura/escrita, como mostra a Figura 4.9. Como em cada transição os cabeçotes são operados de forma independente, é útil ter a opção de um cabeçote ficar imóvel, além das opções de se mover para a esquerda ou direita. No início, a palavra de entrada está na fita 1 a partir

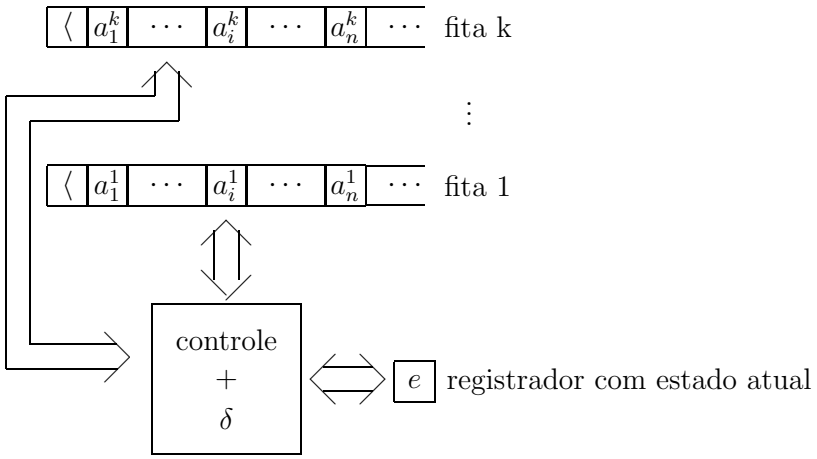



Figura 4.9: Máquina de Turing com múltiplas fitas.

da sua segunda posição, as primeiras posições de todas as fitas contêm $\langle$, e todas as posições após a palavra de entrada na fita 1, assim como todas as outras posições das fitas restantes, contêm $\sqcup$. Todos os cabeçotes começam posicionados na *segunda* posição da fita respectiva. Cada fita tem $\langle$ na sua primeira posição para evitar movimento de seu cabeçote para a esquerda.

Assim, uma máquina de $k$ fitas é uma óctupla $(E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$, onde $E$, $\Sigma$, $\Gamma$, $i$ e $F$ são como em MT's padrão, e $\delta$ é uma função de $E \times \Gamma^k$ para $E \times (\Gamma \times \{\mathrm{D}, \mathrm{E}, \mathrm{I}\})^k$.

Uma configuração instantânea de uma MT com $k$ fitas, de forma similar a uma MT de $k$ trilhas, tem a forma

$$[e, x_1\underline{a_1}y_1, x_2\underline{a_2}y_2, \ldots, x_k\underline{a_k}y_k]$$

onde o conteúdo da fita $i$ é $x_i a_i y_i$. Mas, ao contrário de uma MT de $k$ trilhas, aqui não há a restrição de que $|x_i| = |x_j|$ para $i \neq j$.

A linguagem aceita por uma MT de $k$ fitas, $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$, é o conjunto das palavras $w \in \Sigma^*$ tais que

$$[i, \langle\underline{w}, \langle\underline{\sqcup}, \ldots, \langle\underline{\sqcup}] \overset{*}{\vdash} [e, x_1\underline{a_1}y_1, x_2\underline{a_2}y_2, \ldots, x_k\underline{a_k}y_k]$$

onde $e \in F$ e $\delta(e, a_1, a_2, \ldots, a_k)$ é indefinido.

O uso de várias fitas pode simplificar bastante a obtenção de uma MT, como exemplificado a seguir.

**Exemplo 127** Na Figura 4.10 está exibido o diagrama de estados de uma máquina de duas fitas que reconhece $\{ww^R w \mid w \in \{\mathtt{0}, \mathtt{1}\}^*\}$. Para simplificar o diagrama, está sendo usado o símbolo $*$ para denotar "qualquer símbolo do alfabeto de entrada". Assim, por exemplo, "$*/*$ D, $\sqcup/\sqcup$ I" denota duas transições: "$\mathtt{0}/\mathtt{0}$ D, $\sqcup/\sqcup$ I" e "$\mathtt{1}/\mathtt{1}$ D, $\sqcup/\sqcup$ I".

No ciclo ocasionado pelos estados 0, 1 e 2, para cada três símbolos da palavra de entrada o cabeçote da fita 2 é movido 1 vez para a direita. Em seguida, no estado 3, o terço final da palavra de entrada é escrito na fita 2. Após isto, no estado 4, o terço do

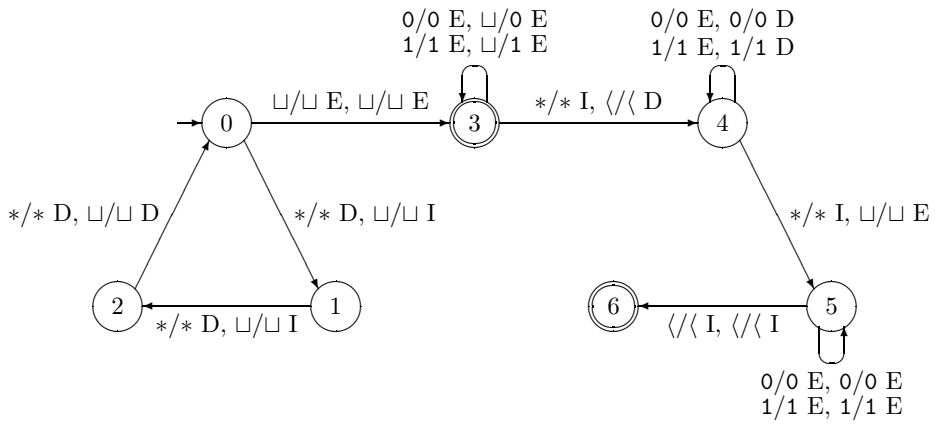



Figura 4.10: Máquina de duas fitas para $\{ww^R w \mid w \in \{0,1\}^*\}$.

meio da palavra de entrada é comparado com o reverso do terço final, que se encontra na fita 2. Finalmente, no estado 5, o primeiro terço da palavra de entrada é comparado com o terço final, que se encontra na fita 2. Tudo isto só envolve uma leitura da palavra de entrada da esquerda para a direita, nos estados 0, 1 e 2, e outra leitura da direita para a esquerda, nos estados 3, 4 e 5. □

Uma MT padrão nada mais é do que uma máquina "multifita" com apenas uma fita. Evidentemente, uma MT padrão pode ser simulada por uma multifita em que todas as transições desprezam todas as fitas, com exceção da fita 1. Por outro lado, uma máquina multifita $M$ pode ser simulada por uma MT padrão. Será esboçado abaixo como simular uma máquina de duas fitas por meio de uma máquina de quatro trilhas. Em geral, uma máquina de $k$ fitas pode ser simulada, de forma análoga, por meio de uma máquina de $2k$ trilhas.

Seja uma máquina de duas fitas $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ e $\mathrm{X} \notin \Gamma$. Será esboçado aqui como se comporta uma MT de quatro trilhas que simule $M$. A trilha 1 terá o conteúdo da fita 1, a trilha 2 terá um `X` para marcar a posição do cabeçote da fita 1 e seu restante estará em branco, e as trilhas 3 e 4 terão o conteúdo da fita 2 e a posição do cabeçote da fita 2. Após $M'$ escrever as representações dos cabeçotes nas trilhas 2 e 4, e escrever $\langle$ no início da trilha 3, transita para o estado $[i, ----\!-\!-]$. Em cada estado da forma $[e, ----\!-\!-]$, onde $e \in E$ (incluindo $[i, ----\!-\!-]$), $M'$ busca, movendo seu cabeçote da esquerda para a direita, os símbolos $a_1$ e $a_2$ das trilhas 1 e 3 cujas posições estão marcadas pelos símbolos `X` das trilhas 2 e 4. Após encontrá-los, $M'$ transita para o estado $[e, a_1a_2 ----]$. Para cada estado desta forma,

- se $\delta(a_1, a_2)$ é indefinido, $M'$ pára no estado $[e, a_1a_2 ----]$ (ou seja, $M'$ não tem transição definida saindo deste estado).

- se $\delta(e, a_1, a_2) = [e', b_1, d_1, b_2, d_2]$, $M'$ busca, movendo seu cabeçote da direita para a esquerda, os símbolos $a_1$ e $a_2$ das trilhas 1 e 3 cujas posições estão marcadas pelos símbolos `X` das trilhas 2 e 4. Substitui $a_1$ por $b_1$ e $a_2$ por $b_2$, e move os símbolos `X` das trilhas 2 e 4 nas direções $d_1$ e $d_2$. Feito isto, $M'$ transita para o estado

$[e, a_1a_2b_1d_1b_2d_2]$. Neste estado, $M'$ volta ao início da fita e transita para o estado $[e', ------]$.

Os estados finais de $M'$ são os estados $[e, a_1a_2 ----]$ para $a_1, a_2 \in \Gamma$ e $e \in F$.

Para concretizar a MT $M'$ esboçada acima, basta acrescentar novos estados da forma $[e, x_1x_2y_1d_1y_2d_2]$, onde $x_1, x_2, y_1, y_2 \in \Gamma \cup \{-\}$ e $d_1, d_2 \in \{\mathtt{D}, \mathtt{E}, \mathtt{I}\}$, na medida em que forem necessários. Por exemplo, se a representação `X` do cabeçote da fita 1 for encontrada antes que o da fita 2 quando $M'$ procura por $a_1$ e $a_2$ da esquerda para a direita, pode-se fazer com que $M'$ transite para $[e, a_1 -----]$. Neste estado, $M'$ procura por $a_2$; ao achá-lo, aí sim, transita para o estado referido no parágrafo anterior, $[e, a_1a_2 ----]$. Por outro lado, se a representação do cabeçote da fita 2 for encontrada antes que o da fita 1, $M'$ transita para $[e, -a_2 ----]$; depois, ao encontrar $a_1$, transita para $[e, a_1a_2 ----]$. E assim por diante.

### 4.2.5 Máquinas não determinísticas

Uma MT não determinística é uma MT que admite mais de uma transição partindo de um certo estado sob um certo símbolo. Assim, podem existir várias computações possíveis para o processamento de uma palavra. Uma palavra é aceita quando *existe* uma computação para a qual a máquina pára em um estado final.

Mais formalmente, uma MT não determinística é uma óctupla $(E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$, onde $E$, $\Sigma$, $\Gamma$, $\langle$, $\sqcup$, $i$ e $F$ são como em MT padrão, e $\delta$ é uma função total de $E \times \Gamma$ para $\mathcal{P}(E \times \Gamma \times \{\mathrm{D}, \mathrm{E}\})$. No caso em que $\delta(e, a) = \emptyset$ para um certo estado $e$ e símbolo $a$, não há transição do estado $e$ sob $a$. Este caso corresponde ao caso em que, em uma MT padrão, $\delta(e, a)$ é indefinido. A linguagem aceita pela máquina é:

$$\{w \in \Sigma^* \mid [i, \langle \underline{w}] \overset{*}{\vdash} [e, x\underline{a}y], \delta(e, a) = \emptyset \text{ e } e \in F\}.$$

**Exemplo 128** A Figura 4.11 apresenta o diagrama de estados de uma MT não determinística que aceita a linguagem $b^*ab^* + c^*ac^*$. O alfabeto de entrada é $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ e o de fita é $\{\mathtt{a}, \mathtt{b}, \mathtt{c}, \sqcup, \langle\}$. A partir do estado $e_1$, por exemplo, existem as seguintes transições:

$\delta(e_1, \mathtt{a}) = \{[e_2, \mathtt{b}, \mathrm{D}], [e_4, \mathtt{c}, \mathrm{D}]\}$ (olha o não determinismo)

$\delta(e_1, \mathtt{b}) = \{[e_1, \mathtt{b}, \mathrm{D}]\}$

$\delta(e_1, \mathtt{c}) = \{[e_1, \mathtt{c}, \mathrm{D}]\}$

$\delta(e_1, \sqcup) = \{\}$

$\delta(e_1, \langle) = \{\}$. □

Será mostrado, a seguir, como simular uma máquina de Turing não determinística $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ por meio de um MT determinística, $M'$, de 3 fitas.

A idéia é simular, de forma sistemática, as computações possíveis da máquina não determinística. Para garantir que *todas* as computações sejam passíveis de simulação, as computações serão simuladas de tal forma que uma computação envolvendo $n + 1$ transições nunca é considerada para simulação antes de todas as que envolvam $n$

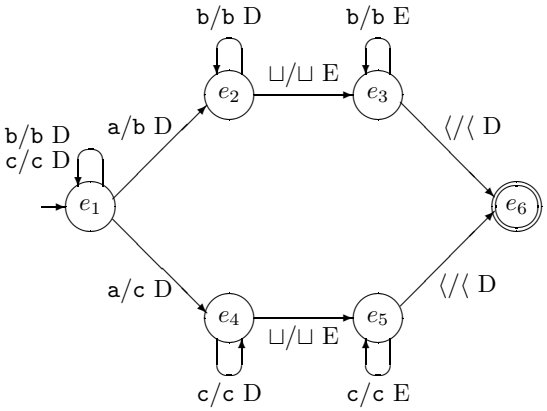



Figura 4.11: Uma Máquina de Turing não determinística.

transições. Com isto, mesmo que haja uma ou mais computações infinitas, a máquina simuladora nunca irá se enveredar indefinidamente por uma delas. Para este propósito, seja, inicialmente, $m$ o número máximo dentre os números de transições sob um mesmo símbolo que emanam de cada estado da máquina não determinística $M$, ou seja, $m$ é o número máximo pertencente ao conjunto $\{|\delta(e,a)| \mid e \in E \text{ e } a \in \Gamma\}$. Por exemplo, para a máquina do Exemplo 128, $m = |\delta(e_1, \mathtt{a})| = 2$. Para cada par $(e,a)$, se $\delta(e,a) \neq \emptyset$:

(a) se o conjunto $\delta(e,a)$ tiver menos de $m$ membros, completá-lo repetindo algum de seus elementos, de forma que ele fique com $m$ elementos[5]; e

(b) numerar as $m$ transições relativas a $\delta(e,a)$ com os números de 1 a $m$ (qualquer ordem serve).

Para a máquina do Exemplo 128, uma possibilidade seria dar o número 1 para a transição $[e_2, \mathtt{b}, \mathrm{D}] \in \delta(e_1, \mathtt{a})$ e 2 para $[e_4, \mathtt{c}, \mathrm{D}] \in \delta(e_1, \mathtt{a})$, e para as outras ter-se-ia pares de transições idênticas com os números 1 e 2. Cada computação de $M$ será representada, na máquina $M'$, por uma palavra no alfabeto $\{1, 2, \ldots, m\}$, que estará escrita na fita 3. Considere a máquina do Exemplo 128, com a numeração já referida, e suponha que a fita 3 contenha a palavra 1221. Se a palavra de entrada for `cac`, então 1221 representa a seguinte computação:

| *Computação* | | | *Transição* | *Número* |
|---|---|---|---|---|
| $[e_1, \langle \underline{\mathtt{c}}\mathtt{ac}\sqcup]$ | $\vdash$ | $[e_1, \langle \mathtt{c}\underline{\mathtt{a}}\mathtt{c}\sqcup]$ | $[e_1, \mathtt{c}, \mathrm{D}] \in \delta(e_1, \mathtt{c})$ | 1 |
| | $\vdash$ | $[e_4, \langle \mathtt{cc}\underline{\mathtt{c}}\sqcup]$ | $[e_4, \mathtt{c}, \mathrm{D}] \in \delta(e_1, \mathtt{a})$ | 2 |
| | $\vdash$ | $[e_4, \langle \mathtt{ccc}\underline{\sqcup}]$ | $[e_4, \mathtt{c}, \mathrm{D}] \in \delta(e_4, \mathtt{c})$ | 2 |
| | $\vdash$ | $[e_5, \langle \mathtt{cc}\underline{\mathtt{c}}\sqcup]$ | $[e_5, \sqcup, \mathrm{E}] \in \delta(e_4, \sqcup)$ | 1 |

Para a palavra de entrada `bab`, 1221 representa a computação:

| *Computação* | | | *Transição* | *Número* |
|---|---|---|---|---|
| $[e_1, \langle \underline{\mathtt{b}}\mathtt{ab}\sqcup]$ | $\vdash$ | $[e_1, \langle \mathtt{b}\underline{\mathtt{a}}\mathtt{b}\sqcup]$ | $[e_1, \mathtt{b}, \mathrm{D}] \in \delta(e_1, \mathtt{b})$ | 1 |
| | $\vdash$ | $[e_4, \langle \mathtt{bc}\underline{\mathtt{b}}\sqcup]$ | $[e_4, \mathtt{c}, \mathrm{D}] \in \delta(e_1, \mathtt{a})$ | 2 |

[5] $\delta(e,a)$ será, neste caso, um *multiconjunto* de $m$ elementos.

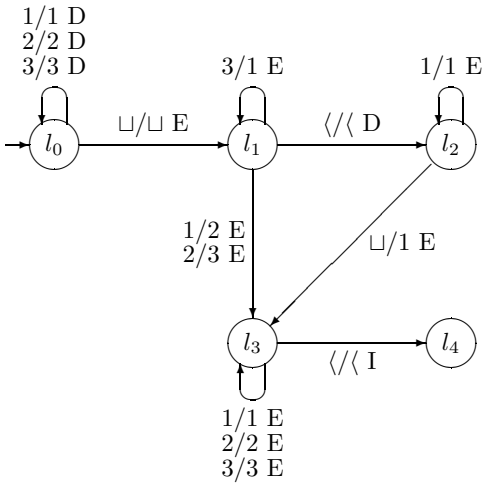



Figura 4.12: Gera próxima palavra de $\{1, 2, 3\}^*$.

Como $\delta(e_4, \mathtt{b}) = \emptyset$, não há como prosseguir: a máquina $M$ pára. Para a palavra `bab`, qualquer seqüência de computações de prefixo 12 é processada apenas até a segunda transição, como acima.

A fita 2 irá conter uma cópia da palavra de entrada no início da simulação de uma computação. Após a simulação de uma computação de $M$ sobre esta cópia, o conteúdo da fita 2 será apagado para, em seguida, ser colocada uma nova cópia da palavra de entrada para a simulação da próxima computação de $M$.

A máquina $M'$ funciona de acordo com o seguinte algoritmo:

1. Inicialize fita 3 com a palavra 1;
2. **ciclo**
   - 2.1 copie a palavra de entrada da fita 1 para a fita 2;
   - 2.2 simule $M$ na fita 2 de acordo com a palavra na fita 3;
     se $M$ parar em estado final, aceite;
   - 2.3 apague a fita 2;
   - 2.4 gere próxima palavra na fita 3

   **fimciclo**.

Os passos 1 e 2.1 são relativamente simples. No passo 2.2, $M'$ pode usar os próprios estados de $M$. Uma transição de $M'$ para simular uma de $M$, $[e', b, d] \in \delta(e, a)$ de número $k$, seria da forma:

$$\delta'(e, \langle, a, k) = [e', [\langle, \mathrm{I}], [b, d], [k, \mathrm{D}]].$$

Além disso, em particular, os estados finais de $M'$ serão os estados finais de $M$: no passo 2.2, dado que $\delta(e, a)$ seja indefinido e $e \in F$ (situação em que $M$ pára em estado final), $M'$ também pára em $e$. Para gerar a próxima palavra na fita 3 (item 2.4 do algoritmo), basta usar a submáquina cujo diagrama de estados está mostrado na Figura 4.12. Lá estão mostradas apenas as transições relativas à fita 3, e está-se supondo que $m = 3$.

Como uma MT padrão é um caso particular de MT não determinística, e para qualquer MT não determinística existe MT padrão equivalente, segue-se que as MT's não determinísticas reconhecem exatamente a classe das LRE's.

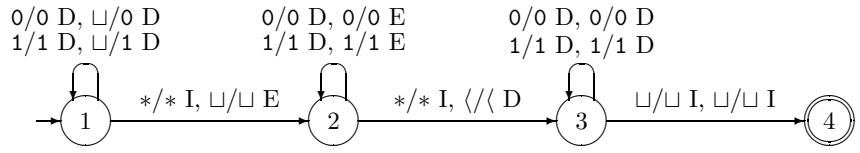



Figura 4.13: MT de duas fitas não determinística para $\{ww^Rw \mid w \in \{0,1\}^*\}$.

Para concluir, pode-se utilizar os incrementos isolados ou uns compostos com os outros, e também pode-se utilizar reconhecimento por entrada em estado final ou por parada com qualquer variação obtida. O exemplo a seguir mostra como uma combinação do uso de duas fitas e de não determinismo simplifica a obtenção de uma MT.

**Exemplo 129** Na Figura 4.13 está o diagrama de estados de uma MT de duas fitas não determinística que reconhece $\{ww^Rw \mid w \in \{0,1\}^*\}$. Tal diagrama deve ser comparado com o da Figura 4.10, página 216, que se refere a uma máquina determinística de duas fitas que reconhece a mesma linguagem. Para simplificar o diagrama, está sendo usado o símbolo "$*$" para denotar "qualquer símbolo do alfabeto de entrada ou $\sqcup$". Observe que o significado de "$*$" aqui é diferente do significado do símbolo "$*$" da Figura 4.13. A opção "$\sqcup$" serve apenas para cobrir o caso em que a palavra de entrada é $\lambda$.

Quando a palavra de entrada é da forma $ww^Rw$ e não é $\lambda$, após copiar $w$ na fita 2 no estado 1, ao ser atingido o símbolo inicial de $w^R$ neste mesmo estado, a máquina "adivinha" que está começando $w^R$ e ativa uma transição para o estado 2. Este é o único ponto em que ocorre não determinismo. Em seguida, no estado 2, o terço do meio da palavra de entrada é assegurado como sendo da forma $w^R$. Depois, no estado 3, o terço final é assegurado como sendo $w$. Apenas se tais condições se verificarem, ou seja, se a palavra de entrada puder ser decomposta nestes três terços, é atingido o estado 4. □

Outras variações, além das apresentadas, também foram propostas na literatura, sendo que para todas elas pode-se verificar que o poder computacional não é maior do que o da MT padrão.

# Exercícios

1. Construa uma MT de 3 trilhas que, recebendo como entrada dois números em notação binária, um na primeira e outro na segunda trilhas, determine a soma na terceira trilha. Faça outra MT que subtraia o número da segunda do número da primeira trilha, colocando o resultado na terceira trilha.

2. Construa uma MT com fita ilimitada em ambas as direções que, começando a fita com duas células contendo o símbolo 0 e com o resto em branco, determine se o número de brancos entre os dois 0's é ímpar. Se for, a MT deve parar em estado de aceitação.

3. Construa uma MT que reconheça a linguagem:

$$\{0^{k_0}10^{k_1}1\ldots0^{k_n}1 \mid n \in \mathbf{N} \text{ e } 0 < k_0 < k_1 < \cdots < k_n\}.$$

   Use uma fita só para assegurar a restrição $0 < k_0 < k_1 < \cdots < k_n$.

4. Seja a MT não determinística $M = (\{a, b, c, d\}, \{0, 1\}, \{\langle, \sqcup, 0, 1\}, \langle, \sqcup, \delta, \{a\}, \{b\})$, onde $\delta$ é assim definida:

$$\delta(a, 0) = \{[b, 0, D], [d, 0, D]\}$$
$$\delta(b, 0) = \{[c, 0, E]\}$$
$$\delta(c, 0) = \{[b, 0, D]\}$$
$$\delta(d, 0) = \{[d, 0, D]\}$$
$$\delta(d, \sqcup) = \{[b, 0, D]\}.$$

   Que linguagem é reconhecida por $M$?

5. Escreva MT's não determinísticas de duas fitas que reconheçam as linguagens:

   (a) $\{xx \mid x \in \{\mathtt{0}, \mathtt{1}\}^*\}$.

   (b) $\{xx^R y \mid x, y \in \{\mathtt{0}, \mathtt{1}\}^* \text{ e } |x| > |y|\}$.

   (c) $\{xyz \mid x, y, z \in \{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*, |x| < |y| < |z|$, $x$ não tem a's, $y$ não tem b's e $z$ não tem c's$\}$.

   Procure obter MT's com o menor número de transições possível.

6. Seja uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, \{f\})$, cuja única diferença com relação a uma MT padrão é que ela tem apenas um estado final. Suponha que a linguagem reconhecida por $M$ é

$$\{w \in \Sigma^* \mid [i, \langle \underline{w}] \overset{*}{\vdash} [f, x\underline{a}y]\}.$$

   Ou seja, para qualquer $w \in \Sigma^*$ $M$ reconhece $w$ se, e somente se, $M$ atinge o estado $f$ ao processar $w$. Mostre que qualquer linguagem recursivamente enumerável pode ser reconhecida por uma MT desse tipo.

7. Seja uma MT que, em cada transição, só pode escrever um símbolo ou mover o cabeçote, mas não ambos. Faça uma definição formal deste tipo de máquina. Depois mostre que tais máquinas reconhecem exatamente as LRE's.

8. Seja um modelo de MT cuja única diferença com relação à MT padrão é que a máquina não pode apagar um símbolo diferente de $\sqcup$. Ou seja, se $\delta(e, a) = [e', \sqcup, d]$, então $a = \sqcup$. Mostre que uma máquina com tal característica tem o mesmo poder de reconhecimento que uma MT padrão.

9. Seja uma máquina em que uma transição depende, não apenas do estado atual e do símbolo sob o cabeçote, mas também do símbolo à direita do cabeçote. Faça uma definição formal de tal tipo de máquina. Mostre como simulá-la mediante uma MT padrão.

10. Seja uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ sendo $\delta$ uma função de $E \times \Gamma$ para $E \times \Gamma \times \{\mathrm{D}, \mathrm{I}\}$. Ou seja, $M$ é uma MT padrão cujo cabeçote pode apenas se mover para a direita (D) ou ficar imóvel (I) em cada transição. $M$ reconhece apenas linguagens regulares, ou existe alguma linguagem não regular que $M$ reconhece?

11. Seja um APN com duas pilhas, como mostrado no Exercício 11 da Seção 3.6, página 196. Mostre que este tipo de máquina reconhece a classe das LRE's. *Sugestão*: para simular uma MT por meio de um APN com duas pilhas, use uma pilha para conter $x^R$ e outra para $ay$, quando a configuração instantânea da MT for $[e, x\underline{a}y]$.

## 4.3 Gramáticas e Máquinas de Turing

Nesta seção, será mostrado que a classe das linguagens geradas pelas gramáticas irrestritas é exatamente a classe das linguagens recursivamente enumeráveis. Além disso, para completar a chamada *hierarquia de Chomsky*, será apresentada uma versão restrita de MT que reconhece exatamente as linguagens geradas por um tipo de gramática denominada *gramática sensível ao contexto*.

Para a consecução da primeira parte, será esboçada a construção de uma MT que aceita uma linguagem gerada por uma gramática irrestrita, e será explicitado como construir uma gramática que gera a linguagem aceita por uma MT.

**Teorema 32** *A linguagem gerada por uma gramática irrestrita é uma LRE.*

*Prova*

Seja uma gramática irrestrita $G = (V, \Sigma, R, P)$. Será mostrado como construir uma MT não determinística de duas fitas, $M$, tal que $L(M) = L(G)$. A fita 1 conterá a palavra de entrada, a qual não será modificada durante todo o processamento, e a fita 2 conterá, em certo instante, uma forma sentencial de $G$.

Segue o algoritmo da máquina $M$. Nos passos 2.1 e 2.2 as seleções lá mencionadas são concretizadas por meio de não determinismo, como será visto à frente.

1. Escreva $P$ (a variável de partida) na fita 2.
2. **ciclo**
   2.1 selecione uma posição $p$ na forma sentencial que está na fita 2;
   2.2 selecione uma regra $u \rightarrow v \in R$;
   2.3 **se** $u$ ocorre a partir da posição $p$ da fita 2 **então**
   2.3.1 substitua $u$ por $v$ na fita 2;
   2.3.2 **se** a forma sentencial na fita 2 é idêntica à palavra
   de entrada na fita 1 **então**
   aceite
   **fimse**
   **senão**
   rejeite
   **fimse**
   **fimciclo**.

Seja $i$ o estado inicial de $M$. Então o passo 1 do algoritmo é levado a efeito pelas transições:

$$\delta(i, a, \sqcup) = \{[j_0, [a, \mathrm{E}], [P, \mathrm{I}]]\} \text{ para cada } a \in \Sigma \cup \{\sqcup\}.$$

No início do ciclo, passo 2.1 do algoritmo, tem-se a seleção não determinística de uma posição $p$ na fita 2 mediante as transições:

$$\delta(j_0, \langle, a) = \{[j_0, [\langle, \mathrm{I}], [a, \mathrm{D}]], [j_1, [\langle, \mathrm{I}], [a, \mathrm{I}]]\} \text{ para cada } a \in \Sigma \cup V.$$

Supondo que $G$ tem $n$ regras, no estado $j_1$ é escolhida, não deterministicamente, uma das regras, da seguinte forma (passo 2.2 do algoritmo):

$$\delta(j_1, \langle, a) = \{[r_1, [\langle, \mathrm{I}], [a, \mathrm{I}]], \ldots, [r_n, [\langle, \mathrm{I}], [a, \mathrm{I}]]\} \text{ para cada } a \in \Sigma \cup V.$$

onde uma transição para o estado $r_t$ corresponde à escolha da $t$-ésima regra. No estado $r_t$, o lado esquerdo da regra $t$ é comparado com o conteúdo da fita 2 a partir da posição atual de seu cabeçote (teste do **se** no passo 2.3). Supondo que o lado esquerdo da regra $t$ é uma palavra $u = a_1 a_2 \ldots a_q$, tal teste seria feito por transições da forma:

$$\delta(r_t, \langle, a_1) = \{[r_t^1, [\langle, \mathrm{I}], [\sqcup, \mathrm{D}]]\}$$

$$\delta(r_t^1, \langle, a_2) = \{[r_t^2, [\langle, \mathrm{I}], [\sqcup, \mathrm{D}]]\}$$

$$\vdots$$

$$\delta(r_t^{q-1}, \langle, a_q) = \{[r_t^q, [\langle, \mathrm{I}], [\sqcup, \mathrm{I}]]\}.$$

Note que, na medida em que é feita a comparação, os símbolos de $u$ vão sendo apagados da fita 2. Se na fita 2 não houver algum destes $a_k$, a máquina pára, correspondendo ao "rejeite" na parte **senão** do comando **se** do passo 2.3. Havendo sucesso na comparação, $u$ terá sido apagada da fita 2 e a máquina entrará no estado $r_t^q$. Neste estado, o lado direito, $v$, da regra $t$ deve ser escrito na fita 2 em substituição a $u$ (passo 2.3.1 do algoritmo). Há três casos a considerar:

(a) $|u| = |v|$. Basta escrever $v$ no espaço em branco onde estava $u$.

(b) $|u| > |v|$. Deve-se escrever $v$ na parte inicial do espaço em branco onde estava $u$, e deslocar a palavra seguinte a este espaço para a esquerda $|u| - |v|$ posições.

(c) $|u| < |v|$. Deve-se deslocar a palavra seguinte ao espaço em branco onde estava $u$ $|v| - |u|$ posições, e escrever $v$ no espaço em branco resultante.

Apesar de não serem explicitadas aqui, transições para cada um destes casos são perfeitamente obteníveis. Suponha que, após a escrita de $v$ na fita 2, a máquina atinja o estado $l_0$, tendo posicionado os cabeçotes na segunda posição de cada fita. Neste estado, de acordo com o passo 2.3.2 do algoritmo, deve-se comparar os conteúdos das fitas 1 e 2, o que pode ser feito mediante as transições:

$$\delta(l_0, a, a) = \{[l_0, [a, \mathrm{D}], [a, \mathrm{D}]]\} \text{ para cada } a \in \Sigma$$

$$\delta(l_0, \sqcup, \sqcup) = \{[f, [\sqcup, \mathrm{I}], [\sqcup, \mathrm{I}]]\}.$$

A última transição ocorre para o estado $f$, único estado final de $M$. Falta considerar o caso em que a comparação levada a efeito no estado $l_0$ não resulta em sucesso. Neste caso, correspondendo à falha do teste do comando **se** em 2.3.2, deve-se reiniciar o ciclo, ou seja, transitar para o estado $j_0$. Para isto, existem as transições:

$$\delta(l_0, a, b) = \{[l_1, [a, \mathrm{E}], [b, \mathrm{E}]]\} \text{ para cada } a \in \Sigma,\ a \neq b$$

$\delta(l_1, a, a) = \{[l_1, [a, \mathrm{E}], [a, \mathrm{E}]]\}$ para cada $a \in \Sigma$

$\delta(l_1, \langle, \langle) = \{[j_0, [\langle, \mathrm{I}], [\langle, \mathrm{D}]]\}$. □

Resta agora mostrar que toda linguagem recursivamente enumerável pode ser gerada por uma gramática irrestrita. Isto será feito no teorema a seguir. Para isto, será conveniente representar uma configuração instantânea $[e, x\underline{a}y]$ pela palavra $xeay\rangle$. Observe que $y$ é seguido por $\rangle$ por motivos que ficarão claros mais à frente. Tal notação não traz confusão, desde que $E \cap \Gamma = \emptyset$, o que será assumido no teorema, sem perda de generalidade.

**Teorema 33** *Uma LRE pode ser gerada por uma gramática irrestrita.*

*Prova*

Seja $L$ uma LRE e seja $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$ uma MT que aceita $L$. Será mostrado como construir, a partir de $M$, uma gramática irrestrita $G = (V, \Sigma, R, P)$ que gera $L$. Existirão regras em $G$ para para três propósitos:

1. gerar todas as formas sentenciais do tipo $w\langle iw\sqcup\rangle$, onde $w \in \Sigma^*$ (os símbolos $i$, $\langle$ e $\rangle$ são *variáveis* em $G$);
2. simular $M$ sobre a configuração instantânea $\langle iw\sqcup\rangle$, deixando o prefixo $w$ inalterado;
3. apagar a configuração instantânea quando ela for do tipo $\langle xeay\rangle$, onde $e \in F$ e $\delta(e, a)$ é indefinido.

Para efeitos de gerar as formas sentenciais do tipo $w\langle iw\sqcup\rangle$ (parte 1), suponha que $\Sigma = \{a_1, a_2, \ldots, a_n\}$. Coloque como novas variáveis em $G$ uma variável para cada $a_i$; sejam $A_1$, $A_2$, $\ldots$, $A_n$ tais variáveis. Coloque também um nova variável $B$. As regras são:

$P \rightarrow B\sqcup\rangle$

$B \rightarrow a_k B A_k$ para $1 \leq k \leq n$ (portanto, $n$ regras)

$B \rightarrow \langle i$

$A_k\sqcup \rightarrow a_k\sqcup$ para $1 \leq k \leq n$ (portanto, $n$ regras)

$A_j a_k \rightarrow a_k A_j$ para $1 \leq k, j \leq n$ (portanto, $n^2$ regras).

A segunda parte, simulação de $M$ sobre a configuração instantânea $\langle iw\sqcup\rangle$, será cumprida pelas regras especificadas a seguir. Note que todos os símbos de $\Gamma$, com exceção dos de $\Sigma$, são variáveis em $G$, assim como os estados de $M$.

para cada transição em $M$ da forma $\delta(e, a) = [e', b, \mathrm{D}]$:

$eac \rightarrow be'c$ para cada $c \in \Gamma - \{\langle\}$
$ea\rangle \rightarrow be'\sqcup\rangle$

para cada transição em $M$ da forma $\delta(e, a) = [e', b, \mathrm{E}]$:

$cea \rightarrow e'cb$ para cada $c \in \Gamma$.

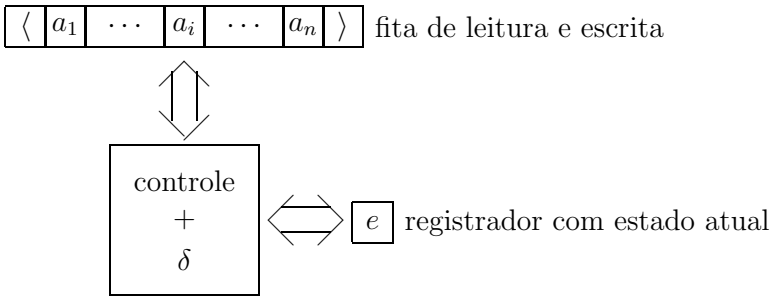



Figura 4.14: Arquitetura de um autômato linearmente limitado.

Para terminar, resta providenciar o apagamento da configuração instantânea quando ela for do tipo $\langle xeay \rangle$, onde $e \in F$ e $\delta(e, a)$ é indefinido (parte 3). Para isto é utilizada uma nova variável #:

para cada par $(e, a)$ tal que $e \in F$ e $\delta(e, a)$ é indefinido:

$ea \rightarrow \#$

$\#c \rightarrow \#$ para cada $c \in \Gamma - \{\langle\}$

$c\# \rightarrow \#$ para cada $c \in \Gamma - \{\langle\}$

$\langle\# \rangle \rightarrow \lambda$ □

No resto desta seção, serão definidas as máquinas reconhecedoras e as gramáticas geradoras das denominadas *linguagens sensíveis ao contexto*.

Na Figura 4.14 está ilustrada a arquitetuta de um *autômato linearmente limitado*. A única diferença com relação a uma MT padrão (além do não determinismo, como será visto), é que a fita é limitada à direita: após a entrada $a_1 a_2 \ldots a_n$, é colocado um símbolo especial, $\rangle$, o qual marca o "final" da fita.[6] Segue uma definição mais precisa.

**Definição 50** *Um* autômato linearmente limitado *(ALL) é uma máquina de Turing não determinística, $M = (E, \Sigma, \Gamma, \langle, \rangle, \sqcup, \delta, i, F)$, onde:*

- $\rangle$ *é um símbolo especial de* $\Gamma$ *que não pode ser escrito na fita;*
- *a configuração inicial é* $[i, \langle \underline{w} \rangle]$*; e*
- *se* $\delta(e, \rangle)$ *é definida, então* $\delta(e, \rangle) = [e', \rangle, E]$ *para algum* $e' \in E$. □

Segue um exemplo.

**Exemplo 130** Está mostrado na Figura 4.15 o diagrama de estados de um ALL que reconhece a linguagem $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n > 0\}$. □

[6] Note que a fita comporta palavra de qualquer tamanho. Mas, após a palavra é colocado o símbolo "$\rangle$", o que coibe o uso de células adicionais.

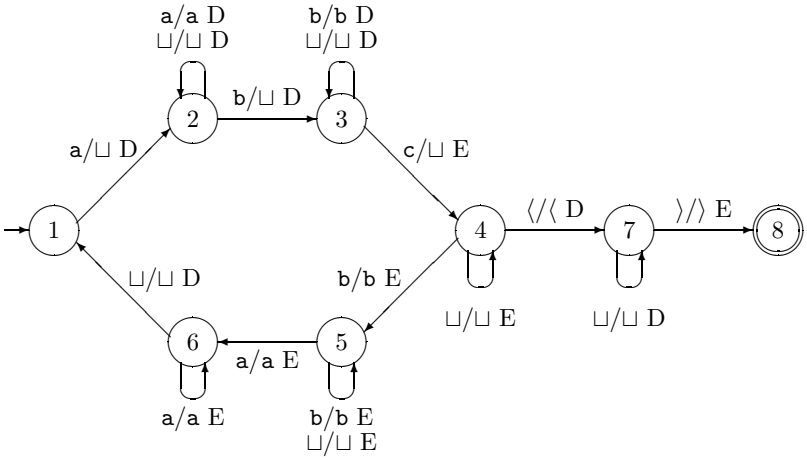



Figura 4.15: Um exemplo de ALL.

Como já foi dito, os ALL's reconhecem as linguagens geradas pelas gramáticas sensíveis ao contexto (GSC's). Segue a definição.

**Definição 51** *Uma* gramática sensível ao contexto *é uma gramática* $(V, \Sigma, R, P)$*, em que cada regra tem a forma* $x \to y$*,* $x, y \in (V \cup \Sigma)^+$ *e* $|x| \leq |y|$*.*[7] □

Assim, pela Definição 51, as formas sentenciais em uma derivação são não decrescentes, ou seja, nunca encolhem. Segue um exemplo.

**Exemplo 131** No Exemplo 49, página 43, foi apresentada uma gramática para a linguagem $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 1\}$. Tal gramática não é sensível ao contexto, visto que o lado direito da regra $A \to \lambda$ é menor do que o lado esquerdo. Uma GSC que reconhece esta mesma linguagem seria:

$$P \ \to \ \mathtt{a}P\mathit{B}\mathtt{c} \mid \mathtt{abc}$$

$$\mathtt{c}\mathit{B} \ \to \ \mathit{B}\mathtt{c}$$

$$\mathtt{b}\mathit{B} \ \to \ \mathtt{bb}$$

□

Segue a definição de linguagem sensível ao contexto (LSC).

**Definição 52** *Uma linguagem é dita ser uma* linguagem sensível ao contexto *se existe uma gramática sensível ao contexto que a gera.* □

Observe que, pela definição acima, uma linguagem que contém $\lambda$ não é uma LSC, já que uma GSC não pode gerar $\lambda$. Para isto, ela teria que permitir, pelo menos uma regra da forma $P \to \lambda$.

Pode-se mostrar que:

[7] O termo "sensível ao contexto" vem do formato das regras em certa forma normal das GSC's, a qual é abordada no Exercício 23 da Seção 4.5, página 234.

(a) Toda LSC é reconhecida por algum ALL.

(b) Se $M$ é um ALL então $L(M) - \{\lambda\}$ é LSC.

Para se mostrar a parte (a), basta explicitar como, dada uma GSC $G$ qualquer, construir um ALL que reconheça $L(G)$. Isto pode ser feito, por exemplo, construindo um ALL com duas trilhas[8] que segue um algoritmo não determinístico similar ao do Teorema 32, página 222. A segunda trilha é usada para armazenar uma forma sentencial, como a segunda fita da MT do Teorema 32. Isto é possível porque uma forma sentencial cujo tamanho ultrapasse o tamanho da palavra de entrada nunca poderá ser usada para gerá-la. A demonstração é proposta como o Exercício 14 da Seção 4.5, página 233.

Para se mostrar a parte (b), basta explicitar como construir uma GSC a partir de um ALL, de forma similar ao que foi feito no Teorema 33, mas garantindo que as regras não tenham lado direito menor que o lado esquerdo. A demonstração é proposta como o Exercício 15 da Seção 4.5, página 233.

Desprezando o caso em que a linguagem contém $\lambda$, a classe das LLC's está *propriamente contida* na classe das LSC's, ou seja, toda LLC é LSC e existe LSC que não é LLC. Para ver isto, basta notar que:

(a) Toda LLC que não contenha $\lambda$ pode ser definida por meio de uma GLC sem regras $\lambda$. E toda GLC sem regras $\lambda$ é uma GSC. Assim, toda LLC sem a palavra $\lambda$ é uma LSC.

(b) Existe *LSC* que não é *LLC*. Por exemplo, existe uma GSC para a linguagem $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 1\}$, como mostrado no Exemplo 131, mas não existe GLC para esta mesma linguagem, conforme pode ser mostrado usando o lema do bombeamento.

Pode-se também mostrar que:

- Toda LSC é recursiva.
- Existe linguagem recursiva que não é LSC.

Diretamente da definição, pode-se concluir que toda linguagem recursiva é LRE. Mais à frente será visto que existem linguagens que são recursivamente enumeráveis, mas não recursivas. Será visto também que existem linguagens que não são LRE's. Assim, o espaço de todas as linguagens sob um alfabeto $\Sigma$ tem a estrutura

$$\text{LReg's} \subset \text{LLC's} \subset \text{LSC's} \subset \text{LRec's} \subset \text{LRE's} \subset \mathcal{P}(\Sigma^*),$$

onde LReg's são as linguagens regulares, LRec's são as recursivas, etc. A Figura 4.16 ilustra a hierarquia em questão.

A classificação das gramáticas nos quatro tipos mostrados, regulares, livres do contexto, sensíveis do contexto e irrestritas[9] é a denominada *hierarquia de Chomsky*.

[8] Na verdade, assim como para MT's em geral, um ALL de várias trilhas é um ALL "normal" que em cada célula comporta uma $n$-upla.

[9] Originalmente, Chomsky classificou-as como gramáticas dos *tipos 0, 1, 2 e 3*, sendo que as do tipo 0 são as irrestritas, as do tipo 1 são as GSC's, etc.

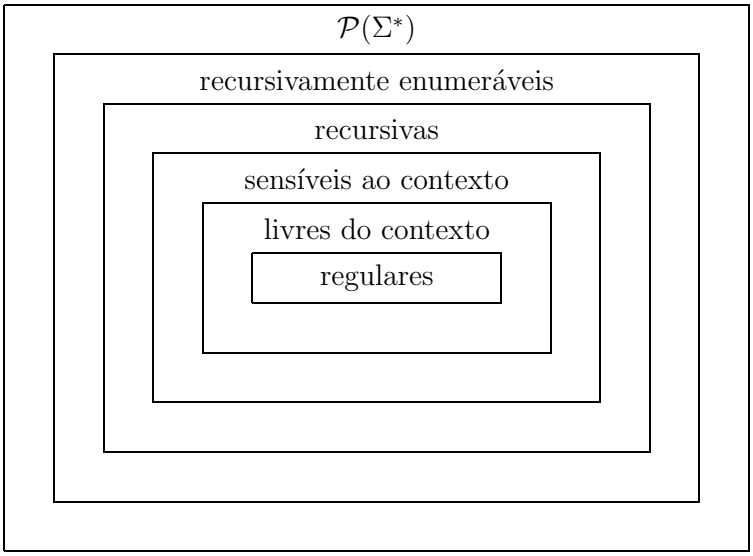



Figura 4.16: Espaço das linguagens em $\mathcal{P}(\Sigma^*)$

# Exercícios

1. Construa gramáticas irrestritas que gerem as linguagens:

   (a) $\{\mathtt{0}^n\mathtt{1}^k\mathtt{0}^n\mathtt{1}^k \mid n, k \geq 0\}$.

   (b) $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^k \mid m < n < k\}$.

   (c) $\{www \mid w \in \{\mathtt{0}, \mathtt{1}\}^*\}$.

2. Mostre que para toda gramática irrestrita existe uma equivalente na qual cada regra tem pelo menos uma variável do lado esquerdo.

3. Mostre que para toda gramática irrestrita existe uma equivalente na qual cada regra tem o lado direito maior ou igual ao lado esquerdo ou é regra $\lambda$.

4. Seja a MT cujo diagrama de estados está mostrado na Figura 4.4(b), página 206. Utilizando o método desenvolvido na prova do Teorema 33, obtenha uma gramática irrestrita para a linguagem reconhecida por tal MT.

5. Construa GSC's para:

   (a) $\{\mathtt{a}^n\mathtt{b}^{n+1}\mathtt{c}^{n+2} \mid n \geq 0\}$.

   (b) $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^k \mid m < n < k\}$.

   Procure obter GSC's com um número mínimo de regras.

6. Construa um ALL que aceite $\{\mathtt{a}^m\mathtt{b}^n\mathtt{c}^k \mid m < n < k\}$.

7. A linguagem $\{ww \mid w \in \{\mathtt{0}, \mathtt{1}\}^+\}$ é uma LSC? Por que?

8. Seja $L$ uma LRE de alfabeto $\Sigma$, e $\#$ um símbolo não pertencente a $\Sigma$. Mostre que existe uma LSC $L'$ de alfabeto $\Sigma \cup \{\#\}$ tal que, para todo $w \in \Sigma^*$:

$$w \in L \text{ se, e somente se, } w\#^k \in L' \text{ para algum } k \geq 0.$$

## 4.4 Propriedades das LRE's e das Linguagens Recursivas

Aqui serão vistas algumas propriedades de fechamento para a classe das linguagens recursivamente enumeráveis e para a classe das linguagens recursivas. Tais propriedades são úteis para mostrar que algumas linguagens são ou não são LRE's ou recursivas.

Várias propriedades de fechamento para LRE's podem ser provadas com relativa facilidade por meio de gramáticas irrestritas. No entanto, elas serão provadas aqui por meio de MT's, com o objetivo de permitir ao leitor praticar mais com relação à diferenciação entre máquinas que param para quaisquer entrada (portanto, reconhecendo linguagem recursiva) e máquinas que, eventualmente, podem não parar (reconhecendo, portanto, LRE não necessariamente recursiva).

**Teorema 34** *A classe das linguagens recursivas é fechada sob união, interseção e complementação.*

*Prova*

Sejam duas MT's $M_1 = (E_1, \Sigma, \Gamma_1, \langle, \sqcup, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \Gamma_2, \langle, \sqcup, \delta_2, i_2, F_2)$ que reconheçam linguagens recursivas. Pode-se construir máquinas que reconhecem $L(M_1) \cup L(M_2)$ e $L(M_1) \cap L(M_2)$ utilizando técnica análoga àquela utilizada na Seção 2.2.3 para construir AFD's para união e interseção, qual seja, a de simular as máquinas $M_1$ e $M_2$ em paralelo. Como lá, a única diferença entre a MT para $L(M_1) \cup L(M_2)$ e a MT para $L(M_1) \cap L(M_2)$ será o conjunto dos estados finais. As máquinas serão da forma $(E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$, onde

- $E = (E_1 \times E_2) \cup \{i, j\}$;
- $\Gamma = \Gamma_1 \cup \Gamma_2$;

Para facilitar, a máquina resultante terá duas fitas; $M_1$ será simulada na primeira e $M_2$ na segunda. Inicialmente, a entrada na fita 1 é copiada na fita 2 por meio das transições:

$\delta(i, a, \sqcup) = [i, [a, \mathrm{D}], [a, \mathrm{D}]]$ para todo $a \in \Sigma$

$\delta(i, \sqcup, \sqcup) = [j, [\sqcup, \mathrm{E}], [\sqcup, \mathrm{E}]]$

$\delta(j, a, a) = [j, [a, \mathrm{E}], [a, \mathrm{E}]]$ para todo $a \in \Sigma$

$\delta(j, \langle, \langle) = [[i_1, i_2], [\langle, \mathrm{D}], [\langle, \mathrm{D}]]$.

Após a cópia, a máquina começa a operar no estado $[i_1, i_2]$. As funções $\delta_1$ e $\delta_2$ contribuem para a definição de $\delta$ da forma a seguir. Para todo $[e_1, e_2] \in E_1 \times E_2$ e todo $[a_1, a_2] \in \Gamma_1 \times \Gamma_2$:

(a) se $\delta_1(e_1, a_1) = [e'_1, b_1, d_1]$ e $\delta_2(e_2, a_2) = [e'_2, b_2, d_2]$, então $\delta([e_1, e_2], a_1, a_2) = [[e'_1, e'_2], [b_1, d_1], [b_2, d_2]]$;

(b) se $\delta_1(e_1, a_1)$ é indefinido e $\delta_2(e_2, a_2)$ é indefinido, $\delta([e_1, e_2], a_1, a_2)$ é indefinido;

(c) se $\delta_1(e_1, a_1) = [e'_1, b_1, d_1]$ e $\delta_2(e_2, a_2)$ é indefinido, então $\delta([e_1, e_2], a_1, a_2) = [[e'_1, e_2], [b_1, d_1], [a_2, \mathrm{I}]]$; e

(d) se $\delta_1(e_1, a_1)$ é indefinido e $\delta_2(e_2, a_2) = [e'_2, b_2, d_2]$, então $\delta([e_1, e_2], a_1, a_2) = [[e_1, e'_2], [a_1, \mathrm{I}], [b_2, d_2]]$.

Pode-se observar que, para qualquer palavra de entrada, a MT resultante pára em um estado $[e_1, e_2]$ se, e somente se, $M_1$ pára no estado $e_1$ e $M_2$ pára no estado $e_2$. Para a MT que reconhece $L(M_1) \cap L(M_2)$, faz-se $F = F_1 \times F_2$. E para a MT que reconhece $L(M_1) \cup L(M_2)$, faz-se $F = (F_1 \times E_2) \cup (E_1 \times F_2)$.

Para finalizar, uma MT que reconhece $\overline{L(M_1)}$ seria $(E_1, \Sigma, \Gamma_1, \langle, \sqcup, \delta_1, i_1, E_1 - F_1)$. □

A seguir, prova-se o fechamento da classe das LRE's com relação às operações de união e interseção.

**Teorema 35** *A classe das LRE's é fechada sob união e interseção.*

*Prova*

Pode-se construir MT's que reconhecem $L(M_1) \cup L(M_2)$ e $L(M_1) \cap L(M_2)$, a partir de MT's quaisquer $M_1 = (E_1, \Sigma, \Gamma_1, \langle, \sqcup, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \Gamma_2, \langle, \sqcup, \delta_2, i_2, F_2)$, de forma análoga à usada na demonstração do Teorema 34. Na verdade, a prova lá apresentada vale, sem modificações, para o caso do reconhecimento de $L(M_1) \cap L(M_2)$. No entanto, a prova lá apresentada deve receber as seguintes modificações para o caso do reconhecimento de $L(M_1) \cup L(M_2)$:

- O item (c) se torna: se $\delta_1(e_1, a_1) = [e'_1, b_1, d_1]$ e $\delta_2(e_2, a_2)$ é indefinido, então:

  (c.1) se $e_2 \in F_2$, então $\delta([e_1, e_2], a_1, a_2)$ é indefinido;

  (c.2) se $e_2 \notin F_2$ $\delta([e_1, e_2], a_1, a_2) = [[e'_1, e_2], [b_1, d_1], [a_2, \mathrm{I}]]$.

- O item (d) tem modificações análogas.

A parte (c.1) garante a parada e aceitação de $M$ no caso em que $M_2$ aceite, mesmo que $M_1$ entre em *loop*. □

As linguagens recursivas e as LRE's são fechadas sob muitas outras operações, mas o resultado mais importante com relação a fechamento, neste contexto, é o *não* fechamento das LRE's sob complementação. Primeiramente, deve-se observar que existem linguagens que não são LRE's. Uma intuição deste fato, já mencionada anteriormente, segue de:

1. MT's podem ser representadas por meio de palavras de uma linguagem sob um certo alfabeto (como será visto no próximo capítulo). Assim, seja $R$ uma linguagem constituída das palavras sob um certo alfabeto $\Sigma$ que representam MT's.

2. Como $\Sigma^*$ é um conjunto enumerável e $R \subseteq \Sigma^*$, $R$ é enumerável. Ou seja, o conjunto das MT's é enumerável, independentemente da linguagem usada para representá-las.

3. O conjunto de todas as linguagens de alfabeto $\Sigma$, $\mathcal{P}(\Sigma^*)$, não é enumerável.

4. Como o conjunto das MT's é enumerável e o conjunto das linguagens não é, segue-se que não há como associar cada linguagem a uma MT (não existe uma função injetiva de $\mathcal{P}(\Sigma^*)$ para $R$). Assim, existem linguagens para as quais não existem MT's.

De qualquer forma, no próximo capítulo será mostrada uma linguagem que não é LRE. E tal linguagem será o complemento de uma LRE.

A técnica utilizada nos Teoremas 34 e 35 pode ser adaptada para provar que, sendo $L$ e $\overline{L}$ LRE's, $L$ (e portanto $\overline{L}$) é recursiva.

**Teorema 36** *Se $L$ e $\overline{L}$ são LRE's, então $L$ é recursiva.*

*Prova*

Sejam $M_1 = (E_1, \Sigma, \Gamma_1, \langle, \sqcup, \delta_1, i_1, F_1)$ e $M_2 = (E_2, \Sigma, \Gamma_2, \langle, \sqcup, \delta_2, i_2, F_2)$ MT's para $L$ e $\overline{L}$, respectivamente. Pode-se construir uma MT de duas fitas que reconhece $L$ e que sempre pare de forma análoga à usada na demonstração do Teorema 35 para o caso $L(M_1) \cup L(M_2)$. A única diferença é que o conjunto de estados finais fica sendo $F_1 \times E_2$. (Para reconhecer $\overline{L}$, tal conjunto seria $E_1 \times F_2$.)

O fato de que $M_1$ pára em estado de $F_1$ se a palavra de entrada $w$ pertence a $L$, garante que a MT resultante pára em estado de $F_1 \times E_2$ se acionada com $w$. E o fato de que $M_2$ pára em estado de $F_2$ se a palavra de entrada $w$ não pertence a $L$, garante que a MT resultante pára em estado de $(E_1 - F_1) \times F_2$ se acionada com $w$. □

# Exercícios

1. Construa uma MT que reconheça $L = \{\mathtt{ab}\}\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}^*$. A partir da MT construída e da MT cujo diagrama de estados está mostrado na Figura 4.4(b), página 206, a qual reconhece $\overline{L}$, use o método da prova do Teorema 36 para construir uma MT para $\overline{L}$.

2. Mostre que toda LSC é recursiva, assim como seu complemento.

3. Seja $L$ uma linguagem *não* recursiva. Mostre que:

   (a) $\overline{L}$ não é recursiva.

   (b) Se $L$ é LRE, então $\overline{L}$ não é LRE.

4. Sejam $L$ uma LRE e $R$ uma linguagem recursiva. Mostre:

   (a) $L - R$ é uma LRE.

   (b) $L - R$ pode não ser recursiva.

   (c) $R - L$ pode não ser uma LRE.

5. Mostre que as LRE's são fechadas sob concatenação e sob fecho de *Kleene* por meio de MT's. *Sugestão*: use *não* determinismo.

## 4.5 Exercícios

1. Construa MT's, utilizando recursos (trilhas, fitas, não determinismo, etc.) que simplifiquem a construção, para reconhecer as linguagens:

   (a) $\{w \in \{\mathtt{a}, \mathtt{b}\}^* \mid w \neq w^R\}$.

   (b) $\{\mathtt{a}^m\mathtt{b}\mathtt{a}^n \mid m - n$ é divisível por $k\}$, onde $k$ é uma constante.

   (c) $\{\mathtt{a}^{2^n} \mid n \geq 0\}$.

   (d) $\{\mathtt{a}^{n^2} \mid n \geq 0\}$.

   (e) $\{\mathtt{a}^n \mid n$ é primo$\}$.

   (f) $\{\mathtt{a}^{n!} \mid n \geq 0\}$.

2. Construa uma MT com fita ilimitada à esquerda e à direita, com alfabeto de fita $\{\mathtt{0}, \mathtt{1}, \sqcup, \langle\}$, que reconheça a linguagem $\{xx \mid x \in \{\mathtt{0}, \mathtt{1}\}^*\}$.

3. Múltiplas trilhas podem ser usadas em ocasiões em que se deseja um reconhecimento *não destrutivo*. Por exemplo, pode-se usar uma trilha adicional de tal forma que a primeira trilha, que contém a palavra de entrada, nunca é modificada. Construa uma MT de duas trilhas para reconhecer a linguagem $\{ww^R \mid w \in \{\mathtt{0}, \mathtt{1}\}^*\}$, que nunca modifique a primeira trilha.

4. Mostre como construir uma MT com duas fitas que simule um APND. Simule a pilha na fita 2. Exemplifique com o APD para $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 0\}$, cujo diagrama de estados está mostrado na Figura 3.4, página 143.

5. Construa uma MT padrão que gere todos os números naturais em notação binária. A MT não deve parar nunca. Ela deve gerar `0`, depois `1`, depois `10`, etc., separados por branco, indefinidamente. A fita deverá ficar assim:

   $$\langle \mathtt{0} \sqcup \mathtt{1} \sqcup \mathtt{10} \sqcup \mathtt{11} \sqcup \mathtt{100} \sqcup \mathtt{101} \sqcup \mathtt{110} \sqcup \cdots$$

6. Construa uma MT com alfabeto de entrada $\{\mathtt{0}, \mathtt{1}\}$, que interprete a palavra de entrada como a representação binária de um número natural e determine a representação unária de tal número. Pode usar qualquer modelo de MT; em particular, pode usar uma fita só para conter o resultado.

7. Construa uma MT com fita ilimitada em ambas as direções que, começando a fita com duas células contendo o símbolo `1` e o resto branco, determine se o número de brancos entre os dois `1`'s é ímpar, e, neste caso, pare em estado de aceitação. Observe que os dois símbolos referidos pode estar em *qualquer lugar* da fita.

8. Completar o esboço apresentado na Seção 4.2.4 para obter uma MT de 4 trilhas equivalente a uma MT de 2 fitas.

9. Escreva MT's não determinísticas que reconheçam as linguagens:

   (a) $\{xww^Ry \mid x, y, w \in \{\mathtt{0}, \mathtt{1}\}^+$ e $|x| > |w| > |y|\}$.

(b) $\{\mathtt{0}^n \mid n$ não é primo$\}$.

10. Completar o esboço apresentado na Seção 4.2.5 para obter uma MT determinística de 3 fitas equivalente a uma MT não determinística.

11. Uma MT multicabeçote é uma MT padrão com vários cabeçotes independentes, mas uma única fita. Faça uma definição formal de tal tipo de máquina. Mostre como simulá-la mediante uma MT padrão.

12. Uma MT bidimensional é uma MT cuja fita tem duas dimensões, ou seja, é uma matriz bidimensional que se estende indefinidamente em duas direções. Com isto, existem mais duas possibilidades de movimentação para o cabeçote, além de *direita*, *esquerda* e *imóvel*: *para cima* e *para baixo*. Faça uma definição formal de MT bidimensional. Em seguida, mostre como simular uma MT bidimensional por meio de algum outro tipo de MT que você conhece.

13. Construa um ALL que aceite $\{\mathtt{a}^{n!} \mid n \geq 0\}$. *Sugestão*: Divida a palavra de entrada por $2, 3, 4, \ldots, n$. Use duas trilhas, uma das quais para conter os divisores $2, 3, 4, \ldots$

14. Mostre que toda LSC pode ser reconhecida por um ALL.

15. Mostre que $L(M) - \{\lambda\}$ é LSC, caso $M$ seja um ALL.

16. Mostre que as LSC's são fechadas ou não sob:

    (a) União.

    (b) Concatenção.

    (c) Fecho de Kleene.

17. Seja $U$ uma linguagem, e seja $C_U : \mathcal{P}(U) \rightarrow \mathcal{P}(U)$ a operação de *complemento relativo a* $U$, tal que $C_U(L) = U - L$. Para cada classe de linguagens, regular, LLC, recursiva e LRE, mostre se a mesma é fechada ou não sob complemento relativo a $U$, para cada um dos seguintes casos:

    (a) $U$ é regular;

    (b) $U$ é LLC;

    (c) $U$ é recursiva;

    (d) $U$ é LRE;

    (e) $U$ não é LRE.

18. Seja $\{L_1, L_2, \ldots, L_n\}$, $n \geq 2$, uma partição de $\Sigma^*$, onde $\Sigma$ é um alfabeto. Mostre que se cada $L_i$ é uma LRE, então cada $L_i$ é uma linguagem recursiva.

19. Seja $L$ uma LRE não recursiva e $M$ uma MT que reconhece $L$. Seja

$$S = \{w \in \overline{L} \mid M \text{ entra em } \textit{loop} \text{ se a entrada é } w\}.$$

    Mostre que $S$ é infinito.

20. Sejam $L_1, L_2, \ldots$ LRE's. Mostre que $\cup_{i \geq 1} L_i$ é LRE ou que pode não ser.

21. Mostre que as LRE's são fechadas sob concatenação e fecho de *Kleene* por meio de gramáticas. (Cuidado! As técnicas simples usadas para GLC's não funcionam aqui sem adaptações!)

22. Seja $L$ uma LRE que não seja recursiva. Mostre que para qualquer MT que reconheça $L$, o conjunto das palavras de entrada para as quais $M$ não pára é infinito.

23. Prove que toda LSC pode ser gerada por uma gramática na qual cada regra tem a forma $xAy \rightarrow xwy$, onde $w \in (V \cup \Sigma)^+$ e $x, y \in (V \cup \Sigma)^*$. *Sugestão*: primeiramente, mostre que para toda GSC existe uma GSC equivalente em que cada regra tem um lado direito com, no máximo, dois símbolos. Em seguida, mostre como obter o mesmo efeito de uma regra da forma $AB \rightarrow CD$ apenas com regras da forma $xAy \rightarrow xwy$.

## 4.6 Notas Bibliográficas

As máquinas de Turing foram concebidas por Turing[Tur36]. Chomsky[Cho56] demonstrou a equivalência destas com gramáticas irrestritas.

Kleene[Kle43] e Post[Pos43][Pos44] apresentaram várias propriedades das linguagens recursivamente enumeráveis.

Os autômatos linearmente limitados foram definidos por Myhill[Myh60]. Em seguida, Landweber[Lan63] e Kuroda[Kur64] abordaram a relação entre autômatos linearmente limitados e gramáticas livres do contexto.

# Capítulo 5

# Decidibilidade

> The idea that there won't be an algorithm to solve it — this is something fundamental that won't ever change — that idea appeals to me.
>
> Stephen Cook *apud* D. Shasha & C. Lazere[SL95]

Como já foi dito nos capítulos anteriores, existem vários problemas indecidíveis, alguns deles de formulação bastante simples. Neste capítulo serão apresentados alguns destes problemas.

Este capítulo será iniciado pela apresentação da denominada tese de Church-Turing, que afirma que a máquina de Turing é um formalismo que captura a noção de computação efetiva. Em seguida, será visto como as instâncias de um problema de decisão podem ser codificadas (representadas) de forma que possam ser alimentadas como entradas para uma MT. Em seguida, na Seção 5.3, será visto que quaisquer MT's podem ser codificadas e apresentadas como entrada para uma MT denominada "máquina de Turing universal", uma MT que simula qualquer MT fornecida como entrada. O primeiro problema de decisão indecidível será analisado na Seção 5.4: o problema da parada para MT's, onde será apresentada também uma linguagem que não é LRE, assim como uma liguagem que é LRE mas não é recursiva. Na Seção 5.5 será apresentada a técnica de redução de problemas, de forma que uma série de problemas indecidíveis sejam determinados como tais, por meio desta técnica, na Seção 5.6.

## 5.1 A Tese de Church-Turing

No capítulo anterior, as MT's foram introduzidas como máquinas para reconhecimento de linguagens. Mas, como ficou lá ressaltado, as MT's podem ser usadas para implementação de funções em geral. Neste caso, a MT recebe como entrada os argumentos e retorna a saída na própria fita. A seguir, são esboçadas algumas considerações a respeito do poder computacional das MT's com relação à computação de funções em geral. No restante do capítulo, cujo propósito maior é a introdução ao mundo dos problemas insolúveis, serão consideradas apenas funções que retornam *sim* ou *não*, as quais podem ser "programadas" por meio de MT's reconhecedoras de linguagens.

Mesmo antes do aparecimento dos primeiros computadores, os matemáticos já se preocupavam com a noção de *computação efetiva*. Trabalhando a partir de uma noção

imprecisa (informal), onde se especificava uma lista de atributos desejáveis para tal conceito, como a possibilidade de execução mecânica, produção da mesma saída para as mesmas entradas, execução em tempo finito, etc., vários formalismos foram propostos no intuito de capturar, de forma precisa (formal), o conceito de computação efetiva. A partir de uma caracterização formal, seria possível, então, mostrar que certos problemas seriam *computáveis*, ou seja, existiriam *algoritmos* para eles, e que outros não seriam computáveis.

É interessante observar que, já naquela época (final da década de 1930), vários formalismos foram propostos que vieram a revelar a mesma expressividade, apesar das suas diferentes "aparências". Alguns destes formalismos são:

- máquinas de Turing;
- sistemas de Post;
- funções $\mu$-recursivas;
- $\lambda$-cálculo.

Os computadores digitais, tanto os mais antigos quanto os mais modernos, foram construídos com um conjunto de instruções que lhes dão um poder computacional idêntico ao das máquinas de Turing e ao dos outros formalismos acima citados.[1] Mais modernamente, linguagens de programação de alto nível, como Java, C, Pascal, etc., com o mesmo poder expressivo, apareceram no intuito de facilitar a tarefa de programação propriamente dita. Mas, não se deve perder de vista que tais linguagens não apresentam maior expressividade do que, por exemplo, as MT's, embora sejam, obviamente, muito mais adequadas do ponto de vista prático para a confecção de algoritmos.

Dentre os diversos formalismos matemáticos propostos, a máquina de Turing é um dos mais aderentes aos computadores atuais, isto é, pode-se considerar que a máquina de Turing captura a parte "essencial", aquela responsável, em última análise, pelo poder computacional dos computadores atuais. Já de um ponto de vista mais geral, como os formalismos e linguagens citados acima são equivalentes do ponto de vista de expressividade, poder-se-ia dizer que a noção de "computação" seria o que existe de comum entre todos eles, ou, por outro lado, cada um deles apresenta uma abordagem diferente para o conceito de computabilidade.

A tese de Church-Turing pode ser assim enunciada:

> *Se uma função é efetivamente computável então ela é computável por meio de uma máquina de Turing.*

Ou, equivalentemente: todo algoritmo pode ser expresso mediante uma MT.

A noção de *computação efetiva* não é definida formalmente. Assim não há como provar que a tese de Church-Turing é correta. No entanto, a proliferação de formalismos nunca mais expressivos que o da máquina de Turing, constitui evidência em seu favor. Dada a equivalência dos vários formalismos e linguagens, a tese de Church-Turing, mesmo

[1] Estritamente falando, o poder computacional associado aos formalismos matemáticos citados é maior do que o de qualquer computador real, devido ao fato de que um computador real tem uma memória limitada.

que implicitamente, equipara a classe das funções computáveis à classe de funções que podem ser expressas em qualquer um deles. Assim, por exemplo, a tese implica que "se uma função é efetivamente computável então ela é computável por meio de um programa escrito na linguagem C".

A tese de Church-Turing, quando particularizada para problemas de decisão, poderia ser assim enunciada: se um problema de decisão tem solução, então existe uma MT que o soluciona. Assim, para mostrar que um problema de decisão não tem solução, basta mostrar que não existe MT que o soluciona. Por outro lado, dada a equivalência dos diversos formalismos, um problema de decisão não tem solução se não for possível expressar uma solução em qualquer um dos mesmos. Assim, por exemplo, se não existir um programa em C que seja uma solução para um problema de decisão, então tal problema de decisão é insolúvel.

Uma característica importante de qualquer um dos formalismos citados é o que se denomina *auto-referência*. Por exemplo, é possível ter MT's que manipulam outras MT's: para isto, basta codificar (ou representar) uma MT usando-se um alfabeto apropriado e supri-la como entrada para outra MT. No caso de uma linguagem de programação, é possível ter programas que manipulam programas: um programa em Java pode receber como entrada um programa escrito em Java e manipulá-lo como *dado*. Na Seção 5.2 será visto como isto pode ser feito para MT's.

A característica mencionada no parágrafo anterior propicia a possibilidade de construir uma *máquina universal*, ou seja, uma MT (ou programa em uma linguagem de programação) que seja capaz de simular uma MT (programa) qualquer suprida como argumento. Na Seção 5.3 será apresenta uma máquina desse tipo: uma *máquina de Turing universal*.

A possibilidade de auto-referência é uma característica fundamental que levou à descoberta de funções não computáveis. Um exemplo disso será visto na Seção 5.4: não existe MT que determine se uma MT arbitrária irá parar ou não para uma certa entrada. Ou ainda: não existe programa em C que determine se um programa em C irá parar ou não para uma certa entrada. Na verdade, vários outros problemas similares, que envolvem o processamento de MT's por MT's são insolúveis, como será mostrado nos capítulos vindouros.

## 5.2 Máquinas de Turing e Problemas de Decisão

Como já foi dito na Seção 1.12, a solução de um problema de decisão $P$ é um algoritmo que dá a resposta correta para cada instância $p \in P$. Dada a tese de Church-Turing, a solução de $P$ pode ser expressa por meio de uma máquina de Turing que, para cada instância $p \in P$, se a resposta para $p$ for "sim", pára em um estado final, e se a resposta for "não", pára em um estado não final. Desta forma, a máquina de Turing deve reconhecer a linguagem *recursiva* que consta de todas as instâncias $p$ para as quais a resposta é "sim".

Implícito no parágrafo anterior está o fato de que uma máquina que solucione um problema $P$ deve receber como entrada uma palavra que *represente* uma instância de $P$. Assim, o primeiro passo para solucionar um problema de decisão $P$ é projetar uma *representação* para as instâncias de $P$ utilizando-se um certo alfabeto $\Sigma$. Quando o PD

consta de uma única instância, na verdade ela não precisa ser representada, ou, por outro lado, qualquer palavra serve para representá-la ($\lambda$, por exemplo). E mais, se o PD consta de um número finito $n$ de instâncias, nenhuma delas precisa ser representada, ou, por outro lado, quaisquer $n$ palavras servem para representar as $n$ instâncias. O problema da representação se torna importante quando o PD tem uma infinidade de instâncias[2].

Cada instância $p$ de um PD $P$ pode ser identificada com uma seqüência $v_1, v_2, \ldots, v_k$ de valores específicos para os $k$ parâmetros de $P$ (e vice-versa). Assim, a representação de $p$ pode ser feita codificando-se a seqüência $v_1, v_2, \ldots, v_k$, o que é feito associando-se uma palavra de $\Sigma^*$ a tal seqüência. Tal associação deve satisfazer os seguintes requisitos:

(1) Para cada instância de $P$ deve existir pelo menos uma palavra de $\Sigma^*$ que a represente.

(2) Cada palavra de $\Sigma^*$ deve representar no máximo uma instância de $P$.

(3) Para cada palavra $w \in \Sigma^*$, deve ser possível determinar se ela representa ou não alguma instância de $P$. Ou seja, o problema de determinar se $w$ representa uma instância deve ser decidível.

**Exemplo 132** Seja o PD "determinar se um número natural $n$ é primo". Seguem duas representações das instâncias deste problema:

(a) Usando-se o alfabeto $\{1\}$, pode-se representar cada instância "determinar se $j$ é primo", onde $j$ é um número natural específico, pela palavra $1^j$. Com isto:

(a.1) Para cada instância, "determinar se $j$ é primo", existe uma palavra de $\{1\}^*$ que a representa: $1^j$.

(a.2) Para cada palavra $w \in \{1\}^*$, é possível determinar se ela representa ou não alguma instância: toda palavra representa uma instância.

(a.3) Cada palavra $1^j \in \Sigma^*$ representa uma instância: a instância "determinar se $j$ é primo".

(b) Usando-se o alfabeto $\{0, 1\}$, pode-se representar cada instância na notação binária convencional, com zeros à esquerda permitidos. Com isto:

(b.1) Cada instância é representada por inúmeras palavras. Por exemplo, as palavras $0^i$, para $i \geq 1$, representam "determinar se 0 é primo"; as palavras $0^i1$, para $i \geq 0$, representam "determinar se 1 é primo"; e assim por diante.

(b.2) Para cada palavra $w \in \{0, 1\}^*$, é possível determinar se ela representa ou não alguma instância: se $w = \lambda$, não representa instância alguma; caso contrário, representa.

(b.3) Cada palavra de $w \in \Sigma^*$ representa no máximo uma instância: se $w = \lambda$, não representa instância alguma; caso contrário, representa a instância "determinar se $\eta(w)$ é primo", onde $\eta(w)$ é o número representado em binário por $w$. Por exemplo, 00110 representa "determinar se 6 é primo", e apenas esta instância.

□

[2]Na verdade, em termos práticos, se um PD tem mais de uma instância, particularmente quando tem muitas instâncias, pode ser conveniente representar suas instâncias explicitamente.

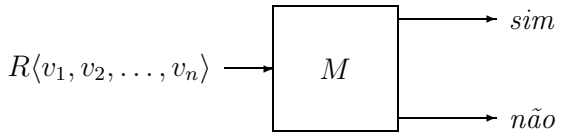



Figura 5.1: Máquina de Turing para um problema de decisão.

A notação $R\langle v_1, v_2, \ldots, v_n\rangle$ será utilizada para designar qualquer palavra $w$ que represente a instância $p$ correspondente à seqüência de valores de parâmetros $v_1$, $v_2$,...,$v_n$. Uma máquina de Turing $M$ que soluciona um PD que receba como entrada $v_1, v_2, \ldots, v_n$, será representada esquematicamente como mostrado na Figura 5.1. Lá mostra-se que uma entrada para $M$ é uma representação $R\langle v_1, v_2, \ldots, v_n\rangle$ para uma instância do PD, e que a saída pode ser *sim* ou *não*. Acima das setas indicando *sim* e *não*, para efeitos de clareza, pode-se colocar descrições do que significa a resposta respectiva (veja, por exemplo, a Figura 5.2). Supondo que a representação para o PD em questão seja feita a partir do alfabeto $\Sigma$, então para cada $w \in \Sigma^*$, a máquina $M$ responde *sim* se (a) $w$ representa alguma instância $p$ (ou seja, se $w$ é $R\langle v_1, v_2, \ldots, v_n\rangle$ para alguma sequência de parâmetros $v_1$, $v_2$,..., $v_n$) e (b) $M$ pára em estado final para a entrada $R\langle v_1, v_2, \ldots, v_n\rangle$; e responde *não* se (a) $w$ não representa qualquer instância ou (b) se $w$ representa alguma instância mas $M$ pára em estado não final para a mesma. Assim, a entrada para $M$, na Figura 5.1, pode ser qualquer $w \in \Sigma^*$, mas a "entrada esperada" é da forma $R\langle v_1, v_2, \ldots, v_n\rangle$. No entanto, daqui para frente, com o objetivo de simplificar a exposição, será suposto que a entrada está realmente no formato $R\langle v_1, v_2, \ldots, v_n\rangle$.[3]

**Exemplo 133** Considere o PD "determinar se uma gramática livre do contexto $G$ gera uma palavra $w$, para $G$ e $w$ arbitrários".

Como ressaltado acima, antes de construir uma máquina de Turing que solucione tal PD, deve-se conceber uma representação para suas instâncias. Para isto, basta mostrar como codificar em um alfabeto $\Sigma$, que pode ser diferente do alfabeto de $G$, os pares $(G, w)$. Neste exemplo, será usado $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.

Seja uma GLC $G = (V, \Gamma, R, P)$, onde $V = \{X_1, X_2, \ldots, X_n\}$ e $\Gamma = \{a_1, a_2, \ldots, a_k\}$. Cada variável $X_i$ será representada pela palavra $\mathtt{1}^i$, e cada terminal $a_j$ será representado pela palavra $\mathtt{1}^{n+j}$. Assim, por exemplo, se $V = \{A, B, C\}$ e $\Gamma = \{a, b\}$, tem-se a seguinte codificação: $A$: $\mathtt{1}$, $B$: $\mathtt{11}$, $C$: $\mathtt{111}$, $a$: $\mathtt{1111}$, $b$: $\mathtt{11111}$. Uma regra de $R$ pode ser codificada colocando-se, em seqüência, os códigos da variável do lado esquerdo seguido dos códigos dos símbolos da palavra do lado direito separados por $\mathtt{0}$. Assim, para o exemplo, uma regra $B \to aAb$ pode ser assim codificada: $\mathtt{1}^2\mathtt{0}\mathtt{1}^4\mathtt{0}\mathtt{1}^1\mathtt{0}\mathtt{1}^5$. Convenciona-se que as variáveis são aquelas cujos códigos aparecem antes do primeiro $\mathtt{0}$, ou seja, os símbolos do lado esquerdo. Será assumido que a variável de partida é codificada por $\mathtt{1}$.

As codificações das regras são colocadas em seqüência separadas por $\mathtt{00}$ (qualquer seqüência serve). A representação da gramática consta de $\mathtt{1}^n$, onde $n$ é o número de variáveis, seguido de $\mathtt{0}$ que, por sua vez, é seguido da representação das regras. A

[3]Tal suposição é coerente com o fato de que o teste para saber se $w$ representa alguma instância pode (e deve!) ser feito *antes* da ativação da MT relativa ao PD propriamente dito.

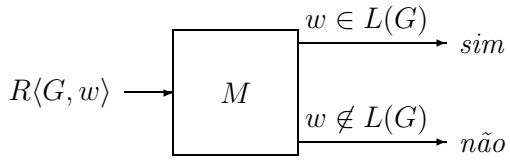



Figura 5.2: Máquina de Turing para o PD do Exemplo 133.

seqüência que representa a gramática é separada da codificação da palavra de entrada $w$ por `000`. Esta última é representada usando `0` para separar as codificações de seus símbolos. Segue abaixo um exemplo de representação de instância baseado nesta codificação.

Seja a instância "determinar se $aba \in L(H)$" para a gramática livre do contexto $H = (\{A, B, C\}, \{a, b\}, R, A)$, onde $R$ consta das regras:

$$A \to aAa \mid B$$

$$B \to aB \mid CC$$

$$C \to b \mid \lambda$$

Usando os códigos definidos acima, sejam $R\langle r_1\rangle = \mathtt{1^101^401^101^4}$ (código da regra $A \to aAa$), $R\langle r_2\rangle = \mathtt{1^101^2}$ (código da regra $A \to B$), etc. Então uma representação para tal instância, $R\langle H, aba\rangle$, seria:

$$\mathtt{1^30}R\langle r_1\rangle\mathtt{00}R\langle r_2\rangle\mathtt{00}R\langle r_3\rangle\mathtt{00}R\langle r_4\rangle\mathtt{00}R\langle r_5\rangle\mathtt{00}R\langle r_6\rangle\mathtt{0001^401^501^4}.$$

Observe que existem outras representações para esta mesma instância; por exemplo, mude a ordem de apresentação das regras. Veja também que é possível testar se qualquer palavra no alfabeto $\{0, 1\}$ representa ou não uma instância. Ou seja, a linguagem

$$\{z \in \{0,1\}^* \mid z \text{ é } R\langle G, w\rangle \text{ para alguma GLC } G \text{ e palavra } w \text{ de } G\}$$

é recursiva.

Uma máquina de Turing que solucionasse o PD em questão seria representada esquematicamente como mostrado na Figura 5.2. Observe que tal PD tem solução se, e somente se, a linguagem $\{R\langle G, w\rangle \mid w \in L(G)\}$ é recursiva. □

# Exercícios

1. Construa representações para os seguintes problemas de decisão:

   (a) Dados um vetor de números naturais $V$ e um número $n$, determinar se $n$ ocorre em $V$.

   (b) Dados um grafo $G$ e dois vértices $v_1$ e $v_2$ de $G$, determinar se existe um caminho de $v_1$ para $v_2$ em $G$.

| *Estado* | *Representação* |
|---|---|
| $e_1 = i$ | $\mathtt{1}$ |
| $e_2$ | $\mathtt{11}$ |
| $\vdots$ | $\vdots$ |
| $e_n$ | $\mathtt{1}^n$ |

| *Símbolo de* $\Gamma$ | *Representação* |
|---|---|
| $a_1 = \langle$ | $\mathtt{1}$ |
| $a_2 = \sqcup$ | $\mathtt{11}$ |
| $\vdots$ | $\vdots$ |
| $a_k$ | $\mathtt{1}^k$ |

Tabela 5.1: Representações dos estados e símbolos do alfabeto.

(c) Dadas duas gramáticas regulares, determinar se elas são equivalentes.

(d) Dado um AFD, determinar se ele reconhece uma linguagem infinita.

(e) Dados um AFD $M$ e uma palavra $w$, determinar se $M$ reconhece $w$.

Utilize o alfabeto $\{0, 1\}$.

2. Para cada um dos problemas de decisão do Exercício 1, mostrar como seria a representação esquemática do mesmo, como foi feito na Figura 5.2 para o problema de decisão do Exemplo 133.

## 5.3 Uma Máquina de Turing Universal

Uma característica indicativa do poder computacional das máquinas de Turing é a possibilidade de construir máquinas de Turing capazes de simular *qualquer* máquina de Turing[4]. Em particular, pode-se simular as máquinas de Turing que são reconhecedoras de linguagens, como é o caso daquelas que solucionam PD's.

Como assinalado na Seção 5.2, o primeiro passo para se construir uma máquina de Turing que solucione um PD é conceber uma representação para suas instâncias. Analogamente, o primeiro passo para se construir uma máquina de Turing que simule qualquer máquina de Turing é conceber uma representação para máquinas de Turing. O exemplo a seguir mostra uma representação possível.

**Exemplo 134** A seguir será mostrada uma possível representação de MT's, de forma que MT's possam ser supridas como entrada para outras MT's. O alfabeto usado na representação será $\{\mathtt{0}, \mathtt{1}\}$. Seja uma MT qualquer $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i, F)$, onde $E = \{e_1, e_2, \ldots, e_n\}$ e $\Gamma = \{a_1, a_2, \ldots, a_k\}$. Suponha que $e_1 = i$, $a_1 = \langle$, $a_2 = \sqcup$ e lembre-se que $\Sigma \subset \Gamma$. Na Tabela 5.1 estão mostradas representações para os estados e símbolos do alfabeto de fita. Com relação à direção de movimentação do cabeçote, $D$ será representada por $\mathtt{1}$ e $E$ por $\mathtt{11}$. Supondo que $F = \{f_1, f_2, \ldots, f_p\}$ e designando por $R\langle x\rangle$ a representação de $x$, seja $x$ um estado, um símbolo de $\Gamma$ ou direção de movimentação do cabeçote, a representação de $M$ tem os seguintes componentes:

- $F$ é representado por uma lista das representações dos estados finais separados por $\mathtt{0}$, ou seja, $R\langle F\rangle = R\langle f_1\rangle \mathtt{0} R\langle f_2\rangle \mathtt{0} \cdots R\langle f_p\rangle$;
- cada transição $t$ da forma $\delta(e_{i_1}, a_{j_1}) = [e_{i_2}, a_{j_2}, d]$ é representada por $R\langle t\rangle = R\langle e_{i_1}\rangle \mathtt{0} R\langle a_{j_1}\rangle \mathtt{0} R\langle e_{i_2}\rangle \mathtt{0} R\langle a$

[4]Inclusive elas próprias!

Finalmente, sendo $t_1, t_2, \ldots, t_s$ as transições de $M$, uma representação de $M$ é:

$$R\langle F\rangle \mathtt{00} R\langle t_1\rangle \mathtt{00} R\langle t_2\rangle \mathtt{00} \cdots R\langle t_s\rangle.$$

□

Um exemplo de representação de MT, usando a representação concebida no Exemplo 134, vem a seguir.

**Exemplo 135** Seja a MT cujo diagrama de estados está mostrado na Figura 4.4(b), página 206, cuja especificação é aqui reproduzida:

$$M = (\{0,1\}, \{\mathtt{a}, \mathtt{b}\}, \{\langle, \sqcup, \mathtt{a}, \mathtt{b}\}, \langle, \sqcup, \delta, \{0,1\})$$

com $\delta$ contendo somente as duas transições:

$t_1$: $\delta(0, \mathtt{a}) = [1, \mathtt{a}, \mathtt{D}]$

$t_2$: $\delta(1, \mathtt{b}) = [0, \mathtt{b}, \mathtt{E}]$.

Então uma representação para $M$ poderia ser obtida assim:

- Estados: $R\langle 0\rangle = \mathtt{1}$, $R\langle 1\rangle = \mathtt{11}$.
- Símbolos: $R\langle\langle\rangle = \mathtt{1}$, $R\langle\sqcup\rangle = \mathtt{11}$, $R\langle \mathtt{a}\rangle = \mathtt{111}$, $R\langle \mathtt{b}\rangle = \mathtt{1111}$.
- Direção: $R\langle \mathtt{D}\rangle = \mathtt{1}$, $R\langle \mathtt{E}\rangle = \mathtt{11}$.
- Transição 1: $\begin{aligned} R\langle t_1\rangle &= R\langle 0\rangle \mathtt{0} R\langle \mathtt{a}\rangle \mathtt{0} R\langle 1\rangle \mathtt{0} R\langle \mathtt{a}\rangle \mathtt{0} R\langle \mathtt{D}\rangle \\ &= \mathtt{10111011011101}. \end{aligned}$
- Transição 2: $\begin{aligned} R\langle t_2\rangle &= R\langle 1\rangle \mathtt{0} R\langle \mathtt{b}\rangle \mathtt{0} R\langle 0\rangle \mathtt{0} R\langle \mathtt{b}\rangle \mathtt{0} R\langle \mathtt{E}\rangle \\ &= \mathtt{11011110101111011}. \end{aligned}$
- Estados finais: $R\langle F\rangle = \mathtt{1011}$.
- $\begin{aligned} R\langle M\rangle &= R\langle F\rangle \mathtt{00} R\langle t_1\rangle \mathtt{00} R\langle t_2\rangle \\ &= \mathtt{10110010111011011101001101111010111101 1}. \end{aligned}$

□

É importante ressaltar, com relação ao exemplo anterior, que, dada uma palavra $w \in \{\mathtt{0}, \mathtt{1}\}^*$, é possível determinar se $w$ é ou não é uma representação de uma MT, o que está de acordo com o requerido no item (3) dos requisitos de uma representação.

Uma *máquina de Turing universal*, uma MT que simula qualquer MT $M$, deve receber como entrada, além de uma representação de $M$, uma representação de uma palavra de entrada de $M$; ou seja, deve receber como entrada, $R\langle M, w\rangle$, onde $w$ é uma palavra no alfabeto de $M$. Assim, para complementar a representação mostrada no Exemplo 134 deve-se projetar uma representação para $w$, palavra de $M$, para, finalmente, especificar como é $R\langle M, w\rangle$. Usando as mesmas convenções do Exemplo 134, e supondo que $w = a_{i_1} a_{i_2} \ldots a_{i_u}$, uma possibilidade é ter $R\langle M, w\rangle = R\langle M\rangle \mathtt{000} R\langle w\rangle$, onde $R\langle M\rangle$ é como especificado no exemplo e $R\langle w\rangle = R\langle a_{i_1}\rangle \mathtt{0} R\langle a_{i_2}\rangle \ldots \mathtt{0} R\langle a_{i_u}\rangle$.

1. Se a entrada não é da forma $R\langle M, w\rangle$, pare em estado não final;
2. copie $R\langle w\rangle$ na fita 2 e posicione cabeçote no início;
3. escreva $R\langle i\rangle$ na fita 3 e posicione cabeçote no início;
4. **ciclo**
   4.1 seja $R\langle a\rangle$ a representação sob o cabeçote da fita 2;
   4.2 seja $R\langle e\rangle$ a representação sob o cabeçote da fita 3;
   4.3 procure $R\langle e\rangle 0 R\langle a\rangle 0 R\langle e'\rangle 0 R\langle a'\rangle 0 R\langle d\rangle$ na fita 1;
   4.4 **se** encontrou **então**
      4.4.1 substitua $R\langle e\rangle$ por $R\langle e'\rangle$ na fita 3
         e volte cabeçote da fita 3 ao seu início;
      4.4.2 substitua $R\langle a\rangle$ por $R\langle a'\rangle$ na fita 2;
      4.4.3 mova cabeçote da fita 2 na direção $d$
   4.5 **senão**
      4.5.1 **se** $e$ é estado final **então**
         4.5.1.1 pare em estado final
         **senão**
         4.5.1.2 pare em estado não final
         **fimse**
   **fimse**

**fimciclo**.

Figura 5.3: Uma MT universal.

Para efeitos de clareza, será especificada uma MT universal, $U$, de 3 fitas, sendo que a primeira receberá a entrada de $U$, ou seja $R\langle M, w\rangle$, a segunda irá fazer o papel da fita de $M$, e a terceira irá conter apenas a representação do estado atual de $M$. Um algoritmo que mostra o comportamento de uma MT universal está exibido na Figura 5.3. A máquina $U$ começa, no passo 1, verificando se o conteúdo da fita 1 é da forma $R\langle M, w\rangle = R\langle M\rangle 000 R\langle w\rangle$. Se não for, $U$ pára em um estado não final. No passo 2, $R\langle w\rangle$ é copiado da fita 1 para a fita 2, inicializando, assim, a fita a ser usada para simulação de $M$. No passo 3, $R\langle i\rangle = 1$ é copiado na fita 3. Depois destas inicializações, $U$ entra em um ciclo no passo 4, onde, em cada passo simula o processamento de uma transição de $M$. No passo 4.3 $U$ pesquisa por $R\langle e\rangle 0 R\langle a\rangle$ na fita 1, onde $R\langle a\rangle$ é a representação sob o cabeçote da fita 2 e $R\langle e\rangle$ é a representação sob o cabeçote da fita 3. Se encontrar, $U$ terá acesso, em seqüência, a $R\langle e'\rangle$, $R\langle a'\rangle$ e $R\langle d\rangle$, que serão usados nos passos 4.4.1, 4.4.2 e 4.4.3 para simular o processamento de uma transição. Se não encontrar, $U$ verifica, no passo 4.5.1, se $R\langle e\rangle$ na fita 3 é a representação de um estado final; para isto, basta pesquisar no início da fita 1. Se $e$ for estado final, há aceitação, caso contrário, não há.

Concluindo, a MT $U$ aceita a linguagem

$$L(U) = \{R\langle M, w\rangle \mid w \in L(M)\}.$$

Se o critério de reconhecimento considerado for *por parada*, tem-se duas "simplificações":

- estados finais estarão ausentes de $R\langle M, w\rangle$;
- o passo 4.5.1 de U (vide Figura 5.3) será simplesmente: pare em estado final.

Chamando-se esta nova MT de $U_P$, ter-se-ia que:

$$U_P \text{ aceita } w \text{ se, e somente se, } M \text{ pára se a entrada é } w,$$

ou seja,

$$L(U_P) = \{R\langle M, w\rangle \mid M \text{ pára se a entrada é } w\}.$$

# Exercícios

1. Faça uma MT que determine se uma palavra de $\{0,1\}^*$ é da forma $R\langle M, w\rangle$ (passo 1 da MT universal), usando a representação do Exemplo 134, página 241.

2. Para MT's com alfabeto de entrada $\{0,1\}$, é possível alterar a representação mostrada no Exemplo 134, página 241, de forma que se tenha $R\langle 0\rangle = 0$ e $R\langle 1\rangle = 1$?[5] Se sim, mostre como. Se não, por que não?

3. Mostre que a linguagem $\{R\langle M\rangle \mid \lambda \in L(M)\}$ é LRE.

4. Mostre que a linguagem $\{R\langle M\rangle \mid L(M) \neq \emptyset\}$ é LRE.

## 5.4 O Problema da Parada

Nesta seção, será mostrado que o célebre problema da parada não é decidível, tanto para máquinas de Turing, quanto para linguagens do tipo Pascal, C, Java, etc. Neste último caso, será lançada mão de uma pseudo-linguagem que contém algumas construções que todas as linguagens de programação usuais têm de uma forma ou de outra.

O problema da parada para MT's pode ser assim enunciado:

> Dadas uma MT arbitrária $M$ e uma palavra arbitrária $w$, determinar se a computação de $M$ com a entrada $w$ pára.

A existência de uma MT universal $U_P$, que simula $M$ para uma entrada $w$, prova que a linguagem $L(U_P)$ é LRE. Ao ser mostrado que o problema da parada é indecidível, ter-se-á mostrado, então, que não existe uma MT que sempre pare e que seja equivalente à MT $U_P$ delineada no final da Seção 5.3, ou, equivalentemente, que $L(U_P)$ não é recursiva.

**Teorema 37** *O problema da parada para MT's é indecidível.*

*Prova*

Será feita uma prova por contradição. Suponha que o problema seja decidível. Neste caso, seja uma MT $P$ que solucionasse o problema. Tal máquina seria como mostrado na Figura 5.4, utilizando a notação introduzida na Seção 5.2. Ora, a partir de tal MT $P$, seria possível construir uma MT $P'$ cujo comportamento se dá como mostrado na Figura 5.5, ou seja, $P'$ entra em *loop* se, e somente se, $P$ pára em um estado final, ou ainda,

$$P' \text{ entra em } \textit{loop} \text{ se, e somente se, } M \text{ pára com entrada } w.$$

[5] Com isto, uma palavra $w$ teria representação $R\langle w\rangle = w$.

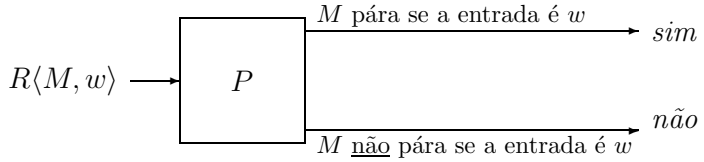



Figura 5.4: Candidato a resolver o Problema da Parada.

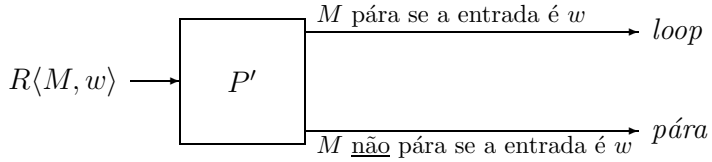



Figura 5.5: MT obtida a partir da MT $P$.

Para isto, basta fazer, para cada par $(f, a)$ tal que $\delta(f, a)$ é indefinido, onde $f$ é estado final de $P$ e $a$ símbolo de fita de $P$, $\delta(f, a) = [l, a, \mathrm{D}]$, onde $l$ é um novo estado. Neste último estado, faz-se a máquina entrar em *loop* assim: $\delta(l, a) = [l, a, \mathrm{D}]$ para todo símbolo de fita $a$ de $P$. Estas seriam as únicas transições a acrescentar a $P$ para se obter $P'$.

A partir de $P'$ pode-se obter uma outra máquina, $P''$, conforme ilustra a Figura 5.6. Para construir $P''$ a partir de $P'$, basta construir transições para (nesta ordem):

1. Duplicar a entrada (por exemplo, utilizando-se a representação desenvolvida na Seção 5.3, a partir de $R\langle M\rangle$, obter $R\langle M, M\rangle = R\langle M\rangle \mathtt{000} R\langle M\rangle$).

2. Agir como $P'$ sobre a entrada acima.

Agora, considere o que acontece se $R\langle P''\rangle$ for submetida como entrada para a MT $P''$: se $P''$ entra em *loop* para a entrada $R\langle P''\rangle$, é porque $P''$ pára se a entrada é $R\langle P''\rangle$; e se $P''$ pára quando a entrada é $R\langle P''\rangle$, é porque $P''$ não pára se a entrada é $R\langle P''\rangle$. Ou seja,

$$P'' \text{ pára com entrada } R\langle P''\rangle$$
$$\text{se e somente se}$$
$$P'' \text{ não pára com entrada } R\langle P''\rangle.$$

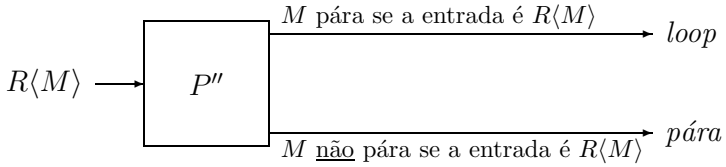



Figura 5.6: MT obtida a partir da MT $P'$.

Contradição! Mas, $P''$ pode ser construída a partir de $P$. Assim, $P$ não pode existir, e, portanto, o problema da parada para MT's é indecidível. □

A prova de que o problema da parada para as linguagens de programação procedurais comuns[6] é indecidível, segue as mesmas linhas que a prova do Teorema 37, como pode ser notado a seguir. Será usada uma pseudo-linguagem de aparência similar à que tem sido usada até aqui na representação dos algoritmos.

**Teorema 38** *O problema da parada para linguagens de programação procedurais comuns é indecidível.*

*Prova*

Suponha que a função lógica (*booleana*) $P$ solucione o PD em questão, de forma que a chamada $P(x, w)$ retorna *verdadeiro* se, e somente se, o procedimento de texto $x$ pára com a entrada (também texto) $w$. No que segue, todos os parâmetros de funções ou procedimentos serão assumidos como "textos", não havendo indicação explícita disto nas suas codificações.

Utilizando-se tal função, pode-se escrever o seguinte procedimento[7]:

**procedimento** $P''(x)$:
    **enquanto** $P(x, x)$ **faça**
    **fimenquanto**
**fim** $P''$.

Seja $T$ este texto do procedimento $P''$. Então: se $P''(T)$ pára é porque $P(T, T)$ retorna *falso*, o que significa que $P''(T)$ não pára; assim, se $P''(T)$ pára, $P''(T)$ não pára! Por outro lado, se $P''(T)$ não pára é porque $P(T, T)$ retorna *verdadeiro*, o que significa que $P''(T)$ pára; assim, se $P''(T)$ não pára, $P''(T)$ pára! Contradição. Conclui-se que a função $P$ não pode existir, ou seja, o problema em questão é indecidível.[8] □

A partir da existência da MT universal e da indecidibilidade do problema da parada para MT's, chega-se aos teoremas abaixo, que se utilizam da linguagem

$$L_P = \{R\langle M, w\rangle \mid M \text{ pára se a entrada é } w\}.$$

**Teorema 39** *A linguagem $L_P$ não é recursiva.*

*Prova*

Segue da indecidibilidade do problema da parada (Teorema 37). □

[6] Algumas linguagens de programação procedurais "comuns": Java, C, Pascal.

[7] Tal procedimento corresponde à MT $P''$ do Teorema 37.

[8] Um procedimento $P'$ correspondente à MT $P'$ do Teorema 37 seria:

**procedimento** $P'(x, w)$:
    **enquanto** $P(x, w)$ **faça**
    **fimenquanto**
**fim** $P'$.

(Ver Exercício 1, página 247.)

O teorema seguinte diz que $L_P$ é LRE, ou seja, o problema da parada é semi-decidível.

**Teorema 40** *A linguagem $L_P$ é recursivamente enumerável.*

*Prova*

Segue diretamente da existência da MT universal $U_P$ concebida no final da Seção 5.3: $L(U_P) = L_P$. □

Assim, tem-se como conseqüência de ambos os Teoremas 39 e 40, que o conjunto das linguagens recursivas é subconjunto próprio do conjunto das LRE's.

Utilizando-se este último resultado (a existência de LRE que não é recursiva) e o Teorema 36, pode-se provar que existem linguagens que não são recursivamente enumeráveis.

**Teorema 41** *A linguagem $\overline{L_P}$ não é recursivamente enumerável.*

*Prova*

Suponha que $\overline{L_P}$ é LRE. Como, pelo Teorema 40, $L_P$ é LRE, segue-se, pelo Teorema 36, que $L_P$ é recursiva. Mas isto contraria o Teorema 39, que diz que $L_P$ não é recursiva. Logo, $\overline{L_P}$ não é LRE. □

# Exercícios

1. Modificar a prova do Teorema 38 para incluir na demonstração o uso do procedimento $P'$, transcrito a seguir, de forma que ela se assemelhe mais com a prova do Teorema 37.

   **procedimento** $P'(x, w)$:
   **enquanto** $P(x, w)$ **faça**
   **fimenquanto**
   **fim** $P'$.

2. Seja a linguagem $\{R\langle M, w\rangle \mid M$ não pára se a entrada é $w\}$. Prove que esta linguagem não é recursivamente enumerável. Observe que esta linguagem é $\overline{L_P}$, excluídas as palavras que não estejam na forma $R\langle M, w\rangle$.

3. Mostre que se o problema da parada fosse decidível, então toda LRE seria recursiva.

## 5.5 Redução de um Problema a Outro

Em consonância com o exercício 1 da Seção 1.12, página 49, define-se que um PD $P$ é *redutível* a um PD $Q$, se existe um algoritmo $\mathcal{R}$ que, recebendo $x$ como entrada, produz um resultado $y$ tal que a resposta de $P$ para a entrada $x$ é idêntica ou complementar à resposta de $Q$ para a entrada $y$, qualquer que seja a entrada $x$. Diz-se, com isto, que o algoritmo $\mathcal{R}$ pode ser usado para *reduzir* o problema $P$ ao problema $Q$.

Usando-se esta definição, supondo os algoritmos expressos por meio de MT's, o PD $P$ pode ser solucionado mediante o algoritmo $\mathcal{R}$ e um algoritmo para o PD Q como mostra

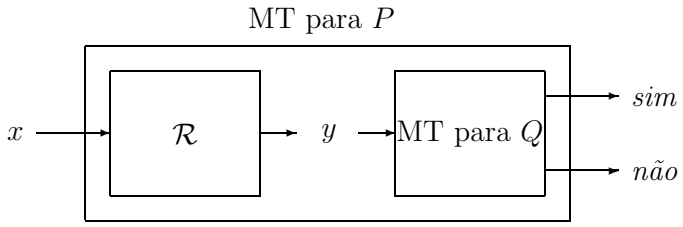



Figura 5.7: Solução de $P$ por redução a $Q$.

a Figura 5.7. Nesta figura, assume-se que a resposta de $P$ para a entrada $x$ é *idêntica* à resposta de $Q$ para a entrada $y$. No caso em que a resposta de $P$ para a entrada $x$ é *complementar* à resposta de $Q$ para a entrada $y$, a seta superior saindo da MT para $Q$ deve se dirigir para a saída *não* e a seta inferior deve se dirigir para a saída *sim*. A MT redutora $\mathcal{R}$, processando a entrada $x$, produz $y$ que é recebida como entrada pela MT que soluciona o PD Q. A partir destas duas MT's, obtém-se sem dificuldade uma MT que soluciona $P$.

A redução de problemas pode ser usada, tanto para provar que um problema é decidível, quanto para provar que um problema é indecidível:

- Se $P$ é redutível a um problema decidível, então $P$ é decidível.
- Se um problema indecidível é redutível a um problema $P$, então $P$ é indecidível.[9]

Segue um exemplo do primeiro tipo.

**Exemplo 136** Seja o problema de determinar se $w \in L(r)$, onde $r$ é uma expressão regular arbitrária. Este problema é decidível, já que pode ser reduzido ao problema, decidível, de determinar se $w \in L(M)$, onde $M$ é um AFD. A prova do Teorema 12, na página 113, mostra como fazer a redução, ou seja, como construir um AFD $M_r$, a partir de uma expressão regular $r$, tal que $w \in L(M_r)$ se, e somente se, $w \in L(r)$. □

A seguir, um exemplo do segundo tipo, ou seja, uma prova que um problema é indecidível pela redução de um problema indecidível a ele. O *problema da fita em branco* é o de determinar se uma MT arbitrária pára quando a fita começa em branco. Em outras palavras: dada uma MT $M$, arbitrária, determinar se $M$ pára quando a entrada é $\lambda$.

**Teorema 42** *O problema da fita em branco é indecidível.*

*Prova*

Observe que o problema da parada tem dois parâmetros: uma MT e uma palavra. Aqui, o problema tem apenas um parâmetro: uma MT. Deve-se determinar se esta MT pára ou não com a entrada específica $\lambda$.

O problema da parada será reduzido ao da fita em branco, estabelecendo-se, assim, a indecidibilidade do problema da fita em branco. A MT redutora (veja a Figura 5.8) produz $R\langle M' \rangle$, a partir de $R\langle M, w \rangle$, de forma que:

[9]Por outro lado, se um PD $P$ pode ser reduzido a um problema indecidível, $P$ pode ser decidível ou não. E se um problema decidível pode ser reduzido a $P$, $P$ pode ser decidível ou não.

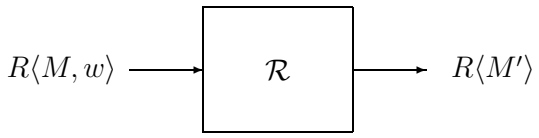



Figura 5.8: Redução do problema da parada ao da fita em branco.

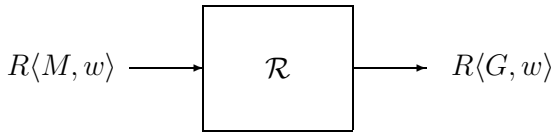



Figura 5.9: Redução do problema da parada ao da geração por gramática.

1. $M'$ escreve $w$;
2. $M'$ volta o cabeçote para o início da fita;
3. $M'$ se comporta como $M$, isto é, o resto da função de transição de $M'$ é idêntico à função de transição de $M$.

Com isto, tem-se que:

$$M \text{ pára se a entrada é } w \text{ se, e somente se } M' \text{ pára se a entrada é } \lambda.$$

□

A seguir, novamente é usada redução do problema da parada a um outro para mostrar que este é indecidível.

**Teorema 43** *O problema de determinar se $w \in L(G)$, para uma gramática irrestrita arbitrária $G$, e $w \in \Sigma^*$, onde $\Sigma$ é o alfabeto de $G$, é indecidível.*

*Prova*

O problema da parada pode ser reduzido a este usando-se a técnica do Teorema 33, página 224. Na Figura 5.9 está ilustrado que o algoritmo (MT) de redução, a partir de $M$ e $w$ ($R\langle M, w\rangle$) obtém uma gramática $G$ e a mesma palavra $w$ ($R\langle G, w\rangle$). Tal algoritmo de redução é dado pela técnica referida do Teorema 33, devendo-se considerar todos os estados de $M$ como estados finais, de forma que $M$ aceita $w$ se, e somente se, $M$ pára se a entrada é $w$. Com isto, a gramática $G$ é tal que:

$$M \text{ pára se a entrada é } w \text{ se, e somente se } G \text{ gera } w.$$

□

Daqui até o final do capítulo, será assumido que o critério de reconhecimento, para todas as MT's mencionadas, é o de parada, ou, equivalentemente, que todos os estados das MT's são estados finais. Assim, $L(M)$ será o mesmo que $L_P(M)$. Isto não introduz restrição significativa, visto que os vários critérios de reconhecimento são inter-redutíveis, como mostrado no Teorema 31, na página 208.

Voltando ao problema da fita em branco, note que ele pertence a uma classe de problemas de decisão a respeito de MT's, isto é, PD's cujo único parâmetro é uma MT. Felizmente, toda uma classe de PD's em que o único parâmetro é uma MT arbitrária pode ser provada como *indecidível*. Todos eles têm o seguinte tipo de enunciado:

> Determinar se a linguagem aceita por uma MT arbitrária $M$ satisfaz à propriedade $P$.

ou, equivalentemente:

> $\{R\langle M\rangle \mid L(M)$ satisfaz $P\}$ é recursiva?

No caso do problema da fita em branco, por exemplo, o enunciado é:

> Determinar se a linguagem aceita por uma MT arbitrária $M$ é tal que $\lambda \in L(M)$.

ou, equivalentemente:

> $\{R\langle M\rangle \mid \lambda \in L(M)\}$ é recursiva?

Para que todos os PD's da classe referida sejam indecidíveis, basta expurgar dois tipos de PD's que são, trivialmente, decidíveis: aqueles em que a propriedade $P$ é sempre verdadeira e aqueles em que $P$ é sempre falsa, ou seja, aqueles em que $\{R\langle M\rangle \mid L(M)$ satisfaz $P\}$ é o conjunto de todas as representações de MT's, e aqueles em que $\{R\langle M\rangle \mid L(M)$ satisfaz $P\} = \emptyset$. Daí, a definição abaixo.

**Definição 53** *Uma propriedade $P$ de LRE's é* trivial *se é satisfeita por toda LRE ou por nenhuma.* □

Assim, por exemplo, para o problema da fita em branco, a propriedade $\lambda \in L(M)$ não é trivial, pois algumas LRE's contêm $\lambda$ e outras não. Outros exemplos de PD's da classe em consideração que, portanto, envolvem propriedades não triviais:

- Determinar se $L(M)$ contém alguma palavra; ou seja, $\{R\langle M\rangle \mid L(M) \neq \emptyset\}$ é recursiva?
- Determinar se $L(M)$ contém todas as palavras; ou seja, $\{R\langle M\rangle \mid L(M) = \Sigma^*\}$ é recursiva?
- Determinar se $L(M)$ é finita; ou seja, $\{R\langle M\rangle \mid L(M)$ é finita$\}$ é recursiva?
- Determinar se $L(M)$ é regular; ou seja, $\{R\langle M\rangle \mid L(M)$ é regular$\}$ é recursiva?
- Determinar se $L(M)$ contém palavra começada com $\mathtt{0}$; ou seja, $\{R\langle M\rangle \mid \mathtt{0}y \in L(M)$ para algum $y \in \Sigma^*\}$ é recursiva?

Antes de apresentar o teorema de Rice, que mostra que tais problemas são indecidíveis, assim como qualquer outro no formato geral apresentado acima, será mostrado à parte que o primeiro problema acima é indecidível, usando uma técnica que modifica um pouquinho aquela usada no problema da fita em branco.

**Teorema 44** *Não existe algoritmo para determinar se a linguagem aceita por uma MT arbitrária $M$ não é $\emptyset$.*

*Prova*

O problema da parada será reduzido a este de forma similar ao que foi feito para o problema da fita em branco no Teorema 42. A MT redutora produz $R\langle M' \rangle$, a partir de $R\langle M, w \rangle$, de forma que:

1. $M'$ apaga a entrada;
2. $M'$ escreve $w$;
3. $M'$ volta o cabeçote para o início da fita;
4. $M'$ se comporta como $M$, isto é, o resto da função de transição de $M'$ é idêntico à função de transição de $M$.

Observe que a única diferença com relação a $M'$ produzida pela redução no Teorema 42, é que aqui há um passo anterior: $M'$ apaga a entrada que está na fita. Com isto, $M'$ ignora qualquer entrada que seja submetida, e se comporta sempre como $M$ com a entrada $w$. Assim, tem-se que:

$$M \text{ pára com a entrada } w \text{ se, e somente se } M' \text{ pára com alguma entrada.}$$

É interessante notar que a mesma redução serve para provar o segundo PD acima, já que $M'$ parando com alguma entrada, pára com todas, e vice-versa:

$$M' \text{ pára com alguma entrada se, e somente se, } M' \text{ pára com qualquer entrada.}$$

□

Segue o teorema de Rice.

**Teorema 45** *(*Teorema de Rice*) Se $P$ é uma propriedade não trivial de LRE's, então $\{R\langle M \rangle \mid L(M)$ satisfaz $P\}$ não é recursiva.*

*Prova*

Seja $P$ uma propriedade não trivial.

*Caso 1*: $\emptyset$ não satisfaz $P$. Seja uma MT $M_X$ tal que $L(M_X)$ satisfaz $P$. Tal MT $M_X$ existe, pois $P$ é não trivial. Observe também que, como suposto, $L(M_X) \neq \emptyset$.

O problema da parada pode ser reduzido ao de determinar se $L(M')$ satisfaz $P$, por meio de uma MT que produz $R\langle M' \rangle$ a partir de $R\langle M, w \rangle$, onde:

1. $M'$ escreve $w$ na fita, após a entrada para $M'$; suponha, então, que a fita fica assim: $\langle x[w \sqcup \ldots$, onde "$\langle$" é o símbolo de início de fita para $M'$ e "[" é símbolo de início de fita para $M$;

2. $M'$ se comporta como $M$ sobre $[w \sqcup \ldots$;

3. quando $M'$ pára, na situação em que $M$ pára, $M'$ se comporta como $M_X$ sobre $\langle x \sqcup \ldots$

Observe que

$$L(M') \text{ satisfaz } P \text{ se, e somente se, } M \text{ pára com entrada } w,$$

pois:

- Se $M$ pára com entrada $w$, $L(M') = L(M_X)$; portanto, $L(M')$ satisfaz $P$.
- Se $M$ não pára com entrada $w$, $L(M') = \emptyset$; dada a suposição inicial deste caso, então $L(M')$ não satisfaz $P$.

Assim, conclui-se que o problema de determinar se $L(M)$ satisfaz $P$, para MT's arbitrárias $M$, é indecidível, ou seja, $\{R\langle M\rangle \mid L(M) \text{ satisfaz } P\}$ não é recursiva.

*Caso 2*: $\emptyset$ satisfaz $P$. Neste caso, $\emptyset$ não satisfaz $\neg P$. E como $P$ não é trivial, $\neg P$ também não é trivial. Pela argumentação do caso 1, $\{R\langle M\rangle \mid L(M) \text{ satisfaz } \neg P\}$ não é recursiva. Como, esta linguagem é o *complemento*[10] de $\{R\langle M\rangle \mid L(M) \text{ satisfaz } P\}$, e as linguagens recursivas são fechadas sob complementação[11], segue-se que $\{R\langle M\rangle \mid L(M) \text{ satisfaz } P\}$ não é recursiva. □

Na próxima seção serão apresentados vários problemas indecidíveis relativos a GLC's. Antes, porém, o problema da parada será reduzido a um problema "intermediário", o qual servirá como ponto de partida para a abordagem daqueles PD's relativos a GLC's. Tal problema é o dito *problema da correspondência de Post*.

**Definição 54** *Um* sistema de correpondência de Post *(SCP) é um par $(\Sigma, P)$, onde $P$ é uma seqüência finita de pares $(x, y)$, onde $x, y \in \Sigma^+$.* □

Uma *solução* para um SCP $S = (\Sigma, P)$ é uma seqüência finita de pares de $P$ tal que a palavra formada pela concatenação dos primeiros elementos dos pares seja idêntica à palavra formada pela concatenação dos segundos elementos dos mesmos pares. Observe que cada par pode aparecer várias vezes na seqüência. Segue uma definição mais formal.

**Definição 55** *Seja um SCP $S = (\Sigma, [(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)])$. Uma* solução para *$S$ é uma seqüência $i_1, i_2, \ldots, i_k$ tal que*

$$x_{i_1} x_{i_2} \ldots x_{i_k} = y_{i_1} y_{i_2} \ldots y_{i_k},$$

*onde $1 \leq i_j \leq n$ para $1 \leq j \leq k$.* □

[10] É o complemento com relação ao conjunto das palavras da forma $R\langle M\rangle$, que é recursivo.
[11] Mesmo relativa a conjunto recursivo, como pode ser facilmente ser verificado.

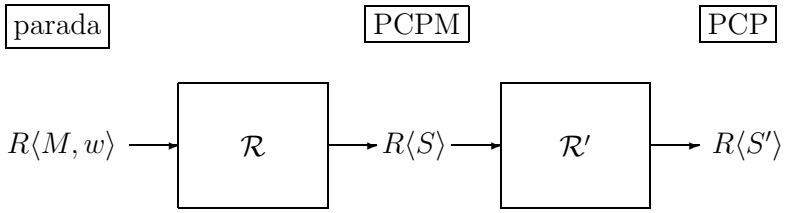



Figura 5.10: Reduções para o PCP.

**Exemplo 137** Seja o SCP $(\{\mathtt{0},\mathtt{1}\},[(\mathtt{10},\mathtt{0}),(\mathtt{0},\mathtt{010}),(\mathtt{01},\mathtt{11})])$. Este SCP tem três pares: $(x_1 = \mathtt{10}, y_1 = \mathtt{0})$, $(x_2 = \mathtt{0}, y_2 = \mathtt{010})$ e $(x_3 = \mathtt{01}, y_3 = \mathtt{11})$ . Uma solução seria a seqüência 2131, pois $x_2x_1x_3x_1 = y_2y_1y_3y_1$: $\mathtt{0\ 10\ 01\ 10} = \mathtt{010\ 0\ 11\ 0}$. Para maior clareza, pode ser conveniente apresentar cada par $(x_i, y_i)$ na forma $\frac{x_i}{y_i}$. Neste caso, a solução referida pode ser apresentada assim:

$$\frac{\mathtt{0}}{\mathtt{010}}\frac{\mathtt{10}}{\mathtt{0}}\frac{\mathtt{01}}{\mathtt{11}}\frac{\mathtt{10}}{\mathtt{0}}.$$

Além destas, quaisquer quantidades de justaposições da seqüência acima é solução. Exemplos: 21312131, 213121312131, etc. □

O PD a ser abordado, o *problema da correspondência de Post* (PCP), é:

*Determinar se um SCP arbitrário tem solução.*

Será mostrado que este problema é indecidível em dois passos: primeiramente, um PD similar ao PCP, denominado PCP modificado (PCPM), será reduzido ao PCP; em seguida, o problema da parada será reduzido ao PCPM. A Figura 5.10 esquematiza as reduções a serem feitas.

O *problema da correspondência de Post modificado* (PCPM), é:

*Determinar se um SCP arbitrário tem solução iniciada com 1.*

Ou seja, dado um SCP $S = (\Sigma, [(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)])$, no PCPM a *solução para* $S$ deve ser uma seqüência iniciada com 1: $1, i_2, \ldots, i_k$, tal que

$$x_1x_{i_2}\ldots x_{i_k} = y_1y_{i_2}\ldots y_{i_k},$$

onde $1 \leq i_j \leq n$ para $1 \leq j \leq k$ (observe que o par $(x_1, y_1)$ pode ser reutilizado).

**Exemplo 138** Seja o SCP $(\{\mathtt{0},\mathtt{1}\},[(\mathtt{0},\mathtt{010}),(\mathtt{10},\mathtt{0}),(\mathtt{01},\mathtt{11})])$, obtido do SCP do Exemplo 137 invertendo-se o primeiro e o segundo pares. Uma solução para este SCP, que satisfaz os requisitos do PCPM, seria: 1232.

Por outro lado, o SCP do Exemplo 137, $(\{\mathtt{0},\mathtt{1}\},[(\mathtt{10},\mathtt{0}),(\mathtt{0},\mathtt{010}),(\mathtt{01},\mathtt{11})])$, não tem solução que satisfaça os requisitos do PCPM, pois no primeiro par uma palavra começa com `1` e a outra com `0`. □

**Teorema 46** *O PCPM é redutível ao PCP.*

*Prova*

Seja um SCP $S = (\Sigma, P)$ com $P = [(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)]$. Seja "$*$" um símbolo não pertencente a $\Sigma$, e sejam:

- $x'_i$ o resultado de colocar "$*$" *após* cada símbolo de $x_i$. Por exemplo, se $x_i = 11010$, então $x_i = 1*1*0*1*0*$.
- $y'_i$ o resultado de colocar "$*$" *antes* de cada símbolo de $y_i$. Por exemplo, se $y_i = 0100$, então $y_i = *0*1*0*0$.

Seja "#" um símbolo não pertencente a $\Sigma$, e seja o SCP $S' = (\Sigma \cup \{*, \#\}, P')$, onde $P'$ é constituído de $n + 2$ pares:

- $(*x'_1, y'_1)$ (a ser o primeiro par de uma solução);
- $(x'_i, y'_i)$ para $1 \leq i \leq n$; e
- $(\#, *\#)$.

Será mostrado, como requerido, que $S$ tem solução começada com $(x_1, y_1)$ se, e somente se, $S'$ tem solução:

($\rightarrow$) Suponha que $S$ tem solução iniciada com $(x_1, y_1)$, e seja

$$1, i_1, i_2, \ldots, i_k$$

uma solução arbitrária de $S$, onde $1 \leq i_j \leq n$. Neste caso, uma solução para $S'$ seria (mostrando-se os pares ao invés de índices):

$$\frac{*x'_1}{y'_1}\frac{x'_{i_1}}{y'_{i_1}}\frac{x'_{i_2}}{y'_{i_2}}\cdots\frac{x'_{i_k}}{y'_{i_k}}\frac{\#}{*\#}.$$

($\leftarrow$) Suponha que $S'$ tem solução. O único par de $P'$ em que seus elementos começam com o mesmo símbolo é $(*x'_1, y'_1)$, e o único par de $P'$ em que seus elementos terminam com o mesmo símbolo é $(\#, *\#)$. Assim, uma solução para $S'$ só pode ter a forma

$$\frac{*x'_1}{y'_1}\frac{x'_{i_1}}{y'_{i_1}}\frac{x'_{i_2}}{y'_{i_2}}\cdots\frac{x'_{i_k}}{y'_{i_k}}\frac{\#}{*\#}.$$

Além disto, existe uma solução desta forma em que $(*x'_1, y'_1)$ ocorre apenas no início e $(\#, *\#)$ ocorre apenas no final. Neste caso,

$$\frac{x_1}{y_1}\frac{x_{i_1}}{y_{i_1}}\frac{x_{i_2}}{y_{i_2}}\cdots\frac{x_{i_k}}{y_{i_k}}$$

é uma solução para $S$, e esta, necessariamente, começa com $(x_1, y_1)$. □

Para provar que o PCPM é indecidível, no Teorema 47, o problema da parada será reduzido a ele. Para isto, dada uma MT $M$ qualquer e uma palavra $w$, deve-se construir um SCP $S$ que tenha solução iniciada por certo par se, e somente se, $M$ pára se a entrada é $w$. A idéia a ser usada é construir $S$ de tal forma que, havendo uma solução para $S$,

$$\frac{x_1}{y_1}\frac{x_{i_2}}{y_{i_2}}\cdots\frac{x_{i_k}}{y_{i_k}},$$

ela seja tal que $x_1 x_{i_2} \cdots x_{i_k}$ "represente" a computação de $M$ para a entrada $w$. Além disto, a solução deve existir somente se $M$ parar quando a entrada é $w$.

**Teorema 47** *Não existe algoritmo para o PCPM.*

*Prova*

O problema da parada será reduzido ao PCPM. Assim, seja uma MT $M = (E, \Sigma, \Gamma, \langle, \sqcup, \delta, i)$ e uma palavra $w \in \Sigma^*$. A partir destas, será produzido um SCP $S = (\Delta, P)$, onde:

- $\Delta = \Gamma \cup \{*, \#\}$;
- o primeiro elemento de $P$ é $(*, *\langle iw*)$;
- os pares restantes de $P$ são:
  - (a) $(c, c)$, para cada $c \in \Gamma$;
    $(*, *)$.
  - (b) Para cada $a, b \in \Gamma$ e $e, e' \in E$:
    $(ea, be')$, se $\delta(e, a) = [e', b, \mathrm{D}]$;
    $(e*, be'*)$, se $\delta(e, \sqcup) = [e', b, \mathrm{D}]$;
    $(cea, e'cb)$, se $\delta(e, a) = [e', b, \mathrm{E}]$, para cada $c \in \Gamma$;
    $(ce*, e'cb*)$, se $\delta(e, \sqcup) = [e', b, \mathrm{E}]$, para cada $c \in \Gamma$.
  - (c) $(ea, \#)$, se $\delta(e, a)$ é indefinido, para cada $e \in E$ e $a \in \Gamma$;
    $(e*, \#*)$, se $\delta(e, \sqcup)$ é indefinido, para cada $e \in E$.
  - (d) $(c\#, \#)$ para cada $c \in \Gamma$;
    $(\#c, \#)$ para cada $c \in \Gamma$.
  - (e) $(*\#*, *)$.

Pode-se mostrar que $M$ pára quando a entrada é $w$ se, e somente se, o SCP $S$ tem solução iniciada com $(*, *\langle iw*)$. □

A seguir, mostra-se um exemplo do uso da técnica desenvolvida na prova do Teorema 47.

**Exemplo 139** Seja a MT cujo diagrama de estados está mostrado na Figura 4.4(b). A partir dela e da palavra $w = \mathtt{aab}$, pode-se construir, usando a técnica do Teorema 47, um SCP cujo primeiro par é $(*, *\langle 0\mathtt{aab}*)$, e os restantes são:

(a) $(\langle, \langle)$, $(\sqcup, \sqcup)$,$(\mathtt{a},\mathtt{a})$,$(\mathtt{b},\mathtt{b})$;
$(*, *)$.

(b) $(0\mathtt{a}, \mathtt{a}1)$,
$(\langle 1\mathtt{b}, 0\langle\mathtt{b})$, $(\sqcup 1\mathtt{b}, 0 \sqcup \mathtt{b})$, $(\mathtt{a}1\mathtt{b}, 0\mathtt{ab})$, $(\mathtt{b}1\mathtt{b}, 0\mathtt{bb})$.

(c) $(0\langle, \#)$, $(0\sqcup, \#)$, $(0\mathtt{b}, \#)$,
$(1\langle, \#)$, $(1\sqcup, \#)$, $(1\mathtt{a}, \#)$,
$(0*, \#*)$, $(1*, \#*)$.

(d) $(\langle\#,\#)$, $(\sqcup\#,\#)$, $(\mathtt{a}\#,\#)$, $(\mathtt{b}\#,\#)$,
$(\#\langle,\#)$, $(\#\sqcup,\#)$, $(\#\mathtt{a},\#)$, $(\#\mathtt{b},\#)$,

(e) $(*\#*,*)$.

Uma solução para o SCP acima, espelhando uma computação de $M$ com a palavra $\mathtt{aab}$, seria construída, passo a passo, assim:

- Começa-se com o par inicial:

$$\frac{*}{*\langle 0\mathtt{aab}*}$$

- Coloca-se o par $(\langle,\langle)$, e, em seguida, $(0\mathtt{a},\mathtt{a}1)$:

$$\frac{*\langle 0\mathtt{a}}{*\langle 0\mathtt{aab}*\langle \mathtt{a}1}$$

- Coloca-se os pares $(\mathtt{a},\mathtt{a})$, $(\mathtt{b},\mathtt{b})$, $(*,*)$, $(\langle,\langle)$ e $(\mathtt{a},\mathtt{a})$:

$$\frac{*\langle 0\mathtt{aab}*\langle \mathtt{a}}{*\langle 0\mathtt{aab}*\langle \mathtt{a}1\mathtt{ab}*\langle \mathtt{a}}$$

- Coloca-se o par $(1\mathtt{a},\#)$:

$$\frac{*\langle 0\mathtt{aab}*\langle \mathtt{a}1\mathtt{a}}{*\langle 0\mathtt{aab}*\langle \mathtt{a}1\mathtt{ab}*\langle \mathtt{a}\#}$$

- Coloca-se os pares $(\mathtt{b},\mathtt{b})$, $(*,*)$, $(\langle,\langle)$ e $(\mathtt{a},\mathtt{a})$:

$$\frac{*\langle 0\mathtt{aab}*\langle \mathtt{a}1\mathtt{ab}*\langle \mathtt{a}}{*\langle 0\mathtt{aab}*\langle \mathtt{a}1\mathtt{ab}*\langle \mathtt{a}\#\mathtt{b}*\langle \mathtt{a}}$$

- Coloca-se o par $(\#\mathtt{b},\#)$:

$$\frac{*\langle 0\mathtt{aab}*\langle \mathtt{a}1\mathtt{ab}*\langle \mathtt{a}\#\mathtt{b}}{*\langle 0\mathtt{aab}*\langle \mathtt{a}1\mathtt{ab}*\langle \mathtt{a}\#\mathtt{b}*\langle \mathtt{a}\#}$$

- Coloca-se os pares $(*,*)$ e $(\langle,\langle)$:

$$\frac{*\langle 0\mathtt{aab}*\langle \mathtt{a}1\mathtt{ab}*\langle \mathtt{a}\#\mathtt{b}*\langle}{*\langle 0\mathtt{aab}*\langle \mathtt{a}1\mathtt{ab}*\langle \mathtt{a}\#\mathtt{b}*\langle \mathtt{a}\#*\langle}$$

- Coloca-se o par $(\mathtt{a}\#, \#)$:

$$\frac{*\langle 0\mathtt{aab} * \langle \mathtt{a1ab} * \langle \mathtt{a}\#\mathtt{b} * \langle \mathtt{a}\#}{*\langle 0\mathtt{aab} * \langle \mathtt{a1ab} * \langle \mathtt{a}\#\mathtt{b} * \langle \mathtt{a}\# * \langle \#}$$

- Coloca-se o par $(*, *)$ e $(\langle \#, \#)$:

$$\frac{*\langle 0\mathtt{aab} * \langle \mathtt{a1ab} * \langle \mathtt{a}\#\mathtt{b} * \langle \mathtt{a}\# * \langle \#}{*\langle 0\mathtt{aab} * \langle \mathtt{a1ab} * \langle \mathtt{a}\#\mathtt{b} * \langle \mathtt{a}\# * \langle \# * \#}$$

- Para finalizar, coloca-se o par $(*\#*, *)$:

$$\frac{*\langle 0\mathtt{aab} * \langle \mathtt{a1ab} * \langle \mathtt{a}\#\mathtt{b} * \langle \mathtt{a}\# * \langle \# * \#*}{*\langle 0\mathtt{aab} * \langle \mathtt{a1ab} * \langle \mathtt{a}\#\mathtt{b} * \langle \mathtt{a}\# * \langle \# * \#*}$$ □

Partindo-se da indecidibilidade do PCP, pode-se provar, com facilidade, que vários problemas relativos a gramáticas livres do contexto são indecidíveis. Este é o assunto da próxima seção.

# Exercícios

1. Para cada PD abaixo, mostre que o mesmo é decidível:
   (a) Dadas uma MT $M$ e uma palavra $w$, determinar se $M$ pára se a entrada é $w$ em, no máximo, 1000 transições.
   (b) Dada uma MT $M$, determinar se $M$ escreve algum símbolo diferente do branco, para a entrada $\lambda$.
   (c) Dada uma MT $M$, determinar se $M$ irá mover o cabeçote para a esquerda alguma vez, para a entrada $\lambda$.
2. Para cada PD abaixo, mostre que o mesmo é indecidível:
   (a) Dados uma MT $M$, uma palavra $w$ e um estado $e$ de $M$, determinar se a computação de $M$ para a entrada $w$ atinge o estado $e$.
   (b) Dados uma MT $M$ e um estado $e$ de $M$, determinar se a computação de $M$ para a entrada $\lambda$ atinge o estado $e$.
   (c) Dada uma MT $M$, determinar se a computação de $M$ para a entrada $\lambda$ "volta" ao estado inicial de $M$.
   (d) Dados uma MT $M$ e um símbolo $a$ de $M$, determinar se a computação de $M$ para a entrada $\lambda$ escreve $a$ na fita em algum momento.
   (e) Dados uma MT $M$ e uma expressão regular $r$, determinar se $L(M) \cap L(r) \neq \emptyset$.

3. Reduza o problema da parada para linguagens de alto nível (*determinar se um programa p com entrada w pára*) ao problema de determinar se um programa (sem entrada) pára (*determinar se um programa p pára*).

4. Seja o problema: *dada uma MT $M$, determinar se $x \in L(M)$, onde $x$ é uma palavra específica.* (Observe que o único parâmetro deste problema é $M$.)

   (a) Pode-se usar o teorema de Rice para mostrar que este problema é indecidível? Justifique.

   (b) Reduza o problema da parada a este.

   (c) A linguagem $L_x = \{R\langle M\rangle \mid x \in L(M)\}$ é recursivamente enumerável? Justifique.

5. Seja a MT $M = (\{A, B, C\}, \{\mathtt{0}, \mathtt{1}\}, \{\langle, \sqcup, \mathtt{0}, \mathtt{1}\}, \langle, \sqcup, \delta, A)$, onde $\delta$ é dada por:

$$\begin{array}{lll} \delta(A, \mathtt{0}) = [A, \mathtt{0}, \mathtt{D}], & \delta(B, \mathtt{1}) = [A, \mathtt{1}, \mathtt{D}], & \delta(C, \mathtt{1}) = [C, \sqcup, \mathtt{E}], \\ \delta(A, \mathtt{1}) = [B, \mathtt{1}, \mathtt{D}], & \delta(B, \sqcup) = [C, \mathtt{1}, \mathtt{E}]. & \end{array}$$

   (a) Construa um SCP a partir de $M$ e $w = \mathtt{0011}$, como indicado na prova do Teorema 47.

   (b) Mostre como é obtida uma solução para o SCP construído.

6. Prove que o PCP para SCP's com alfabeto de um único símbolo é decidível.

## 5.6 Alguns Problemas Indecidíveis Sobre GLC's

A seguir será mostrado como construir duas GLC's, $G_x$ e $G_y$, a partir de qualquer SCP, que são úteis para mostrar que alguns problemas de decisão relativos a GLC's não têm solução. Assim, seja um SCP $S = (\Sigma, P)$, onde $P$ tem $n$ pares $(x_1, y_1)$, $(x_2, y_2)$, ..., $(x_n, y_n)$. Sejam também $n$ símbolos distintos $s_1$, $s_2$, ..., $s_n$, sendo que nenhum é membro de $\Sigma$. Estes últimos servem para "indexar" os pares. Assim, $s_1$ indexa $(x_1, y_1)$, $s_2$ indexa $(x_2, y_2)$, etc. Seguem as definições das duas GLC's:

- $G_x = (\{P_x\}, \Sigma \cup \{s_1, s_2, \ldots, s_n\}, R_x, P_x)$, onde $R_x$ consta das $2n$ regras:

$$\begin{array}{rcl} P_x & \to & x_i P_x s_i, \text{ para cada } 1 \leq i \leq n, \text{ e} \\ P_x & \to & x_i s_i, \text{ para cada } 1 \leq i \leq n. \end{array}$$

- $G_y = (\{P_y\}, \Sigma \cup \{s_1, s_2, \ldots, s_n\}, R_y, P_y)$, onde $R_y$ consta das $2n$ regras:

$$\begin{array}{rcl} P_y & \to & y_i P_y s_i, \text{ para cada } 1 \leq i \leq n, \text{ e} \\ P_y & \to & y_i s_i, \text{ para cada } 1 \leq i \leq n. \end{array}$$

Veja que se $i_1 i_2 \ldots i_k$ é uma solução de $S$, então

$$P_x \stackrel{*}{\Rightarrow} x_{i_1} x_{i_2} \ldots x_{i_k} s_{i_k} s_{i_{k-1}} \ldots s_{i_1} \text{ e } P_y \stackrel{*}{\Rightarrow} y_{i_1} y_{i_2} \ldots y_{i_k} s_{i_k} s_{i_{k-1}} \ldots s_{i_1}$$
$$\text{e } x_{i_1} x_{i_2} \ldots x_{i_k} = y_{i_1} y_{i_2} \ldots y_{i_k}.$$

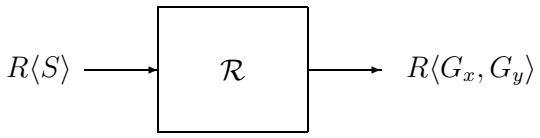



Figura 5.11: Redução do PCP à disjunção de LLC's.

e vice-versa. Conclui-se, então, que:

$$S \text{ tem solução se, e somente se, } L(G_x) \cap L(G_y) \neq \phi.$$

Isto permite concluir o resultado a seguir.

**Teorema 48** *Não existe algoritmo para determinar se as linguagens de duas GLC's são disjuntas.*

*Prova*

O PCP pode ser reduzido a este construindo-se $G_x$ e $G_y$, conforme esquematizado na Figura 5.11, de forma que um SCP $S$ tem solução se, e somente se, $L(G_x) \cap L(G_y) \neq \phi$. □

Embora as LLC's não sejam fechadas sob complementação, o fato é que existem GLC's para $\overline{L(G_x)}$ e $\overline{L(G_y)}$ (vide Exercício 2 no final desta seção, página 260). Com base nisto, segue o próximo teorema.

**Teorema 49** *Não existe algoritmo para determinar se $L(G) = \Sigma^*$, para uma GLC $G$ arbitrária.*

*Prova*

O PCP pode ser reduzido a este construindo-se uma GLC para $\overline{L(G_U)} \cup \overline{L(G_V)}$, pois:

$$\overline{L(G_x)} \cup \overline{L(G_y)} = \Sigma^* \text{ se, e somente se, } S \text{ não tem solução,}$$

pois:

$$\overline{L(G_x)} \cup \overline{L(G_y)} = \Sigma^* \text{ se, e somente se, } \overline{\overline{L(G_x)} \cup \overline{L(G_y)}} = \phi$$

e

$$\overline{\overline{L(G_x)} \cup \overline{L(G_y)}} = \phi \text{ se, e somente se, } L(G_x) \cap L(G_y) = \phi.$$

□

**Teorema 50** *Não existe algoritmo para determinar se uma GLC arbitrária é ambígua.*

*Prova*

O PCP pode ser reduzido a este produzindo-se, a partir de um SCP $S$, a gramática

$$G = (\{S, P_x, P_y\}, \Sigma \cup \{s_1, s_2, \ldots, s_n\}, R_x \cup R_y \cup \{P \to P_x, P \to P_y\}, P)$$

pois:

$$G \text{ é ambígua se, e somente se, } L(G_x) \cap L(G_y) \neq \phi.$$

□

## Exercícios

1. Construa as gramáticas $G_x$ e $G_y$, como definidas no início da seção, correspondentes ao SCP $(\{\mathtt{0},\mathtt{1}\}, P)$, onde $P$ consta dos pares $(\mathtt{01},\mathtt{011}), (\mathtt{001},\mathtt{01}), (\mathtt{10},\mathtt{00})$.

2. Construa uma gramática para a linguagem $\overline{L(G_x)}$, onde $G_x$ é a GLC definida no início da seção.

3. Mostre que o seguinte problema é ou não decidível: dadas uma GR $G_R$ e uma GLC $G_L$, determinar se $L(G_R) \cap L(G_L) = \emptyset$.

4. Mostre que o seguinte problema não é decidível: dadas duas GLC's $G_1$ e $G_2$, determinar se $L(G_1) \cap L(G_2)$ é finito.

5. Mostre que o seguinte problema não é decidível: dada uma GLC $G$, determinar se $\overline{L(G)}$ é finito.

6. Mostre que são indecidíveis os problemas de determinar, dadas duas GLC's $G_1$ e $G_2$, que:

   (a) $L(G_1) = L(G_2)$.

   (b) $L(G_1) \subseteq L(G_2)$.

## 5.7 Exercícios

1. Mostre que o seguinte problema é indecidível: *dada uma máquina de Turing $M$, determinar se $M$ move seu cobeçote para o início da fita (local em que está o símbolo especial "$\langle$") em algum momento.*

2. Para cada PD abaixo, mostre que o mesmo é ou não é decidível:

   (a) Determinar se $L(M) \cap L(P) = \emptyset$, para um AF $M$ e um AP $P$.

   (b) Determinar se $L(M) = L(P)$, para um AF $M$ e um AP $P$.

   (c) Determinar se existe uma MT $M$ que denota $L(r)$, para uma ER $r$.

   (d) Determinar se existe uma ER $r$ que denota $L(M)$, para uma MT $M$.

3. Mostre que o PCP para SCP's com alfabeto de dois símbolos é indecidível. Para isto, reduza o PCP a este problema.

4. Um sistema semi-Thue é um par $S = (\Sigma, R)$ onde $\Sigma$ é um alfabeto e $R$ é um conjunto de regras da forma $u \to v$, onde $u \in \Sigma^+$ e $u \in \Sigma^*$. Seja o seguinte problema associado a sistemas semi-Thue:

   > *Dados um sistema semi-Thue $S$ e duas palavras $x \in \Sigma^+$ e $y \in \Sigma^*$, determinar se $y$ pode ser derivada a partir de $x$.*

A noção de derivação aqui é similar àquela usada no contexto de gramáticas, onde cada passo da derivação resulta da aplicação de uma regra. Mostrar que o PD enunciado acima é indecidível, reduzindo:

(a) O problema da parada para MT's a ele.

(b) O problema de determinar se uma gramática irrestrita $G$ gera uma palavra $w$ a ele.

5. Seja $G_x$ uma GLC obtida como mostrado no início da Seção 5.6, página 258. Descreva AP's para reconhecer:

   (a) $L(G_x)$.

   (b) $\overline{L(G_x)}$.

6. Pelo Teorema de Rice, as seguintes linguagens não são recursivas:

   (a) $\{R\langle M\rangle \mid \lambda \in L(M)\}$.

   (b) $\{R\langle M\rangle \mid L(M)$ aceita uma palavra de tamanho $10\}$.

   (c) $\{R\langle M\rangle \mid L(M)$ é finita$\}$.

   (d) $\{R\langle M\rangle \mid L(M)$ é regular$\}$.

   Elas são recursivamente enumeráveis?

## 5.8 Notas Bibliográficas

O primeiro autor a formular a denominada tese de Church-Turing foi Church em [Chu36], seguido por Turing em [Tur36]. Em [Tur36] aparecem também as máquinas de Turing universais e uma prova da indecidibilidade do problema da parada.

Referências importantes sobre decidibilidade após o aparecimento dos primeiros computadores são [Dav58], [Rog67] e [Min67].

O teorema de Rice foi demonstrado por Rice[Ric53][Ric56].

Várias propridades indecidíveis com relação a linguagens livres do contexto foram apresentadas por Bar-Hillel, Perles e Shamir[BPS61], Ginsburg e Rose[GR63b] e Hartmanis e Hopcroft[HH68].

# Bibliografia


[BA01] M. Ben-Ari. *Mathematical Logic for Computer Science*. Springer-Verlag, second edition, 2001.

[Bac59] J. W. Backus. The syntax and semantics of the proposed international algebraic language of the zurich ACM-GAMM conference. In *Proceedings of the International Conference on Information Processing*, pages 125–132. UNESCO, 1959.

[BF96] E. Burke and E. Foxley. *Logic and its Applications*. International Series in Computer Science. Prentice Hall, 1996.

[BPS61] Y. Bar-Hillel, M. Perles, and E. Shamir. On formal properties of simple phrase structure languages. *Zeitschrift fur Phonetik, Sprachwissenschaft, und Kommunikationsforshung*, 14:143–172, 1961.

[Cho56] N. Chomsky. Three models for the description of languages. *IRE Transactions on Information Theory*, 2(3):113–124, 1956.

[Cho59] N. Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

[Cho62] N. Chomsky. Context-free grammars and pushdown storage. Technical report, MIT Research Laboratory in Electronics, Cambridge, MA, 1962.

[Chu36] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58:345–363, 1936.

[Cop81] I. M. Copi. *Introdução à Lógica*. Mestre Jou, 1981.

[Dav58] M. Davis. *Computability and Unsolvability*. McGraw-Hill, 1958.

[Dav95] A. Davison. *Humour the Computer*. The MIT Press, 1995.

[Dav00] M. Davis. *The Universal Computer: The road from Leibniz to Turing*. W.W. Norton & Co., 2000.

[Dea97] N. Dean. *The Essence of Discrete Mathematics*. Prentice Hall, 1997.

[dS02] J. N. de Souza. *Lógica para Ciência da Computação*. Campus, 2002.

[End00] H. B. Enderton. *A Mathematical Introduction to Logic*. Science & Technology Books, second edition, 2000.

[Epp90] S. Epp. *Discrete Mathematics with Applications*. Wadsworth, 1990.

[EPR81] A. Ehrenfeucht, R. Parikh, and G. Rozenberg. Pumping lemmas and regular sets. *SIAM Journal on Computing*, 10:536–541, 1981.

[Eve63] J. Evey. Application of pushdown store machines. In *Proceedings of the 1963 Fall Joint Computer Conference*, pages 215–227. AFIPS Press, 1963.

[FB94] R. W. Floyd and R. Beigel. *The Language of Machines: An Introduction to Computability and Formal Languages*. Computer Science Press, 1994.

[Fis63] P. C. Fischer. On computability by certain classes of restricted turing machines. In *Proceedings of the Fourth Annual IEEE Symposium on Switching Circuit Theory and Logical Design*, pages 23–32, 1963.

[GG66] S. Ginsburg and S. A. Greibach. Deterministic context-free languages. *Information and Control*, 9(6):563–582, 1966.

[GH98] R. Greenlaw and H. J. Hoover. *Fundamentals of the Theory of Computation: Principles ans Practice*. Morgan Kaufmann, 1998.

[Gin66] S. Ginsburg. *The Mathematical Theory of Context-Free Languages*. McGraw-Hill, 1966.

[GR63a] S. Ginsburg and G.F. Rose. Operations which preserve definability in languages. *Journal of the Association for Computing Machinery*, 10(2):175–195, 1963.

[GR63b] S. Ginsburg and G.F. Rose. Some recursively unsolvable problems in algol-like languages. *Journal of the Association for Computing Machinery*, 10(1):29–47, 1963.

[GR66] S. Ginsburg and G.F. Rose. Preservation of languages by transducers. *Information and Control*, 9(2):153–176, 1966.

[Gre65] S. A. Greibach. A new normal form theorem for context-free phrase structure grammars. *Journal of the Association for Computing Machinery*, 12(1):42–52, 1965.

[Gri94] R. P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction*. Addison Wesley, third edition, 1994.

[GS63] S. Ginsburg and E. H. Spanier. Quocients of context-free languages. *Journal of the Association for Computing Machinery*, 10(4):487–492, 1963.

[Hai91] L. Haines. *Generation and recognition of formal languages*. PhD thesis, MIT, Cambridge, MA, 1991.

[Hal91] P. Halmos. *Naive Set Theory*. Springer-Verlag, 1991.

[HH68] J. Hartmanis and J. E. Hopcroft. Structure of undecidable problems in automata theory. In *Proceedings of the Ninth Annual IEEE Symposium on Switching and Automata Theory*, pages 327–333, 1968.

[HMU01] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, second edition, 2001.

[Hod01] W. Hodges. *Logic: An introduction to Elementary Logic*. Penguin Books, second edition, 2001.

[Hop71] J. E. Hopcroft. An $n \log n$ algorithm for minimizing the states in a finite automaton. In Z. Kohavi, editor, *The Theory of Machines and Computation*, pages 189–196. Academic Press, 1971.

[HU79] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 1979.

[Huf54] D. A. Huffman. The syntesis of sequential switcing circuits. *Journal of the Franklin Institute*, 257(3-4):161–190 and 275–303, 1954.

[Jaf78] J. Jaffe. A necessary and sufficient pumping lemma for regular languages. *SIGACT News*, 10:48–49, 1978.

[Kle43] S. C. Kleene. Recursive predicates and quantifiers. *Transactions of the American Mathematical Society*, 53:41–74, 1943.

[Kle56] S. C. Kleene. Representation of events in nerve nets and finite automata. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 3–41. Princeton University Press, 1956.

[Koz97] D. C. Kozen. *Automata and Computability*. Springer-Verlag, 1997.

[Kur64] S. Y. Kuroda. Classes of languages and linear bounded automota. *Information and Control*, 7(2), 1964.

[Lan63] P. S. Landweber. Three theorems on phrase structure grammars of type 1. *Information and Control*, 6(2):131–136, 1963.

[Lin97] P. Linz. *An Introduction to Formal Languages and Automata*. Jones and Bartlett, 1997.

[LP98] H. R. Lewis and C. H. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall, 1998.

[Mar91] J. C. Martin. *Introduction to Languages and the Theory of Computation*. McGraw-Hill, 1991.

[Mea55] G. H. Mealy. A method for synthesizing sequential circuits. *Bell System Technical Journal*, 34(5):1045–1079, 1955.

[Men87] E. Mendelson. *Introduction to Mathematical Logic*. Wadsworth & Brooks/Cole, third edition, 1987.

[Min67] M. L. Minsky. *Computation: Finite and Infinite Machines*. Prentice Hall, 1967.

[Moo56] E. F. Moore. Gendanken experiments on sequential machines. In C. E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 129–153. Princeton University Press, 1956.

[Mor97] B. M. Moret. *The Theory of Computation*. Addison-Wesley, 1997.

[MP43] W. S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.

[MY60] R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers*, 9(1):39–47, 1960.

[Myh60] J. Myhill. Linear bounded automata. Technical Report Technical Note WADD 60-165, Wright Paterson AFB, Dayton, Ohio, 1960.

[Nau63] P. Naur. Revised report on the algorithmic language algol 60. *Communications of the Association for Computing Machinery*, 6(1):1–17, 1963.

[Nis99] N. Nissanke. *Introductory Logic and Sets for Computer Scientists*. Addison-Wesley, 1999.

[Oet61] A. G. Oettinger. Automatic syntactic analysis and the pushdown store. In *Proceedings of the Symposia on Applied Mathematics*, volume 12. American Mathematical Society, 1961.

[Ogd68] W. G. Ogden. A helpfull result for proving inherent ambiguity. *Mathematical Systems Theory*, 2(3):191–194, 1968.

[Pos43] E. Post. Formal reductions of the general combinatorial decision problem. *American Journal of Mathematics*, 65:197–215, 1943.

[Pos44] E. Post. Recursively enumerable sets of positive natural numbers and their decision problems. *Bulletin of the American Mathematical Society*, 50:284–316, 1944.

[Ric53] H. G. Rice. Classes of recursively enumerable sets and their decision problems. *Transactions of the American Mathematical Society*, 89:25–59, 1953.

[Ric56] H. G. Rice. On completely recursively enumerable classes and their key arrays. *Journal of Symbolic Logic*, 21:304–341, 1956.

[Rog67] H. Rogers. *The Theory of Recursive Functions and Effective Computability*. McGraw-Hill, 1967.

[Ros99] K. H. Rosen. *Discrete Mathematics and Its Applications*. McGraw-Hill, fourth edition, 1999.

[RS59] M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):115–125, 1959.

[Sch60] S. Scheinberg. Nore on the boolean properties of context-free languages. *Information and Control*, 3(4):372–375, 1960.

[Sch63] M. P. Schützenberger. On context-free languages and pushdown automata. *Information and Control*, 6(3):246–264, 1963.

[Sip97] M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Co., 1997.

[SL95] D. Shasha and C. Lazere. *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*. Copernicus, 1995.

[Sup57] P. Suppes. *Introduction to Logic*. Van Nostrand Co, 1957.

[SW82] D. Stanat and S. Weiss. A pumping lemma for regular languages. *SIGACT News*, 14:36–37, 1982.

[Tur36] A. M. Turing. On computable numbers with an application to the entscheidungsproblem. In *Proceedings of the London Mathematical Society*, volume 2:42, pages 230–265, 1936. A correction: 43, pp. 544-546.

[Vel94] D. J. Velleman. *How To Prove It: A Structured Approach*. Cambridge University Press, 1994.

[Wes96] D. B. West. *Introduction to Graph Theory*. Prentice Hall, 1996.

# Índice