

Sumário

1	Introdução	1
1.1	Algoritmos	5
1.2	Tese de Church-Turing	8
1.3	Teoremas de Gödel	10
1.4	Conclusões e leituras recomendadas	21
1.5	Exercícios	22
2	Noções Preliminares	23
2.1	Conjuntos	23
2.2	Funções	29
2.3	Relações	36
2.4	Equivalência e Congruência	41
2.5	Relações de Ordem	43
2.6	Indução Finita	45
2.7	Alfabetos e Linguagens	49
2.8	Objetos Finitos e Espaços	53
2.9	Conclusões e leituras recomendadas	56
2.10	Exercícios	57

3	Funções Recursivas	60
3.1	Funções Recursivas Primitivas	62
3.2	A Linguagem Básica	75
3.3	Funções Aritméticas	77
3.4	Manipulação de Tuplas	84
3.5	Funcionais Especiais	86
3.6	Processamento de Cadeias	90
3.7	Relação entre Alfabetos	100
3.8	Repetição	103
3.9	Funções Recursivas	106
3.10	Conclusões e leituras recomendadas	111
3.11	Exercícios	111
4	Lógica Simbólica	113
4.1	Lógica Proposicional	114
4.2	Forma Normal na Lógica Proposicional	116
4.3	Lógica de Primeira Ordem	122
4.4	Forma Normal Prenex	127
4.5	Forma de Normal de Skolen	130
4.6	Teorema de Herbrand	132
4.7	O Princípio da Resolução	139
4.8	Conclusões e leituras recomendadas	151
4.9	Exercícios	152
5	Sistemas Formais	161
5.1	Sistemas Formais	162

5.2	Sistemas de Produções de Post	167
5.3	Linguagens Formais e Gramáticas	170
5.4	Hierarquia de Chomsky	176
5.5	Gramáticas Sensíveis ao Contexto	179
5.6	Gramáticas Livres de Contexto	182
5.7	Gramáticas Regulares	185
5.8	Aritmética Rudimentar	196
5.9	Conclusões e leituras recomendadas	197
5.10	Exercícios	197
6	Autômatos e Linguagens	203
6.1	Autômato Finito Determinístico - DFA	204
6.2	Autômato Finito Não Determinístico - NFA	207
6.3	Conversão de NFA para DFA	212
6.4	Minimização de Autômatos	215
6.5	Conclusões e leituras recomendadas	228
6.6	Exercícios	228
7	Máquina de Turing	234
7.1	Máquina Clássica	234
7.2	Máquinas de Turing Combinadas	243
7.3	Variações da Máquina de Turing	247
7.4	Máquina Universal de Turing	253
7.5	Decidibilidade da Máquina de Turing	259
7.6	Reflexões de Fronteira	268
7.7	Conclusões e leituras recomendadas	271

7.8 Exercícios	272
Referências Bibliográficas	277

Lista de Figuras

1.1	Círculos concêntricos da Computação	2
1.2	Métodos efetivos para resolução de problemas	6
1.3	Esquema ilustrativo da Tese de Church-Turing	8
2.1	Diagrama de Venn de operações com conjuntos: (a) $A \cup B$; (b) $A \cap B$; (c) $A - B$; (d) \bar{A}	27
2.2	Função de A em B	30
2.3	Funções: (a) Sobrejetora; (b) Injetora.	31
2.4	Composição de f e g	31
2.5	Relações: (i) os planos coordenados e uma região representando uma relação; (ii) o grafo da relação binária $R = \{ \langle a, b \rangle, \langle a, c \rangle, \langle b, d \rangle, \langle a, d \rangle, \langle c, d \rangle, \langle d, e \rangle \}$	38
2.6	Composição de R e S , dada por $S \circ R$	39
2.7	Representação gráfica de uma relação	39
2.8	Partição em um conjunto A e f -congruência	42
2.9	Exemplos de ordem	44
3.1	Composição de funções	71
3.2	Hierarquia de funções computáveis.	108
4.1	Árvore semântica para $S = \{P(x, a), \neg Q(x) \vee P(f(x), b)\}$	135

4.2	Árvore semântica para $S = \{\neg P(x) \vee Q(x), P(f(y)), \neg Q(f(y))\}$: (a) cheia, (b) fechada.	136
4.3	Os ângulos interiores alternados formados pela diagonal de um trapezóide são iguais.	147
5.1	Derivação de Teoremas	167
5.2	Árvore de Derivação Sintática de uma sentença em Português	171
5.3	Autômato finito	179
5.4	Hierarquia de Chomsky	180
6.1	Autômato Finito ou Máquina de Estados Finita	204
6.2	Diagrama de estados do Exemplo 6.1	206
6.3	Autômato finito determinístico reconhecedor de strings que não contenham 3 b 's consecutivos	207
6.4	Exemplo de autômato finito não determinístico	208
6.5	Possíveis configurações para o autômato da Figura 6.4	209
6.6	Autômato reconhecedor de strings com o símbolo 1 na 3 ^a posição de trás para frente da string: (a) NFA; (b) DFA.	211
6.7	Autômato não determinístico reconhecedor de strings contendo 01 no final.	213
6.8	Autômato determinístico reconhecedor de strings contendo 01 no final: (a) representação com estados inatingíveis; (b) representação final.	214
6.9	Exemplo de minimização: (a) Autômato finito determinístico não minimi- zado; (b) DFA minimizado.	216
6.10	Exemplo de não equivalência entre estados.	217
6.11	Exemplo de particionamento.	218
6.12	Minimização de DFA: (a) autômato cíclico; (b) autômato minimizado.	222
6.13	Autômatos do Exercício 6.2	230

6.14	Autômatos do Exercício 6.13	231
6.15	Autômato do Exercício 6.14	232
6.16	Autômato do Exercício 6.15	232
6.17	Autômato do Exercício 6.16	233
7.1	Máquina de Turing representada através de um diagrama de estados: <i>shif-</i> <i>tadora</i>	239
7.2	Máquina de copiadora do Exemplo 7.1.	242
7.3	Máquina de fita <i>multitrack</i> : exemplo com duas trilhas.	250
7.4	Máquina de várias fitas.	253
7.5	Máquina Universal de Turing.	257
7.6	Hierarquia de Chomsky complementada.	260
7.7	Cenários de aceitação de uma Máquina de Turing.	261
7.8	Máquinas: (a) reconhecedora; (b) decisora.	265
7.9	Limites da computabilidade.	267
7.10	Limites da computabilidade e alguns problemas.	268

Lista de Tabelas

1.1	Sistemas de contagem curiosos.	3
4.1	Tabela verdade do \neg , \wedge , \vee , \rightarrow e \leftrightarrow	115
4.2	Tabela verdade de uma tautologia.	116
4.3	Tabela verdade de uma contradição.	116
4.4	Tabela verdade de uma equivalência lógica.	117
4.5	Regras de equivalência para normalização.	117
4.6	Demonstração de consequência lógica utilizando tabela verdade. . . .	120
4.7	Demonstração de consequência lógica utilizando tabela verdade es- tendida.	120
4.8	Demonstração de consequência lógica utilizando tabela verdade e con- tradição.	121
4.9	Regras de equivalência para normalização prenex.	128

Lista de Algoritmos

2.1	Determinação da cardinalidade finita de um conjunto	32
4.1	Transformação de fórmulas para a forma normal prenex	129
4.2	Algoritmo imediato para a implementação do Teorema de Herbrand . .	139
4.3	Unificação de expressões	144
5.1	Programa ilustrativo para construção de uma gramática livre de con- texto proposto no Exercício 5.10	200
6.1	Minimização de um DFA $M = \langle k, \Sigma, \delta, s, F \rangle$, segundo Hopcroft [1]. . .	219
7.1	Algoritmo de funcionamento da Máquina Universal de Turing.	258

Capítulo 1

Introdução

A Teoria da Computação tem fundamental importância na formação de Engenheiros de Computação. Ela não só proporciona o embasamento teórico necessário para um correto e amplo entendimento da Computação, como também proporciona o desenvolvimento do raciocínio lógico e formal, do pensamento indutivo e recursivo e da capacidade de abstração. Além disto, introduz os conceitos fundamentais que são desenvolvidos em outras áreas, tais como: linguagens formais, semântica formal, compiladores, orientação a objetos, algoritmos, complexidade computacional entre outros.

Os profissionais de áreas correlatas à Computação se posicionam em áreas de atuação de acordo com suas especialidades, tal como ilustrado na Figura 1.1. O círculo mais interno enfatiza as atividades relacionadas com o hardware e comumente está associado aos Engenheiros Eletrônicos.

No círculo de software básico encontram-se os Engenheiros de Computação. Nesta região agrupam-se as atividades relacionadas com sistemas operacionais tais como o acesso ao hardware e tarefas com alta abstração para o usuário, bem como aquelas relacionadas com tradutores, isto é, programas que aceitam outros programas escritos em uma linguagem e os reproduzem em uma outra linguagem. Os Engenheiros de Computação também atuam no círculo seguinte, o de software de suporte, juntamente com os Engenheiros de Software. Neste círculo predominam os sistemas gerenciadores de bancos de dados, as linguagens de quarta geração, e

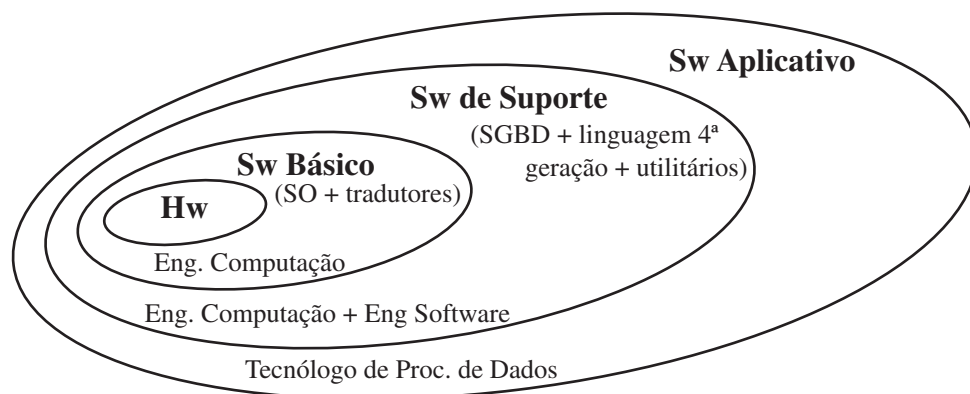


Figura 1.1: Círculos concêntricos da Computação.

os utilitários que implementam tarefas básicas para a utilização do sistema operacional, muitas vezes confundidos com o próprio sistema (geralmente distribuídos juntamente com o sistema operacional). Por fim, o círculo mais externo corresponde ao software aplicativo, o local de origem dos Tecnólogos de Processamento de Dados.

A Figura 1.1 apresenta didaticamente uma estratificação entre as áreas de atuação, entretanto, na prática estas fronteiras não são bem definidas. Dentre estas fronteiras, a única que recebe um nome é aquela entre o hardware e o software básico, chamada firmware. Os profissionais podem transpor livremente estas fronteiras, mas em geral, quanto mais eles se afastam de seu círculo de atuação, maior deverá ser o esforço para se inserir no novo meio.

Independente de suas áreas de atuação, os profissionais de todos os círculos concêntricos da Computação buscam uma solução para um problema utilizando sistemas interpretados ou compilados, que implementam um algoritmo. Apesar de termos como sistemas compilados, interpretados e algoritmo ainda não terem sido oportunamente definidos, tem-se que a busca pela solução para o problema passa, em geral, através da capacidade de representação numérica do ser humano.

O modo através do qual é feita esta representação varia de cultura para cultura, de acordo com as necessidades da sociedade. Culturas indígenas, como por exemplo a Tupi, contam apenas até 10 pois sua sociedade não demanda mais que isso da aritmética. Sob esta ótica, necessidade, cultura e aritmética se combinam

produzindo sistemas de contagem curiosos, tal como o apresentado na Tabela 1.1. Entretanto não são somente as sociedades marginais a nossa era que empregam sistemas de contagem pouco convencionais. Sociedades modernas, tal como a francesa, também podem utilizar sistemas de contagem pouco tradicionais (ver novamente Tabela 1.1).

Tabela 1.1: Sistemas de contagem curiosos.

Tupi			Francês		
Cardinal	Representação	Entendimento	Cardinal	Representação	Entendimento
1	oïepe		10	dix	
2	mokôï		20	vingt	
3	mosapÿr		30	trente	
4	irundyk		40	quarante	
5	xe pó	minha mão	50	cinquante	
6	xe pó oïepe	minha mão e um	60	soixante	
7	xe pó mokôï	minha mão e dois	70	soixante dix	sessenta dez
8	xe pó mosapÿr	minha mão e três	80	quatre vingts	quatro vintes
9	xe pó irundyk	minha mão e quatro	90	quatre vingts dix	quatro vintes dez
10	mokôï pó	duas mãos	100	cent	

A noção de número natural tem uma origem empírica e é uma das mais antigas criações do espírito humano, através da operação de contar. Certas gravuras feitas em cavernas, compostas de traços e pontos, parecem indicar sinais de numeração. Os progressos feitos na arte de contar, ou seja, no que se denomina Aritmética, é uma medida do progresso na vida econômica da humanidade. Segundo Monteiro[2], os indígenas Bakumu (talvez extintos) não sabiam contar além de 30 ou 40, pois seus contratos comerciais ou de matrimônio não ultrapassavam 30 ou 40 unidades. Além disto, certos costumes de registrar o resultado de contagem, que são provavelmente pré-históricos, ainda hoje são utilizados, mesmo por pessoas educadas. Por exemplo é usual se contar pontos em jogos usando-se figuras como a seguir para representar o resultado da contagem 1 a 5.



A evolução na representação dos números naturais desenvolveu, pouco a pouco, certas crenças de fundo místico, como na escola Pitagórica: “o número é a alma das coisas” ou “o número três representa a divindade”, etc. A partir do uso empírico da Aritmética os homens desenvolveram leis de cálculo sobre as representações inventadas. Há pelo menos 5.000 anos a humanidade sabe calcular com os números naturais. No Egito antigo e na Babilônia existiam calculadores profissionais, chamados **escribas** pelos egípcios e **logísticos** pelos gregos. Ifrah [3] apresenta detalhadamente a história dos números.

Depois dos egípcios e babilônios, foram os gregos que mais contribuíram para o desenvolvimento da Aritmética. Euclides iniciou uma ordenação sistemática dos conhecimentos sobre a Aritmética nos Elementos, onde já se pode observar passagens com demonstrações formais de certas regras de cálculo. A preponderância da Geometria, o apelo constante ao uso de figuras para representação geométrica e o desprezo pela “prática” paralizaram o desenvolvimento da Aritmética e geraram uma estagnação no desenvolvimento das técnicas de cálculo. O desenvolvimento de simbolismos é praticamente o único avanço que surge deste período até o Renascimento.

A história das notações aritméticas passou por períodos distintos. O primeiro é conhecido como período retórico em que os problemas da Aritmética eram resolvidos por uma seqüência de raciocínios expressos inteiramente por meio de discurso em linguagem natural (português, francês, etc). Neste sentido, o problema enunciado da seguinte forma “*Eu possuo vinte animais, sendo uma dúzia de ovelhas. Quantos camelos eu possuo?*”, seria resolvido através do discurso “*Supõe-se que ele possui ovelhas e camelos somente. Logo, a quantidade de camelos é resultante da contagem de todos animais, excluindo as ovelhas, isto é, oito camelos*”.

No período sincopado, durante a Idade Média, a Aritmética passou a adotar uma notação em que usavam-se abreviaturas para algumas operações e quantidades, como por exemplo, “*camelos = 20 - 1dz*”. Por fim, nasce a Aritmética simbólica. Pode-se dizer, que com Viète, em seu livro “Logística Speciosa”, propõe o uso de certas letras para os valores desconhecidos e outras para as constantes ou valores conhecidos. Este tipo de notação é empregada até os dias de hoje (“ $x = 20 - 12$ ”).

A criação de um sistema de notações adequadas ao cálculo, nasceu da necessidade de abreviar e simplificar a resolução de diversos problemas que surgiam na vida humana, e determinou um grande desenvolvimento da Aritmética. Apareceram então as regras fixas que permitem calcular com rapidez e segurança, poupando o espírito e a imaginação (Leibniz), e um dos resultados é a Mecanização do Cálculo.

1.1 Algoritmos

Descartes acreditava no emprego sistemático do cálculo algébrico como um método poderoso e universal para resolver *todos os problemas*. Esta crença, junta-se a de outros e surgem as primeiras idéias sobre máquinas universais, capazes de resolver todos os problemas. Esta era uma crença de mentes poderosas que deixaram obras respeitáveis em Matemática e ciências em geral.

A noção de computabilidade foi pela primeira vez formalizada por Gödel em 1931, através do desenvolvimento da teoria das funções recursivas empregando funções primitivas recursivas, definidas anteriormente por Dedekind em 1888 [4]. A recursividade, segundo Gödel [5] pode ser assim definida: as funções recursivas primitivas são precisamente aquelas funções aritméticas que podem ser derivadas do núcleo da recursividade por meio de um número finito de operações mecânicas específicas [6]. Esta forma de observar a questão tornou possível referenciar objetos infinitos a partir de regras finitas de construção.

Alonzo Church, em 1936, utilizou as funções recursivas que Gödel introduziu nos seus estudos sobre os teoremas da incompletude de modo a criar o conceito de algoritmo. O *lambda-calculus* foi materializado através de diversas linguagens funcionais de computador, considerando que seu reticulado de iterações não era mais difícil para um computador do que qualquer outra operação mecânica [7].

A preocupação constante em minimizar o esforço humano gerou o desenvolvimento de máquinas que passaram a substituir o homem na realização de tarefas, que são consideradas como atividades *inteligentes*, por exemplo: jogar xadrez, demonstrar teoremas, etc. Hoje poderíamos reservar o qualificativo *inteligente* apenas para atividades que parecem depender fortemente de fatores perceptivos. As tentativas

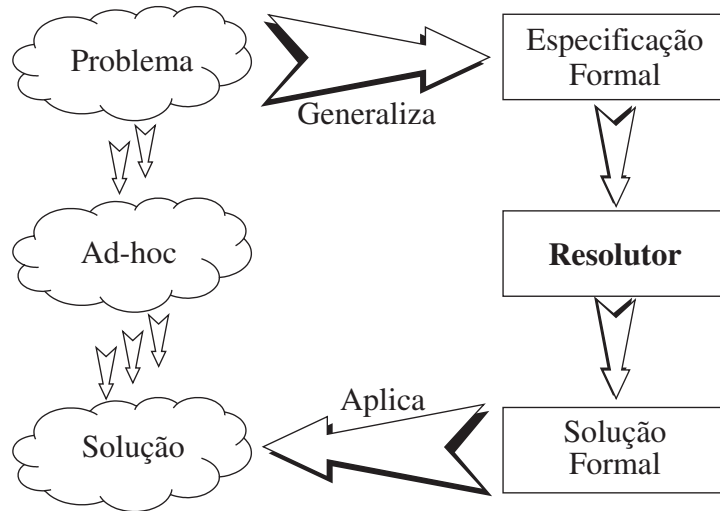


Figura 1.2: Métodos efetivos para resolução de problemas.

de mecanização de atividades desta natureza, como por exemplo a formação, identificação e manipulação de conceitos abstratos, tem apresentado resultados pouco satisfatórios. Na verdade não existem argumentos definitivos em tais questões. O que existe é uma atitude generalizada em não aceitar como *inteligente* nenhuma atividade passível de tratamento *sistemático*, ou executada pelo uso de *procedimentos efetivos*. A questão é, portanto, saber que atividades gozam desta prerrogativa, ou seja, que atividades são *computáveis*.

A teoria da computação é responsável pela formalização e explicação dos conceitos mencionados. Ela precedeu o surgimento dos computadores e seu desenvolvimento é absolutamente independente do estado corrente da tecnologia. Esta teoria baseia-se na definição e construção de máquinas abstratas e no estudo do poder destas máquinas na solução de problemas.

O homem tem, através dos tempos, tentado construir métodos efetivos para resolução de problemas. Os problemas podem ser resolvidos um a um, ou agrupados em classes de forma que, dada a solução de um dos elementos, os demais estarão solucionados. A segunda hipótese requer a concepção de métodos gerais a que chamamos *resolutores*.

A Figura 1.2 ilustra as abordagens direta, pelo uso de *métodos ad hoc*, ou seja

pela ausência de métodos e a indireta obtida pela utilização de *mecanismos* desenvolvidos com o objetivo de resolver classes cada vez mais abrangentes de problemas, aos quais damos o nome de *resolutores*.

Alguns destes resolutores foram delineados na antiguidade histórica, sob a forma de *algoritmos* e seu uso faz parte dos currículos escolares em todo o mundo. Aprendemos a resolver problemas aritméticos como adição, subtração, multiplicação, divisão e até mesmo a extração de raízes, sem nunca nos perguntarmos o *porquê* de tais *métodos*, com a confiança que somente as crianças possuem. Nos tornamos adultos e aprendemos outros *métodos* para problemas mais complicados e na maioria dos casos com a mesma crença no acerto dos mesmos. Muitas vezes os resolutores são aparelhos mecânicos como os ábacos, régua de cálculo ou máquinas de calcular ou aparelhos eletrônicos como calculadoras eletrônicas. O uso de tais aparelhos exige um aprendizado de uma linguagem ou pelo menos a identificação de botões ou manivelas e uma seqüência de operação, ou seja, de um algoritmo de uso. De um modo informal, podemos dizer que, o que chamamos de algoritmo, é nada mais que uma seqüência finita de instruções escritas ou faladas de alguma linguagem, normalmente uma língua natural, por exemplo o português.

Nos anos 20 e 30 houve um grande interesse na noção de algoritmo, no sentido de tornar esta noção intuitiva em um conceito preciso, para propiciar o estudo de suas propriedades de maneira rigorosa. Vários matemáticos - na Inglaterra (Alan Turing), nos Estados Unidos (Alonzo Church, Emile Post), na Áustria (Kurt Gödel) na União Soviética (Andrey Markov) - desenvolveram definições da noção de algoritmo através o desenvolvimento de certas máquinas abstratas e de linguagens artificiais. Todas as propostas feitas para definir *algoritmo*, até os dias atuais, foram mostradas equivalentes. O autor de uma dessas linguagens, o americano Alonzo Church evidenciou este fato, e isto o levou a formular o que passou a ser conhecida como a **Tese de Church**: "tais formalismos são caracterizações tão gerais da noção do efetivamente computável quanto consistentes com o entendimento intuitivo usual".

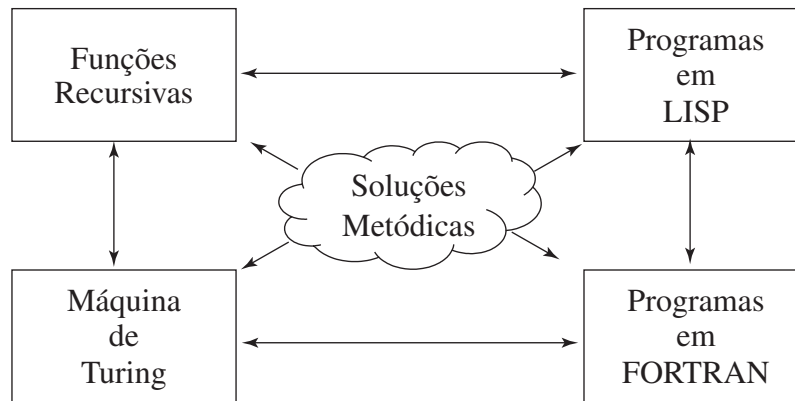


Figura 1.3: Esquema ilustrativo da Tese de Church-Turing.

1.2 Tese de Church-Turing

A Tese de Church, ou hipótese de Church-Turing, ilustrada na Figura 1.3, não é um teorema mas sim um resultado epistêmico cuja aceitação é quase universal. O conceito intuitivo de computável é identificado com uma certa classe de funções aritméticas chamadas funções recursivas, cuja caracterização é um dos objetivos deste livro.

O leitor pode pensar que se a hipótese não pode ter sua veracidade comprovada de forma direta, então talvez ela possa ser refutada. Assim, para negar a tese basta encontrar um procedimento que não pudesse demonstradamente ser computado por uma Máquina de Turing. Até o presente momento isto não ocorreu, e ainda, em virtude do grande número de dados experimentais favoráveis, esta tese tem sido aceita pelos estudiosos do assunto. Além disto, as diversas tentativas independentes de formalizar o conceito de algoritmo resultaram em formalismos que podem ser demonstrados como equivalentes ao de Turing.

Neste sentido, a Tese de Church é uma hipótese sobre a natureza mecânica do ato de calcular, relacionando-se diretamente com o computador, e com os tipos de algoritmos eles podem executar. Assim, toda função considerada sistematizável pode ser computada por uma Máquina de Turing. Programas podem ser traduzidos em uma Máquina de Turing, bem como, qualquer Máquina de Turing pode ser traduzida para uma linguagem de programação. Conseqüentemente, qualquer linguagem de

programação de propósito geral é suficiente para expressar qualquer algoritmo, seja ele qual for.

Até o presente, ninguém foi capaz de produzir um modelo computacional que tenha maior poder que o de Turing. Trata-se de um modelo mais geral que os computadores atuais, pois não é limitado por questões restritivas como espaço de armazenamento e memória.

Uma consequência positiva da Tese de Church é que se o ser humano consegue resolver um determinado problema, de forma consciente, sem adivinhação ou mímica, então é possível construir uma máquina que também o resolva. Uma aplicação prática disto é o campo de estudo conhecido como Inteligência Artificial ¹. O lado negativo é que os problemas que não podem ser resolvidos pela Máquina de Turing, também não serão resolvidos exclusivamente pelo raciocínio humano. Desta forma, existem problemas que possuem soluções, mas que nunca serão encontradas através do raciocínio humano.

Gödel trabalhou muito tempo com questões filosóficas relativas ao contraste entre mentes e máquinas [9]. Para ele a mente humana é incapaz de mecanizar todas as intuições matemáticas, fato este que poderia ser chamado de incompletude da Matemática, intimamente relacionado com o problema da incompletude. Outro resultado afirma que, ou a mente humana consegue ultrapassar qualquer máquina ou, então, existem questões da teoria dos números que são indecidíveis para a mente humana. Entretanto, Gödel, assim como Hilbert, se recusava a admitir o comportamento absolutamente irracional, de acordo com o qual a mente seria capaz de formular questões que, pela sua própria natureza, lhe seja impossível responder.

¹O termo Inteligência Artificial é um nome brilhantemente criado por McCarthy em 1956 para chamar a atenção de investidores para aquela área de conhecimento. McCarthy disse que tratava-se da capacidade de uma máquina de realizar funções que se fossem realizadas pelo ser humano seriam consideradas inteligentes. Ora, se a máquina consegue desempenhar essa tarefa, é porque esta mesma é sistemática, e por definição nunca pode ser dita inteligente. Contudo, apesar deste cheque-mate à respeitabilidade [8] o termo é um bom nome para vender a idéias para o público leigo.

1.3 Teoremas de Gödel

Em 1931, Gödel² tornou públicos seus teoremas relacionados com a indecidibilidade e incompletude da matemática. Estes teoremas se contrapunham ao otimismo científico dominante da época que acreditava poder resumir toda a matemática, e até mesmo outras ciências naturais, em um único sistema formal livre de contradições. Seus teoremas delinearam novos contornos ao domínio da matemática e influenciam não só as ciências correlatas, como a computação, mas também outros campos do conhecimento humano, como a filosofia.

No fim do século XIX e início do século XX a matemática era profundamente influenciada pelas idéias do matemático alemão David Hilbert (1862-1943) pertencente a uma escola chamada formalista que atuou ao longo de todo o século XX. Em 1900, durante o Congresso Internacional de Matemáticos, realizado em Paris, ele propôs uma coleção de vinte e três problemas inconclusos que direcionaram grande parte da pesquisa matemática no decorrer do século XX, alguns destes problemas ainda hoje não são resolvidos, e outros o foram apenas parcialmente. Em 1920, ele sugeriu explicitamente uma linha de pesquisa na qual se defendia que os desenvolvimentos destes problemas aconteceriam a partir de uma amostra finita de proposições, devidamente selecionadas, chamadas axiomas. Além disto, tal sistema axiomático provavelmente seria consistente, ou seja, livre de contradições. A partir destes axiomas, os formalistas, seguidores de Hilbert, acreditavam ser possível desenvolver uma teoria única que contemplasse toda a matemática de forma a torná-la completa.

Ora, a matemática é amplamente utilizada para formalizar a modelagem de

²Kurt Gödel (1906-1978), conhecido como senhor “Por que?”, nasceu em Brno, atualmente República Tcheca. Ainda na infância sofre um ataque de febre reumática, o que lhe debilitou físico e psicologicamente tornando-o hipocondríaco durante toda a vida. Em 1939, foi incluído na lista negra nazista talvez por ser intelectual, talvez devido ao seu círculo de amizades com judeus, talvez por fazer parte do Círculo de Viena, e assim sendo fugiu para a Universidade de Princeton nos Estados Unidos da América. Não era uma pessoa muito sociável e cultivou poucas relações de amizade, entre elas a de Albert Einstein com quem trabalhou na Teoria da Relatividade. Com o passar dos anos, sua hipocondria foi piorando, até que morreu de fome no hospital de Princeton, ocasião em que se convenceu de que o estavam envenenando pela alimentação.

fenômenos da natureza. Sob esta ótica, o otimismo científico apontava para a possibilidade de modelagem de todos os problemas, inclusive daqueles relativos às ciências naturais. Além disto, Hilbert defendia também, que um problema matemático deveria admitir obrigatoriamente uma solução exata, seja ela de forma direta ou através da demonstração de sua impossibilidade de resolução. Esta característica de singularidade de uma solução passou a ser conhecida como decidibilidade de um problema. Quando se afirma que uma proposição é verdadeira ou falsa, está-se enunciando um dos princípios da lógica clássica (lógica de predicados de primeira ordem). Trata-se do princípio do terceiro excluído (*tertium non datur*) pois tal proposição não é ao mesmo tempo verdadeira e falsa, tampouco nem verdadeira, nem falsa. Informalmente tem-se: “O que é, é, o que não é, não é, e não há uma terceira opção”.

Entretanto, a decidibilidade se contrapõe à existência de paradoxos que são facilmente criados, como por exemplo, a seguinte sentença: “*Esta afirmação é falsa*”. É fácil perceber que os seguidores de Hilbert claramente enfrentaram dificuldades para construir uma matemática livre de contradições, decidível, completa e consistente. Neste sentido, torna-se importante compreender algumas definições de termos comuns a este assunto.

Definição 1.1 Um conjunto de enunciados verdadeiros, selecionados para serem axiomas de um sistema qualquer, é dito *completo* se é possível obter novamente, através de demonstrações, todos os enunciados verdadeiros do objeto de estudo que se propõe axiomatizar.

Definição 1.2 Um sistema axiomático é *consistente* se não é possível provar, a partir dos axiomas, uma contradição, isto é, um enunciado e sua negação.

Definição 1.3 Uma classe de enunciados que não podem ser demonstrados nem refutados dentro de um sistema axiomático é chamada de *indecidíveis*. Se em um sistema axiomático existe algum problema indecidível, então este sistema é incompleto.

Uma etapa importante da axiomatização da matemática proposta por Hilbert foi realizada pelo matemático italiano Giuseppe Piano (1858-1932), em 1889.

Apesar de outros matemáticos terem apresentado anteriormente³ tais sistemas de axiomatização foi Peano que conseguiu fazê-lo através de uma meta-linguagem de primeira ordem, ou seja, através de uma linguagem onde aparecem apenas predicados e proposições aplicados a objetos. Um exemplo é a definição de identidade feita através de duas propriedades de relacionamento:

- $a = a$ (reflexão); $(a = b) \rightarrow (b = a)$ (simetria); $(a = b \wedge b = c) \rightarrow a = c$ (transitividade)
- $a = b \rightarrow \varphi(a) = \varphi(b)$ (se a igual a b , b possuirá a mesma propriedade de a)

A principal característica da aritmética de Peano é a definição de sucessor. Através dela, os seguintes axiomas apresentam as propriedades fundamentais dos números naturais \mathbb{N} .

1. $\neg \underline{suc}(x) = 0$: O 0 não é sucessor de nenhum número natural;
2. $\underline{suc}(x) = \underline{suc}(y) \rightarrow x = y$: Se dois números naturais x e y têm o mesmo sucessor então eles são iguais;
3. $[p(0) \wedge (\forall x)(p(x) \rightarrow p(\underline{suc}(x)))] \rightarrow (\forall x_1), p(x_1)$: Trata-se do princípio da indução matemática (que será explorado mais profundamente na seção 2.6), isto é, se a propriedade é válida para o elemento inicial e a sentença “*se a propriedade p é verdadeira para um número x , então ela também é verdadeira para o sucessor*”.

A fim de construir um sistema formal que demonstra a existência de teoremas indecidíveis, Gödel criou uma linguagem estritamente numérica⁴ associando os sinais primitivos da aritmética de Peano a números naturais primos.

³Em 1888, Dedekind apresentou uma axiomatização similar à apresentada por Peano um ano depois. Entretanto, Ao contrário de Peano, Dedekind usa uma linguagem de segunda ordem, onde expressões fazem referência a outras expressões e não apenas objetos.

⁴O processo consiste em traduzir o enunciado da meta-linguagem de Peano para a linguagem-objeto da aritmética e por isso é notadamente denominado aritmetização [10]

O problema a seguir, conhecido como quebra-cabeça godeliano, proveniente de Raymond Smullyan [11], expressa a idéia geral do teorema de Gödel. Suponha a existência de uma máquina que imprime setenças, ou cadeias, compostas pelos símbolos

$$\neg, P, N$$

As sentenças que a máquina é capaz de imprimir estão em uma das quatro formas a seguir:

- PX
- PNX
- $\neg PX$
- $\neg PNX$

onde X é uma cadeia qualquer.

A máquina que questão segue as seguintes regras:

1. PX é verdadeiro \Leftrightarrow a máquina imprime X ;
2. PNX é verdadeiro \Leftrightarrow a máquina imprime XX ;
3. $\neg PX$ é verdadeiro \Leftrightarrow a máquina não imprime X ;
4. $\neg PNX$ é verdadeiro \Leftrightarrow a máquina não imprime XX ;
5. $\neg X$ é a negação de X . Para todas as cadeias X , ou elas são verdadeiras, ou sua negação é verdadeira;
6. Todas as sentenças impressas pela máquina são verdadeiras;
7. A máquina nunca imprime sentenças falsas.

Este tipo de construção é um caso de auto-referência: *o programa imprime sentenças que definem o que o programa pode e não pode imprimir*, portanto o programa descreve seu próprio comportamento. Seja um program deste tipo, que

imprime apenas sentenças verdadeiras sobre si mesmo. É possível que o programa seja capaz de imprimir todas as sentenças verdadeiras?

Considere a combinação $\neg PN\neg PN$, e suponha que a máquina imprime esta cadeia. Utilizando a regra 2, e assumindo $X = \neg PN$, tem-se que é verdadeira a sentença $\neg PN\neg PN \equiv \neg PNX \Leftrightarrow \neg XX$. Desfazendo a substituição, obtém-se $\neg XX \equiv \neg\neg PN\neg PN \equiv PN\neg PN$ é verdadeiro porque a hipótese era que a máquina imprime a sentença $\neg PN\neg PN$. Além disto, pela regra 5, a negação de $PN\neg PN$, isto é, $\neg PN\neg PN$ é falsa. Ora, pela regra 7, a máquina nunca imprime sentenças falsas, então ela não deveria imprimir $\neg PN\neg PN$. Assim, a suposição de que a máquina imprime $\neg PN\neg PN$ está incorreta.

Desta forma, supõem-se então que a máquina não imprime $\neg PN\neg PN$. Entretanto, se essa impressão não ocorre, e tomando a regra 4, tem-se que $\neg PNX \equiv \text{verdadeiro}$. Por substituição $X = \neg PN$, tem-se $\neg PN\neg PN \equiv \text{verdadeiro}$ também. Assim, $\neg PN\neg PN$ é verdadeira, mas a máquina não imprime a sentença. Assim, a suposição de que a máquina não imprime $\neg PN\neg PN$ está incorreta. Logo, tem-se duas hipóteses:

- a sentença é imprimível, mas ela é falsa, o que configura uma contradição, pois o programa só imprime sentenças verdadeiras;
- a sentença é verdadeira, mas ela não imprimível, o que fere o propósito da máquina.

A analogia entre este enigma e o teorema da incompletude está no paralelo entre um programa que imprime cadeias verdadeira e um sistema formal que gera teoremas verdadeiros.

O artigo original de Gödel pode ser encontrado em [12]. Na primeira seção daquele artigo, o esboço da demonstração indica claramente a conseqüência final do teorema e o caminho a ser tomado para chegar a ela. No entanto, são necessárias dezenas de definições e teoremas para completar rigorosamente a prova. À luz da Tese de Church, pode-se dispensar o arcabouço formal e apresentar apenas a genialidade da idéia global.

O princípio fundamental é o de auto-referência: Gödel parte do sistema formal para a aritmética, definido em Principia Mathematica (PM) [13], e “reescreve-o” utilizando a linguagem dos números naturais. Este processo de *aritmética* permite que os conceitos e as propriedades daquele sistema formal sejam transformados em conceitos e propriedades sobre os números naturais.

Em suma, a aritmética de Gödel é um mapeamento entre os elementos do sistema formal e os números naturais. A seguir, apresenta-se uma pequena variação do mapeamento original utilizado por Gödel, encontrado em Nagel and Newman [14]. Inicialmente, tem-se os símbolos do sistema formal, na tabela a seguir. A correspondência é feita com os 7 primeiros números ímpares.

símbolo	significado	número
\neg	negação	1
\rightarrow	se-então	3
\exists	existe	5
0	zero	7
<u>suc</u>	função sucessor	9
(abre parêntese - separador	11
)	fecha parêntese - separador	13

Considera-se também a existência de 3 tipos de variáveis: (1) x_1, x_2, \dots que podem ser substituídas por números; (2) $\alpha_1, \alpha_2, \dots$ que podem ser substituídas por fórmulas; e (3) P_1, P_2, \dots que podem ser substituídas por predicados. A numeração das variáveis é como mostra a tabela a seguir.

variável	número	exemplo
x_i	i-ésimo primo > 13	17, 19, 23, 29, ...
α_i	i-ésimo primo > 13 ao quadrado	$17^2, 19^2, 23^2, 29^2, \dots$
P_i	i-ésimo primo > 13 ao cubo	$17^3, 19^3, 23^3, 29^3, \dots$

O número de Gödel de uma expressão de k símbolos é o produto dos k

primeiros primos elevados aos números de Gödel de seus constituintes. Por exemplo:

$$\begin{array}{ccc} \neg & & \alpha_1 \\ \downarrow & & \downarrow \\ 1 & & 17^2 \\ 2^1 & \times & 3^{17^2} \end{array}$$

ou ainda, o exemplo:

$$\begin{array}{ccccccc} P_1 & (& \underline{suc} & (& x_1 &) &) \\ \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \downarrow & \\ 17^3 & 11 & 9 & 11 & 17 & 13 & 13 \\ 2^{17^3} & \times & 3^{11} & \times & 5^9 & \times & 7^{11} & \times & 11^{17} & \times & 13^{13} & \times & 17^{13} \end{array}$$

O número de Gödel de uma sequência de k expressões é o produto dos k primeiros primos elevados aos números de Gödel de cada expressão. Por exemplo, se $n = 2^1 \times 3^{17^2}$ e $m = 2^{17^3} \times 3^{11} \times 5^9 \times 7^{11} \times 11^{17} \times 13^{13} \times 17^{13}$ então o número de Gödel da sequência $\neg\alpha_1, P_1(s(x_1))$ é $2^n \times 3^m$.

A aritmetização das seqüências de fórmulas pode ser aplicada a qualquer seqüência finita de fórmulas, especialmente nas demonstrações axiomáticas, cujas definições são obtidas por indução. Embora Gödel objetivasse demonstrar a relação de consistência através da aritmética de Peano, Guerreiro [10] ressalta que tal método permite que qualquer linguagem seja reproduzida de maneira semelhante, especialmente as linguagens de programação.

Além do fato que um número de Gödel representar apenas um símbolo ou fórmula, outra característica importante desse método é o fato de que qualquer número de Gödel pode ser revertido novamente na fórmula ou símbolo original, bastando para isso decompor esse número em números primos.

Como será observado a seguir, a tabela de substituição proposta por Gödel em [15] é suficiente para tradução de axiomas da aritmética de Peano. Mais tarde, Stephen Cole Kleene, um dos alunos de Gödel em seu curso apresentado na Universidade de Princeton em 1934, desenvolveria a teoria das funções recursivas (objeto de estudo no Capítulo 3) propondo um conjunto de funções iniciais e um conjunto de construtores de funções que podem ser encadeados, traduzindo assim qualquer função computável [16].

A primeira vista, pode parecer que a associação é insuficiente para um algoritmo completo ou mesmo para traduzir os axiomas da aritmética de Peano. Até mesmo por isso, foi acrescentada a associação com primos maiores que 13 para obter os exemplos já expostos. Gödel sobrepõe estas dificuldades através do conceito de funções recursivas primitivas (ver Capítulo 3) onde, uma vez definidas funções muito simples e estabelecendo algumas regras, é possível derivar qualquer função que possua um número finito de etapas.

Kleene amplia esse resultado, afirmando que uma função (ou programa) é computável se puder ser obtida a partir de algumas funções básicas e da definição de alguns construtores, ou seja, ele define o que se pode processar por um computador. Embora esse resultado seja eminentemente teórico [16], o trabalho necessário para verificar a computabilidade de funções é muito simples.

Como estes modelos de computação serão estudados nos próximos capítulos, o foco no momento será o processo de Gödel. Para reconhecer uma expressão, dado seu número de Gödel n , basta observar certas regras, como:

1. se $n \leq 13$ é ímpar, a expressão é um símbolo;
2. se $n > 13$ é ímpar e é potência de um número primo, a expressão é uma variável;
3. se n é par, decompondo-se n em fatores primos:
 - (a) se for o produto de primos sucessivos com expoentes ímpares, então potencialmente é uma fórmula;
 - (b) se for o produto de primos sucessivos com expoentes pares, então potencialmente é uma seqüência de mais de uma fórmula.

Outras condições devem ser observadas para que um natural seja número de Gödel para uma fórmula. Por exemplo, $30 = 2^1 \times 3^1 \times 5^1$ corresponde a $\neg\neg\neg$, que não é uma fórmula no sistema formal pretendido. De qualquer modo, o leitor pode imaginar como construir um programa que enumere/reconheça fórmulas de um sistema formal com os símbolos apresentados, segundo esta aritmetização.

Para aritmetizar o conceito de derivável por Modus Ponens, observe o exemplo:

fórmula	número
$P_2(x_1) \rightarrow P_1(x_1)$	$\frac{2^{19^3} \times 3^{11} \times 5^{17} \times 7^{13} \times 11^3 \times 13^{17^3} \times 17^{11} \times 19^{19} \times 23^{13}}{2^{19^3} \times 3^{11} \times 5^{17} \times 7^{13}}$
$P_2(x_1)$	$2^{19^3} \times 3^{11} \times 5^{17} \times 7^{13}$
$P_1(x_1)$	$2^{17^3} \times 3^{11} \times 5^{19} \times 7^{13}$

O número da segunda premissa é o produto dos 4 fatores primos iniciais da primeira premissa. Os expoentes da conclusão são os mesmos dos fatores primos da primeira premissa, após o fator 11^3 correspondente ao símbolo \rightarrow . Novamente, o leitor deve imaginar como construir um programa que aplique Modus Ponens sobre dois números de Gödel. Este programa computa a função característica da relação *DerivávelPorMP*(x, y, z).

Para aritmetizar a regra de substituição de uma variável por um termo, observe o exemplo:

fórmula	número
$P_1(x_1)$	$2^{17^3} \times 3^{11} \times 5^{17} \times 7^{13}$
$\underline{suc}(0)$	$2^9 \times 3^{11} \times 5^7 \times 7^{13}$
$P_1(\underline{suc}(0))$	$2^{17^3} \times 3^{11} \times 5^9 \times 7^{11} \times 11^7 \times 13^{13} \times 17^{13}$

Há, de fato, uma substituição do fator 5^{17} correspondente ao x_1 , com a inserção dos expoentes do número de $\underline{suc}(0)$ aplicados aos primos a partir do 5, e os expoentes dos fatores à direita do 5^{17} foram deslocados para primos após o 13. O programa assim descrito computa a função característica da relação *DerivávelPorSUB*(x, y).

Extrapolando estas observações, Gödel criou predicados aritméticos para conceitos tais como *SerFórmula*, *SerAxioma*, *SerTeorema*, todos eles conjuntos recursivos. Para chegar ao ponto crucial do teorema, foram utilizados, especialmente:

1. a função $sub(y, 17, y)$, que calcula o número de Gödel da fórmula obtida a partir da fórmula de número y ($P_1(x_1)$), pela substituição da variável x_1 por

y novamente. Por exemplo:

$$\begin{array}{ccc} \text{sub}(2^{17^3} \times 3^{11} \times 5^{17} \times 7^{13} & , & 17 & , & 2^{17^3} \times 3^{11} \times 5^{19} \times 7^{13} &) \\ \downarrow & & \downarrow & & \downarrow & \\ P_1(x_1) & & x_1 & & 2^{17^3} \times 3^{11} \times 5^{17} \times 7^{13} & \end{array}$$

é o número de Gödel de $P_1(2^{17^3} \times 3^{11} \times 5^{17} \times 7^{13})$, isto é, $P_1(P_1(x_1))$.

2. o predicado $\acute{E}ProvaPara(y, z)$ que representa que a seqüência de fórmulas de número de Gödel y é uma dedução para a fórmula de número de Gödel z .

Considere então a fórmula:

$$\neg \exists x_2 \acute{E}ProvaPara(x_2, x_3) \cdots (G_0)$$

que diz, essencialmente, “não existe uma dedução para a fórmula de número de Gödel x_3 ”. Substituindo x_3 por $\text{sub}(x_1, 17, x_1)$ temos

$$\neg \exists x_2 \acute{E}ProvaPara(x_2, \text{sub}(x_1, 17, x_1)) \cdots (G_1)$$

que diz, essencialmente, “não existe uma dedução para a fórmula de número de Gödel $\text{sub}(x_1, 17, x_1)$ ”. Em seguida, seja n o número de Gödel de (G_1) . Portanto, o valor de $\text{sub}(n, 17, n)$ é o número de Gödel de

$$\neg \exists x_2 \acute{E}ProvaPara(x_2, \text{sub}(n, 17, n)) \cdots (G)$$

que diz, essencialmente, “não existe uma dedução para (G) ”.

Analisando o conteúdo da fórmula (G) , temos duas possibilidades:

1. se existe uma dedução no sistema formal da aritmética para (G) , então é possível deduzir uma contradição dentro do sistema;
2. se não existe uma dedução para (G) então (G) é verdadeira, porém não é possível deduzir todas as verdades aritméticas dentro do sistema, já que (G) não é dedutível.

Diante desta situação, Gödel se viu numa situação delicada em que era necessário fazer uma escolha: (1) ou que existem enunciados verdadeiros que não podem ser demonstrados; (2) ou que existem demonstrações podem provar a afirmação

e a negação de um mesmo enunciado. Sob esta ótica, Gödel foi obrigado a postular os Teoremas da Incompletude e da Consistência, descritos a seguir.

Teorema 1.1 [Teorema da Incompletude] Todo sistema axiomático consistente e recursivo para a aritmética possui enunciados indecidíveis. Em particular, se os axiomas do sistema são enunciados verdadeiros, pode existir um enunciado verdadeiro que não é demonstrável dentro deste sistema.

Teorema 1.2 [Teorema da Consistência] Um enunciado que expressa a consistência de um sistema axiomático recursivo para a aritmética não é demonstrável dentro deste sistema.

Após este trabalho, vários matemáticos e filósofos avançaram sobre os teoremas ampliando suas implicações para matemática (Alfred Tarski, 1902-1983; Paul Cohen, 1934-2007; John Rosse, 1907-1989), para a informática (Johann Von Neumann, 1903-1952; Alonzo Church, 1903-1995; Alan Turing, 1912-1954; Joseph Bernard Kruskal, 1928-2010) e para a filosofia (Hao Wang, 1921-1995), tais como:

- Nenhum programa de computador pode demonstrar todas as proposições verdadeiras da matemática. Uma vez que a recursividade corresponde a um dos paradigmas da Computabilidade (Tese de Church-Turing) é natural que a Ciência da Computação assuma que existem enunciados que não podem ser provados ou refutados, e ainda, que o princípio de bivalência do valor verdade deve ser rejeitado;
- Nenhuma programa de computador pode ser, ao mesmo tempo, não contraditório e completo;
- Não existe programa capaz de verificar, de antemão, a presença de laços infinitos em um programa, isto é, não existe um algoritmo que pode ser aplicado a qualquer programa arbitrário, com uma entrada, para decidir se este programa para ou não com a dada entrada.

A imediata recepção pelos matemáticos da época à apresentação dos teore-

mas de Gödel ⁵ foi cautelosa. Essa cautela se justificava pela divergência não só aos resultados matemáticos apresentados na mesma época, mas também ao direcionamento “hilbertiano” que os trabalhos tomavam em geral. Um claro exemplo pode ser encontrado na citação que Hilbert realizou apenas dois dias depois no mesmo congresso (ainda ignorante aos resultados alcançados por Gödel): “Para o matemático, não existe *ignorabinus* (ignoraremos) e, na minha opinião, o mesmo vale para as ciências naturais. A verdadeira razão porque ainda não conseguimos encontrar um problema insolúvel reside, segundo creio, no fato deles não existirem. (...)”, conforme Guerreiro [10].

Essa multiplicidade de implicações que torna o trabalho de Gödel atrativo a matemáticos e leigos. Como se pode verificar em Kubrusly [17] e Guerreiro [10] seus teoremas inspiram as mais diversas conclusões, enveredando até a teologia, área abordada matematicamente por Gödel no fim de sua vida. Contudo, é a “derrocada da pretensa segurança” [10] da matemática o resultado geral que mais se sobressai para ambos os grupos. O desenvolvimento da matemática e de ciências correlatas como a informática, traduzem, de certa forma, o desenvolvimento do próprio pensamento humano e, por isso, da própria humanidade. As demonstrações de Gödel em 1931 marcaram um momento de inflexão da matemática moderna, indo mais longe que mera demonstração da falibilidade da disciplina: Gödel expande o domínio da matemática promovendo uma nova abordagem através de sua lógica e sua repercussão é claramente visível nos dias de hoje através da informática.

1.4 Conclusões e leituras recomendadas

Sem dúvida, o mais conhecido e respeitado dos especialistas em Gödel é Smullyan, cuja bibliografia sobre o tema de indecidibilidade é bastante extensa. Entre seus livros encontram-se textos extremamente profundos como [11] e [18]; e livros lúdicos, que ilustram o conceito através de enigmas e charadas, tais como [19], [20]. Goldstein [21] também oferece um material didático e profundo sobre a

⁵Congresso sobre teoria do conhecimento nas ciências exatas (1931) realizado na cidade bohemia de Königsberg, hoje Kaliningrado, anexada a Rússia após a II Guerra Mundial.

prova de Gödel. O texto de Hofstadter [22] contém discussões sobre auto-referência, comparando os trabalhos de Gödel na matemática, Escher nas artes plásticas e Bach na música.

Especulações sobre o significado e as consequências do Teorema da Incompletude, tanto em matemática como em filosofia são abundantes. Algumas destas consequências são aceitas com restrições sérias, por parte da comunidade científica, e o próprio Gödel em trabalhos que só foram publicados após sua morte em Janeiro de 1978 [23], discute exaustivamente questões que variam, desde o conceito de inteligência até a possibilidade (na qual ele acreditava) da existência de Deus. Dentre as obras que abordam o legado lógico-filosófico de Gödel recomenda-se fortemente a coleção [24]. O texto *Reflections on Kurt Gödel* [25] contém uma excelente biografia comentada pelo grande lógico Hao Wang.

1.5 Exercícios

1.1 Explique o que se entende por atividade inteligente e atividade computável.

1.2 Qual é a proposta da Tese de Church-Turing?

1.3 Calcule o número de Gödel associado à expressão $P_2(x_1) \rightarrow \alpha_1$.

1.4 Determine a expressão associado ao número de Gödel 37.968.750.

Capítulo 2

Noções Preliminares

Neste capítulo serão introduzidas algumas noções matemáticas básicas para o entendimento dos demais capítulos. A apresentação do material é feita de modo informal, e o leitor deve procurar na literatura citada nas referências esclarecimentos maiores, quando isto se tornar necessário.

2.1 Conjuntos

As idéias da Matemática, em geral, envolvem um processo evolucionista no qual vários estudiosos trabalham paralelamente. Entretanto, a Teoria dos Conjuntos [26, 27] não compartilha esta característica com as demais áreas da Matemática, pois ela foi criada pelo matemático russo-alemão chamado George Cantor¹. Em 1872, Cantor conheceu o matemático Richard Dedekind, cujo pensamento abstrato e lógico influenciou as idéias desenvolvidas por Cantor, e que mais tarde iriam al-

¹George Ferdinand Ludwig Philipp Cantor nasceu em 03 de março de 1845 e morreu em 06 de janeiro de 1918. Nascido na Rússia, mudou-se para Alemanha com dez anos de idade. Foi responsável por fazer a distinção entre conjuntos enumeráveis e não enumeráveis. Provou que o conjunto dos números racionais \mathbb{Q} é enumerável e que o conjunto dos números reais \mathbb{R} é não enumerável, usando o célebre argumento da diagonal de Cantor. Foi o primeiro a utilizar o símbolo \mathbb{R} como representação dos conjuntos reais. Durante boa parte de sua vida sofreu ataques de depressão que o conduziram a morte em um hospital psiquiátrico. A descoberta do Paradoxo de Russell contribuiu de forma significativa para a sua falência nervosa.

terar o que hoje se conhece com Matemática Moderna. Em um artigo de 1874 no Crelle's Journal, ele demonstrou que os números reais têm uma correspondência um para um com os números naturais, enquanto que para estes últimos não existe tal correspondência. Sob esta ótica, o ato de contar é um aspecto comum entre a Teoria dos Conjuntos e a Computação, e por este motivo, torna-se desejável rever alguns conceitos iniciais sobre a mesma.

Um conjunto é uma coleção de objetos distintos, usualmente caracterizado por enumeração ou por uma propriedade que seus elementos *possuem, gozam, ou satisfazem*. Conjuntos são denotados por letras maiúsculas latinas ou gregas, com ou sem índices, assim A , Γ e B_1 podem ser usados como nomes de conjuntos. Os objetos que constituem um conjunto A são membros ou elementos do conjunto.

Com respeito à notação, deve-se estabelecer que:

1. se a é um elemento de A , diz-se que a pertence a A ou $a \in A$; abreviadamente, se vários elementos, digamos, a, b e c são elementos de um conjunto A , pode-se dizer $a, b, c \in A$;
2. se um conjunto A tem como únicos elementos a, b e c , podemos representá-lo por $A = \{a, b, c\}$;
3. uma propriedade que certos elementos gozam é representada por $P(x)$. Se forem considerados valores para x , por exemplo 1, representado por $P(1)$, le-se “ P de 1”. Assim o conjunto cujos elementos satisfazem uma propriedade P é representado por $A = \{x \in B | P(x)\}$. Conjuntos especificados desta forma são lidos como: “ A é o conjunto dos elementos x de B tal que $P(x)$ ”;
4. se os elementos de um conjunto são indexados, por exemplo pelos números naturais de 1 a n , usamos a notação $A = \{a_1, a_2, \dots, a_n\}$; se o conjunto de índices de A for um outro conjunto, digamos I , cujos elementos não foram especificados, dizemos $A = \{a_i\}_{i \in I}$.

Existem conjuntos que possuem um grande número de elementos, por exemplo o conjunto de todos os jornais de todas as cidades do mundo que foram publica-

dos desde o início da imprensa até o dia de hoje. Note que embora não saibamos o número de elementos, estes estão bem especificados por uma propriedade definidora.

Exemplo 2.1

pares $\{x \in \mathbb{N} \mid x \text{ é um número natural e } x \text{ é divisível } 2\}$

ímpares $\{x \in \mathbb{N} \mid x \text{ é um número natural e } x \text{ não é divisível } 2\}$

Um conjunto que não tem elementos é dito ser vazio, e é denotado por \emptyset . Conjuntos vazios em geral surgem quando é especificado por uma propriedade que não é satisfeita por nenhum elemento. Por exemplo $A = \{x \mid x \neq x\}$.

Definição 2.1

1. Diz-se que A é um subconjunto de B se e somente se para todo x , se $x \in A$ então $x \in B$ e denotamos por $A \subseteq B$; neste caso diz-se também que B é um superconjunto de A , denotado por $B \supseteq A$. Formalmente tem-se² $A \subseteq B \Leftrightarrow \text{para todo } x, \text{ se } x \in A \text{ então } x \in B$.
2. Diz-se que A é igual a B , o que é denotado por $A = B$ se e somente se $A \subseteq B$ e $B \subseteq A$. Formalmente $A = B \Leftrightarrow A \subseteq B \text{ e } B \subseteq A$.
3. Diz-se que A é um subconjunto próprio de B , o que é denotado por $A \subset B$, se e somente se A é um subconjunto de B , mas A não é igual a B . Formalmente $A \subset B \Leftrightarrow A \subseteq B \text{ e } B \not\subseteq A$.

Exemplo 2.2 $\{a, b, c\} \subset \{b, c, a, e\}$; $\text{pares} \subset \mathbb{N}$, $\text{ímpares} \subset \mathbb{N}$.

Lema 2.1 São válidas as seguintes propriedades:

$$(p1) \quad A \subseteq B \text{ e } B \subseteq C \Rightarrow A \subseteq C$$

$$(p2) \quad A \subset B \text{ e } B \subset C \Rightarrow A \subset C$$

$$(p3) \quad A \subset B \text{ e } B \subset A \Rightarrow A = B$$

²Neste capítulo é utilizado \Leftrightarrow para representar a expressão “se e somente se” e \Rightarrow para representar a expressão “se...então...”.

Demonstração (p1): Uma vez que $A \subseteq B$ e $B \subseteq C$ tem-se

$$\left. \begin{array}{l} \forall x \in A \rightarrow x \in B \\ \forall x \in B \rightarrow x \in C \end{array} \right| \Rightarrow \begin{array}{l} \forall x \in A \rightarrow x \in C \\ A \subseteq C \end{array}$$

Demonstração (p2): Uma vez que $A \subset B$ e $B \subset C$ tem-se

$$\left. \begin{array}{l} (i) \forall x \in A \rightarrow x \in B \\ (ii) \exists y \in B \mid y \notin A \\ (iii) \forall y \in B \rightarrow y \in C \\ (iv) \exists z \in C \mid z \notin B \end{array} \right| \Rightarrow \begin{array}{l} (i) \text{ e } (iii) \forall x \in A \rightarrow x \in C \\ (iv) \text{ e } (ii) \exists z \in C \mid z \notin A \\ A \subset C \end{array}$$

Demonstração (p3): $\forall x \in A \rightarrow x \in B$, mas se $z \in B$, não necessariamente $z \in A$. $\forall z \in B \rightarrow z \in A$, logo para satisfazer a condição anterior $A = B$.

Definição 2.2 [Conjunto potência] Seja A um conjunto. O conjunto de subconjuntos de A é chamado de conjunto das partes de A ou conjunto potência de A , denotado por $\wp(A)$. Formalmente $\wp(A) = \{X \mid X \subseteq A\}$.

Note que $\wp(A)$ é um conjunto de conjuntos, diz-se que tais conjuntos são famílias de conjuntos.

Definição 2.3 [Operações sobre conjuntos]

União Sejam A e B conjuntos. A união de A e B denotada por $A \cup B$ é definida por $A \cup B = \{x \mid x \in A \text{ ou } x \in B\}$. Seja $\{A_i\}_{i \in I}$ uma família de conjuntos com índice I . Então a união da família é definida por

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

Se $I = 1, 2, \dots, n$ escreve-se

$$\bigcup_{i=1}^{i=n} A_i \text{ ou } \bigcup_{i=1}^n A_i$$

Interseção Sejam A e B conjuntos. A interseção de A e B denotada por $A \cap B$ é definida por $A \cap B = \{x \mid x \in A \text{ e } x \in B\}$. Seja $\{A_i\}_{i \in I}$ uma família de conjuntos com índice I . Então a interseção da família é definida por

$$\bigcap_{i \in I} A_i = \{x \mid \text{para todo } i \in I \text{ } x \in A_i\}$$

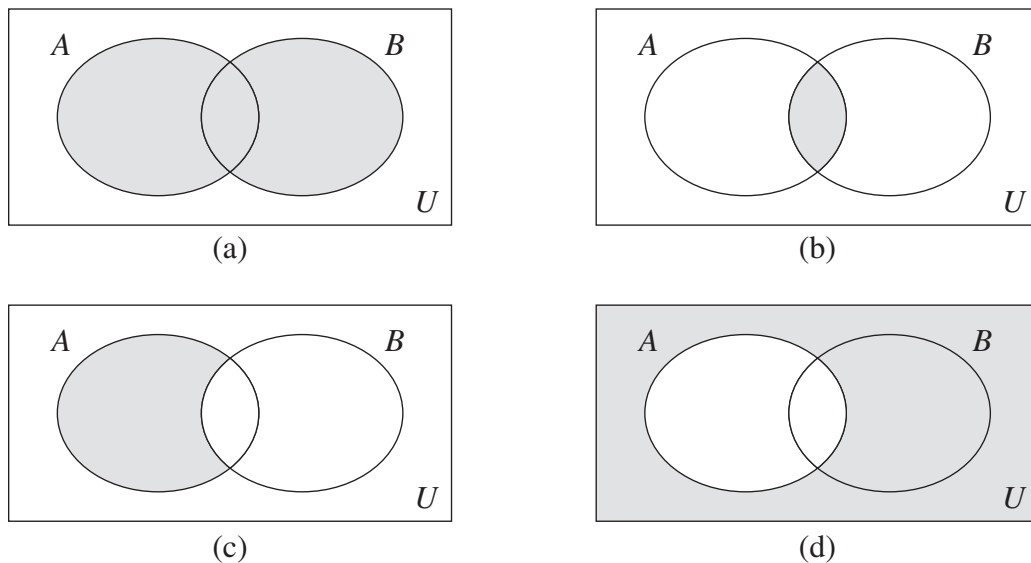


Figura 2.1: Diagrama de Venn de operações com conjuntos: (a) $A \cup B$; (b) $A \cap B$; (c) $A - B$; (d) \bar{A} .

Se $I = 1, 2, \dots, n$ escreve-se

$$\bigcap_{i=1}^{i=n} A_i \text{ ou } \bigcap_{i=1}^n A_i$$

Diferença Sejam A e B conjuntos. A diferença de A e B , denotada por $A - B$, é definida por $A - B = \{x \mid x \in A \text{ e } x \notin B\}$.

Complemento Seja A um conjunto. O complemento de A , denotado por \bar{A} , é definido por $\bar{A} = \{x \mid x \notin A\}$.

Potência Seja A um conjunto. O conjunto potência de A (também conhecido como conjunto das partes de A), denotado por \mathcal{P} , é definido por $\mathcal{P} = \{x \mid x \subseteq A\}$.

Em geral, serão estudadas propriedades de subconjuntos de um conjunto U ou “conjunto universo”. Portanto, nas definições feitas até o momento, presupõe-se sempre que para um conjunto $A \subseteq U$, $A = \{x \in U \mid P(x)\}$. No caso do complemento de A , $\bar{A} = U - A$. Alguns autores se referem à diferença $A - B$ como o complemento de B com relação a A . A figura 2.1 apresenta diagramas de Venn ilustrativos das operações de união, interseção, diferença e complemento.

Exemplo 2.3

1. $\{1, 2, 3\} \cup \{3, 4\} = \{1, 2, 3, 4\}$
2. $\{1, 2, 3, 5\} \cap \{3, 4, 5\} = \{3, 5\}$
3. $\{1, 2, 3\} - \{3, 4\} = \{1, 2\}$
4. $\overline{\text{pares}} = \text{ímpares}$, com relação a \mathbb{N}

Lema 2.2 São válidas as seguintes propriedades:

- (p4) $A \subseteq B \Rightarrow A \cup B = B$
- (p5) $A \subseteq B \Rightarrow A \cap B = A$
- (p6) $A \subseteq B \Rightarrow \bar{B} \subseteq \bar{A}$
- (p7) $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
- (p8) $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
- (p9) $\overline{A \cup B} = \bar{A} \cap \bar{B}$
- (p10) $\overline{A \cap B} = \bar{A} \cup \bar{B}$

Ainda que o formalismo matemático sugira uma Teoria dos Conjuntos completa e consistente, o leitor deve ter cuidado para não aceitá-la de forma incondicional e ingênua. Bertrand Russell descobriu um paradoxo na teoria (Paradoxo de Russell) que nunca foi resolvido. Considere o conjunto C como sendo "o conjunto de *todos* os conjuntos que não se contêm a si próprios como membros". Formalmente, A é elemento de C se e só se A não é elemento de A , isto é, $C = \{A | A \notin A\}$.

Pela Teoria dos Conjuntos, C é um conjunto perfeitamente bem definido. Contudo, se C contém a si mesmo, então, por definição, ele não é membro de C . Por outro lado, supondo que C não contenha a si mesmo, então ele precisa ser elemento de C , uma declaração aparentemente verdadeira que leva a uma contradição lógica.

Outra formulação interessante deste paradoxo é conhecida como paradoxo do barbeiro. Imagine um barbeiro que recebe a atribuição de barbear todos os homens, e somente homens quem não barbeiam a si mesmos. Observe que existem apenas dois conjuntos distintos, aquele no qual os homens barbeiam a si mesmos, e aquele no qual os homens não barbeiam a si mesmos. Se o barbeiro pertence ao conjunto dos homens que barbeiam a si mesmo, então ele, como barbeiro não poderia se barbear. Ora, mas como ele não pode se barbear, então ele, como barbeiro, deve cumprir sua obrigação de se barbear. Parece evidente que a única solução para este

problema, é que o problema não deveria existir.

Kurt Gödel usou o paradoxo do barbeiro para provar o seu teorema da incompletude. Alan Turing também demonstrou a indecidibilidade do problema da parada usando o mesmo paradoxo.

2.2 Funções

A Computação emprega funções em larga escala, particularmente aquelas que transformam um conjunto finito em outro conjunto finito, funções estas conhecidas como funções discretas. Um programa de computador pode ser entendido como uma função que utiliza dados como argumentos de entrada, e que produz um resultado associado a estes argumentos.

Sejam A e B conjuntos quaisquer; uma função f de A em B é uma *regra* que associa a certos elementos de A um único elemento de B . Esta unicidade significa que se f associa $a \in A$ a $b_1 \in B$ e a $b_2 \in B$ então $b_1 = b_2$.

Para dizer que f é uma função de A em B usa-se a notação $f : A \rightarrow B$. O conjunto de todas as funções de A em B é denotado por B^A , ou seja $B^A = \{f \mid f : A \rightarrow B\}$.

Seja A um conjunto, então chama-se de operação n -ária uma função $f : A^n \rightarrow A$.

Se $f : A \rightarrow B$ então diz-se que A é o conjunto de partida de f e B o conjunto de chegada de f . O subconjunto de A cujos elementos são associados a elementos de B é chamado de domínio de f , denotado por $dom(f)$. O conjunto B , por sua vez é chamado de contra-domínio de f , denotado por $ctr(f)$. O subconjunto de B , aos quais aqueles elementos de A foram associados, é chamado de imagem de f , que denota-se por $img(f)$. Denota-se o elemento de B associado a um elemento $a \in A$ por $f(a)$, tal como a Figura 2.2.

Definição 2.4 Seja $f : A \rightarrow B$ e $g : A \rightarrow B$. Diz-se que

1. f é uma sobrejeção, ou que f é uma função de A sobre B , se e somente se

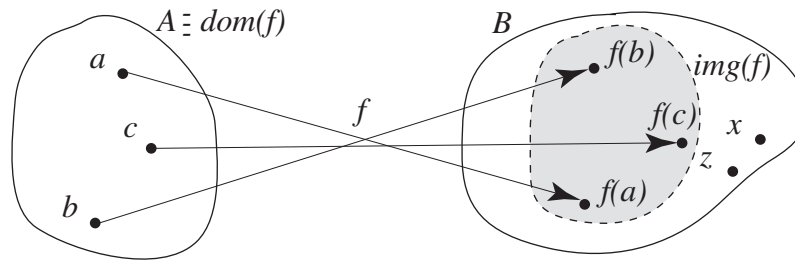


Figura 2.2: Função de A em B .

$img(f) = B$ (ver Figura 2.3a).

2. f é uma injeção ou injetiva de A em B se e somente se para cada elemento do conjunto A corresponde um elemento distinto do conjunto B , isto é, para todo $x, y \in A$ se $f(x) = f(y)$ então $x = y$ (ver Figura 2.3b).
3. f é uma bijeção se e somente se f é uma injeção e uma sobrejeção.
4. f é igual a g , denotado por $f = g$ se e somente se para todo $x \in dom(f)$ $f(x) = g(x)$, e além disso $dom(f) = dom(g)$
5. f é uma função total de A em B se e somente se $dom(f) = A$.
6. f é uma função parcial de A em B se nem todos os $x \in A$ pertencem ao $dom(f)$.
7. se $x \in dom(f)$ então $f(x) \downarrow$ (f converge em x); se $x \in A - dom(f)$ então $f(x) \uparrow$ (f diverge em x).

Exemplo 2.4 A seguir são exibidos alguns tipos de funções:

Funções no discreto Por exemplo funções do conjunto dos naturais \mathbb{N} no conjunto dos naturais $f : \mathbb{N} \rightarrow \mathbb{N}$

1. $f : \mathbb{N} \rightarrow \mathbb{N}$, definida por: para todo $x \in \mathbb{N}$ $f(x) = x + 1$
2. $f : \mathbb{N} \rightarrow \mathbb{N}$, definida por: para todo $x \in \mathbb{N}$ $f(x) = 2.x$
3. $f : \mathbb{N} \rightarrow \mathbb{N}$, definida por: para todo $x \in \mathbb{N}$, tal que 3 divide x $f(x) = x/3$

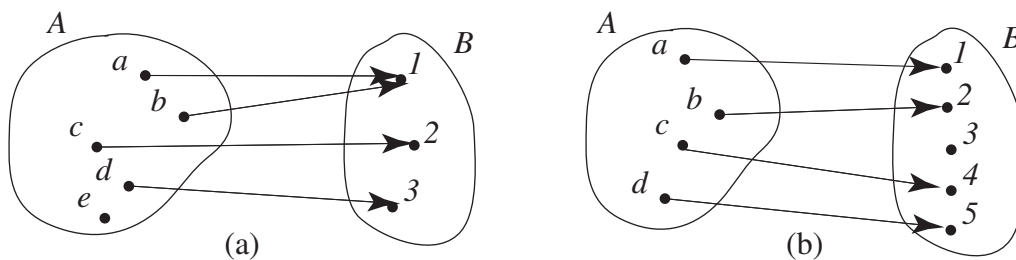


Figura 2.3: Funções: (a) Sobrejetora; (b) Injetora.

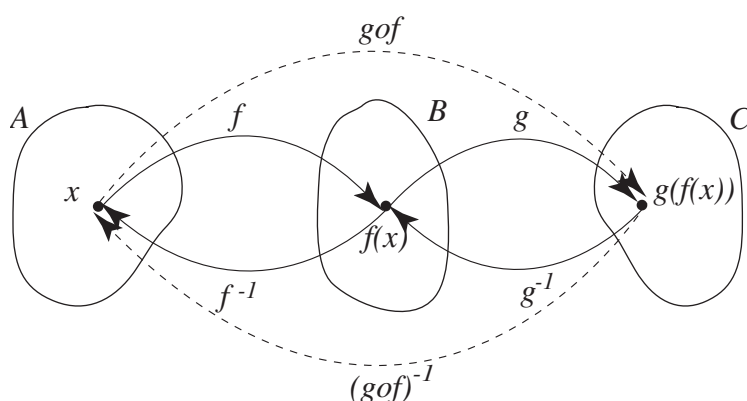


Figura 2.4: Composição de f e g .

As funções 1. e 2. são funções totais não sobrejetivas, 3. é uma função parcial pois $\text{dom}(f) = \{0, 3, 6, 9, \dots\}$

Funções no contínuo Por exemplo funções $f : \mathbb{R} \rightarrow \mathbb{R}$, onde \mathbb{R} é o conjunto dos reais.

Definição 2.5 Seja $f : A \rightarrow B$ uma função injetiva. A inversa de f , denotada por f^{-1} é uma função de B em A definida por $f^{-1}(x) = y$ se e somente se $f(y) = x$. Formalmente $f^{-1}(x) = y \Leftrightarrow f(y) = x$.

Definição 2.6 Sejam as funções $f : A \rightarrow B$ e $g : B \rightarrow C$. A composição de f e g , $g \circ f$, é a função de A em C tal que $g \circ f(x) = g(f(x))$ (ver Figura 2.4).

Lema 2.3 Sejam as funções $f : A \rightarrow B$ e $g : B \rightarrow C$ funções injetivas. Então $(g \circ f)^{-1} = f^{-1} \circ g^{-1}$ (ver Figura 2.4).

Definição 2.7 Seja $A \subseteq B$ um conjunto. Uma função $f : A \rightarrow \{0, 1\}$ é a função característica para A se e somente se

$$\text{para todo } x \in B \quad f(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{se } x \notin A \end{cases}$$

Funções características de um conjunto A são, em geral, representadas por χ_A .

Exemplo 2.5 Se o resto da divisão de x por y for dado por $\text{resto}(x, y)$ então a seguinte função é a função característica do conjunto dos números ímpares:

$$\chi_{\text{ímpares}}(x) = \text{resto}(x, 2)$$

e para os pares:

$$\chi_{\text{pares}}(x) = 1 - \text{resto}(x, 2)$$

Se este processo pode ser representado por uma função sobrejetora $h : \mathbb{N} \rightarrow A$, onde h associa a cada número natural um elemento de A , então diz-se que A é contável, caso contrário é não contável. Se o processo de contagem termina, ou seja, se existe uma bijeção $h : \{1, \dots, n\} \rightarrow A$, $n \in \mathbb{N}$, diz-se que a cardinalidade de A , denotado por $\#(A)$, é n (finita). O Algoritmo 2.1 apresenta um resolutor para o processo de contagem.

Data: Conjunto A cuja quantidade de elementos deseja-se obter

Result: Cardinalidade finita do conjunto A

iContagem = 0;

while $A \neq \emptyset$ **do**

Retirar um elemento de A ;

iContagem = iContagem + 1;

end

Cardinalidade finita do conjunto = iContagem;

Algoritmo 2.1: Determinação da cardinalidade finita de um conjunto

Se o processo de contagem não termina, diz-se que A tem cardinalidade infinita. Por exemplo todo subconjunto de \mathbb{N} tem cardinalidade infinita que é representada por \aleph_0 (aleph 0). Na realidade, no caso de \mathbb{N} , após a retirada de n elementos não importante quão grande é n ainda sobrá um conjunto com cardinalidade \aleph_0 . O conjunto dos pares e o dos ímpares também têm cardinalidade \aleph_0 , e ambos são

partes de \mathbb{N} . Observe que se um conjunto pode ser colocado em correspondência biunívoca, então ele também possui cardinalidade \aleph_0 .

Definição 2.8 Um conjunto A é enumerável (contável) se ele é finito ou se tem cardinalidade \aleph_0 . Uma enumeração para um conjunto χ enumerável infinito representado por uma seqüência $\chi = a_1, a_2, a_3, \dots$ é uma bijeção $f(a_i) = i \in \mathbb{N}$.

Exemplo 2.6 \mathbb{Z} é enumerável.

$$f : \mathbb{Z} \rightarrow \mathbb{N}$$

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

□

Exemplo 2.7 Pares ordenados envolvendo números naturais, $\mathbb{N} \times \mathbb{N}$ é enumerável. Para demonstrar, basta ordenar os pares ordenados como se segue e em seguida atribuir etiquetas de números naturais às suas posições.

$$\begin{array}{ccccccc} \swarrow & & & & & & \\ \langle 1,1 \rangle & \langle 1,2 \rangle & \langle 1,3 \rangle & \langle 1,4 \rangle & \dots & & \\ \swarrow & & & & & & \\ \langle 2,1 \rangle & \langle 2,2 \rangle & \langle 2,3 \rangle & \langle 2,4 \rangle & \dots & & \\ \swarrow & & & & & & \\ \langle 3,1 \rangle & \langle 3,2 \rangle & \langle 3,3 \rangle & \langle 3,4 \rangle & \dots & & \\ \swarrow & & & & & & \\ \langle 4,1 \rangle & \langle 4,2 \rangle & \langle 4,3 \rangle & \langle 4,4 \rangle & \dots & & \\ & \vdots & \vdots & \vdots & \vdots & & \end{array}$$

□

Exemplo 2.8 O conjunto de todas as palavras formadas a partir de um alfabeto $\Sigma = \sigma_1, \sigma_2, \dots, \sigma_n$, conhecido por Σ^* , é enumerável pela função:

$$f : \Sigma^* \rightarrow \mathbb{N}$$

$$f(\text{palavra}) = \sum_{i=1}^k \text{posição do caracter}_i \times n^i$$

onde k é o tamanho da palavra.

□

Exemplo 2.9 O conjunto dos programas de qualquer linguagem de programação (LP) é enumerável, pois qualquer LP possui um alfabeto Σ tal que o programa está em Σ^* .

□

Definição 2.9 Um conjunto A é dito não enumerável se $\#(A) > \aleph_0$.

A prova da existência de conjuntos não enumeráveis foi feita por Cantor (1874) através de um método que ficou conhecido como diagonalização de Cantor, ilustrado no exemplo a seguir.

Exemplo 2.10 O conjunto \mathbb{R} é não enumerável. A idéia para essa demonstração é que se existir uma bijeção $\mathbb{R} \rightarrow \mathbb{N}$, seria possível compor uma bijeção de $]0, 1[\rightarrow \mathbb{R}$ e então concluir que $]0, 1[$ é enumerável. Contudo, sabe-se que $]0, 1[$ não é enumerável. Assim, suponha uma demonstração por contradição. Suponha que $]0, 1[$ é enumerável, e seja uma bijeção $f : \mathbb{N} \rightarrow]0, 1[$. Assim, cada número natural n é mapeado sobre um número decimal:

$$\begin{array}{lcl} 0 & \mapsto & 0. a_{00} a_{01} a_{02} \\ 1 & \mapsto & 0. a_{10} a_{11} a_{12} \\ 2 & \mapsto & 0. a_{20} a_{21} a_{22} \\ \vdots & & \vdots \end{array}$$

A intenção é mostrar que f não é uma bijeção. A idéia de que \mathbb{N} tem cardinalidade \aleph_0 , é preciso mostrar que $]0, 1[$ tem cardinalidade maior que \aleph_0 . Desta forma, o argumento será que f não pode ser sobrejetora, logo é possível encontrar um número que não é mapeado.

$$\begin{array}{lcl} 0 & \mapsto & 0. \mathbf{a}_{00} a_{01} a_{02} \\ 1 & \mapsto & 0. a_{10} \mathbf{a}_{11} a_{12} \\ 2 & \mapsto & 0. a_{20} a_{21} \mathbf{a}_{22} \\ \vdots & & \vdots \end{array}$$

onde $b = 0.b_0b_1b_2 \dots$ e $b_0 \neq a_{00}, b_1 \neq a_{11}, \dots, b_i \neq a_{ii}$

Suponha $i \in \mathbb{N}$ tal que $f(i) = b$. Note que $f(i)$ é mapeado sobre o número real $0.a_{i0}a_{i1}a_{i2} \dots a_{ii} \dots$. Porém, por construção de b , $b_i \neq a_{ii}$, logo $f(i) \neq b$. Assim, tem-se que b não está mais na lista. Se b não está na lista, f não é sobrejetora, e também não é bijetora. \square

Teorema 2.1 Seja A um conjunto qualquer, $\#(A) = \aleph_i$. Então $\#(\mathcal{P}(A)) > \#(A)$ e $\#(\mathcal{P}(A)) = \aleph_{i+1}$.

Basta provar que $\#(A) \leq \#(\mathcal{P}(A))$. Para isso, é suficiente mostrar que não existe função bijetora de $A \rightarrow \mathcal{P}(A)$. Seja então uma $f : A \rightarrow \mathcal{P}(A)$ bijetiva. Seja $x \in A$ de modo que $f(x) = y'$, onde $\#(y') = 1$. Observe que todos os elementos de A estão mapeados sobre os elementos de $\mathcal{P}(A)$ com cardinalidade 1. Resta mapear os elementos de $\mathcal{P}(A)$ cuja cardinalidade é maior que 1. Como todos os elementos de A já foram utilizados, é necessário tomar novamente um x de modo que $f(x) = y''$ onde $\#(y'') > 1$. Contudo, para fazer isto, tem-se um $x \in A$ tal que $f(x) = y'$ e $f(x) = y''$, logo f não é uma função e muito menos bijetiva. \square

Neste sentido, conjuntos infinitos de cardinalidades sucessivamente maiores podem ser obtidos pela aplicação sucessiva da operação conjunto-potência. Considere os conjuntos $A, B = \mathcal{P}(A), C = \mathcal{P}(B), D = \mathcal{P}(C)$ etc. Então, $\#(A) < \#(B) < \#(C) < \#(D) < \dots$. De acordo com a teoria de Cantor, \aleph é o conjunto que possui a menor cardinalidade entre todos os conjuntos infinitos, a qual é denotada por \aleph_0 , o primeiro número da sua série transfinita. Por consequência, $\aleph < \mathcal{P}(\aleph)$.

Teorema 2.2 Sejam A e B dois conjuntos, $B \subseteq A$. Se $\#(A) = \aleph_0$, então $\#(A) \leq \aleph_0$.

Se $\#(A) = \aleph_0$, então existe uma função bijetora entre o conjunto dos números naturais \mathbb{N} e o conjunto A (e vice-versa). Logo, existe uma função injetora e total f_1 que associa elementos de A e \mathbb{N} , conforme a seguir:

$$\begin{array}{ccccccc}
 B : & - & a_1 & - & \dots & a_n & \dots \\
 f_2 : & & \downarrow & & & \downarrow & \\
 A : & a_0 & a_1 & a_2 & \dots & a_n & \dots \\
 f_1 : & \downarrow & \downarrow & \downarrow & & \downarrow & \\
 \mathbb{N} : & 1 & 2 & 3 & \dots & n & \dots
 \end{array}$$

Observe também que se B é subconjunto de A , é possível associar cada elemento de B ao mesmo elemento de A através de uma função injetora e total f_2 . Já a composição das funções f_1 e f_2 mostra que existe uma função injetora e total de B para \mathbb{N} . Logo, $\#(B) \leq \#(\mathbb{N})$, ou seja, $\#(B) \leq \aleph_0$. Em outras palavras, qualquer subconjunto (finito ou infinito) de um conjunto enumerável é também um conjunto enumerável. \square

Teorema 2.3 Sejam A e B dois conjuntos quaisquer. Se $\#(A) = \aleph_0$ e $\#(B) = \aleph_0$, então $\#(A \cup B) = \aleph_0$.

Se A e B são conjuntos enumeráveis (finitos ou infinitos), então seus elementos podem ser ordenados da seguinte forma:

$$\begin{aligned} A : & a_0, a_1, a_2, \dots, a_{n-1}, a_n, a_{n+1}, \dots \\ B : & b_0, b_1, b_2, \dots, b_{n-1}, b_n, b_{n+1}, \dots \end{aligned}$$

A enumeração dos elementos de $A \cup B$ pode ser feita através do seguinte modo:

$$A \cup B : a_0, b_0, a_1, b_1, a_2, b_2, \dots, a_{n-1}, b_{n-1}, a_n, b_n, a_{n+1}, b_{n+1}, \dots$$

Desta forma, $A \cup B$ é um conjunto enumerável e $\#(A \cup B) = \aleph_0$, e a união de dois conjuntos enumeráveis é sempre um conjunto enumerável. \square

Teorema 2.4 Sejam A e B dois conjuntos quaisquer. Se $\#(A) = \aleph_0$ e $\#(B) = \aleph_0$, então $\#(A \cap B) \leq \aleph_0$.

Se $A \subseteq B$, então $A \cap B = A$ e $\#(A \cap B) = \#(A) = \aleph_0$ por hipótese. Se, por outro lado, $B \subseteq A$, então $A \cap B = B$ e $\#(A \cap B) = \#(B) = \aleph_0$ por hipótese. Finalmente, se nenhuma dessas duas condições for verdadeira, então $(A \cap B) \subseteq A$ e, pelo Teorema 2.2, $\#(A \cap B) \leq \aleph_0$. Portanto, em qualquer caso $\#(A \cap B) \leq \aleph_0$. \square

Teorema 2.5 Sejam A e B dois conjuntos, $B \subseteq A$. Se $\#(A) = \aleph_1$ e $\#(B) = \aleph_0$, então $\#(A - B) = \aleph_1$.

Suponha-se que $\#(A - B) = \aleph_0$. Então, de acordo com o Teorema 2.3, $\#((A - B) \cup B) = \aleph_0$, o que contradiz a hipótese de que $\#(A) = \aleph_1$, pois $(A - B) \cup B = A$. Como $B \subseteq A$, e portanto, $\#(B) \leq \#(A)$, conclui-se que $\#(A - B) = \aleph_1$. \square

2.3 Relações

Freqüentemente, é utilizado a noção de relações entre duas ou mais coisas. Por exemplo, Batman se relaciona com Robin como parceiro; UFRJ e Rio de Janeiro como localização ou, a relação de “estar entre” pode ser verificada entre três objetos quaisquer do espaço físico. Informalmente é exigido que exista uma conexão entre

as coisas que se relacionam. Esta idéia vaga de conexão é incorporada formalmente pelo conceito de *par ordenado*. Fundamentalmente, um par ordenado $\langle a, b \rangle$ possui a seguinte propriedade³:

$$\langle x_1, x_2 \rangle = \langle y_1, y_2 \rangle \text{ se e somente se } x_1 = y_1 \text{ e } x_2 = y_2$$

A noção de par ordenado é estendida pela noção de n -tupla ordenada, denotada por $\langle a_1, a_2, \dots, a_n \rangle$ com a propriedade:

$$\langle x_1, x_2, \dots, x_n \rangle = \langle y_1, y_2, \dots, y_n \rangle \Leftrightarrow x_1 = y_1, \dots, x_n = y_n$$

Em vez de n -tupla ordenada diz-se também n -*pla* ou em geral *pla*. Por razões de economia e facilidade de escrita e leitura, representa-se as *plas* de letras indexadas $\langle x_1, x_2, \dots, x_n \rangle$ por \underline{x}_n , considerando o primeiro elemento como possuindo índice 1 e o último índice n .

Definição 2.10 [Produto Cartesiano] Sejam A e B conjuntos. Define-se o produto cartesiano⁴ de A e B , denotado por $A \times B$ como sendo o conjunto de todos os pares ordenados $\langle x, y \rangle$ com $x \in A$ e $y \in B$, ou seja, $A \times B = \{ \langle x, y \rangle \mid x \in A \text{ e } y \in B \}$.

Exemplo 2.11 Seja $A = \{0, 1\}$ e $B = \{1, 2, 3\}$ dois conjuntos quaisquer. O produto cartesiano de A por B , e vice-versa, é dado por:

$$A \times B = \{ \langle 0, 1 \rangle, \langle 0, 2 \rangle, \langle 0, 3 \rangle, \langle 1, 1 \rangle, \langle 1, 2 \rangle, \langle 1, 3 \rangle \}$$

$$B \times A = \{ \langle 1, 0 \rangle, \langle 1, 1 \rangle, \langle 2, 0 \rangle, \langle 2, 1 \rangle, \langle 3, 0 \rangle, \langle 3, 1 \rangle \}$$

Se A e B são conjuntos então uma relação binária R entre A e B é qualquer subconjunto do produto cartesiano de A e B , ou seja $R \subseteq A \times B$. Quando $A = B$ diz-se que R é uma relação em A ou $R \subseteq A^2$. A Figura 2.5 mostra: (i) os planos coordenados e uma região representando uma relação; (ii) o grafo da relação binária $R = \{ \langle a, b \rangle, \langle a, c \rangle, \langle b, d \rangle, \langle a, d \rangle, \langle c, d \rangle, \langle d, e \rangle \}$.

³Esta propriedade é derivada da definição formal de par ordenado $\langle a, b \rangle = \{ \{a\}, \{a, b\} \}$, veja [28]

⁴O nome cartesiano é uma homenagem a René Descartes, quem historicamente representou funções utilizando eixos ortogonais, o eixo horizontal representando o domínio da função e o vertical o contradomínio. Assim qualquer região destes planos de coordenadas é uma relação, incluindo as funções.

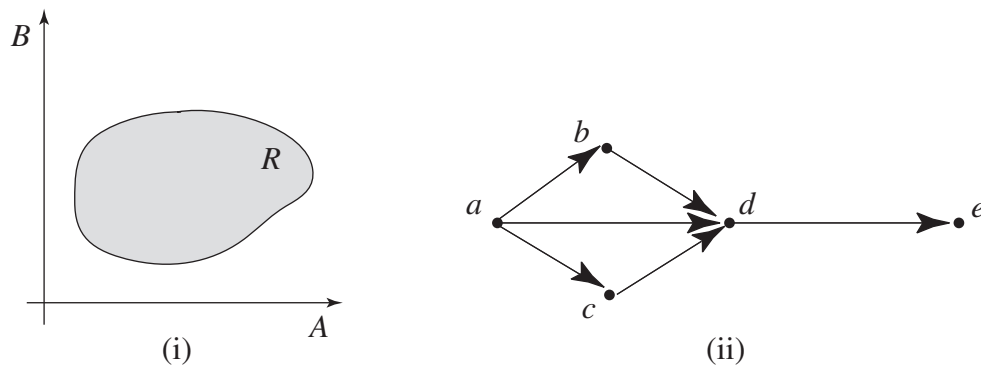


Figura 2.5: Relações: (i) os planos coordenados e uma região representando uma relação; (ii) o grafo da relação binária $R = \{ \langle a, b \rangle, \langle a, c \rangle, \langle b, d \rangle, \langle a, d \rangle, \langle c, d \rangle, \langle d, e \rangle \}$.

Do mesmo modo pode-se definir relações n -árias em uma família de conjuntos $\{A_i\}_{1 \leq i \leq n}$, como qualquer subconjunto R do produto cartesiano, isto é,

$$\prod_{i=1}^n A_i$$

O natural n é a *aridade* da relação R .

Definição 2.11 [Composição] Sejam A , B e C conjuntos, R uma relação em $A \times B$ e S uma relação em $B \times C$. Então, a composição $S \circ R$ é uma relação em $A \times C$ definida por $\{ \langle a, c \rangle \mid \langle a, b \rangle \in R \text{ e } \langle b, c \rangle \in S \}$, e ilustrada na Figura 2.6.

A forma tradicional de representação de uma relação é através das n -tuplas ordenadas. Entretanto, existem outros modos de representação que privilegiam determinados tipos de visão, particularmente no que diz respeito às propriedades das relações que serão descritas na seção a seguir.

A representação gráfica, por exemplo, privilegia a noção de espacialização da relação. Supondo a e b elementos de um conjunto A , e R uma relação de A em A tem-se que existirá uma seta entre a e b , se e somente se, $\langle a, b \rangle \in R$. A Figura 2.7 apresenta a espacialização de uma relação em A^2 . Neste caso a relação é dada por $R = \{ \langle 1, 2 \rangle, \langle 2, 1 \rangle, \langle 2, 3 \rangle, \langle 3, 1 \rangle, \langle 3, 3 \rangle \}$.

A representação matricial das relações é interessante, sob o ponto de vista matemático, sempre que a complexidade da representação gráfica dificulta a compreensão da relação. Assim, seja $A = \{a_1, a_2, \dots, a_n\}$ e $B = \{b_1, b_2, \dots, b_m\}$ conjuntos

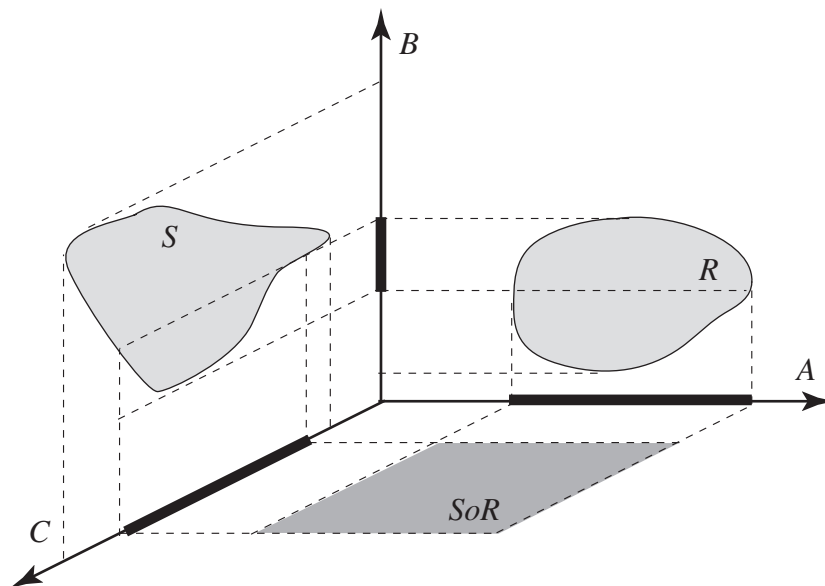


Figura 2.6: Composição de R e S , dada por $S \circ R$.

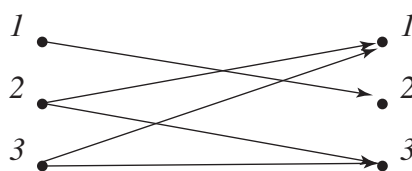


Figura 2.7: Representação gráfica de uma relação.

quaisquer tais que $R: A \rightarrow B$. A matriz $M_{n \times m}$, representativa da relação R é dada por:

$$\begin{array}{ll} M[i, j] = F, & \langle a_i, b_j \rangle \notin R \\ & V, \quad \langle a_i, b_j \rangle \in R \end{array}$$

Exemplo 2.12 A Relação $R = \{\langle 1, 2 \rangle, \langle 2, 1 \rangle, \langle 2, 3 \rangle, \langle 3, 1 \rangle, \langle 3, 3 \rangle\}$, descrita na Figura 2.7, pode ser escrita na forma matricial:

$$R = \begin{bmatrix} F & V & F \\ V & F & V \\ V & F & V \end{bmatrix}$$

Exemplo 2.13 A Relação $S = \{\langle 1, a \rangle, \langle 1, b \rangle, \langle 2, a \rangle, \langle 3, b \rangle\}$, sobre os conjuntos $A = \{1, 2, 3\}$ e $B = \{a, b\}$ 2.7, pode ser escrita na forma matricial:

$$S = \begin{bmatrix} V & V \\ V & F \\ F & V \end{bmatrix}$$

Definição 2.12 [Matriz Produto Lógico] Seja uma relação R com matriz de representação $M_R(m \times n)$, e uma relação S com matriz de representação $M_S(n \times p)$. A relação $S \circ R$ possui uma matriz de representação $M_{S \circ R}(m \times p)$ tal que:

$$\begin{aligned}
M_{S \circ R}[i, j] &= [M_R(i, 1) \wedge M_S(1, j)] \vee \\
&\quad [M_R(i, 2) \wedge M_S(2, j)] \vee \\
&\quad \dots \vee \\
&\quad [M_R(i, n) \wedge M_S(n, j)]
\end{aligned}$$

onde $i \in \{1, \dots, m\}$ e $j \in \{1, \dots, p\}$.

Exemplo 2.14 Seja a relação $R = \{\langle 1, 2 \rangle, \langle 2, 1 \rangle, \langle 2, 3 \rangle, \langle 3, 1 \rangle, \langle 3, 3 \rangle\}$, e a relação $S = \{\langle 1, a \rangle, \langle 1, b \rangle, \langle 2, a \rangle, \langle 3, b \rangle\}$. A aplicação da relação R , seguida da aplicação S irá produzir a relação $S \circ R = \{\langle 1, a \rangle, \langle 2, a \rangle, \langle 2, b \rangle, \langle 3, a \rangle, \langle 3, b \rangle\}$ (sugere-se ao leitor

utilizar a representação gráfica para obter $S \circ R$). Utilizando-se a matriz produto lógico, pode-se obter facilmente a relação $S \circ R$ através da mera multiplicação das representações matriciais de R e S , tal como a seguir:

$$\begin{matrix} & R & & S & = & S \circ R \\ \begin{bmatrix} F & V & F \\ V & F & V \\ V & F & V \end{bmatrix} & & \begin{bmatrix} V & V \\ V & F \\ F & V \end{bmatrix} & & \begin{bmatrix} V & F \\ V & V \\ V & V \end{bmatrix} \end{matrix}$$

2.4 Equivalência e Congruência

Definição 2.13 Seja R uma relação em um conjunto A . R é:

Reflexiva Se para todo $x \in A$ $\langle x, x \rangle \in R$.

Simétrica Se para todo $x, y \in A$ se $\langle x, y \rangle \in R$ então $\langle y, x \rangle \in R$.

Transitiva Se para todo $x, y, z \in A$ se $\langle x, y \rangle \in R$ e $\langle y, z \rangle \in R$ então $\langle x, z \rangle \in R$.

Anti-simétrica Se para todo $x, y \in A$ se $\langle x, y \rangle \in R$ e $\langle y, x \rangle \in R$ então $x = y$.

Definição 2.14 Seja R uma relação em um conjunto A . Diz-se que R é uma relação de equivalência se R é reflexiva, simétrica e transitiva.

Exemplo 2.15

- A relação $=$ é o caso óbvio de relação de equivalência.
- A relação $R = \{\langle x, y \rangle \in \mathbb{N}^2 \mid \text{ter divisores comuns}\}$ não é uma relação de equivalência, R é evidentemente reflexiva e simétrica, mas não é transitiva, por exemplo $\langle 12, 15 \rangle \in R$, $\langle 15, 25 \rangle \in R$ porém $\langle 12, 25 \rangle \notin R$.
- A relação $R = \{\langle x, y \rangle \in \mathbb{N}^2 \mid \text{ter os mesmos divisores primos}\}$ é uma relação de equivalência.

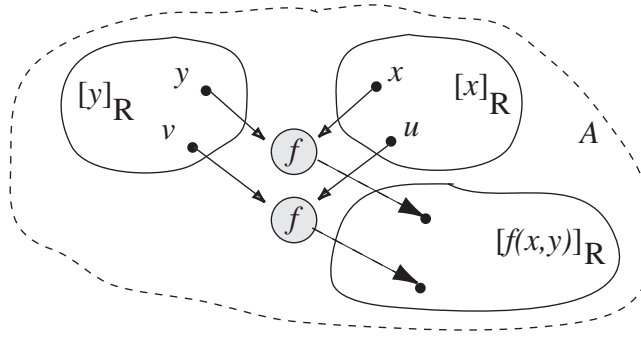


Figura 2.8: Partição em um conjunto A e f -congruência.

Definição 2.15 Seja R uma relação de equivalência (reflexiva, simétrica e transitiva) em um conjunto A e $x \in A$. Define-se a classe de equivalência de x com respeito a R por

$$[x]_R = \{y \in A \mid \langle x, y \rangle \in R\}$$

.

Lema 2.4 Seja R uma relação de equivalência em um conjunto A e $x \in A$. Então:

1. Se $\langle x, y \rangle \in R$ então $[x]_R = [y]_R$.
2. $[x]_R \cap [y]_R \neq \emptyset$ se e somente se $[x]_R = [y]_R$.
3. $\bigcup_{x \in A} [x]_R = A$

Definição 2.16 [Partição] Seja A um conjunto e $\{A_i\}_{i \in I}$ uma família de subconjuntos de A . Diz-se que $\{A_i\}_{i \in I}$ é uma partição de A se e somente se:

$$\bigcup_{i \in I} A_i = A \text{ e para todo } i, j \in I \text{ e } i \neq j \text{ } A_i \cap A_j = \emptyset$$

Lema 2.5 Seja R uma relação de equivalência em um conjunto A . A família de classes de equivalência de A com respeito a R $\{[x]_R\}_{x \in A}$ é uma partição de A .

Demonstração: É uma consequência do lema 2.4 e definição 2.16.

Definição 2.17 Seja A um conjunto, R uma relação de equivalência em A e $f : A^2 \rightarrow A$, diz-se que R é uma relação de congruência com respeito a f ou uma f -congruência se e somente se para todo $x, y, u, v \in A$ se $\langle x, u \rangle, \langle y, v \rangle \in R$ então $\langle f(x, y), f(u, v) \rangle \in R$. Ver Figura 2.8.

Exemplo 2.16 O conjunto \mathbb{N} pode ser partido pela relação \equiv_3 definida por $x \equiv_3 y \Leftrightarrow \text{resto}(x, 3) = \text{resto}(y, 3)$ onde $\text{resto}(x, y)$ é a função que nos fornece o resto da divisão de x por y . É fácil notar que \equiv_3 é uma relação de congruência com respeito a $+$.

2.5 Relações de Ordem

Definição 2.18 Diz-se que uma relação R em um conjunto A é uma relação de **ordem parcial** em A , ou simplesmente uma ordem em A se e somente se R é reflexiva, anti-simétrica e transitiva em A . A relação de ordem é dita ser **total** se e somente se para todo $x, y \in A$ $\langle x, y \rangle \in R$ ou $\langle y, x \rangle \in R$, se diz que esta ordem é linear. Relações de ordem parcial são em geral denotadas por \preceq (**precede**). Se $x \preceq y$ e $x \neq y$ dizemos que x **estritamente precede** y , o que denotamos por $x \prec y$.

Exemplo 2.17 As seguintes relações em \mathbb{N} são ordens parciais:

1. \geq é a mais conhecida das ordens parciais e é total.
2. xMy significando x é um múltiplo de y . Esta não é uma ordem total, por exemplo 3 não é múltiplo de 2 nem vice versa.

Definição 2.19 Uma estrutura de ordem ou simplesmente uma ordem é um par $\langle A, \preceq \rangle$, onde A é um conjunto e \preceq uma relação de ordem parcial em A .

Definição 2.20 Seja $\langle A, \preceq \rangle$ uma ordem e $X \subseteq A$. Diz-se que

1. $a \in A$ é um **limite inferior** de X se e somente se para todo $x \in X$, $a \preceq x$.
2. $a \in A$ é um **limite superior** de X se e somente se para todo $x \in X$, $x \preceq a$.

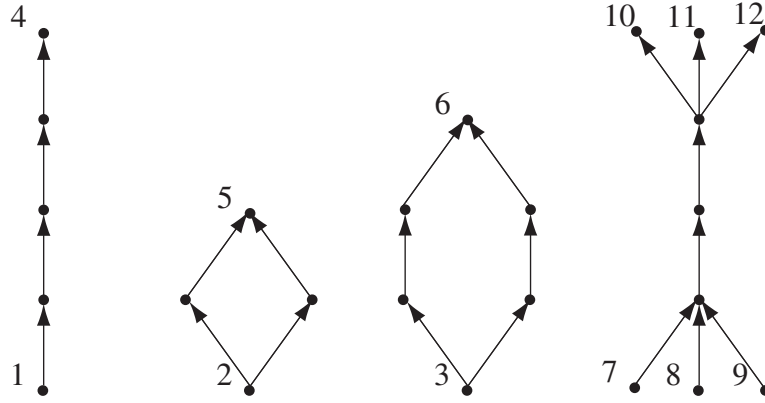


Figura 2.9: Exemplos de ordem.

3. $a \in A$ é o **ínfimo** de X se e somente se, para todo $x \in X$, $a \preceq x$ e além disso se, y é um limite inferior de X , $y \preceq a$. Ou seja, a é o maior dos limites inferiores (é único, se existir).
4. $a \in A$ é o **supremo** de X se e somente se, para todo $x \in X$, $x \preceq a$ e além disso, se y é um limite superior de X , $a \preceq y$. Ou seja, a é o menor dos limites superiores (é único, se existir).
5. $a \in X$ é um **minimal** em X se e somente se para todo $x \in X$, $x \preceq a$ então $x = a$. Ou seja, não existe $x \in X$ tal que $x \prec a$; na Figura 2.9 7, 8 e 9 são minimais.
6. $a \in X$ é **mínimo** em X se e somente se para todo $x \in X$, $a \preceq x$; na Figura 2.9 1, 2 e 3 são mínimos.
7. $a \in X$ é um **maximal** em X se e somente se para todo $x \in X$, $a \preceq x$ então $x = a$. Ou seja, não existe $x \in X$ tal que $a \prec x$; na Figura 2.9 10, 11 e 12 são maximais.
8. $a \in X$ é **máximo** em X se e somente se para todo $x \in X$, $x \preceq a$; na Figura 2.9 4, 5 e 6 são máximos.

Definição 2.21 Uma estrutura abstrata é uma tripla $\mathcal{E} = \langle D, \mathcal{R}, \mathcal{F} \rangle$, onde

1. D é um conjunto não vazio

2. \mathcal{R} é um conjunto de relações $r \subseteq D^n$, $n > 0$, n é a aridade de r .
3. \mathcal{F} é um conjunto de funções $f : D^n \rightarrow D$, $n \geq 0$, n é a aridade de f .

Definição 2.22 Sejam $\mathcal{E}_1 = \langle D_1, \mathcal{R}_1, \mathcal{F}_1 \rangle$ e $\mathcal{E}_2 = \langle D_2, \mathcal{R}_2, \mathcal{F}_2 \rangle$ estruturas abstratas. Diz-se que \mathcal{E}_1 é compatível com \mathcal{E}_2 se e somente se

1. Para cada $f \in \mathcal{F}_1$ de aridade n existe $f^* \in \mathcal{F}_2$ de aridade n , e
2. Para cada $r \in \mathcal{R}_1$ de aridade n existe $r^* \in \mathcal{R}_2$ de aridade n .
3. Diz-se que \mathcal{E}_1 e \mathcal{E}_2 são compatíveis se e somente se \mathcal{E}_1 é compatível com \mathcal{E}_2 e \mathcal{E}_2 é compatível com \mathcal{E}_1 .

Definição 2.23 Sejam $\mathcal{E}_1 = \langle D_1, \mathcal{R}_1, \mathcal{F}_1 \rangle$ e $\mathcal{E}_2 = \langle D_2, \mathcal{R}_2, \mathcal{F}_2 \rangle$ estruturas abstratas compatíveis e $h : D_1 \rightarrow D_2$ uma função. Diz-se que h é um homomorfismo de \mathcal{E}_1 em \mathcal{E}_2 se e somente se:

1. Para todo $f \in \mathcal{F}_1$, $f^* \in \mathcal{F}_2$ de aridade n e $a_1, a_2, \dots, a_n \in D_1$

$$h(f(a_1, a_2, \dots, a_n)) = f^*(h(a_1), h(a_2), \dots, h(a_n))$$

2. Para todo $r \in \mathcal{R}_1$, $r^* \in \mathcal{R}_2$ de aridade n e $a_1, a_2, \dots, a_n \in D_1$

$$\langle a_1, a_2, \dots, a_n \rangle \in r \Rightarrow \langle h(a_1), h(a_2), \dots, h(a_n) \rangle \in r^*$$

Se h é uma bijeção diz-se que h é um isomorfismo de \mathcal{E}_1 em \mathcal{E}_2 . Note que se existe um isomorfismo h de \mathcal{E}_1 em \mathcal{E}_2 , h é uma bijeção, portanto h^{-1} é um isomorfismo de \mathcal{E}_2 em \mathcal{E}_1 . Diz-se que \mathcal{E}_1 e \mathcal{E}_2 são estruturas abstratas isomorfas.

Estruturas abstratas $\langle D, \mathcal{R}, \mathcal{F} \rangle$, tal que $\mathcal{R} = \{=\}$ são chamadas de estruturas algébricas ou simplesmente álgebras.

2.6 Indução Finita

As propriedades principais do conjunto dos números naturais são:

1. Os números naturais podem ser gerados a partir do número natural 0 via a operação de sucessor.
2. Quando uma propriedade numérica ocorre para um número natural, e também ocorre para o próximo número natural da geração, então a propriedade acontecerá para todos os números naturais subsequentes.

A segunda propriedade, chamada de princípio da indução finita, é tão geral que merece uma consideração especial.

Princípio da indução finita *Seja P uma propriedade de números naturais. Se 0 possui a propriedade P , e ainda, quando n possui a propriedade a mesma P juntamente com $n + 1$, então tem-se que todo natural tem a propriedade P .*

O princípio da indução é usado para demonstrar asserções P sobre números naturais, e o procedimento de demonstração tem os seguintes passos:

- (a) Base da indução (BI): mostrar que 0 satisfaz a asserção P .
- (b) Hipótese de indução (HI): supor que o número natural k satisfaz a asserção P , e demonstrar que:
- (c) Passo de indução (PI): $k + 1$ satisfaz a asserção P
- (d) Conclusão da indução (CI): de (a), (b) e (c) concluir que todo natural n satisfaz a asserção P .

Exemplo 2.18 Prove que $0 + 1 + 2 + 3 + \dots + k = \frac{k \cdot (k+1)}{2}$.

BI Se $k = 0$ temos $0 = \frac{0 \cdot (0+1)}{2}$

HI Suponha válido para $k = n$, ou seja $0 + 1 + \dots + n = \frac{n \cdot (n+1)}{2}$

PI Para $k = n + 1$, temos

$$\begin{aligned}
 0 + 1 + 2 + \dots + n + (n + 1) &= \frac{n \cdot (n+1)}{2} + (n + 1) \quad \text{por HI} \\
 &= \frac{n \cdot (n+1)}{2} + \frac{2 \cdot (n+1)}{2} \\
 &= \frac{(n+1) \cdot (n+2)}{2} \\
 &= \frac{(n+1) \cdot ((n+1)+1)}{2}
 \end{aligned}$$

CI A propriedade é válida.

Para que o princípio da indução finita possa ser utilizado na demonstração de uma propriedade de um conjunto devemos ter que este é **bem-ordenado**, como os \mathbb{N} . Um conjunto A é bem-ordenado se existe uma relação de ordem total \preceq para A e todo subconjunto não-vazio X de A possui um único elemento minimal.

Em geral, podemos aplicar o princípio da indução a conjuntos **bem fundados**, um conceito mais abrangente que bem-ordenado. Um conjunto A é bem-fundado se existe uma relação de ordem parcial \preceq para A e todo subconjunto não-vazio X de A possui um elemento minimal. Neste caso pode-se ter um número infinito de elementos minimais. Esta propriedade é fundamental para que seja possível utilizar indução em um conjunto pois caso contrário não se poderia verificar a base de indução, que são os elementos minimais (no caso do conjunto \mathbb{N} , o 0). Para os conjuntos bem-fundados tem-se o seguinte princípio geral da indução, ou indução completa.

Princípio geral da indução *Seja $< A, \preceq >$ uma ordem com A bem-fundado e seja P uma propriedade dos elementos de A . Se todo $l \preceq k$, $l, k \in A$, tem a propriedade P , e k tem a propriedade P então todo $x \in A$ tem a propriedade P .*

Indução é muito utilizada para definir objetos, conjuntos, relações e funções. Isto acontece quando temos uma expressão geral que define tais indivíduos como membros de uma classe ou família. Observe o exemplo a seguir.

Exemplo 2.19 Seja $A = \{\square, \diamond\}$, C o conjunto de *concatenações* dos elementos de A e $F = \{f\}$ onde:

$$\begin{aligned} f(\square x) &= \square x \diamond \\ f(\diamond x) &= \diamond x \square \end{aligned}$$

assim tem-se a seguinte definição de um conjunto chamado O :

- (a) $A = \{\square, \diamond\} \subseteq O$;
- (b) Se $x \in O$ então $f(x) \in O$;
- (c) Os únicos elementos de O são os objetos satisfazendo os itens (a) e (b) acima.

É fácil verificar que $\square\diamond, \diamond\square$ são elementos de O , porém $\square\square$ não é um elemento de O .

No Exemplo 2.19, o conjunto O é indutivo em A . O item a. é a cláusula básica da definição, o item b. é a cláusula de indução e c. a cláusula de fechamento. A é o conjunto básico ou inicial e f é a função geradora. Diz-se também que O é definido por indução a partir de A por f .

Observe no próximo exemplo, a definição indutiva dos números naturais a partir da função sucessor.

Exemplo 2.20 O conjunto dos números naturais \mathbb{N} é uma classe indutiva. O conjunto A é o conjunto $\{0\}$, e a função é a operação de sucessor, que soma 1 a um número natural:

- (a) 0 é um número natural;
- (b) Se a é um número natural então o sucessor de a é um número natural.
- (c) Os únicos números naturais são os objetos satisfazendo os itens (a) e (b) acima

Os exemplos 2.19 e 2.20 obedecem ao esquema geral de definições indutivas, apresentado a seguir.

Definição 2.24

Conjunto Indutivo Sejam D e $A \subset D$ conjuntos, e F um conjunto de funções $f : D^k \rightarrow D, k > 0$. Diz-se que um conjunto B é indutivo em A , se e somente:

1. $A \subseteq B$;
2. Se $a_1, a_2, \dots, a_k \in B$ então $f(a_1, a_2, \dots, a_k) \in B$ para toda $f \in F$;

Fecho Indutivo Diz-se que B é definido por indução se B é a interseção de todos os conjuntos indutivos em A , também chamado “fecho indutivo” de A sob F .

Observe que o fecho indutivo é o menor conjunto indutivo em A . Alternativamente, pode-se ter uma definição mais construtiva do fecho indutivo de um conjunto, como se segue.

Definição 2.25 Sejam D e $A \subset D$ conjuntos, e F um conjunto de funções $f : D^k \rightarrow D$, $k > 0$. Dizemos que um conjunto B é definido por indução em A , se e somente existe uma seqüência B_0, B_1, \dots tal que:

1. $B_0 = A$
2. $B_{i+1} = B_i \cup \{f(a_1, a_2, \dots, a_k) | f \in F, a_1, a_2, \dots, a_k \in B_i\}$
3. $B = \bigcup_{i \geq 0} B_i$

Fica para o leitor a demonstração de que as duas definições acima se equivalem. Frequentemente serão definidos conjuntos de objetos possuindo uma estrutura formal, tal como o conjunto do exemplo 2.19. Pretende-se demonstrar propriedades destes conjuntos.

Exemplo 2.21 Sejam os conjuntos A, C e O como definidos no exemplo 2.19. Então, todo elemento de O é $\underbrace{\square \diamond \dots \diamond}_n$ ou $\diamond \underbrace{\square \dots \square}_n$, $n \in \mathbb{N}$.

- (a) Base de Indução (BI): é válido para o conjunto inicial $A = \{\square, \diamond\}$: basta fazer $n = 0$
- (b) Hipótese de Indução (HI): supondo a_1 satisfaz a propriedade para algum n
- (c) Passo de Indução (PI): então:

1. $f(\underbrace{\square \diamond \dots \diamond}_n) = \underbrace{\square \diamond \dots \diamond \diamond}_{n+1}$; ou
2. $f(\diamond \underbrace{\square \dots \square}_n) = \diamond \underbrace{\square \dots \square \square}_{n+1}$

satisfazem a propriedade para $n + 1$.

- (d) Conclusão de Indução (CI): (a), (b) e (c) satisfazem a asserção.

2.7 Alfabetos e Linguagens

Os dados no computador são organizados através de seqüências de bits, representados pelos símbolos 0 e 1. Sob esta ótica, entende-se como desejável estudar

a matemática de cadeias de símbolos. Desta forma, define-se como alfabeto (Σ) um conjunto, em geral finito, de símbolos (σ) chamados de letras. Entende-se por símbolo todo sinal gráfico satisfazendo os seguintes critérios:

1. As letras devem possuir uma estrutura espacial que facilite sua reprodução e reconhecimento. Por exemplo: $\square, \triangle, |, *$. Como contra-exemplo o leitor pode imaginar figuras complicadas como rubricas pessoais, tais como $\mathfrak{b}\mathfrak{h}$.
2. As letras devem possuir uma estrutura que impossibilite decomposições horizontais. Assim, “ $||$ ” não seria uma escolha apropriada, pois é composta horizontalmente de dois sinais iguais (“ $|$ ” e “ $|$ ”); no entanto, “ $-$ ” e “ $==$ ” seriam escolhas adequadas - apesar de poderem ser decompostas verticalmente.
3. Para a construção de alguns sistemas, existe a necessidade de um suprimento infinito de letras, assim deve-se exigir que elas possam ser produzidas de modo uniforme.

$\Sigma_1 = \{*, |\}$ e $\Sigma_2 = \{\square, \triangle, \odot\}$ satisfazem os critérios 1., 2. e 3. descritos acima.

Exemplo 2.22 O alfabeto romano é dado por $\{a, b, \dots, z\}$, o alfabeto binário é composto por $\{0, 1\}$, e alfabeto hexadecimal por $\{0, 1, \dots, 9, A, B, \dots, F\}$

Expressões ou cadeias são seqüências de letras justapostas horizontalmente, tendo seus limites claramente identificados por inter espaço separador - com mesma função que o espaço em branco na escrita convencional. Assim, uma string em um alfabeto Σ é uma seqüência finita de símbolos deste alfabeto. O comprimento de uma string w , denotado por $|w|$, é a contagem de símbolos pertencentes a string.

Exemplo 2.23 $|101| = 3$ e $|\text{inconstitucionalissimamente}| = 27$

Uma string que não possua nenhum símbolo, isto é, de comprimento 0, é chamada de string vazia, e é denotada por Λ . Assim, seja Σ um alfabeto, uma cadeia (expressão, string) em Σ é:

1. Λ , é uma cadeia nula e seu comprimento $|\Lambda| = 0$, ou

2. uma letra $\sigma \in \Sigma$ e seu comprimento $|\sigma| = 1$.
3. Se x é uma cadeia em Σ e $\sigma \in \Sigma$ uma letra, então $x\sigma$ é uma cadeia em Σ . Se o comprimento $|x| = n$ então o comprimento de $|x\sigma| = n + 1$.

Deste modo, $\square\Delta$ e $\square\odot\square\Delta$ são expressões no alfabeto Σ_2 . Além disto, o conjunto de todas as strings, incluindo Λ , sobre o alfabeto Σ é denotado por Σ^* . Assim, o conjunto Σ^* para $\Sigma = \{a, b\}$ é dado por $\Sigma^* = \{\Lambda, a, b, aa, bb, ab, ba, aaa, \dots\}$

A operação de *concatenação* que se resume a justapor um símbolo a uma cadeia. Além disto, esta operação é associativa. Por exemplo, a concatenação de $\square\Delta$ com \square é a cadeia $\square\Delta\square$. Podemos generalizar esta operação para a concatenação de duas cadeias; assim o comprimento da cadeia resultante é a soma dos comprimentos das cadeias concatenadas. Por exemplo, $\square\Delta$ concatenada com $\square\odot\square\Delta$ é $\square\Delta\square\odot\square\Delta$.

Palavras são expressões que respeitam determinados critérios explícitos para sua formação. Desta forma pode-se explicitar o seguinte critério de formação para as palavras em Σ_1 :

Critério 1: As únicas palavras são as expressões onde não apareçam mais que duas ocorrências sucessivas de “*” e não menos que duas de “|” em seqüência.

Utilizando-se este critério, conclui-se (informalmente) que, $**||*$ é uma palavra, e que $**|*$ não é, pois possui menos que duas ocorrências sucessivas de “|”.

Seja a string w . Um símbolo $\sigma \in \Sigma$ na posição $j \in [1, |w|]$ é chamado de ocorrência de σ , e é denotado por $w(j) = \sigma$. A palavra *casa*, por exemplo, possui as seguintes ocorrências: $w(1) = c$, $w(2) = a$, $w(3) = s$ e $w(4) = a$. É importante ressaltar que as ocorrências $w(2)$ e $w(4)$ estão associadas a um mesmo símbolo a , porém, ainda assim são ocorrências distintas.

O inverso de uma string w , denotado por w^R , é a string escrita de trás para frente. Por exemplo, para a string $w = casa$, tem-se que seu inverso é dado por $w^R = asac$. Neste sentido a definição 2.26 apresenta a formalização deste conceito.

Definição 2.26 O inverso w^R de uma string w é definido como:

1. Se $|w| = 0$ então $w^R = w = \Lambda$;
2. Se $|w| = n + 1 > 0$, $w = ua$, $a \in \Sigma$ então $w^R = au^R$;

Algumas strings recebem nomes particulares de acordo com suas propriedades. As strings de letras nas quais $w^R = w$ são chamadas de palíndromes, como por exemplo, “reviver”, “osso”, “arara”. A cultura popular também consagrou algumas expressões como palíndromes, ainda que não detanhem o rigor matemático aqui apresentado, por conta da presença de espaços em branco. Entre estas, podemos citar “luz azul”, “socorram-me subi no ônibus em Marrocos” e “seco de raiva coloco no colo caviar e doces”.

As strings de números, por sua vez, nas quais $w^R = w$ são chamadas de capicua, como por exemplo, 10022001 (que poderia ser a data 10Fev2001), ou ainda o número 9 no alfabeto binário (1001).

Exemplo 2.24 Mostre que dados duas strings quaisquer x e w , $(xw)^R = w^R x^R$, como por exemplo $(casa)^R = (sa)^R (ca)^R = asac$.

Demonstração por Indução

1. Base de Indução (BI):

$$|x| = 0 \rightarrow x = \Lambda$$

$$(xw)^R = (\Lambda w)^R = w^R = w^R \Lambda = w^R \Lambda^R = w^R x^R$$

2. Hipótese de Indução (HI): $(xw)^R = w^R x^R$ para $|x| = n - 1$

3. Passo de Indução (PI):

$$\text{Seja } y = ax, |y| = n, |x| = n - 1 \text{ e } |a| = 1$$

$$(yw)^R = ((ax)w)^R = (a(xw))^R = (xw)^R a = w^R x^R a = w^R (ax)^R = w^R y^R$$

4. Conclusão de Indução (CI): (1), (2) e (3) demonstram a asserção.

Definição 2.27 [Linguagem] Seja Σ um alfabeto. Então:

1. Define-se o conjunto Σ^* da seguinte maneira: seja $\Sigma^0 = \{\Lambda\}$ e $\Sigma^{i+1} = \Sigma^i \cup \{x\sigma \mid x \in \Sigma^i \text{ e } \sigma \in \Sigma\}$, então $\Sigma^* = \bigcup_{i=0}^{\infty} \Sigma^i$. Ou seja, Σ^* é o conjunto de todas as cadeias em Σ .

2. Diz-se que \mathcal{L} é uma **linguagem** em Σ se e somente se $\mathcal{L} \subseteq \Sigma^*$ (em geral, uma linguagem é um conjunto definido por uma propriedade).

Exemplo 2.25 Dado o alfabeto $\Sigma = \{\square, \triangle, \odot\}$

$$\mathcal{L} = \{\square, \square\triangle, \square\odot, \triangle\}$$

é uma linguagem em Σ .

2.8 Objetos Finitos e Espaços

Um objeto é finito se sua especificação requer apenas uma quantidade finita de informação. Um número natural é um objeto finito que pode ser especificado pela sua representação arábica. Nem todo número real é um objeto finito. Por exemplo, existem números reais transcendentess, cuja especificação poderia ser dada apenas exibindo uma série infinita de termos⁵. Uma n -tupla de objetos finitos é também um objeto finito, desde que pode ser especificada pela enumeração finita de suas n componentes. Uma classe finita de objetos finitos é também um objeto finito, mas uma classe infinita de objetos em geral não é um objeto finito. Um **espaço** é qualquer conjunto X de objetos finitos tal que, dado um objeto finito qualquer, sempre é possível verificar efetivamente sua pertinência a X . Por exemplo:

1. A classe de números naturais;
2. Se A e B são espaços então $A \times B$;
3. A classe dos números naturais divisível por 5.

Note que a classe de funções $f : \mathbb{N} \rightarrow \mathbb{N}$ não é um espaço pois seus objetos não são em geral finitos. Um dos objetivos deste livro é estudar que classe de funções possui representações finitárias, sendo, portanto objetos finitos.

⁵É claro que os transcendentess conhecidos têm uma especificação aparentemente finita, por exemplo π , como sendo a razão entre o dobro do comprimento de qualquer circunferência para o seu raio, mas estamos aqui aceitando como um fato que trata-se de um caso particular e não uma regra.

O objetivo é obter métodos que tornam conceitos abstratos, tais como conjuntos, funções e relações, em objetos com uma representação concreta de natureza simbólica. Um sistema de representação é uma *correspondência* entre objetos simbólicos e objetos conceituais. Por exemplo, os sistemas de representação numéricos fazem corresponder, a cadeias de símbolos, números.

O sistema usual de representação dos naturais é o decimal. Utiliza-se um alfabeto $\Sigma_{10} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, portanto os números naturais positivos são cadeias em Σ_{10}^* . A estas cadeias associamos um valor através de uma função $\nu : \Sigma_{10}^* \rightarrow \mathbb{N}$, da seguinte maneira:

- Para todo $x \in \Sigma_{10}$, $\nu(x) = x$
- Se $x = \sigma_k \sigma_{k-1} \cdots \sigma_1 \sigma_0$ então $\nu(x) = \sum_{i=0}^k \nu(\sigma_i) \times 10^i$

Exemplo 2.26 ⁶ $\nu(\mathbf{1024}) = 1 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0 = 1024$

- 1024 é um número natural, algo de natureza abstrata ou conceitual, representante da classe de todos os conjuntos contendo 1024 objetos.
- **1024** é uma cadeia de símbolos e portanto algo de natureza simbólica.

A diferença entre a representação de um número, uma cadeia, e seu valor fica mais clara se for considerada a representação **Romana**, em que os símbolos utilizados, ou seja, o alfabeto é o conjunto $\{I, V, X, L, C, M\}$. Assim, $\text{valor}(IV) = 4$ mostra, inclusive, que a função de avaliação *valor* é diferente de ν .

A *contagem* dos elementos de um conjunto fornece sua cardinalidade e a *valorização* fornece o valor de uma cadeia. No caso dos sistemas de representação com um símbolo de valor 0 as correspondentes funções ν são não injetivas. Isto é devido ao possível posicionamento do 0 à esquerda, como nas cadeias $\{5, 05, 005, \dots\}$. Nos sistemas n -ádicos, cuja definição apresentaremos a seguir, não existe em Σ_n nenhum símbolo que tenha valor 0, assim a função ν_n é injetiva. Isto permite tratar as cadeias de tais sistemas como representantes únicas de seus respectivos valores, e vice-versa.

⁶a cadeia **1024** está em negrito para acentuar a diferença com o valor abstrato 1024

Definição 2.28 Um sistema de representação n -ádico é constituído de:

- Σ_n : um alfabeto ordenado com n símbolos
- W : conjunto de todas as palavras de Σ_n
- \mathcal{W}_i : conjunto de todas as i -tuplas de W
- W^0 : conjunto contendo apenas a 0-tupla $\langle \rangle$
- ν_n : *função de avaliação*

O conjunto de todas as tuplas, de todas as dimensões, \mathcal{W} é definido como:

$$\begin{aligned}\mathcal{W}_1 &= W^1 &= W \\ \mathcal{W}_{i+1} &= W^{i+1} &= W^i \times W \\ \mathcal{W} &= \bigcup_{i=0}^{\infty} \mathcal{W}_i\end{aligned}$$

A função ν_n é definida como:

$$\begin{aligned}\nu_n() &= 0 \text{ a palavra nula tem valor } 0 \\ \nu_n(s_j) &= j \text{ se } s_j \in \Sigma_n = \{s_1, s_2, s_3, \dots, s_n\} \\ \nu_n(\sigma_k \sigma_{k-1} \dots \sigma_1 \sigma_0) &= \sum_{i=0}^k \nu(\sigma_i) \times n^i\end{aligned}$$

Exemplo 2.27 Define-se um sistema 3-ádico de representação. Naturalmente, é escolhido $\Sigma_3 = \{1, 2, 3\}$. Logo, $W = \Sigma_3^*$. A função ν_3 é definida como:

$$\begin{aligned}\nu_3() &= 0 \text{ a palavra nula tem valor } 0 \\ \nu_3(1) &= 1 \\ \nu_3(2) &= 2 \\ \nu_3(3) &= 3 \\ \nu_3(\sigma_k \dots \sigma_0) &= \nu_3(\sigma_k) \times 3^k \times \dots \times \sigma_0 \times 3^0\end{aligned}$$

Freqüentemente é usada a notação W em vez de W_n , quando ficar claro que se está trabalhando com um alfabeto fixado Σ em que o número de letras não é importante. Os sistemas n -ádicos foram escolhidos como o sistema de representação numérico para os espaços \mathcal{W} pelas seguintes razões:

1. A função ν é uma bijeção.
2. A cadeia vazia 0 corresponde ao 0 dos naturais.

Estas propriedades de um sistema n -ádico, tal como definido, nos possibilitará trabalhar tanto no plano onde serão representados e formalizados os conceitos como no plano onde estes conceitos são formados a partir do processo de abstração [29]. No plano onde os conceitos são formados tanto o resultado das abstrações (no caso a cardinalidade) como o valor (interpretação) é um único conceito, por exemplo as coleções com cinco objetos têm **5** como cardinal, e no plano de representação dos conceitos as cadeias $\{5, 05, 005, \dots\}$ têm valor 5.

Por exemplo, sejam as cadeias 12 e 21 cujos valores são 4 e 5, respectivamente num sistema 2-ádico em $\Sigma_2 = \{1, 2\}$:

- a operação aritmética $12 + 21 = 121$ seria mais confortavelmente calculada, por razões de aculturação, no sistema decimal. Seria efetuada a operação $4 + 5 = 9$ e depois o resultado é convertido para sua representação 121;
- a operação de concatenação de 12 com 21 é mais facilmente realizada simbolicamente, gerando 1221. Posteriormente será apresentado que concatenação também é uma operação aritmética.

2.9 Conclusões e leituras recomendadas

Neste capítulo foi feita uma breve revisão dos pre-requisitos de Matemática Discreta necessários para o estudo de computabilidade. O autor insiste que o leitor deva estar bastante familiarizado com este material antes de prosseguir.

O material apresentado neste capítulo pode ser encontrada em inúmeros textos de Teoria dos Conjuntos. Dentre os clássicos encontram-se [30], [31], [32] e [28]. Para maior aprofundamento em Álgebra Universal tem-se que [33] e [34] são textos com um tratamento extremamente preciso e extensivo. Para um apanhado geral e objetivo da base matemática necessária para um curso de lógica computacional, o leitor deve fazer referência ao segundo capítulo de [35]. Para um curso completo de Estruturas Discretas recomenda-se fortemente [36]. Alguns dos aspectos abordados na seção 2.8 podem ser encontrados em diversos textos de fundamentos, principalmente [37] e [38].

2.10 Exercícios

2.1 Demonstre as propriedades do lema 2.2.

2.2 Para cada um dos conjuntos a seguir, liste (a) os elementos de X e, (b) os elementos de $\mathcal{P}(X)$.

(i) $X = \{1, 2\}$

(ii) $X = \{1, 2, \{1, 2\}\}$

(iii) $X = \{1, 2, \{1, 3\}\}$

2.3 Para $i \in \mathbb{N}$, seja $A_1 = \{0, 1, \dots, i\}$ e seja $B_i = \{0, i\}$. Quem são os conjuntos $\bigcup_i A_i$, $\bigcup_i B_i$, $\bigcap_i A_i$ e $\bigcap_i B_i$?

2.4 Prove que o conjunto dos quadrados dos números pares $A = \{x | \exists y (x = (2y)^2)\}$ é subconjunto próprio do conjunto de múltiplos de quatro $B = \{x | \exists y (x = 4y)\}$.

2.5 Demonstre as propriedades do lema 2.2.

2.6 Prove que uma função $f : A \rightarrow B$ é injetiva se e somente se, para toda a função $g, h : C \rightarrow A$, se $f \circ g = f \circ h$ então $g = h$. Prove que uma função $f : A \rightarrow B$ é sobrejetiva se e somente se, para toda a função $g, h : B \rightarrow C$, se $g \circ f = h \circ f$ então $g = h$.

2.7 Demonstre o lema 2.3.

2.8 Mostre, por contradição, que a função inversa de uma bijeção f , é única.

2.9 Represente graficamente a relação \leq para o conjunto $\{1, 2, 3\}$.

2.10 Seja R uma relação binária reflexiva em A^2 . Seja R^c definida como $R^c = \{ \langle x, y \rangle \in A^2 \mid \langle y, x \rangle \in R \}$. Mostre que $R' = R \cup R^c$ é a menor relação reflexiva e simétrica contendo R . no conjunto $A = \{a, b, c, d\}$.

2.11 Determine se as relações a seguir são simétricas, reflexivas e transitivas:

(a) “Igualdade” = em um conjunto A não vazio.

- (b) “Menor que” $<$ em \mathbb{Z} .
- (c) A relação $R = \{\langle 1, 1 \rangle, \langle 2, 3 \rangle, \langle 3, 1 \rangle\}$ no conjunto $A = \{1, 2, 3\}$.
- (d) A relação $R = \emptyset$ no conjunto $A = \{1, 2, 3\}$.
- (e) A relação $R = \{\langle a, b \rangle, \langle c, d \rangle\}$
- (f) A relação $R = \{\langle x, y \rangle \in \mathbb{Z}^2 \mid x - y \text{ é múltiplo de } 3\}$
- (g) A relação $R = \{\langle x, y \rangle \in A \mid x \text{ é paralela a } y\}$, A é o conjunto de retas em \mathbb{R}^2
- (h) A relação $R = \{\langle x, y \rangle \in \mathbb{N}^2 \mid x, y \text{ são pares}\}$

2.12 Suponha que R e S sejam relações binárias em A . Para cada uma das propriedades a seguir, se R e S possuem tais propriedades, explique se $R \cup S$ também possui. E $R \cap S$?

- (i) Reflexividade
- (ii) Simetria
- (iii) Transitividade

2.13 Demonstre o lema 2.4.

2.14 Sejam R_1 e R_2 relações de equivalência em A . Mostre que, se $R \circ S = S \circ R$, então $R \circ S$ é a menor relação de equivalência contendo R e S .

2.15 Sejam R e S relações em A . Prove:

- (i) $[x]_{(R \cap S)} = [x]_R \cap [x]_S$
- (ii) $[x]_{(R \cup S)} = [x]_R \cup [x]_S$

2.16 Sejam $\langle A, \preceq \rangle$ e $\langle B, \preceq' \rangle$ duas ordens parciais. Uma função $f : A \rightarrow B$ é monotônica se, para todo $a, b \in A$, se $a \preceq b$ então $f(a) \preceq' f(b)$. Mostre que a composição de funções monotônicas é monotônica. Mostre que se f é monotônica e m é o mínimo de um subconjunto S de A , então $f(m)$ é o mínimo de $f(x)$, $x \in S$.

2.17 Prove que não existem cadeias $x \in \{a, b\}^*$ tal que $xa = bx$.

2.18 Prove que $(w^R)^R = w$ para qualquer string w .

2.19 Prove que se v é uma substring de w então v^R é uma substring de w^R .

2.20 Prove que $\sum_{i=1}^n i^4 = \frac{n}{30}(n+1)(2n+1)(3n^2+3n-1)$.

Capítulo 3

Funções Recursivas

Neste capítulo será estudada uma classe especial de funções, as *funções recursivas* que têm como conjuntos de partida \mathbb{N}^n para algum $n > 0$ e como conjunto de chegada \mathbb{N} . A motivação para esta classe de funções é de capturar a intuição sobre funções computáveis. Estas funções serão apresentadas de maneira abstrata e informal, como é usual em matemática.

A partir das funções recursivas serão apresentados certos conceitos matemáticos básicos para o estudo de computabilidade, relacionados aos domínios das funções. Serão abordados os *sistemas de representação*, maneiras de fazer corresponder termos abstratos a *representações simbólicas*. Também será demonstrada a independência entre a propriedade “computável” de uma função e o sistema de representação adotado para o conjunto de partida e chegada, tanto com respeito ao conjunto de símbolos quanto à dimensão.

Sob esta ótica, observe que no início do século XX, a crença na possibilidade de resolução de qualquer problema matemático era amplamente aceita, principalmente devido ao matemático David Hilbert. Sua ambição de *mecanizar* o tratamento de problemas matemáticos contribuiu fortemente para a reformulação da matemática, na direção de sistemas axiomáticos, nos moldes da geometria Euclidiana.

Entretanto, em 1931 Kurt Gödel publicou o resultado da *incompletude da aritmética* estabelecendo a não existência de procedimentos efetivos para julgar a

veracidade ou falsidade de asserções (potenciais teoremas) a respeito da teoria dos números naturais. Na demonstração deste resultado, tão fundamental para a matemática, Gödel utilizou funções cujo cálculo é feito através de procedimentos finitos, com passos bem determinados. Gödel e Jacques Herbrand definiram um conjunto de funções, hoje conhecido como *conjunto das funções recursivas*, que capturam a noção de cálculo ou computação finita.

Em 1936, Alonzo Church demonstrou a equivalência entre o conjunto de funções definido por Gödel e Herbrand, e uma outra caracterização de procedimento efetivo devida a Church e Stephen Kleene, o λ -calculus. Esta equivalência levou Church a conjecturar que a definição de procedimento efetivo é completamente caracterizada pelas funções recursivas. Esta conjectura é conhecida como *Tese de Church* e até hoje é amplamente aceita.

Os formalismos usados para especificar algoritmos podem ser classificados em *Operacional*, *Axiomático* e *Denotacional* (ou *Funcional*). O formalismo Operacional define uma máquina abstrata baseada em estados, em instruções primitivas e na especificação de como cada instrução modifica cada estado. No formalismo Axiomático, associam-se regras às componentes da linguagem. As regras permitem afirmar o que será verdadeiro após a ocorrência de cláusulas, considerando o que era verdadeiro antes da ocorrência. Por fim, o formalismo Denotacional trata-se de uma função construída a partir de funções elementares de forma composicional no sentido em que o algoritmo denotado pela função pode ser determinado em termos de suas funções componentes.

A classe das funções recursivas pode ser representada por uma gama bastante variada de formalismos. Neste capítulo será utilizada uma linguagem básica cuja origem pode ser encontrada em Eilenberg [39] ou em Brainerd [38]. O material apresentado é uma teoria mais *concreta*, ou seja, mais *formal* e construtiva, na qual as funções serão definidas *dentro* de um plano simbólico e de tal modo que os referentes conceituais fiquem bem determinados.

Quando se fala do sucessor de um número natural, consideramos a função $\underline{suc} : \mathbb{N} \rightarrow \mathbb{N}$ tal que $\underline{suc}(x) = x + 1$ como se o conjunto \mathbb{N} fosse totalmente conhecido e pressupondo um conhecimento anterior do significado da operação de somar 1.

É claro que sabemos efetuar tal operação na representação arábica decimal, mas esta representação não foi nem sequer mencionada na definição de *suc*, e ilustra a necessidade premente de um cuidado mais formal. Nos próximos capítulos serão definidas as funções recursivas via sistemas de representação diversificados.

Serão introduzidas uma série de funções que realizam transformações no espaço \mathcal{W} e que servem de base para a construção de linguagens, máquinas e programas. As funções são apresentadas na seguinte ordem: 1) funções iniciais, que servem de base para todas as demais; 2) funcionais, ou seja, funções que geram funções a partir de funções (composição, combinação e expoentização); 3) A partir de 1) e 2) serão construídas funções aritméticas, funções que manipulam *tuplas*, funções para processamento de cadeias e alguns funcionais de particular interesse computacional. Finalmente é apresentado o funcional repetição, que completa a linguagem básica (LB). O leitor deve prosseguir como se estivesse aprendendo uma nova linguagem de programação funcional.

A apresentação das funções é feita paralelamente com o uso ícones, que são chamadas de componentes atômicos e moleculares. Esta linguagem gráfica assemelha-se aos gráficos conhecidos como circuitos estudados em eletricidade.

3.1 Funções Recursivas Primitivas

Uma das maneiras usuais de apresentar funções matemáticas é através de uma *definição recursiva*: um conjunto de valores da função é explicitado e os demais valores são obtidos destes iniciais. Por exemplo, a conhecida sequência de Fibonacci $1, 1, 2, 3, 5, 8, 13, \dots$ é definida como se segue:

$$\begin{aligned} Fib(0) &= 1 \\ Fib(1) &= 1 \\ Fib(x+2) &= Fib(x+1) + Fib(x) \end{aligned}$$

A definição recursiva é simples e possui operações matemáticas pouco custosas, ao contrário de outras formas de representação. Note a diferença significativa

da definição recursiva para a algébrica a seguir:

$$Fib(x) = \frac{\sqrt{5}}{5} \left(\frac{1 + \sqrt{5}}{2} \right)^{x+1} - \frac{\sqrt{5}}{5} \left(\frac{1 - \sqrt{5}}{2} \right)^{x+1}$$

Esta última definição apresenta operações matemáticas extremamente demoradas, tais como divisão, raiz quadrada e exponenciação. Contudo, existe um ponto de inflexão a partir do qual a representação algébrica conduz a um resultado mais rapidamente do que a recursiva. Isto ocorre quando o valor de x é alto, pois neste caso, a recursão precisaria passar por todos os $x - 1$ resultados até computar o valor de $Fib(x)$.

Será utilizado o mesmo método de *definição recursiva* para definir a classe de funções (primitivas) recursivas. Começa-se pela definição das funções iniciais, cuja simplicidade é óbvia, que são de indiscutível *calculabilidade*. As funções iniciais apresentadas nesta seção são formalismos que representam objetos finitos e realizáveis¹.

Definição 3.1 As seguintes funções são chamadas iniciais:

Zero $\underline{zero} : \mathbb{N} \rightarrow \mathbb{N}$, tal que para todo $x \in \mathbb{N}$

$$\underline{zero}() = 0$$

Projeção para cada $n > 0$ e cada $1 \leq i \leq n$

$\underline{pr} : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$, tal que para todo $\underline{x}_n = \langle x_1, x_2, \dots, x_n \rangle \in \mathbb{N}^n$

$$\underline{pr}(n, i, \underline{x}_n) = x_i$$

A função de projeção também é chamada de função de seleção. Além disto, a

$\underline{pr}(1, 1, x) = x$ é muitas vezes é chamada de função identidade $id(x) = x$.

¹São nomes de *processadores* elementares, de um ponto de vista apenas conceitual é que são funções, e portanto abstrações. A idéia é que tais processadores elementares sejam suficientemente simples de tal modo que seja fácil imaginar sua possível existência no plano real, e devem ser suficientes para se constituírem de base para a construção de processadores mais complexos que representam a classe completa das funções recursivas. Assim em todas as definições se está tratando da função (entidade abstrata) que os processadores reais computam e que são representadas por uma linguagem. São usados os mesmos símbolos para representar tais processadores e as correspondentes funções.

Sucessor $\underline{suc} : \mathbb{N} \rightarrow \mathbb{N}$, tal que para todo $x \in \mathbb{N}$

$$\underline{suc}(x) = y, \quad \nu(y) = x + 1$$

Observe que a função adição ainda não foi definida na hierarquia de funções recursivas, por isto não deve ser utilizada na definição da função $\underline{suc}(x)$. O leitor deve entender que o resultado desta função é y , cuja avaliação significa o elemento posterior a x em uma enumeração qualquer, que aqui foi simplificada representado por $x + 1$.

As funções iniciais são representadas pelas seguintes componentes atômicas:

$$\text{zero} \dots \quad \boxed{z} \rightarrow \quad \text{projção} \dots \quad \boxed{\pi} \rightarrow \quad \text{sucessor} \dots \quad \bullet \rightarrow \boxed{s} \rightarrow$$

A idéia é que $\{ \boxed{z} \rightarrow, \bullet \rightarrow \boxed{s} \rightarrow, \boxed{\pi} \rightarrow \}$ é o conjunto de componentes básicas atômicas para a construção de máquinas mais complexas. Estas componentes são fornecidas como caixas pretas obedecendo às especificações dadas pelas definições abstratas. Pode-se também imaginar $\boxed{z} \rightarrow$ como um agente que cria um registro contendo a palavra nula 0 cujo valor é 0. As demais componentes apenas transformam ou destroem registros criados anteriormente. Assim, segundo esta interpretação, apenas $\boxed{z} \rightarrow$ tem utilidade quando considerada isoladamente.

Os números naturais, por exemplo, podem ser definidos como uma sucessão de componentes básicas atômicas justapostas uma após a outra:

0 : função primitiva recursiva $\underline{zero}()$, isto é, $\boxed{z} \rightarrow$.

1 : função primitiva recursiva $\underline{suc}(\underline{zero}())$, isto é, $\boxed{z} \rightarrow \bullet \rightarrow \boxed{s} \rightarrow$.


2 : função primitiva recursiva $\underline{suc}(\underline{suc}(\underline{zero}()))$, isto é, $\boxed{z} \rightarrow \bullet \rightarrow \boxed{s} \rightarrow \bullet \rightarrow \boxed{s} \rightarrow$.

n : função primitiva recursiva $\underbrace{\underline{suc}(\dots \underline{suc}(\underline{zero}()) \dots)}_n$, isto é, $\boxed{z} \rightarrow \underbrace{\bullet \rightarrow \boxed{s} \rightarrow \dots \bullet \rightarrow \boxed{s} \rightarrow}_n$.

De modo análogo, os valores booleanos também podem ser definidos como uma sucessão de componentes básicas atômicas justapostas uma após a outra:

Falso : função primitiva recursiva $\underline{zero}()$, isto é, $\boxed{z} \rightarrow$.

Verdadeiro : função primitiva recursiva $\underline{suc}(\underline{zero}())$, isto é, .

A função identidade $id(x) = x$, apesar de ser derivada de $\underline{pr}(1, 1, x) = x$ e não ser uma função inicial, também terá uma componente básica atômica de modo a simplificar a representação. Esta componente será .

A seguir são definidas maneiras de *construir* funções a partir de funções anteriormente construídas, no caso básico, as iniciais.

Definição 3.2 [Recursão Primitiva] Sejam $f : \mathbb{N}^n \rightarrow \mathbb{N}$ e $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$, diz-se que $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ é definida por recursão primitiva se os valores de h são obtidos por

$$\begin{aligned} h(0, x_1, \dots, x_n) &= f(x_1, \dots, x_n) \\ h(y + 1, x_1, \dots, x_n) &= g(y, h(y, x_1, \dots, x_n), x_1, \dots, x_n) \end{aligned}$$

Exemplo 3.1 A adição de números naturais pode ser definida por:

$$\begin{aligned} 0 + x &= x \\ (y + 1) + x &= (y + x) + 1 \end{aligned}$$

Assim, toma-se $f = \underline{pr}(1, 1, ??) : \mathbb{N}^3 \rightarrow \mathbb{N}$ e $g = \underline{suc} \circ \underline{pr}(3, 2, ??, ??, ??) : \mathbb{N}^5 \rightarrow \mathbb{N}$, obtidas por composição a partir das funções iniciais, pode-se escrever:

$$\begin{aligned} h(0, x) &= \underline{soma}(0, x) = f(x) = \underline{pr}(1, 1, x) \\ h(y + 1, x) &= \underline{soma}(y + 1, x) = g(y, h(y, x), x) = \underline{suc}(\underline{pr}(3, 2, y, \underline{soma}(y, x), x)) \\ &= \underline{suc}(\underline{soma}(y, x)) \end{aligned}$$

É fácil ver que para todo x e todo y , $\underline{soma}(x, y) = x + y$, portanto \underline{soma} é a adição de números naturais. E, além disso \underline{soma} foi obtida por composição e recursão primitiva a partir das funções iniciais. Neste caso, partindo da segunda parcela, faz-se sucessor tantas vezes quantas forem indicadas pela primeira parcela. Note também que para calcular a soma de dois números, por exemplo 3 e 5 seria computado:

$$\begin{aligned} h(3, 5) &= 3 + 5 \\ \underline{soma}(0, 5) &= \underline{pr}(1, 1, 5) = 5 \\ \underline{soma}(1, 5) &= \underline{suc}(\underline{pr}(3, 2, 0, \underline{soma}(0, 5), 5)) = \underline{suc}(\underline{soma}(0, 5)) = \underline{suc}(5) = 6 \\ \underline{soma}(2, 5) &= \underline{suc}(\underline{pr}(3, 2, 1, \underline{soma}(1, 5), 5)) = \underline{suc}(\underline{soma}(1, 5)) = \underline{suc}(6) = 7 \\ \underline{soma}(3, 5) &= \underline{suc}(\underline{pr}(3, 2, 2, \underline{soma}(2, 5), 5)) = \underline{suc}(\underline{soma}(2, 5)) = \underline{suc}(7) = 8 \end{aligned}$$

Definição 3.3 [Função Recursiva Primitiva] Uma função $f : \mathbb{N}^n \rightarrow \mathbb{N}$ é *recursiva primitiva* se f for uma função inicial ou for obtida pela utilização de composição e recursão primitiva a partir das funções iniciais.

Para mostrar que uma função $f : \mathbb{N}^n \rightarrow \mathbb{N}$ é recursiva primitiva, basta exibir a sua definição informal como feito com a adição. Assim para a multiplicação tem-se:

$$\begin{array}{lcl} 0.x & = & 0 \\ (y+1).x & = & y.x + x \end{array} \quad \left\| \quad \begin{array}{lcl} h(0, x) & = & f(x) \\ h((y+1), x) & = & g(y, h(y, x), x) \end{array} \right.$$

sem precisar explicitar a forma

$$\begin{aligned} \underline{produto}(0, x) &= \underline{zero}() \\ \underline{produto}(y+1, x) &= \underline{soma}(\underline{pr}(3, 2, y, \underline{produto}(y, x), x), \underline{pr}(3, 3, y, \underline{produto}(y, x), x)) \\ &= \underline{soma}(\underline{produto}(y, x), x) \end{aligned}$$

que obedece a definição formal de recursão primitiva.

Exemplo 3.2 Definição informal das seguintes funções:

Predecessor \underline{pred} , função que retorna o antecessor de um número de uma lista \underline{x} de n objetos.

$$\begin{aligned} \underline{pred}(0) &= 0 \\ \underline{pred}(x) &= \underline{pr}(\underline{suc}(n), x, \underline{zero}(), \underline{x}) \end{aligned}$$

Note que para calcular o predecessor de 3, seria computado:

$$\underline{pred}(3) = \underline{pr}(\underline{suc}(n), 3, 0, 1, 2, 3, 4, \dots, n)$$

Subtração própria \underline{monus} , função cujo resultado é $(x - y)$ se $x \geq y$ e 0 de $x < y$

$$\begin{aligned} \underline{monus}(x, 0) &= \underline{pr}(1, 1, x) \\ \underline{monus}(x, y+1) &= \underline{pred}(\underline{monus}(x, y)) \end{aligned}$$

Note que para calcular $3 - 2$, deve ser computado:

$$\begin{aligned} \underline{monus}(3, 0) &= 3 \\ \underline{monus}(3, 1) &= \underline{pred}(\underline{monus}(3, 0)) = \underline{pred}(3) = 2 \\ \underline{monus}(3, 2) &= \underline{pred}(\underline{monus}(3, 1)) = \underline{pred}(2) = 1 \end{aligned}$$

Lema 3.1 As funções a seguir são recursivas primitivas:

(a) $\underline{exp}(x, y) = x^y, y \geq 0$

(b) $\underline{!}(x) = x!$

(c) $\underline{somatório}(x) = \sum_{n=0}^x n$

(d) $\underline{modulo}(x, y) = |x - y| = \begin{cases} x - y & \text{se } x \geq y \\ y - x & \text{se } x < y \end{cases}$

(e) $\underline{diferente}(x, y) = \begin{cases} 1 & \text{se } x \neq y \\ 0 & \text{se } x = y \end{cases}$

(f) $\underline{igual}(x, y) = \begin{cases} 1 & \text{se } x = y \\ 0 & \text{se } x \neq y \end{cases}$

(g) $\underline{menorigual}(x, y) = \begin{cases} 1 & \text{se verdadeiro} \\ 0 & \text{se falso} \end{cases}$

(h) $\underline{menor}(x, y) = \begin{cases} 1 & \text{se verdadeiro} \\ 0 & \text{se falso} \end{cases}$

(i) $\underline{maior}(x, y) = \begin{cases} 1 & \text{se verdadeiro} \\ 0 & \text{se falso} \end{cases}$

(j) $\underline{maiorigual}(x, y) = \begin{cases} 1 & \text{se verdadeiro} \\ 0 & \text{se falso} \end{cases}$

(k) $\underline{max}(x, y) = \text{maior número entre } x \text{ e } y$

(l) $\underline{min}(x, y) = \text{menor número entre } x \text{ e } y$

(m) $\underline{quo}(x, y) = x \text{ div } y$

(n) $\underline{not}(x) = \begin{cases} 1 & \text{se } x = 0 \\ 0 & \text{se qualquer outro valor de } x \end{cases}$

(o) $\underline{and}(x, y)$ onde $x, y = 0$ ou 1

(p) $\underline{or}(x, y)$ onde $x, y = 0$ ou 1

$$(q) \ \underline{if}(x, y, z) \begin{cases} y & \text{se } x \neq 0 \\ z & \text{se } x = 0 \end{cases}$$

(r) $\underline{mod}(x, y)$, resto da divisão de x por y

(s) $\underline{for}(i, n, c) = \text{for}(k = i; k \leq n; k++)c;$

(t) $\underline{dowhile}(c, i, n) = \text{do } c \ \text{while}(i \leq n);$

Lema 3.2 Seja $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$. As funções:

$$(a) \cdots \sum_{y=0}^{y=m} f(\underline{x}_n, y) = f(\underline{x}_n, 0) + f(\underline{x}_n, 1) + \cdots + f(\underline{x}_n, m)$$

$$(b) \cdots \prod_{y=0}^{y=m} f(\underline{x}_n, y) = f(\underline{x}_n, 0) \cdot f(\underline{x}_n, 1) \cdot \cdots \cdot f(\underline{x}_n, m)$$

$$(c) \cdots \bigvee_{y=0}^{y=m} f(\underline{x}_n, y) = f(\underline{x}_n, 0) \vee f(\underline{x}_n, 1) \vee \cdots \vee f(\underline{x}_n, m)$$

$$(d) \cdots \bigwedge_{y=0}^{y=m} f(\underline{x}_n, y) = f(\underline{x}_n, 0) \wedge f(\underline{x}_n, 1) \wedge \cdots \wedge f(\underline{x}_n, m)$$

onde $x \vee y = \vee(x, y)$ e $x \wedge y = \wedge(x, y)$, são recursivas primitivas

Demonstração:

(a) Seja a função g definida por

$$\begin{aligned} g(0) &= f(0) \\ g(x+1) &= g(x) + f(x+1) \end{aligned}$$

É claro que

$$g(n) = \sum_{x=0}^{x=n} f(x)$$

(b) semelhante à (a) bastando tomar g como sendo:

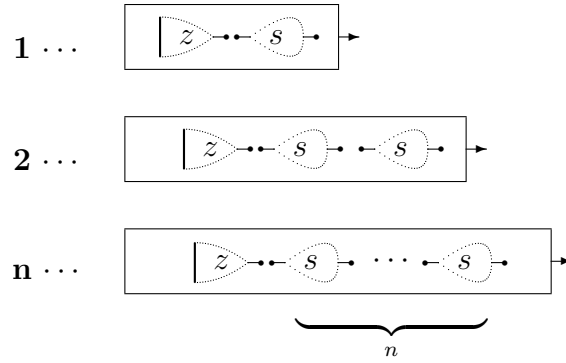
$$\begin{aligned} g(0) &= f(0) \\ g(x+1) &= g(x) \cdot f(x+1) \end{aligned}$$

(c),(d) Sugere-se que o leitor demonstre.

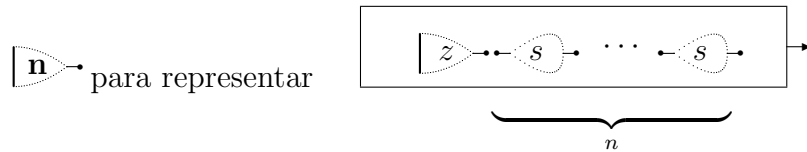
Se for desejado criar espaços de representação e transformá-los deve-se buscar envelopamentos que permitam juntar tais componentes atômicas em objetos estruturados. Se for considerado juntar componentes e objetos já preparados em *série*, pode-se obter componentes moleculares com estrutura linear.

Exemplo 3.3 A função constante $f(x) = 2$ pode ser obtida por composição como $f(x) = \underline{suc}(\underline{suc}(\underline{zero}()))$

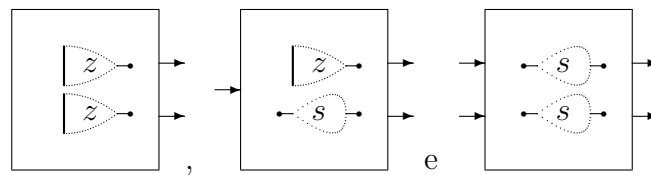
Assim pode-se ter estruturas como a seguir:



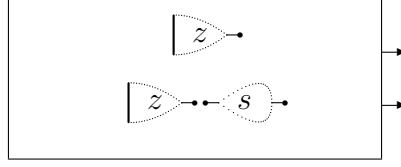
Estas componentes moleculares foram obtidas via uma *disposição linear* em que a saída de uma componente atômica é usada como entrada para a próxima, da esquerda para a direita. Note que se pode assim criar registros contendo todos os elementos do espaço \mathcal{W} . Estas componentes moleculares representam os números naturais e pode-se utilizar figuras mais simples,



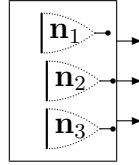
Uma outra maneira de se obter componentes moleculares é por estruturação vertical ou em *paralelo*, por exemplo:



que representam combinações em paralelo de 2 componentes atômicas. Pode-se ter combinações em paralelo de componentes moleculares e atômicas:



Pode-se ter componentes moleculares para representar qualquer tupla $\langle x_1, x_2, \dots, x_r \rangle$ de \mathcal{W}_r . Por exemplo, para $r = 3$ e tomando $\nu(x_i) = n_i$, tem-se que a componente molecular a seguir cria três registros contendo n_1 , n_2 e n_3 .



Para obter funções entre espaços quaisquer utiliza-se esquemas de construção de funções a partir de funções. Em geral, se houver uma classe \mathcal{F} de funções definidas em um conjunto A_1 e tomando valores em um conjunto A_2 , diz-se que \mathcal{F} é uma subclasse da classe $A_2^{A_1}$, que é a classe de todas as funções de A_1 em A_2 . Um funcional é qualquer função que tome como parâmetros outras funções, por exemplo F de $(A_2^{A_1})^n$ em $A_2^{A_1}$. No caso particular que está sendo estudado, tem-se funções em $\mathcal{W}^{\mathcal{W}}$. Assim nestes esquemas de construção de funções os funcionais são de $(\mathcal{W}^{\mathcal{W}})^n$ em $\mathcal{W}^{\mathcal{W}}$.

Definição 3.4 [Composição] Sejam as funções $f : \mathbb{N}^m \rightarrow \mathbb{N}$, $g_1, \dots, g_m : \mathbb{N}^n \rightarrow \mathbb{N}$. A composição h de f com g_1, \dots, g_m é a função $h : \mathbb{N}^n \rightarrow \mathbb{N}$, definida por:

$$h(x_1, \dots, x_n) = f(g_1(x_1, \dots, x_n), \dots, g_m(x_1, \dots, x_n))$$

A composição de uma função f com uma outra g é usualmente denotada por $f \circ g$, e ilustrada na Figura 3.1.

Definição 3.5 Sejam as funções $f : \mathcal{W}_r \rightarrow \mathcal{W}_s$ e $g : \mathcal{W}_s \rightarrow \mathcal{W}_t$. A composição de f e g é a função $g \circ f : \mathcal{W}_r \rightarrow \mathcal{W}_t$ definida por:

$$g \circ f(\underline{x}_r) = g(f(\underline{x}_r))$$

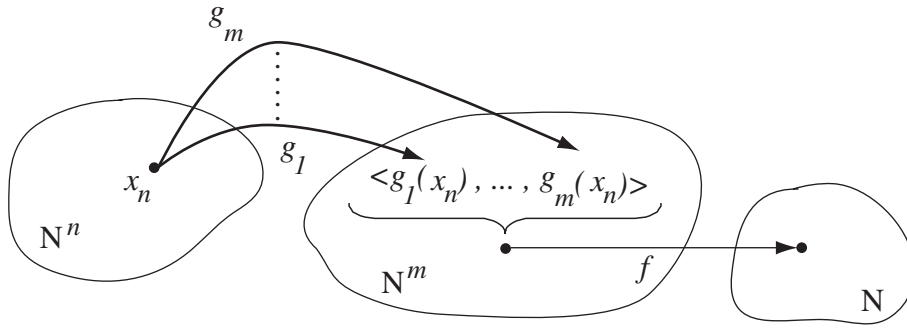
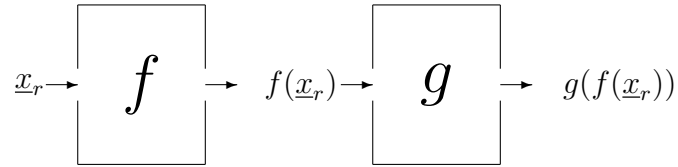


Figura 3.1: Composição de funções.

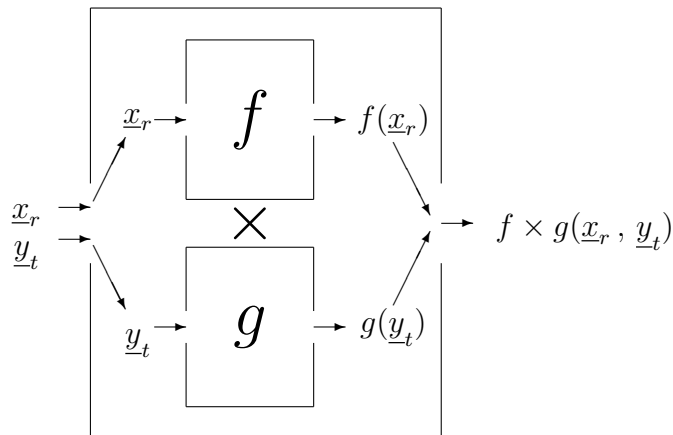


Exemplo 3.4 A partir das funções iniciais pode-se obter, pelo uso da composição:

$$\begin{aligned}
 s \circ z &= 1 \\
 s \circ s \circ z &= 2 \\
 \underbrace{s \circ s \circ \dots \circ s}_n \circ z &= n
 \end{aligned}$$

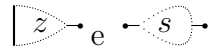
Definição 3.6 Sejam as funções $f : \mathcal{W}_r \rightarrow \mathcal{W}_s$ e $g : \mathcal{W}_t \rightarrow \mathcal{W}_u$. A combinação de f com g é a função $f \times g : \mathcal{W}_{r+t} \rightarrow \mathcal{W}_{s+u}$, definida por:

$$f \times g(\underline{x}_r, \underline{y}_t) = \langle f(\underline{x}_r), g(\underline{y}_t) \rangle$$



Todas as tuplas de \mathcal{W}_2 podem agora ser representadas, do mesmo modo que são representados os elementos de \mathcal{W}_1 . Assim $\langle 0, 0 \rangle$ é representado por $z \times z$. As tuplas de \mathcal{W}_3 também podem ser representadas, por exemplo $\langle 0, 3, 1 \rangle$ é representado por $z \times s \circ s \circ s \circ z \times s \circ z$. Em geral, os pontos do espaço \mathcal{W}_k são representados por $n_1 \times n_2 \times \dots \times n_k$ e os n_i por $s^{n_i} \circ z$. Note que os pontos do espaço \mathcal{W} agora podem ser representados como r -tuplas de palavras no alfabeto Σ , ou como componentes moleculares, ou ainda como expressões formadas por composição e combinação das funções iniciais.

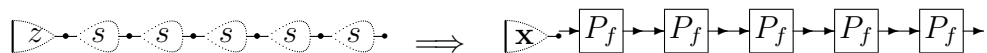
Todas as tuplas do espaço \mathcal{W} podem ser realizadas (dependendo de interpretação) no plano real por objetos compostos pelas componentes atômicas



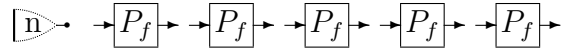
através dos esquemas de composição (componentes arranjados em série) e combinação (componentes arranjados em paralelo). Por exemplo, considerando o alfabeto $\Sigma = \{\square, \diamond\}$, com $\nu_2(\square) = 1$ e $\nu_2(\diamond) = 2$, pode-se construir $\mathcal{W}_1 = \{\square, \diamond, \square\square, \square\diamond, \dots\}$. Assim, tem-se que $\nu_2(\square\diamond) = 4$ e sua componente molecular é $\boxed{z} \rightarrow \bullet \bullet \bullet \bullet \rightarrow$. De um modo geral uma palavra de \mathcal{W}_1 de valor n corresponde ao componente molecular $\boxed{z} \rightarrow \underbrace{\bullet \bullet \bullet \bullet \rightarrow}_n$.

Note que não existe circularidade nesta representação, pois o n é utilizado como recurso metalinguístico para explicar a representação. Assim tem-se representações finitárias para objetos reais, seja através da linguagem de representação numérica, seja como componente molecular.

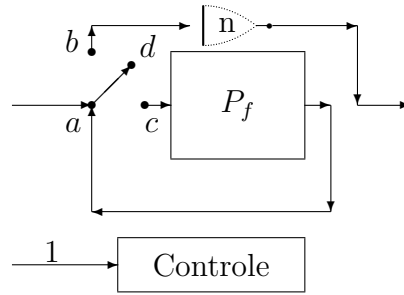
Se for considerado uma componente P_f realizando $f : \mathcal{W}_1 \rightarrow \mathcal{W}_1$ e deseja-se reaplicar tal função, por exemplo 5 vezes a uma certa palavra x de \mathcal{W}_1 , pode-se realizar a componente correspondente substituindo $\boxed{z} \rightarrow$ por $\boxed{x} \rightarrow$ e cada $\bullet \bullet \bullet \bullet \rightarrow$ pela componente P_f .



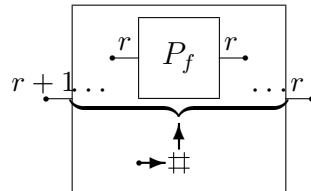
Esta substituição corresponderia a reaplicar a função f a x . Para se obter a função que seria aplicada a uma entrada qualquer x basta substituir $\boxed{z} \rightarrow$ por $\boxed{n} \rightarrow$, obtendo-se assim a componente molecular



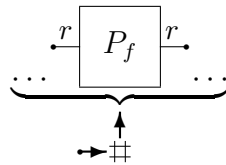
A figura a seguir representa uma maneira em que a operação de substituição pode ser realizada utilizando-se apenas um componente molecular correspondendo a P_f e o componente atômico \boxed{n} . A chave em a está ligada inicialmente na posição d . O controle recebe como entrada a representação do número de vezes que P_f deve ser reaplicado. Se a entrada for \boxed{z} então a chave em a é ligada na a posição b ; se a entrada for diferente de 0 então o controle liga a chave para a posição c e em seguida retorna a chave para a posição d aguardando a próxima entrada atômica. Isto é como ligar a chave cada vez que se passa uma conta do ábaco para a direita.



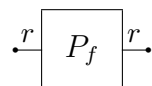
Pode-se representar esta figura de modo mais simples por



onde indica-se o número de entradas para P_f . O 1 somado a r significa mais uma entrada para o controle, e



é para ser entendido como a repetição do componente P_f , o número de vezes indicado por #.



Se $P_f^\# : \mathcal{W}_{r+1} \rightarrow \mathcal{W}_1$ for a função computada por este componente, então tem-se:

$$\begin{aligned} P_f^\#(\underline{x}_r, 0) &= \underline{x}_r \\ P_f^\#(\underline{x}_r, y + 1) &= P_f(P_f^\#(\underline{x}_r, y)) \end{aligned}$$

evidenciando que esta maneira de realizar este tipo de repetição de uma função coincide com o esquema de recursão primitiva. Mais tarde este tópico será rediscutido. Por enquanto define-se o esquema de obtenção de novas funções que é motivado por esta discussão.

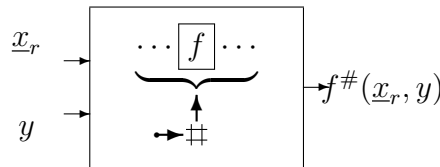
Definição 3.7 [Expoentização] Seja a função $f : \mathcal{W}_r \rightarrow \mathcal{W}_r$. A expoentização de f é a função $f^\# : \mathcal{W}_{r+1} \rightarrow \mathcal{W}_r$, definida por:

$$\begin{aligned} f^\#(\underline{x}_r, 0) &= \underline{x}_r \\ f^\#(\underline{x}_r, y + 1) &= f(f^\#(\underline{x}_r, y)) \end{aligned}$$

Diz-se, também, que $f^\#$ é definida por recursão primitiva a partir de f . Observe que:

$$\begin{aligned} f^\#(x, 0) &= x \\ f^\#(x, 1) &= f(x) \\ f^\#(x, 2) &= f(f(x)) \\ &\vdots \end{aligned}$$

A componente molecular para a expoentização de uma função f é representada pela figura:



A rigor seria necessário também dizer que a função *expoentização* é a composição da função f consigo mesma, tantas vezes quanto o valor de y , aplicada a x_r . Logo, se f é computável, $f^\#$ também é computável.

Informalmente, as funções recursivas primitivas são aquelas obtidas das funções iniciais e aplicação dos esquemas de composição (\circ), combinação (\times) e expoentização ($\#$), o que gera uma coleção muito grande de funções.

Definição 3.8 [Funções recursivas primitivas]

1. Diz-se que uma função $f : \mathcal{W} \rightarrow \mathcal{W}$ é recursiva primitiva se existe uma seqüência f_1, \dots, f_n de funções em $\mathcal{W}^{\mathcal{W}}$ tal que $f = f_n$ e para todo i , $1 \leq i < n$ f_i é uma função inicial ou f_i satisfaz um dos itens abaixo:

$$(I) \quad f_i = f_j \circ f_k \quad 1 \leq j, k < i$$

$$(II) \quad f_i = f_j \times f_k \quad 1 \leq j, k < i$$

$$(III) \quad f_i = (f_j)^\# \quad 1 \leq j < i$$

2. Seja F um funcional de $(\mathcal{W}^{\mathcal{W}})^n$ em $\mathcal{W}^{\mathcal{W}}$. Diz-se que F é recursivo primitivo se e somente se para toda $f \in \text{dom}(F)$, se f recursiva primitiva então $F(f)$ é recursiva primitiva.

Seja $f = \iota^\# \circ (s \circ z \times s \circ z)$, então a seguinte seqüência de funções evidencia que f é uma função recursiva primitiva:

$$\begin{array}{ll} f_1 & \cdots \quad z \quad \cdots (\textit{inicial}) \\ f_2 & \cdots \quad s \quad \cdots (\textit{inicial}) \\ f_3 & \cdots \quad s \circ z \quad \cdots (f_1 \circ f_2) \\ f_4 & \cdots \quad s \circ z \times s \circ z \quad \cdots (f_3 \times f_3) \\ f_5 & \cdots \quad \iota \quad \cdots (\textit{inicial}) \\ f_6 & \cdots \quad \iota^\# \quad \cdots (f_5^\#) \\ f_7 & \cdots \quad (\iota)^\# \circ (s \circ z \times s \circ z) \quad \cdots (f_6 \circ f_4) \end{array}$$

Chega-se a um ponto em que se tem esboçada uma linguagem de programação, definida a seguir.

3.2 A Linguagem Básica

Nesta seção será introduzida uma linguagem de programação chamada de linguagem básica- ∇ ou $LB-\nabla$. Esta linguagem é naturalmente obtida ao considerarem-se as expressões formadas a partir das funções iniciais π, z, s e os esquemas de composição \circ , combinação \times , expoentização $\#$. O esquema de repetição ∇ é definido na seção 3.8 e estende a linguagem posteriormente.

Definição 3.9 Um programa $LB - \nabla$ é uma expressão que satisfaz a seguinte definição formal:

$$\begin{aligned}
< \text{expressão} > &::= < \text{inicial} > \mid \\
& \quad (< \text{expressão} > \circ < \text{expressão} >) \mid \\
& \quad (< \text{expressão} > \times < \text{expressão} >) \mid \\
& \quad < \text{expressão} >^\# \\
< \text{inicial} > &::= \pi \mid z \mid s
\end{aligned}$$

e outras condições que não foi formulada gramaticalmente. Por exemplo, $s \circ \pi$ satisfaz a especificação dada para expressão $LB - \nabla$, mas não é um programa em $LB - \nabla$, tendo em vista que $s : \mathcal{W}_1 \rightarrow \mathcal{W}_1$ e $\pi : \mathcal{W}_1 \rightarrow \mathcal{W}_0$, e portanto, s não pode ser aplicada sobre o domínio \mathcal{W}_0 resultante de π . Tais restrições são de natureza semântica e sempre dizem respeito à composição de funções.

Cada expressão $LB - \nabla$ está associada a uma função, como é definido a seguir.

Definição 3.10 Seja α uma expressão $LB - \nabla$, então a função computada por α denotada por $|\alpha|$ é definida da seguinte maneira:

1. Se $\alpha \equiv \iota$ então $|\alpha|(x) = x$;
2. Se $\alpha \equiv z$ então $|\alpha|(<>) = 0$;
3. Se $\alpha \equiv \pi$ então $|\alpha|(x) = <>$;
4. Se $\alpha \equiv s$ então $|\alpha|(x) = x + 1$;
5. Se $\alpha \equiv (\beta \circ \gamma)$, $\beta \in \mathcal{W}_s^{\mathcal{W}_r}$ e $\gamma \in \mathcal{W}_t^{\mathcal{W}_s}$ então

$$|\alpha|(\underline{x}_r) = |\beta|(|\gamma|(\underline{x}_r))$$

6. Se $\alpha \equiv (\beta \times \gamma)$, $\beta \in \mathcal{W}_s^{\mathcal{W}_r}$ e $\gamma \in \mathcal{W}_t^{\mathcal{W}_v}$ então

$$|\alpha|(\underline{x}_r, \underline{y}_v) = < |\beta|(\underline{x}_r), |\gamma|(\underline{x}_v) >$$

7. Se $\alpha \equiv \beta^\#$, $\beta \in \mathcal{W}_r^{\mathcal{W}_r}$ então

$$\begin{aligned}
|\alpha|(\underline{x}_r, 0) &= \underline{x}_r \\
|\alpha|(\underline{x}_r, y + 1) &= |\beta|(|\alpha|(\underline{x}_r, y))
\end{aligned}$$

8. As únicas funções computadas por expressões $LB - \nabla$ são expressões em $LB - \nabla$ que satisfazem 1 a 7 acima.

Nas seções a seguir são implementadas nesta linguagem uma série de funções mais complexas, todas elas recursivas primitivas, já que esta linguagem só implementa funções desta natureza. O método seguido é o seguinte:

1. Apresenta-se cada função f de maneira informal;
2. Apresenta-se o lema afirmando que f é recursiva primitiva, de tal modo que na demonstração do lema surge a implementação de f . A rigor, tal demonstração deve constar de duas partes:
 - (a) prova de que a implementação de f concorda com sua definição;
 - (b) seqüência de funções de acordo com a definição 3.8.

Apenas para as primeiras funções seguiu-se rigorosamente a demonstração do lema. Na maioria dos casos apenas exibe-se a implementação deixando para o leitor completar os detalhes.

3.3 Funções Aritméticas

A seguir será construído um conjunto de funções aritméticas baseadas na linguagem $LB - \nabla$. Fica evidente a dificuldade de compor e combinar funções definidas e tomando valores em espaços diversos. Por exemplo, seja $f, g, h : \mathcal{W}_1 \rightarrow \mathcal{W}_1$. Para computar $f \circ h$ e $g \circ h$ é interessante computar $(f \times g) \circ h$, porém h só tem uma saída. A função δ abaixo resolve este problema.

Definição 3.11 Define-se a função $\delta : \mathcal{W}_1 \rightarrow \mathcal{W}_2$ como:

$$\delta(x) = \langle x, x \rangle$$

Lema 3.3 δ é recursiva primitiva.

Demonstração:

1. Temos que $\delta = (s \times s)^\# \circ (z \times z \times \iota)$, que provamos por indução.

$$(a) \quad \delta(0) = (s \times s)^\# \circ (z \times z \times \iota)(0) = (s \times s)^\#(0, 0, 0) = \langle 0, 0 \rangle$$

(b) Suponha que $\delta(x) = \langle x, x \rangle$, então:

$$\begin{aligned} \delta(x+1) &= (s \times s)^\# \circ (z \times z \times \iota)(x+1) = (s \times s)^\#(0, 0, x+1) = \\ &= (s \times s) \circ (s \times s)^\#(0, 0, x) = (s \times s) \circ (s \times s)^\#(z \times z \times \iota)(x) = (s \times s) \circ \delta(x) = \\ &= (s \times s)(x, x) = \langle x+1, x+1 \rangle. \end{aligned}$$

2. A seqüência f_1, \dots, f_8 a seguir completa a demonstração que δ é recursiva primitiva, pela definição 3.8:

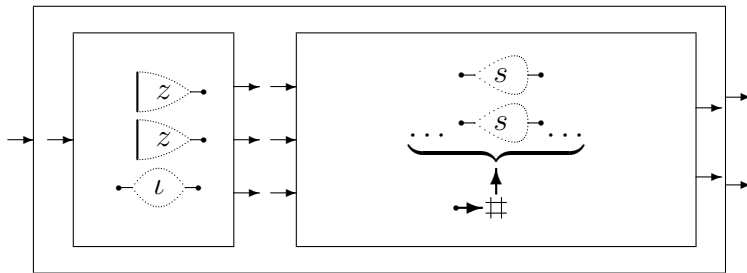
f_1	\dots	z	\dots	$(inicial)$
f_2	\dots	$z \times z$	\dots	$(f_1 \times f_1)$
f_3	\dots	ι	\dots	$(inicial)$
f_4	\dots	$(z \times z) \times \iota$	\dots	$(f_2 \times f_3)$
f_5	\dots	s	\dots	$(inicial)$
f_6	\dots	$s \times s$	\dots	$(f_5 \times f_5)$
f_7	\dots	$(s \times s)^\#$	\dots	$(f_6^\#)$
f_8	\dots	$(s \times s)^\# \circ ((z \times z) \times \iota)$	\dots	$(f_7 \circ f_4)$

□

Utiliza-se a seguinte componente para representar δ



que corresponde à componente molecular



Para definir as funções aritméticas é necessário fazer a análise dos algoritmos aritméticos que dificilmente são explicitamente utilizados. Começa-se pela soma.

Somar dois números, por exemplo m e n , é incrementar o primeiro (segundo) de 1 tantas vezes quanto é o valor do segundo (primeiro). Isto é o mesmo que aplicar a função sucessor s ao primeiro (segundo) número tantas vezes consecutivas quanto é o valor do segundo (primeiro).

É importante observar que a função sucessor é a única das funções iniciais que modifica o valor da representação, ou seja se $s(x) = y$ então $\nu(y) = \nu(x) + 1$, ou melhor $\nu(s(x)) = \nu(x) + 1$. Claro que se está considerando uma *base* fixa, ou ainda que o alfabeto de símbolos é fixado. Assim:

$$+(x, y) = \overbrace{s(s(\cdots (s(x)) \cdots))}^{\nu(y)}$$

Vê-se, portanto, que se tem na linguagem funcional uma maneira de representar isto através da exponetização.

Definição 3.12 Define-se $+: \mathcal{W}_2 \rightarrow \mathcal{W}_1$ como:

$$\begin{aligned} +(x, 0) &= x \\ +(x, s(y)) &= s(+(x, y)) \end{aligned}$$

Lema 3.4 $+: \mathcal{W}_2 \rightarrow \mathcal{W}_1$ é recursiva primitiva.

Demonstração:

1. Tem-se que $+= s^\#$, o que é provado por indução.

$$(a) \quad +(x, 0) = s^\#(x, 0) = x$$

(b) Suponha que $+(x, y) = s^\#(x, y) = x + y$, então:

$$+(x, y + 1) = s^\#(x, y + 1) = s(s^\#(x, y)) \text{ e por hipótese de indução:}$$

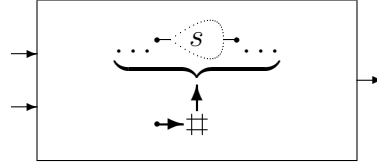
$$s(s^\#(x, y)) = s(+(x, y)) = (x + y) + 1$$

2. A seqüência $f_1, \cdots f_2$ a seguir completa a demonstração que $+$ é recursiva primitiva, pela definição 3.8:

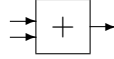
$$\begin{array}{llll} f_1 & \cdots & s & \cdots (inicial) \\ f_2 & \cdots & s^\# & \cdots (f_1^\#) \end{array}$$

□

A componente molecular de $+$ é



que é representado por



O leitor deve considerar que esta maneira de representar funções aritméticas dispensa o uso de variáveis. Esta observação é que torna esta linguagem funcional, isto é, as funções são objetos com representações finitárias tão boas quanto às que se está acostumado a ter para os números naturais. Aqui, um dos objetivos principais é evidenciar que, se uma função é *calculável* ou *computável*, então ela tem uma representação finitária.

Veja agora como realizar o produto de naturais, tendo a seguinte definição.

Definição 3.13 Definimos $. : \mathcal{W}_2 \rightarrow \mathcal{W}_1$ como

$$\begin{aligned} .(x, 0) &= 0 \\ .(x, y + 1) &= +(. (x, y), x) \end{aligned}$$

Aparentemente deve-se repetir o argumento anterior, utilizado para a soma: o produto de dois números naturais x e y é o resultado da aplicação da função soma, que já se tem como um objeto, ao primeiro número x tantas vezes quanto é o valor do segundo número y . Assim:

$$\times = +^{\#} \dots (E_1)$$

A representação E_1 está incorreta. No caso da soma utiliza-se a função s , que necessita apenas de um argumento, ou seja é uma função de aridade 1. Portanto, o resultado obtido em cada momento da seqüência de aplicações é suficiente para continuar. No caso do produto, devemos reiterar a aplicação da soma que precisa

de dois argumentos. Veja que já na primeira aplicação ambos os valores estão destruídos, restando apenas o resultado.

A solução do problema da destruição de *dados* já deve ser familiar em computação. Deve-se preservar os valores que não possam ser destruídos, criando um espaço para guardar os resultados obtidos. Esta é a idéia central de um acumulador. Antes de resolver o problema do produto de dois números naturais, será construído um novo dispositivo computacional, utilizando apenas os recursos que já se tem, ou seja, as funções iniciais e os esquemas de composição, combinação e expoentização.

Definição 3.14 Seja $f : \mathcal{W}_2 \rightarrow \mathcal{W}_1$. Define-se a função acumulador de f , denotada por f_α , por $f_\alpha(x, y) = \langle f(x, y), y \rangle$

Exemplo 3.5 $+_\alpha(x, y) = \langle +(x, y), y \rangle$.

Lema 3.5 Seja $f : \mathcal{W}_2 \rightarrow \mathcal{W}_1$. Se f é recursiva primitiva então a função acumulador de $f : \mathcal{W}_2 \rightarrow \mathcal{W}_1$, f_α é recursiva primitiva.

Demonstração:

1. Tem-se que $f_\alpha = (f \times \iota) \circ (\iota \times \delta)$, o que é provado a seguir:

$$\begin{aligned} f_\alpha(x, y) &= (f \times \iota) \circ (\iota \times \delta)(x, y) \\ &= (f \times \iota)(x, y, y) \\ &= \langle f(x, y), y \rangle \end{aligned}$$

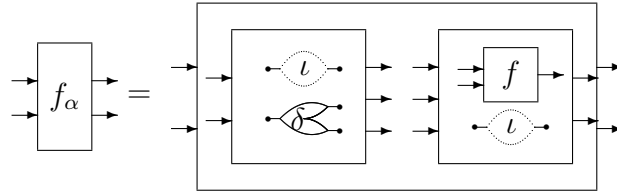
2. A seqüência de funções abaixo completam a demonstração que f_α é recursiva

primitiva, pela definição 3.8:

$$\begin{array}{llll}
f_1 & & & \\
\vdots & & & \\
f_j & \cdots & f & \\
f_{j+1} & \cdots & z & \cdots (inicial) \\
f_{j+2} & \cdots & z \times z & \cdots (f_{j+1} \times f_{j+1}) \\
f_{j+3} & \cdots & \iota & \cdots (inicial) \\
f_{j+4} & \cdots & (z \times z) \times \iota & \cdots (f_{j+2} \times f_{j+3}) \\
f_{j+5} & \cdots & s & \cdots (inicial) \\
f_{j+6} & \cdots & (s \times s) & \cdots (f_{j+5} \times f_{j+5}) \\
f_{j+7} & \cdots & (s \times s)^\# & \cdots (f_{j+6}^\#) \\
f_{j+8} & \cdots & (s \times s)^\# \circ ((z \times z) \times \iota) & \cdots (f_{j+7} \circ f_{j+4}) \\
f_{j+9} & \cdots & f \times \iota & \cdots (f_j \times f_{j+3}) \\
f_{j+10} & \cdots & \iota \times \delta & \cdots (f_{j+3} \times f_{j+8}) \\
f_{j+11} & \cdots & (f \times \iota) \circ (\iota \times \delta) & \cdots (f_{j+9} \circ f_{j+10})
\end{array}$$

□

A componente molecular para o funcional acumulador é:

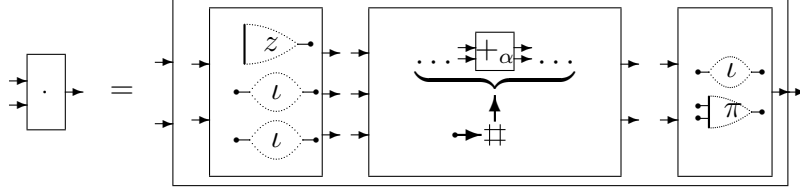


Agora com o auxílio do funcional acumulador, pode-se apresentar a implementação do produto.

Lema 3.6 $. : \mathcal{W}_2 \rightarrow \mathcal{W}_1$ é recursiva primitiva.

Demonstração: Tem-se que $. = (\iota \times \pi) \circ +_\alpha^\# \circ (z \times \iota \times \iota)$. Deixa-se para o leitor verificar esta igualdade e exibir uma dedução para $. .$ □

A componente molecular para $. .$ é:



A próxima função fornece o predecessor de seu argumento. Esta função é utilizada na implementação da subtração.

Definição 3.15 Definimos a função $\circ_1 : \mathcal{W}_1 \rightarrow \mathcal{W}_1$ como

$$\circ_1(x) = \begin{cases} x - 1 & \text{se } x \geq 1 \\ 0 & \text{se } x = 0 \end{cases}$$

Lema 3.7 \circ_1 é uma função recursiva primitiva.

Demonstração: A idéia da implementação é uma função que gera a partir de um número natural x a seqüência:

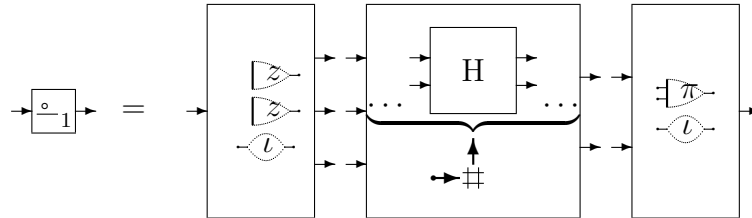
$$\langle 0, 0 \rangle, \langle 1, 0 \rangle, \langle 2, 1 \rangle, \dots, \langle x, x - 1 \rangle$$

e então do último par da seqüência obtém-se o resultado por *projeção*, que neste caso é a função $\pi \times \iota$. Deixa-se a verificação de que a expressão a seguir é uma possível representação para \circ_1 .

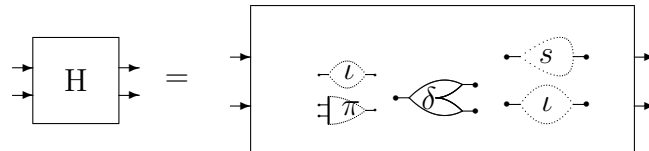
$$\circ_1 = (\pi \times \iota) \circ (((s \times \iota) \circ \delta \circ (\iota \times \pi))^{\#} \circ (z \times z \times \iota))$$

□

A componente molecular para \circ_1 é:



onde



Prossegue-se apresentando a subtração.

Definição 3.16 Define-se a função $\circ : \mathcal{W}_2 \rightarrow \mathcal{W}_1$ como

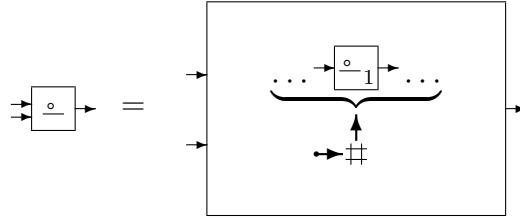
$$\circ(x, y) = \begin{cases} x - y & \text{se } x \geq y \\ 0 & \text{se } x < y \end{cases}$$

Lema 3.8 $\circ : \mathcal{W}_2 \rightarrow \mathcal{W}_1$ é uma função recursiva primitiva.

Demonstração: Tem-se que $\circ = \circ_1^\#$. Deixa-se para o leitor verificar esta igualdade e exibir uma dedução para \circ .

□

A correspondente componente molecular é:



3.4 Manipulação de Tuplas

A necessidade de lidar com argumentos de funções motiva a introdução de funções $f : \mathcal{W} \rightarrow \mathcal{W}$, que aplicadas a uma r-tupla $\langle x_1, x_2, \dots, x_r \rangle$, geram uma s-tupla $\langle x_{i_1}, x_{i_2}, \dots, x_{i_s} \rangle$, $i_1, i_2, \dots, i_s \in \{1, 2, 3, \dots, r\}$. Estas funções fazem reordenações e redimensionamentos da tupla da entrada. Um exemplo que foi discutido é a função δ . Faz-se referência a este tipo de função como *funções arranjo*.

Definição 3.17 Definimos as funções arranjo $f : \mathcal{W}_r \rightarrow \mathcal{W}_s$ como:

$$f(x_1, x_2, \dots, x_r) = \langle x_{i_1}, x_{i_2}, \dots, x_{i_s} \rangle$$

onde $i_1, i_2, \dots, i_s \in \{1, 2, 3, \dots, r\}$ e utiliza-se a seguinte notação:

$$\begin{pmatrix} & & & r & & \\ i_1 & i_2 & \cdots & i_{s-1} & i_s & \end{pmatrix} (x_1, x_2, \dots, x_r) = \langle x_{i_1}, x_{i_2}, \dots, x_{i_s} \rangle$$

Exemplo 3.6 Apresenta-se a seguir alguns casos de função arranjo.

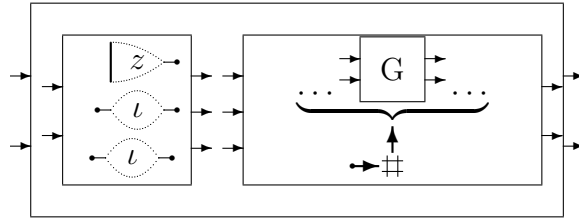
- Função diagonal: $\delta : \mathcal{W}_1 \rightarrow \mathcal{W}_2$, $\delta(x) = \langle x, x \rangle$

$$\delta = \begin{pmatrix} & 1 \\ 1 & \end{pmatrix}$$

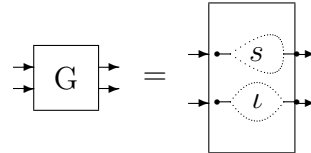
- Função troca: ${}_1\xi_2 : \mathcal{W}_2 \rightarrow \mathcal{W}_2$, ${}_1\xi_2(x_1, x_2) = \langle x_2, x_1 \rangle$

$${}_1\xi_2 = \begin{pmatrix} & 2 \\ 2 & 1 \end{pmatrix}$$

Verifique que ${}_1\xi_2 = (s \times \iota)^\# \circ (z \times \iota \times \iota)$. A correspondente componente molecular é:



onde



- $\iota_r : \mathcal{W}_r \rightarrow \mathcal{W}_r$, $\iota_r(x_1, x_2, \dots, x_r) = \langle x_1, x_2, \dots, x_r \rangle$

$$\iota_r = \begin{pmatrix} & r \\ 1 & 2 & \dots & r-1 & r \end{pmatrix}$$

- $\pi_r : \mathcal{W}_r \rightarrow \mathcal{W}_{r-1}$, $\pi_r(x_1, x_2, \dots, x_r) = \langle x_1, x_2, \dots, x_{r-1} \rangle$

$$\pi_r = \begin{pmatrix} & r \\ 1 & 2 & \dots & r-1 \end{pmatrix}$$

Lema 3.9 As funções arranjo são funções recursivas primitivas.

Demonstração: Inicialmente, pode-se observar que a função arranjo aplicada a uma $r - tupla$ x_1, x_2, \dots, x_r copia os elementos i_1, i_2, \dots, i_t da $r - tupla$. Assim, inicialmente deve-se conseguir funções $\theta_j^k : \mathcal{W}_k \rightarrow \mathcal{W}_{k+1}$ que copiam o j -ésimo elemento de uma $k - tupla$ como $k + 1$ -ésimo elemento, ou seja:

$$\theta_j^k(x_1, x_2, \dots, x_j, \dots, x_k) = \langle x_1, x_2, \dots, x_j, \dots, x_k, x_j \rangle$$

Pode-se obter a tupla $\langle x_1, x_2, \dots, x_r, x_{i_1}, x_{i_2}, \dots, x_{i_t} \rangle$ a partir da tupla $\langle x_1, x_2, \dots, x_r \rangle$, pela composição de θ_j^k 's, ou seja:

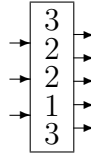
$$\theta_{i_t}^{r+t-1} \circ \dots \circ \theta_{i_2}^{r+1} \circ \theta_{i_1}^r(x_1, x_2, \dots, x_r) = \langle x_1, x_2, \dots, x_r, x_{i_1}, x_{i_2}, \dots, x_{i_t} \rangle$$

agora basta eliminar as r primeiras componentes. Assim:

$$\begin{pmatrix} & r & \\ i_1 & \dots & i_t \end{pmatrix} = (\overbrace{\pi \times \dots \times \pi}^r \times \overbrace{\iota \times \dots \times \iota}^t) \circ \theta_{i_t}^{r+t-1} \circ \dots \circ \theta_{i_2}^{r+1} \circ \theta_{i_1}^r$$

A síntese das funções θ_j^k fica como exercício para o leitor. □

Cada função arranjo tem uma correspondente componente molecular, e indica-se tal componente de forma simplificada. Por exemplo:



é a componente molecular para a função arranjo

$$\begin{pmatrix} & 3 & \\ 3 & 2 & 2 & 1 & 3 \end{pmatrix}$$

3.5 Funcionais Especiais

Uma das estruturas de programação mais comuns, e que ocorre basicamente em todas as linguagens de programação mais amigáveis, é a estrutura de comando *if then else*. Este dispositivo já faz parte da cultura de qualquer programador. Ao final desta seção, será construído o funcional caso para simulá-lo. Inicialmente, apresenta-se alguns funcionais auxiliares na construção principal.

Freqüentemente deseja-se aplicar duas funções à mesma $r - tupla$, isto pode ser conseguido através do funcional definido a seguir.

Lema 3.10 Sejam as funções $f : \mathcal{W}_r \rightarrow \mathcal{W}_s$ e $g : \mathcal{W}_r \rightarrow \mathcal{W}_t$. Se f e g são recursivas primitivas então $\langle f, g \rangle : \mathcal{W}_r \rightarrow \mathcal{W}_{s+t}$, definida por

$$\langle f, g \rangle (\underline{x}_r) = \langle f(\underline{x}_r), g(\underline{x}_r) \rangle$$

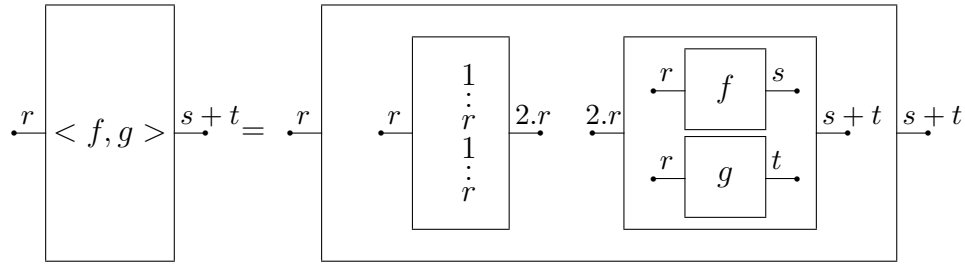
é recursiva primitiva.

Demonstração: É fácil ver que:

$$\langle f, g \rangle = (f \times g) \circ \begin{pmatrix} & r \\ 1 & \dots & r & 1 & \dots & r \end{pmatrix}$$

□

A correspondente componente molecular é:



Pode-se também aplicar uma função binária a uma $r - tupla$ e uma $1 - tupla$ fixada.

Lema 3.11 Seja $f : \mathcal{W}_2 \rightarrow \mathcal{W}_1$ uma função recursiva primitiva, então a função $\boxed{f}_r : \mathcal{W}_{r+1} \rightarrow \mathcal{W}_r$, definida por:

$$\boxed{f}_r(\underline{x}_r, y) = \langle f(x_1, y), f(x_2, y), \dots, f(x_r, y) \rangle$$

é recursiva primitiva.

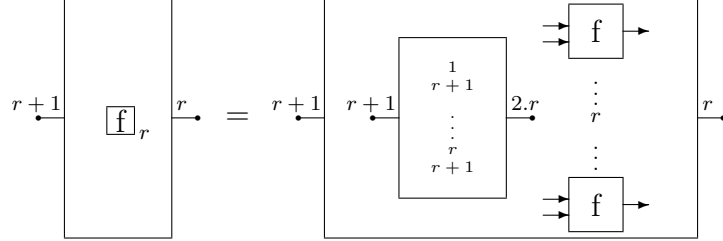
Demonstração:

$$\boxed{f}_r = (\overbrace{f \times f \cdots f \times f}^r) \circ \begin{pmatrix} & r+1 \\ 1 & r+1 & 2 & r+1 & \dots & r & r+1 \end{pmatrix}$$

□

Exemplo 3.7 $\boxplus_r(\underline{x}_r, y) = \langle +(x_1, y), +(x_2, y), \dots, +(x_r, y) \rangle$

A correspondente componente molecular é:



Às vezes deseja-se operar uma função binária com todos os elementos de duas r -tuplas. O lema a seguir mostra que isto pode ser feito preservando recursão primitiva.

Lema 3.12 Seja $f : \mathcal{W}_2 \rightarrow \mathcal{W}_1$ uma função recursiva primitiva, então a função $\boxplus_r : \mathcal{W}_{r+r} \rightarrow \mathcal{W}_r$, definida por:

$$\boxplus_r(\underline{x}_r, \underline{y}_r) = \langle f(x_1, y_1), f(x_2, y_2), \dots, f(x_r, y_r) \rangle$$

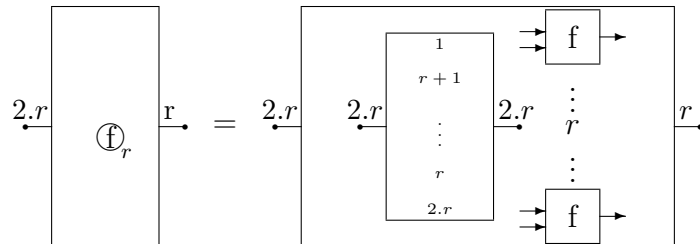
é recursiva primitiva.

Demonstração:

$$\boxplus_r = \overbrace{(f \times f \cdots f \times f)}^r \circ \begin{pmatrix} & & 2.r \\ 1 & r+1 & 2 & r+2 & \cdots & r & 2r \end{pmatrix}$$

□

A correspondente componente molecular é:



Exemplo 3.8

$$\oplus_r(\underline{x}_r, \underline{y}_r) = \langle +(x_1, y_1), +(x_2, y_2), \dots, +(x_r, y_r) \rangle$$

$$\odot_r(\underline{x}_r, \underline{y}_r) = \langle \cdot(x_1, y_1), \cdot(x_2, y_2), \dots, \cdot(x_r, y_r) \rangle$$

$$\exp_r(\underline{x}_r, \underline{y}_r) = \langle \exp(x_1, y_1), \exp(x_2, y_2), \dots, \exp(x_r, y_r) \rangle$$

Lema 3.13 Sejam as funções $f : \mathcal{W}_r \rightarrow \mathcal{W}_1$ e $g : \mathcal{W}_r \rightarrow \mathcal{W}_s$. Se f e g são recursivas primitivas então a função, definida por

$$f \rightarrow g(\underline{x}_r) = \begin{cases} g(\underline{x}_r) & \text{se } f(\underline{x}_r) > 0 \\ \underline{0}_s & \text{se } f(\underline{x}_r) = 0 \end{cases}$$

é recursiva primitiva.

Demonstração: Seja:

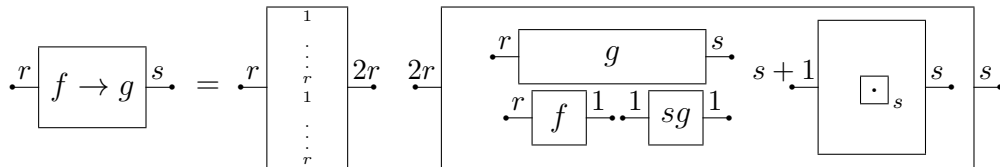
$$sg(x) = \begin{cases} 1 & \text{se } x > 0 \\ 0 & \text{se } x = 0 \end{cases}$$

Verifique que:

$$f \rightarrow g = \boxed{\cdot}_s \circ (\langle g, sg \circ f \rangle)$$

Deixa-se a demonstração de que sg é recursiva primitiva para o leitor. \square

A correspondente componente molecular é:



Os funcionais definidos até agora preparam a apresentação de um funcional muito útil, a que será chamado de caso para motivar a associação de seu entendimento com a estrutura de programação *case* presente na maioria das linguagens de programação.

Definição 3.18 [funcional caso] Sejam as funções $f : \mathcal{W}_r \rightarrow \mathcal{W}_1$, $g_1, g_2 : \mathcal{W}_s \rightarrow \mathcal{W}_t$, define-se caso : $\mathcal{W}_1^{\mathcal{W}_r} \times \mathcal{W}_t^{\mathcal{W}_s} \times \mathcal{W}_t^{\mathcal{W}_s} \rightarrow \mathcal{W}_t^{\mathcal{W}_{r+s}}$, por

$$\underline{caso}(f, g_1, g_2)(\underline{x}_r, \underline{y}_s) = \begin{cases} g_1(\underline{y}_s) & \text{se } f(\underline{x}_r) = 0 \\ g_2(\underline{y}_s) & \text{se } f(\underline{x}_r) > 0 \end{cases}$$

Lema 3.14 Se $f : \mathcal{W}_r \rightarrow \mathcal{W}_1$, $g_1, g_2 : \mathcal{W}_s \rightarrow \mathcal{W}_t$ são recursivas primitivas então $\underline{caso}(f, g_1, g_2)$ é recursiva primitiva.

Demonstração: Seja

$$\overline{sg}(x) = \begin{cases} 0 & \text{se } x > 0 \\ 1 & \text{se } x = 0 \end{cases}$$

Note que:

$$\underline{caso}(f, g_1, g_2)(\underline{x}_r, \underline{y}_s) = \oplus_t(\square_t(g_1(\underline{y}_s), \overline{sg}(f(\underline{x}_r))), \square_t(g_2(\underline{y}_s), sg(f(\underline{x}_r))))$$

pois toda vez que $f(\underline{x}_r) = 0$, seu sinal sg é 0, logo todas as componentes de $g_1(\underline{y}_s)$ serão multiplicadas por 1, enquanto as componentes de $g_2(\underline{y}_s)$ serão multiplicadas por 0. Na soma componente a componente, 0 será somado às componentes de $g_1(\underline{y}_s)$, portanto o resultado final será exatamente $g_1(\underline{y}_s)$. Se $f(\underline{x}_r) > 0$, vale o raciocínio contrário. Portanto

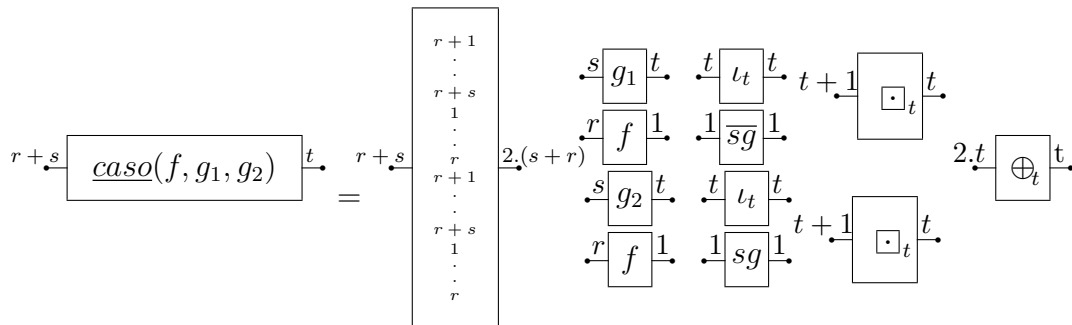
$$\underline{caso}(f, g_1, g_2) = \oplus_t \circ (\square_t \times \square_t) \circ (\iota_t \times \overline{sg} \times \iota_t \times sg) \circ (g_1 \times f \times g_2 \times f) \circ H$$

onde

$$H = \begin{pmatrix} & & & r+s \\ r+1 \cdots r+s & 1 \cdots r & & r+1 \cdots r+s & 1 \cdots r \end{pmatrix}$$

Deixa-se a prova de que \overline{sg} é recursiva primitiva para o leitor. \square

A correspondente componenete molecular de \underline{caso} é:



3.6 Processamento de Cadeias

Usualmente as linguagens de programação oferecem tipos de dados, de tal modo que as operações aritméticas pertencem a um tipo bem determinado, por

exemplo *Integer*, e as operações sobre cadeias pertencem a um outro tipo, por exemplo *String*. Entretanto no nível de máquina estes tipos se unificam em seqüências de 0's e 1's. O leitor deve ter em mente esta distinção para compreender o que se segue.

As funções a seguir manipulam cadeias. É importante observar que todas as funções aritméticas vistas até agora, foram implementadas sem um sistema de representação fixado. Apenas a definição da função inicial s contém o pressuposto de que o sistema é n -ádico² e $\nu(s(x)) = \nu(x) + 1$. Verifique que para qualquer operação aritmética \odot , $\nu(\odot(x, y)) = \nu(x) \odot \nu(y)$, ou seja, ν é um isomorfismo entre os planos simbólico e conceitual. Deseja-se que todas as demais operações sobre cadeias gozem desta propriedade. Para tanto deve-se fixar o número de letras do alfabeto Σ do sistema de representação.

Como visto no capítulo 2 dado um alfabeto Σ , uma cadeia em Σ é:

- (1) Λ é uma cadeia nula e seu comprimento é 0, ou
- (2) uma letra $\sigma \in \Sigma$ e seu comprimento é 1.
- (3) Se x é uma cadeia em Σ e $\sigma \in \Sigma$, então $x\sigma$ é uma cadeia em Σ . Se o comprimento de x é n então o comprimento de $x\sigma$ é $n + 1$.
- (4) As únicas cadeias em Σ são as expressões satisfazendo os critérios (1), (2) e (3).

Esta definição motiva uma ordem na geração de cadeias, a partir da cadeia 0 e da ordenação do alfabeto Σ . Por exemplo para $\Sigma_2 = \{1, 2\}$ temos as seguintes 20 primeiras cadeias:

$\Lambda, 1, 2, 11, 12, 21, 22, 111, 112, 121, 122, 211, 212, 221, 222, 1111, 1112, 1121, 1122$

Contando da esquerda para a direita, a partir de Λ , a cadeia x de ordem k é tal que $\nu(x) = k$. Então a função $s(x)$ poderá ser computada, tanto pegando a cadeia seguinte a x , quanto pegando o elemento de ordem $\nu(x) + 1$ na lista. Daqui por diante será fixado n como o número de letras do alfabeto Σ .

²Num sistema decimal s não seria uma função, pois $s(x) = y, s(x) = 0y, s(x) = 00y$ e assim por diante, satisfazem $\nu(s(x)) = \nu(x) + 1$. Teria-se que adicionar a condição de que y é o menor possível.

Definição 3.19 [cadeia inicial e símbolo final]

1. Define-se $\underline{inic} : \Sigma^* \rightarrow \Sigma^*$ como:

$$\underline{inic}(x) = \begin{cases} x' & \text{se } x = x'\sigma, \sigma \in \Sigma \\ \Lambda & \text{se } x = \Lambda \end{cases}$$

2. Define-se $\underline{fim} : \Sigma^* \rightarrow \Sigma$ como:

$$\underline{fim}(x) = \begin{cases} \sigma & \text{se } x = x'\sigma, \sigma \in \Sigma \\ \Lambda & \text{se } x = \Lambda \end{cases}$$

Por exemplo: $\underline{inic}(1223) = 122$, $\underline{fim}(1223) = 3$.

Lema 3.15 $\underline{inic} : \Sigma^* \rightarrow \Sigma^*$ e $\underline{fim} : \Sigma^* \rightarrow \Sigma$ são funções recursivas primitivas.

Demonstração:

1. A função \underline{inic} é obtida por operações aritméticas. Observando que, se $x = \sigma_k \sigma_{k-1} \cdots \sigma_1 \sigma_0$ então $\underline{inic}(x) = \sigma_k \sigma_{k-1} \cdots \sigma_1$. Como

$$\nu(x) = \left(\sum_{j=0}^{j=k} \nu(\sigma_j) \cdot n^j \right)$$

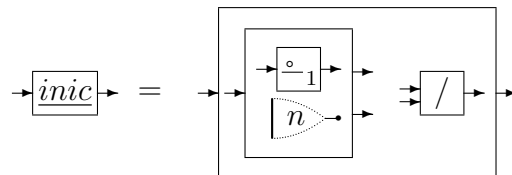
e

$$\nu(\underline{inic}(x)) = \left(\sum_{j=1}^{j=k} \nu(\sigma_j) \cdot n^{j-1} \right)$$

e portanto

$$\nu(\underline{inic}(x)) = \frac{\nu(x) - \nu(\sigma_0)}{n}$$

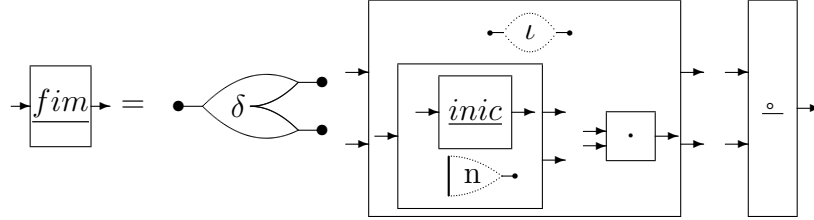
Como $\underline{inic}(x)$ não é injetora, já que para todos os possíveis valores de σ_0 temos o mesmo valor de $\underline{inic}(x)$, basta calcular para $\sigma_0 = 1$, o menor valor em Σ . Logo tem-se que $\underline{inic}(x) = (x - 1)/n$, ou seja $\underline{inic} = / \circ (\overset{\circ}{-}_1 \times n)$, bastando demonstrar que a divisão inteira ($/$) é recursiva primitiva (ver a lista de exercícios). A correspondente componente molecular é:



2. Temos que $\underline{fim}(x) = x \circ \underline{inic}(x).n$, e portanto

$$\underline{fim} = \circ \circ (\iota \times \cdot \circ (\underline{inic} \times n)) \circ \delta$$

A correspondente componente molecular é:



□

Definição 3.20 [comprimento da cadeia]

Define-se $\underline{compr} : \Sigma^* \rightarrow \Sigma^*$ como

$$\underline{compr}(x) = \begin{cases} 0 & \text{se } x = \Lambda \\ \underline{compr}(x') + 1 & \text{se } x = x'\sigma, \sigma \in \Sigma \end{cases}$$

Por exemplo: $\underline{compr}(1221) = 4$.

Lema 3.16 $\underline{compr} : \Sigma^* \rightarrow \Sigma^*$ é uma função recursiva primitiva.

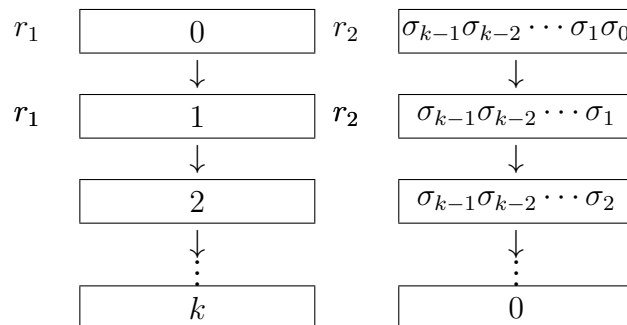
Demonstração: Se $\underline{compr}(x) = k$ então k é o número de símbolos de Σ em x , ou seja $x = \sigma_{k-1}\sigma_{k-2} \cdots \sigma_1\sigma_0$. A idéia intuitiva para computar $\underline{compr}(x)$ é a seguinte:

(a) Cria-se dois registros r_1 e r_2 que conterão 0 e x , respectivamente.

$$r_1 \quad \boxed{0} \quad r_2 \quad \boxed{\sigma_{k-1}\sigma_{k-2} \cdots \sigma_1\sigma_0}$$

(b) Para cada retirada à direita do registro r_2 de um símbolo adiciona-se 1 ao registro

r_1



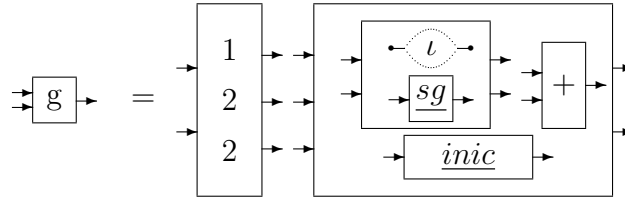
Não se sabe de antemão quando o registro r_2 está vazio, assim o algoritmo deve ser tal que repete (b) até que isto aconteça. Define-se uma função $g : \mathcal{W}_2 \rightarrow \mathcal{W}_2$ por

$$g(x, y) = \begin{cases} \langle x + 1, \underline{inic}(y) \rangle & \text{se } y \neq 0 \\ \langle x, y \rangle & \text{se } y = 0 \end{cases}$$

Deixa-se para o leitor verificar que

$$g = ((+ \circ (\iota \times \underline{sg})) \times \underline{inic}) \circ \begin{pmatrix} 2 \\ 1 & 2 & 2 \end{pmatrix}$$

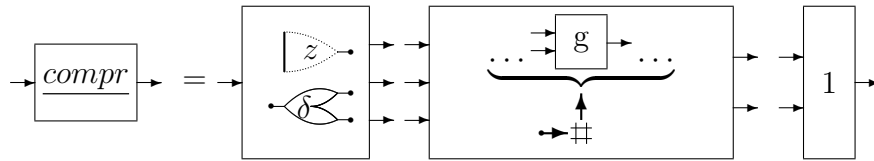
satisfaz a definição de g . A componente molecular para g é:



Notando que $\nu(x) \geq \underline{compr}(x)$, utiliza-se $\nu(x)$ como limite superior do número de vezes que g será aplicado, logo tem-se que

$$\underline{compr}(x) = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \circ g^\# \circ (z \times \delta)(x)$$

cuja correspondente componente molecular é:



Assim chega-se à expressão:

$$\underline{compr} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \circ (((+ \circ (\iota \times \underline{sg})) \times \underline{inic}) \circ \begin{pmatrix} 2 \\ 1 & 2 & 2 \end{pmatrix})^\# \circ (z \times \delta)$$

□

Definição 3.21 [concatenação]

Define-se $\frown: \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ como

$$\begin{aligned}\frown(x, 0) &= x \\ \frown(x, y\sigma) &= \frown(x, y)\sigma\end{aligned}$$

Por exemplo: $\frown(122, 112) = 122112$.

Lema 3.17 A função \frown é recursiva primitiva.

Demonstração: A função \frown depende do alfabeto, ou seja para cada n existe uma função \frown_n . No entanto, considera-se que n é fixado e verifica-se que:

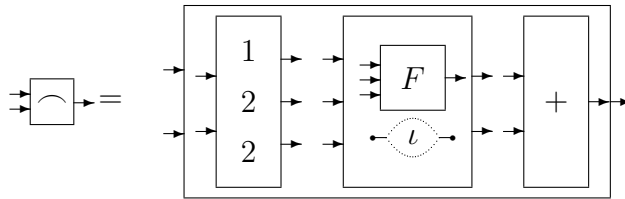
$$\frown(x, y) = x.n^{\underline{compr}(y)} + y$$

e portanto

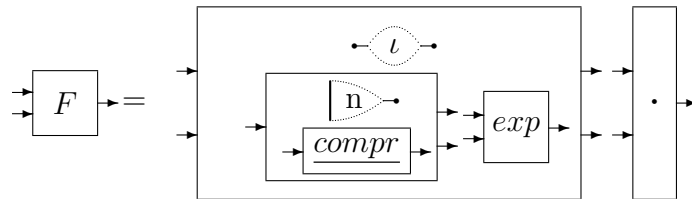
$$\frown = + \circ ((. \circ (\iota \times \underline{exp} \circ (n \times \underline{compr}))) \times \iota) \circ \begin{pmatrix} & 2 \\ 1 & 2 & 2 \end{pmatrix}$$

bastando demonstrar que a exponenciação da aritmética é recursiva primitiva (ver a lista de exercícios ao final do capítulo) \square

A correspondente componente molecular é:



onde



Definição 3.22 [cadeia reversa]

Define-se $\rho : \Sigma^* \rightarrow \Sigma^*$ como

$$\begin{aligned}\rho(\Lambda) &= \Lambda \\ \rho(x\sigma) &= \sigma\rho(x)\end{aligned}$$

Por exemplo: $\rho(12112) = 21121$.

Lema 3.18 A função $\rho : \Sigma^* \rightarrow \Sigma^*$ é recursiva primitiva.

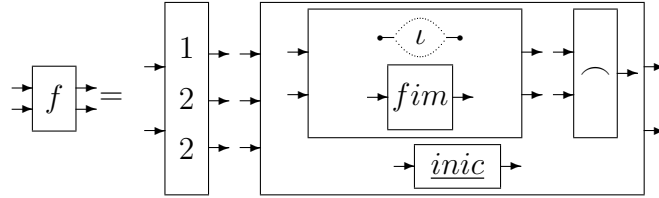
Demonstração: Utiliza-se uma função auxiliar $f : \Sigma^* \times \Sigma^* \rightarrow \Sigma^* \times \Sigma^*$ tal que

$$\begin{aligned}f(x, 0) &= \langle x, 0 \rangle \\ f(x, y\sigma) &= \langle x\sigma, y \rangle \quad \sigma \in \Sigma\end{aligned}$$

então

$$f = ((\neg \circ (\iota \times \underline{fim})) \times \underline{inic}) \circ \begin{pmatrix} & 2 \\ 1 & 2 & 2 \end{pmatrix}$$

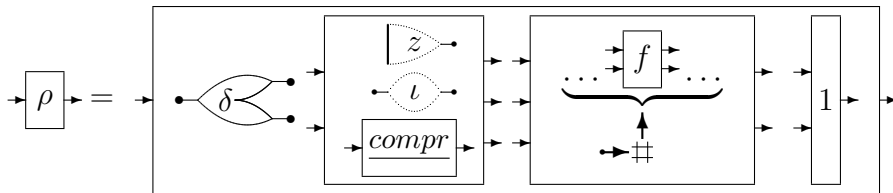
com componente molecular



e portanto

$$\rho = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \circ f^\# \circ (z \times \iota \times \underline{compr}) \circ \delta$$

com componente molecular:



□

Definição 3.23 [cadeia final]

Define-se $\underline{corte} : \Sigma^* \rightarrow \Sigma^*$ como

$$\underline{corte}(x) = \begin{cases} x' & \text{se } x = \sigma x', \sigma \in \Sigma \\ \Lambda & \text{se } x = \Lambda \end{cases}$$

Por exemplo: $\underline{corte}(1221) = 221$.

Lema 3.19 $\underline{corte} : \Sigma^* \rightarrow \Sigma^*$ é uma função recursiva primitiva.

Demonstração: $\underline{corte} = \rho \circ \underline{inic} \circ \rho$. □

Definição 3.24 [primeiro símbolo]

Define-se $\underline{prim}s : \Sigma^* \rightarrow \Sigma$ como

$$\underline{prim}s(x) = \begin{cases} \sigma & \text{se } x = \sigma x', \sigma \in \Sigma \\ \Lambda & \text{se } x = \Lambda \end{cases}$$

Por exemplo: $\underline{prim}s(1221) = 1$.

Lema 3.20 $\underline{prim}s : \Sigma^* \rightarrow \Sigma$ é uma função recursiva primitiva.

Demonstração: $\underline{prim}s = \underline{fim} \circ \rho$. □

Definição 3.25 [segmento inicial de tamanho y]

Define-se $\underline{segin} : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ como

$$\begin{aligned} \underline{segin}(\sigma_k \cdots \sigma_0, \Lambda) &= \Lambda \\ \underline{segin}(\sigma_k \cdots \sigma_0, y) &= \begin{cases} \sigma_k \cdots \sigma_0 & \text{se } k < y \\ \sigma_k \cdots \sigma_{k-y+1} & \text{se } k \geq y \end{cases} \end{aligned}$$

Por exemplo: $\underline{segin}(abcdef, 3) = abc$.

Lema 3.21 $\underline{segin} : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ é uma função recursiva primitiva.

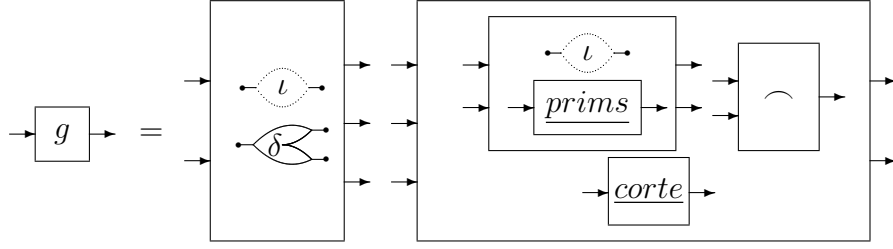
Demonstração: Seja g definida por:

$$\begin{aligned} g(x, 0) &= \langle x, 0 \rangle \\ g(x, \sigma y) &= \langle x\sigma, y \rangle \quad \sigma \in \Sigma \end{aligned}$$

deixa-se para o leitor verificar que

$$g = ((\neg \circ (\iota \times \underline{prim}s)) \times \underline{corte}) \circ (\iota \times \delta)$$

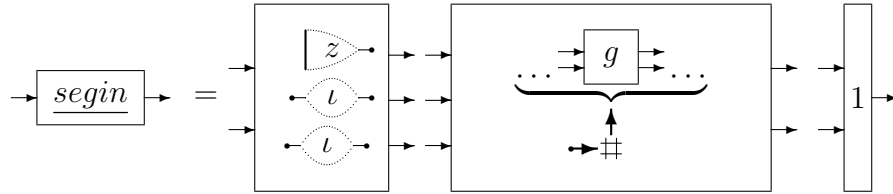
e que tem como componente molecular



e portanto

$$\underline{segin} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \circ g^\# \circ (z \times \iota \times \iota)$$

A correspondente componente molecular é:



□

Definição 3.26 [segmento final de tamanho $\underline{compr}(x) - y$]

Define-se $\underline{sefin} : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ como

$$\begin{aligned} \underline{sefin}(\sigma_k \cdots \sigma_0, 0) &= \sigma_k \cdots \sigma_0 \\ \underline{sefin}(\sigma_k \cdots \sigma_0, y) &= \begin{cases} 0 & \text{se } k < y \\ \sigma_{y-1} \cdots \sigma_0 & \text{se } k \geq y \end{cases} \end{aligned}$$

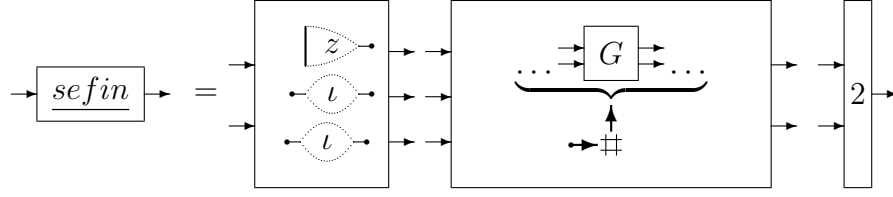
Por exemplo: $\underline{sefin}(abcdef, 2) = ef$.

Lema 3.22 $\underline{sefin} : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ é uma função recursiva primitiva.

Demonstração: Pode-se verificar que

$$\underline{sefin} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} \circ g^\# \circ (z \times \iota \times \iota)$$

onde g é a mesma função do item anterior, a componente molecular é:



□

Definição 3.27 [x é segmento inicial de y]

Define-se $\underline{inicia} : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ como

$$\underline{inicia}(x, y) = \begin{cases} 1 & \text{se } y = xz, \text{ para algum } z \in \Sigma^* \\ 0 & \text{caso contrário} \end{cases}$$

Por exemplo: $\underline{inicia}(abcdef, ab) = 1$ e $\underline{inicia}(abcdef, abd) = 0$.

Lema 3.23 $\underline{inicia} : \Sigma^* \times \Sigma^* \rightarrow \Sigma^*$ é uma função recursiva primitiva.

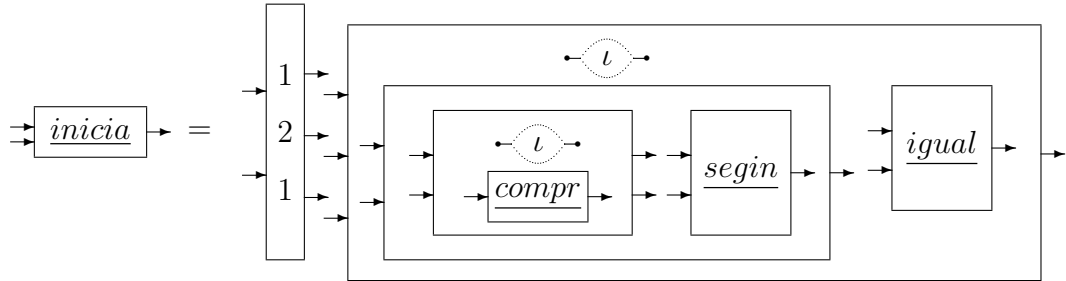
Demonstração: Pode-se verificar que

$$\underline{inicia} = \underline{igual} \circ (\iota \times \underline{segin} \circ (\iota \times \underline{compr})) \circ \begin{pmatrix} & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

pois

$$\begin{aligned} \underline{inicia}(x, y) &= \underline{igual} \circ (\iota \times \underline{segin} \circ (\iota \times \underline{compr}))(x, y, x) \\ &= \underline{igual}(x, \underline{segin}(y, \underline{compr}(x))) \end{aligned}$$

A correspondente componente molecular é:



□

3.7 Relação entre Alfabetos

Estudou-se que os alfabetos $\Sigma_n = \{1, 2, \dots, n\}$ são conjuntos finitos de símbolos, onde os números $1, 2, \dots, n$ são na realidade os valores de certos sinais gráficos. Por exemplo, pode-se ter um alfabeto com 3 sinais gráficos, $\{\square, \diamond, \odot\}$, aos quais pode-se atribuir valores 1, 2, 3 para significar uma ordem arbitrária.

Foi apresentado também que a função $\nu : \Sigma^* \rightarrow \mathbb{N}$ é uma extensão da função que dá o valor de cada símbolo enquanto indivíduo. Assim para o alfabeto $\{\square, \diamond, \odot\}$ tem-se que $\nu(\square\diamond) = 5$. Observa-se que não se atribui a nenhum símbolo o valor 0, portanto trabalha-se com sistemas de representação n-ádicos.

Considera-se, na próxima definição, a relação que existe entre palavras de diferentes alfabetos quando o valor é o mesmo.

Definição 3.28 Define-se a função $\kappa_{n,m} : \Sigma_n^* \rightarrow \Sigma_m^*$, por

$$\kappa_{n,m}(x) = y \text{ se e somente se } \nu_n(x) = \nu_m(y)$$

Por exemplo, para $n = 5$ e $m = 2$ tem-se $\Sigma_5 = \{1, 2, 3, 4, 5\}$, $\Sigma_2 = \{1, 2\}$ e $\Sigma_2^* = \{1, 2, 11, 12, 21, 22, \dots\}$. Logo, $\kappa_{5,2}(1) = 1$, $\kappa_{5,2}(2) = 2$, $\kappa_{5,2}(3) = 11$, $\kappa_{5,2}(4) = 12$. Analogamente, para $n = 2$ e $m = 5$ tem-se $\kappa_{2,5}(1) = 1$, $\kappa_{2,5}(2) = 2$, $\kappa_{2,5}(11) = 3$.

Como consequência, o valor de uma palavra em um alfabeto Σ_n quando representado em um outro alfabeto m é exatamente a função $\kappa_{n,m}$, ou seja

$$\kappa_{n,m}(\sigma_k \sigma_{k-1} \dots \sigma_1 \sigma_0) = \nu_m(\sigma_k \sigma_{k-1} \dots \sigma_1 \sigma_0)$$

Quando define-se a função de valorização ν , considera-se um caso específico onde as operações $+$ e produto \cdot eram funções abstratas, e cujos valores era representados em notação decimal. Agora deve-se considerar estas operações como sendo realizadas em um alfabeto qualquer, assim usa-se a notação $+_m, \cdot_m$ para representar soma e produto no sistema de numeração com m símbolos. Por exemplo:

$$\begin{aligned} +_2(11, 1) &= 12, \quad +_2(11, 2) = 21, \quad +_2(11, 11) = 22, \quad +_2(11, 12) = 111 \\ \cdot_2(1, 2) &= 2, \quad \cdot_2(11, 2) = 22, \quad \cdot_2(12, 2) = 112 \end{aligned}$$

Assim, sendo ${}_m \sum$ o somatório na base m :

$${}_m \sum_{j=0}^k x_j = x_0 +_m x_1 +_m \cdots +_m x_k$$

e $(x)_m^j$ a exponenciação na base m :

$$(x)_m^j = \overbrace{x \cdot_m \cdots \cdot_m x}^j$$

tem-se que

$$\kappa_{n,m}(\sigma_k \sigma_{k-1} \cdots \sigma_1 \sigma_0) = {}_m \sum_{j=0}^k \kappa_{n,m}(\sigma_j) \cdot_m (\kappa_{n,m}(n))_m^j$$

Exemplo 3.9 O cálculo de $\kappa_{3,5}(1332)$ é dado por:

$$\begin{aligned} & \kappa_{3,5}(2) \cdot_5 (\kappa_{3,5}(3))_5^0 +_5 \kappa_{3,5}(3) \cdot_5 (\kappa_{3,5}(3))_5^1 +_5 \kappa_{3,5}(3) \cdot_5 (\kappa_{3,5}(3))_5^2 +_5 \kappa_{3,5}(1) \cdot_5 (\kappa_{3,5}(3))_5^3 \\ & \nu_5(2) \cdot_5 (\nu_5(3))_5^0 +_5 \nu_5(3) \cdot_5 (\nu_5(3))_5^1 +_5 \nu_5(3) \cdot_5 (\nu_5(3))_5^2 +_5 \nu_5(1) \cdot_5 (\nu_5(3))_5^3 \\ & 2 \cdot_5 (3)_5^0 +_5 3 \cdot_5 (3)_5^1 +_5 3 \cdot_5 (3)_5^2 +_5 1 \cdot_5 (3)_5^3 \\ & 2 \cdot_5 1 +_5 3 \cdot_5 3 +_5 3 \cdot_5 14 +_5 1 \cdot_5 52 \\ & 2 +_5 3 +_5 52 +_5 52 \\ & 214 \end{aligned}$$

Logo, $\kappa_{3,5}(1332) = 214$. O leitor pode verificar que $\kappa_{5,3}(214) = 1332$

Lema 3.24 $\kappa_{n,m} : \Sigma_n^* \rightarrow \Sigma_m^*$ é recursiva primitiva.

Demonstração: Supondo $m > n$, sem perda de generalidade. Tem-se que $\kappa_{n,m}(\sigma) = \sigma$ para todo $\sigma \in \Sigma_n$, logo $\kappa_{n,m}(\sigma_k \cdots \sigma_0) = {}_m \sum_{j=0}^k \sigma_j \cdot n^j$. Seja f definida por

$$\begin{aligned} f(x, 0) &= \langle x, 0 \rangle \\ f(x, \sigma y) &= \langle x \cdot n + \sigma, y \rangle \end{aligned}$$

então, f é recursiva primitiva e sua expressão LB é:

$$f = (+ \circ (\cdot \circ (\iota \times n) \times \underline{prims}) \times \underline{corte}) \circ \begin{pmatrix} 2 \\ 1 & 2 & 2 \end{pmatrix}$$

Assim

$$\kappa_{n,m} = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \circ f^\# \circ (z \times \iota \times \underline{compr}) \circ \begin{pmatrix} 2 \\ 1 & 1 \end{pmatrix}$$

□

Para permitir que a função $\kappa_{n,m}$ esteja definida para qualquer cadeia em Σ_n^* ou em Σ_m^* , define-se sua extensão como a seguir.

Definição 3.29 Sejam os alfabetos Σ_n e Σ_m e seja $\Sigma = \Sigma_{\max\{m,n\}}$. Define-se a extensão natural $\bar{\kappa}_{n,m} : \Sigma^* \rightarrow \Sigma^*$ de $\kappa_{n,m}$, por

$$\bar{\kappa}_{n,m}(x) = \begin{cases} \kappa_{n,m}(x) & \text{se } x \in \Sigma_n^* \\ 0 & \text{se } x \in (\Sigma - \Sigma_n)^* \end{cases}$$

Para simplificar a notação, suponha $m > n$ e assim $\Sigma_m = \Sigma$. Para mostrar que $\bar{\kappa}_{n,m}$ é recursiva primitiva tem-se necessidade de provar que

$$f(x) = \begin{cases} 1 & \text{se } x \in \Sigma^* \\ 0 & \text{se } x \in (\Sigma - \Sigma_n)^* \end{cases}$$

é recursiva primitiva. Daí, tem-se $\bar{\kappa}_{n,m} = \underline{caso}(f, z, \kappa_{n,m}) \circ \begin{pmatrix} & 2 \\ 1 & 1 \end{pmatrix}$. Para provar que f é recursiva primitiva precisa-se mostrar que as funções que testam pertinência a certos conjuntos são funções recursivas primitivas.

Lema 3.25 Seja Σ um alfabeto, então:

1. χ_Σ (função característica de Σ) é recursiva primitiva;
2. χ_{Σ^*} (função característica de Σ^*) é recursiva primitiva.

Demonstração:

1. $\chi_\Sigma = sg \circ (\circ \circ (s \circ n \times \iota))$
2. Seja $f : \Sigma^{*2} \rightarrow \Sigma^{*3}$, definida por $f(x, \sigma y) = \langle x, \sigma, y \rangle$, $\sigma \in \Sigma$, então f é recursiva primitiva pois

$$f = (\iota \times \underline{prim_s} \times \underline{corte}) \circ \begin{pmatrix} & 2 \\ 1 & 2 & 2 \end{pmatrix}$$

Seja $g = (\cdot \circ (\iota \times \chi_{\Sigma_n}) \times \iota) \circ f$, então

$$g(x, \sigma y) = (\cdot \circ (\iota \times \chi_{\Sigma_n}) \times \iota)(x, \sigma, y) = \langle x \cdot \chi_{\Sigma_n}(\sigma), y \rangle$$

Deixa-se a definição de χ_{Σ^*} , a partir da expoentização de g , como exercício para o leitor. □

Agora pode-se concluir as discussões sobre $\bar{\kappa}_{n,m}$.

Lema 3.26 $\bar{\kappa}_{n,m} : \Sigma \rightarrow \Sigma$ é uma função recursiva primitiva.

Demonstração: Foi dito que se

$$f(x) = \begin{cases} 1 & \text{se } x \in \Sigma^* \\ 0 & \text{se } x \in (\Sigma - \Sigma_n)^* \end{cases}$$

é recursiva primitiva, então $\bar{\kappa}_{n,m} = \underline{caso}(f, z, \kappa_{n,m}) \circ \begin{pmatrix} & 2 \\ 1 & 1 \end{pmatrix}$ é recursiva primitiva. Como f é exatamente χ_{Σ^*} , o lema está demonstrado.

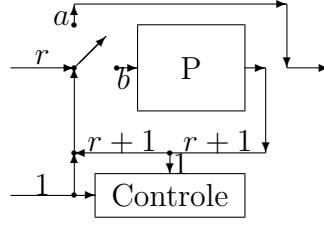
□

3.8 Repetição

O funcional expoentização realiza a reaplicação repetida de uma função. A repetição é controlada pela contagem de um índice natural que decresce de 1 cada vez que f é aplicada. Logo o processo é finito. Agora será considerada uma modificação no mecanismo de substituição de tal modo que o controle passa a chave para a posição a apenas quando recebe 1 e passa para a posição b quando recebe qualquer outra entrada. Note que agora o controle não computa o valor da entrada pois após a primeira execução da função f a última componente de $f(\underline{x}_{r+1})$ é a próxima entrada.

Definição 3.30 [Repetição] Seja a função $f : \mathcal{W}_{r+1} \rightarrow \mathcal{W}_{r+1}$ a repetição de f (também chamada minimização), $f^\nabla : \mathcal{W}_{r+1} \rightarrow \mathcal{W}_r$, é a função definida por:

$$f^\nabla(\underline{x}_r, y) = \begin{cases} x'_r & \text{se e somente se, para algum } k \geq 0, f^\#(\underline{x}_r, y, k) = \langle x'_r, 1 \rangle \\ & \text{e, para todo } l < k, f^\#(\underline{x}_r, y, l) \neq \langle x''_r, 1 \rangle \\ \text{indefinido} & \text{se não existe tal } k \geq 0 \end{cases}$$



Exemplo 3.10 Seja a função $f : \mathcal{W}_2 \rightarrow \mathcal{W}_2$, $f(x, y) = \langle x + 1, y \div 3 \rangle$, então $g : \mathcal{W}_1 \rightarrow \mathcal{W}_1$ definida por

$$g(y) = f^\nabla(0, y + 1)$$

é a função que dá o quociente da divisão de y por 3 se y for divisível por 3 e *diverge* em caso contrário.

$$g(6) = f^\nabla(0, 7) = 2$$

pois $f(0, 7) = \langle 1, 4 \rangle$, $f(f(0, 7)) = f(1, 4) = \langle 2, 1 \rangle$

$$g(7) = f^\nabla(0, 8) \uparrow$$

pois

$$f(0, 8) = \langle 1, 5 \rangle$$

$$f(f(0, 8)) = f(1, 5) = \langle 2, 2 \rangle$$

$$f(f(f(0, 8))) = f(f(1, 5)) = f(2, 2) = \langle 3, 0 \rangle$$

$$f(f(f(f(0, 8)))) = f(f(f(1, 5))) = f(f(2, 2)) = f(3, 0) = \langle 4, 0 \rangle$$

\vdots

Definição 3.31 [função parcialmente recursiva] Dizemos que uma função $f : \mathcal{W} \rightarrow \mathcal{W}$ é parcialmente recursiva se e somente se existe uma sequência de funções g_1, g_2, \dots, g_n tal que $f = g_n$ e para todo i , $1 \leq i \leq n$, g_i é inicial ou:

$$g_i = g_{i_1} \circ g_{i_2} \quad 1 \leq i_1, i_2 < i, \text{ ou}$$

$$g_i = g_{i_1} \times g_{i_2} \quad 1 \leq i_1, i_2 < i, \text{ ou}$$

$$g_i = g_{i_1}^\# \quad 1 \leq i_1 < i$$

$$g_i = g_{i_1}^\nabla \quad 1 \leq i_1 < i$$

Agora pode-se estender a linguagem básica $LB - \nabla$, com a adiç o do novo funcional. Um programa LB   uma express o que satisfaz a seguinte defini o formal:

$$\begin{aligned}
< \text{express o} > &::= < \text{inicial} > \mid \\
&(< \text{express o} > \circ < \text{express o} >) \mid \\
&(< \text{express o} > \times < \text{express o} >) \mid \\
&< \text{express o} >^\# \mid \\
&< \text{express o} >^\nabla \\
< \text{inicial} > &::= \pi \mid z \mid s
\end{aligned}$$

Defini o 3.32 Seja α uma express o LB, ent o a fun o computada por α denotada por $|\alpha|$   definida da seguinte maneira:

1. Se $\alpha \equiv \iota$ ent o $|\alpha|(x) = x$;
2. Se $\alpha \equiv z$ ent o $|\alpha|(<>) = 0$;
3. Se $\alpha \equiv \pi$ ent o $|\alpha|(x) = <>$;
4. Se $\alpha \equiv s$ ent o $|\alpha|(x) = x + 1$;
5. Se $\alpha \equiv (\beta \circ \gamma)$, $\beta \in \mathcal{W}_s^{\mathcal{W}_r}$ e $\gamma \in \mathcal{W}_t^{\mathcal{W}_s}$ ent o

$$|\alpha|(\underline{x}_r) = |\beta|(|\gamma|(\underline{x}_r))$$

6. Se $\alpha \equiv (\beta \times \gamma)$, $\beta \in \mathcal{W}_s^{\mathcal{W}_r}$ e $\gamma \in \mathcal{W}_t^{\mathcal{W}_v}$ ent o

$$|\alpha|(\underline{x}_r, \underline{y}_v) = < |\beta|(\underline{x}_r), |\gamma|(\underline{x}_v) >$$

7. Se $\alpha \equiv \beta^\#$, $\beta \in \mathcal{W}_r^{\mathcal{W}_r}$ ent o

$$\begin{aligned}
|\alpha|(\underline{x}_r, 0) &= \underline{x}_r \\
|\alpha|(\underline{x}_r, y + 1) &= |\beta|(|\alpha|(\underline{x}_r, y))
\end{aligned}$$

8. Se $\alpha \equiv \beta^\nabla$, $\beta \in \mathcal{W}_{r+1}^{\mathcal{W}_{r+1}}$ ent o

$$|\alpha|(\underline{x}_r, y) = \begin{cases} \underline{x}'_r & \text{se para algum } k \geq 0, |\beta^\#|(\underline{x}_r, y, k) = < \underline{x}'_r, 1 > \\ & \text{e para todo } l < k, |\beta^\#|(\underline{x}_r, y, l) \neq < \underline{x}''_r, 1 > \\ \text{indefinido, se n o existe tal } k \geq 0 \end{cases}$$

9. As  nicas fun es computadas por express es $LB - \nabla$ s o express es em $LB - \nabla$ que satisfazem 1 a 8.

3.9 Funções Recursivas

Como foi dito no início deste capítulo deseja-se construir uma classe de funções que capture a noção intuitiva de função *computável*. Pode parecer que a tarefa foi terminada, e que a classe das funções primitivas recursivas coincide exatamente com a classe das funções *computáveis*. Note que qualquer função recursiva primitiva $f : \mathbb{N}^n \rightarrow \mathbb{N}$ é uma função total, ou seja $\text{dom}(f) = \mathbb{N}^n$, e durante algum tempo esta classe recebeu o nome de classe das funções recursivas. Posteriormente foram descobertas funções $f : \mathbb{N}^n \rightarrow \mathbb{N}$, que, embora funções totais computáveis, não poderiam ser obtidas a partir das funções iniciais pela utilização de composição e recursão primitiva. Estas funções, embora definidas por métodos bastante simples e facilmente reconhecidos como *computáveis* são extremamente complicadas para serem avaliadas mesmos para argumentos pequenos. Um exemplo é a função de Ackermann³:

$$\left. \begin{array}{lll} r.1 & a(0, y) & = y + 1 \\ r.2 & a(x + 1, 0) & = a(x, 1) \\ r.3 & a(x + 1, y + 1) & = a(x, a(x + 1, y)) \end{array} \right\} \text{Não usa as funções iniciais}$$

Note que se está também utilizando um esquema de recursão, e que para dados naturais m e n é sempre possível *computar* $a(m, n)$. Para entender o que se quer dizer com avaliação complicada pode-se exemplificar com a computação de $a(1, 2)$. Primeiramente observe que o início do processo de computação depende de uma escolha criteriosa de que equação, entre $r.1$, $r.2$ ou $r.3$, será utilizada. Esta escolha depende dos valores m e n . No exemplo não pode-se começar nem por $r.1$ nem por $r.2$ e sim por $r.3$. Substituindo x por 0 e y por 1 tem-se $a(1, 2) = a(0, a(1, 1))$, assim para avaliar $a(1, 2)$ depende-se da avaliação de $a(1, 1)$. A avaliação de $a(1, 1)$ deve iniciar também por $r.3$. Faz-se as substituições de x por 0 e de y por 0 obtém-se $a(1, 1) = a(0, a(1, 0))$, que depende de $a(1, 0)$. Escolhendo

³A função de Ackermann é o exemplo mais simples de uma função total bem definida que é computável mas não é primitiva recursiva, fornecendo um contra-exemplo para a crença do começo do século XX de que toda função computável era também primitiva recursiva. Trata-se de uma recursão aninhada especial, com recursão dupla, onde uma definição recursiva refere-se a si própria mais de uma vez. Ela cresce mais rápido do que uma função exponencial, ou até mesmo que uma função exponencial múltipla.

$r.2$ obtém-se $a(1, 0) = a(0, 1)$ e agora $a(0, 1) = 1 + 1 = 2$ pela equação $r.1$. Assim, $a(1, 0) = 2$ e portanto $a(1, 1) = a(0, 2)$ e por $r.1$, $a(0, 2) = 3$. Logo, $a(1, 1) = 3$, assim $a(1, 2) = a(0, 3)$ que por $r.1$ é igual a 4. O leitor pode tentar computar $a(2, 3)$.

$$\begin{aligned} r.3 : \quad & a(1, 2) = a(0, a(1, 1)) \\ r.3 : \quad & a(1, 1) = a(0, a(1, 0)) \\ r.2 : \quad & a(1, 0) = a(0, 1) \\ r.1 : \quad & a(0, 1) = 1 + 1 = 2 \end{aligned}$$

$$\begin{aligned} & a(0, 1) = 2 \\ & a(1, 0) = a(0, 1) = 2 \\ r.1 : \quad & a(1, 1) = a(0, 2) = 2 + 1 = 3 \\ r.1 : \quad & a(1, 2) = a(0, 3) = 3 + 1 = 4 \end{aligned}$$

É interessante observar neste ponto de nossos estudos que, permitindo esquemas de *recursão* geral, será possível eventualmente definir funções que são *computáveis* para alguns valores, mas que eventualmente *divergem*. Por exemplo a função definida por

$$\left. \begin{aligned} r.1 \quad & b(0, y) = b(y + 1, y + 1) \\ r.2 \quad & b(x + 1, y) = b(x, b(x, y + 1)) \end{aligned} \right\} \text{Não usa as funções iniciais}$$

é recursiva, mas totalmente divergente. Neste sentido, sugere-se que o leitor efetue a computação de $b(1, 1)$.

As funções iniciais, que são computáveis elementares, podem ser usadas para construir funções recursivas primitivas também computáveis. Note que a classe de funções recursivas primitivas não esgota a classe de funções computáveis, pois algumas como a função de Ackermann e a divisão, não são totais. Desta forma, pode-se estabelecer uma hierarquia para as funções tal como ilustrado na Figura 3.2.

O estudo detalhado dos esquemas de recursão geral foge ao escopo deste capítulo, no entanto o leitor pode consultar o capítulo 3 de Carvalho e Oliveira [40]. De qualquer modo, o fato de existirem funções computáveis, que não são recursivas primitivas, definidas por tais esquemas, e que tais esquemas podem definir

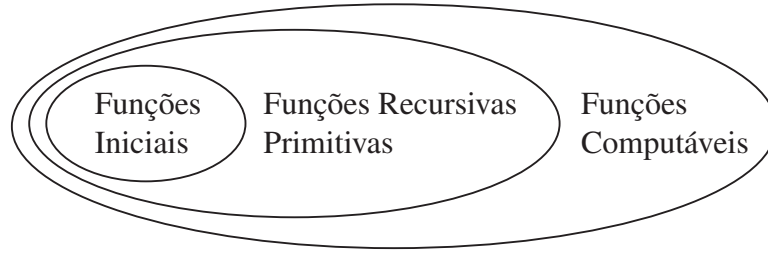


Figura 3.2: Hierarquia de funções computáveis.

funções cujos processos de computação eventualmente divergem, sugere que além de composição e recursão primitiva, deve-se definir um outro esquema funcional. A seguir apresenta-se o esquema de *minimização*, que gera funções não totais.

Definição 3.33 Seja $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ uma função total. Diz-se que $h : \mathbb{N}^n \rightarrow \mathbb{N}$ é definida por minimização de f se e somente se os valores de h são obtidos por

$$h(x_1, \dots, x_n) = \begin{cases} \min(A_f) & \text{se } A_f \neq \emptyset \\ \uparrow \text{ (diverge) } & \text{se } A_f = \emptyset \end{cases}$$

onde $A_f = \{y | f(y, x_1, \dots, x_n) = 1\}$. Denota-se a minimização de f por

$$h(x_1, \dots, x_n) = \mu y (f(y, x_1, \dots, x_n))$$

Exemplo 3.11 Função parcial obtida por minimização :

$$x/3 = \mu y (\underline{\text{igual}}(y \times 3, x))$$

Verifique que para $x = 7$ a função $x/3$ diverge.

Definição 3.34 Diz-se que uma função $f : \mathbb{N}^n \rightarrow \mathbb{N}$ é recursiva parcial se f for uma função inicial ou for obtida pela utilização de composição, recursão primitiva e minimização a partir das funções iniciais. Se acontecer que f é uma função total, diz-se apenas que f é uma função recursiva.

Suponha que se deseja limitar a funções totais. Pode-se usar um outro esquema de minimização definido a seguir.

Definição 3.35 Sejam $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ e $g : \mathbb{N}^n \rightarrow \mathbb{N}$. Define-se uma função $(\mu fg) : \mathbb{N}^n \rightarrow \mathbb{N}$, chamada de minimização de f limitada por g da seguinte maneira:

$$(\mu fg)(\underline{x}_n) = \begin{cases} \min(A_{\underline{x}_n}^g) & \text{se } A_{\underline{x}_n}^g \neq \emptyset \\ 0 & \text{se } A_{\underline{x}_n}^g = \emptyset \end{cases}$$

onde $A_{\underline{x}_n}^g = \{y \mid y < g(\underline{x}_n) \wedge f(\underline{x}_n, y) = 1 \text{ e } f(\underline{x}_n, z) \downarrow \text{ para todo } z < y\}$.

Lema 3.27 Se $f : \mathbb{N}^n \rightarrow \mathbb{N}$ e $g : \mathbb{N}^m \rightarrow \mathbb{N}$ são recursivas primitivas, então a função $(g \rightarrow f) : \mathbb{N}^{n+m} \rightarrow \mathbb{N}$ definida por:

$$(g \rightarrow f)(\underline{x}_n, \underline{y}_m) = \begin{cases} 0 & \text{se } g(\underline{y}_m) = 0 \\ f(\underline{x}_n) & \text{se } g(\underline{y}_m) > 0 \end{cases}$$

é recursiva primitiva.

Demonstração: Deixa-se para o leitor como exercício.

Lema 3.28 Sejam $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ e $g : \mathbb{N}^n \rightarrow \mathbb{N}$. Se f e g são recursivas primitivas então (μfg) é uma função recursiva primitiva.

Demonstração: Inicialmente note que se f e g são recursivas primitivas então $f(\underline{x}_n, z) \downarrow$ e $y < g(\underline{x}_n) \downarrow$ portanto a definição de $A_{\underline{x}_n}^g$ fica sendo

$$A_{\underline{x}_n}^g = \{y \mid y < g(\underline{x}_n) \wedge f(\underline{x}_n, y) = 1 \text{ para todo } z < y\}$$

Note, também, que a função h' abaixo testa se $A_{\underline{x}_n}^g$ é um conjunto vazio:

$$h'(\underline{x}_n) = \bigvee_{v=0}^{g(\underline{x}_n)} (f(\underline{x}_n, v) = 1)$$

Suponha agora que $A_{\underline{x}_n}^g \neq \emptyset$ e seja $y_0 = \min(A_{\underline{x}_n}^g)$ e h a função definida por:

$$h(\underline{x}_n, y) = \bar{s}g\left(\bigvee_{v=0}^y f(\underline{x}_n, v) = 1\right)$$

Pode-se verificar que

$$h(\underline{x}_n, y) = \begin{cases} 1 & \text{se } y < y_0 \\ 0 & \text{se } y \geq y_0 \end{cases}$$

Assim

$$y_0 = \sum_{y=0}^{g(\underline{x}_m)} h(\underline{x}_n, y)$$

Tem-se, portanto,

$$(\mu f g)(\underline{x}_n) = h'(\underline{x}_n) \rightarrow \sum_{y=0}^{g(\underline{x}_n)} h(\underline{x}_n, y)$$

Os *conjuntos recursivos* são aqueles conjuntos de números naturais para os quais existe um procedimento efetivo para verificar pertinência. São também chamados conjuntos decidíveis. Foi apresentado na definição 2.7 que a função característica de um conjunto $A \subseteq \mathbb{N}$ é a função:

$$\chi_A : \mathbb{N} \rightarrow \{0, 1\}$$

$$\chi_A(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{se } x \notin A \end{cases}$$

A definição a seguir caracteriza os conjuntos recursivos via suas funções características.

Definição 3.36 Seja A um conjunto de números naturais. Diz-se que A é um conjunto recursivo se e somente sua função característica $\chi_A : A \rightarrow \{0, 1\}$ é uma função recursiva.

$$\chi_A : A \rightarrow \{0, 1\}$$

$$\chi_A(x) = \begin{cases} 1 & \text{se } x \in A \\ 0 & \text{se } x \notin A \end{cases}$$

O conjunto recursivo também pode ser chamado de decidível, computável e ainda solúvel.

Alguns conjuntos têm funções características expressas por algoritmos, o estudo destes será visto posteriormente.

Exemplo 3.12 Os seguintes conjuntos são recursivos:

Conjunto de ímpares e pares Já foi estudado que suas funções características são funções recursivas.

Conjunto de números primos A função característica é o algoritmo conhecido como “Crivo de Eratóstenes”.

Conjuntos finitos Se $A = \{i_1, i_2, \dots, i_n\}$ então sua função característica é dada por

$$\chi_A(x) = \underline{igual}(x, i_1) \vee \underline{igual}(x, i_2) \vee \dots \vee \underline{igual}(x, i_n)$$

Definição 3.37 Um conjunto A de naturais é dito ser recursivamente enumerável (r.e) se e somente se existe uma função recursiva $f : \mathbb{N} \rightarrow \mathbb{N}$ tal que $A = \text{img}(f)$.

$$\chi_A : A \rightarrow \{0, 1\}$$

$$\chi_A(x) = \begin{cases} 1 & \text{se } x \in A \\ \text{indefinido/não pára} & \text{se } x \notin A \end{cases}$$

O conjunto recursivamente enumerável também pode ser chamado de computavelmente enumerável, semi-decidível, demonstrável ou Turing-reconhecível.

3.10 Conclusões e leituras recomendadas

Foi apresentada uma caracterização das funções recursivas com uma linguagem funcional LB e esquemas gráficos que auxiliam na visualização e compreensão da computação das funções. A linguagem básica utilizada neste capítulo tem origem em Brainerd [41]. No entanto, para uma apresentação mais tradicional, o leitor não terá dificuldades em encontrar bibliografia abordando a teoria das funções recursivas.

Entre os textos mais consagrados devemos destacar Kleen [42], Rogers [43], Davis [44] e Boolos [45]. A diferença mais evidente entre a caracterização aqui apresentada e as mais tradicionais é devida à simplicidade do presente conjunto de funções iniciais e de funcionais básicos. Em geral, considera-se um conjunto infinito de funções iniciais (todas as projeções e todas as constantes, por exemplo) e a composição envolve um número qualquer de funções.

3.11 Exercícios

3.1 Demonstre os itens c a q do lema 3.1.

3.2 Demonstre os itens c a d do lema 3.2.

3.3 Demonstre que o lema 3.2 vale quando substituímos o limite superior para $g(m)$ onde m é uma função recursiva primitiva $g : \mathbb{N}^n \rightarrow \mathbb{N}$

3.4 Utilizando a função $x/3$ do exemplo 3.11 verifique que para $x = 7$ a função $x/3$ diverge.

3.5 Escrever a série de Fibonacci através de funções recursivas.

3.6 Demonstre o lema 3.27.

Capítulo 4

Lógica Simbólica

Com o advento da inteligência artificial, houve um incremento no interesse por processos automatizados, ou semi-automatizados, para a escrita de programas, atendimento à estímulos e, em últimas palavras, prova sistemática de teoremas. As bases da prova automática de teoremas foram desenvolvidas em 1930 por Herbrand, cujo método foi impossível de ser aplicado até a invenção do computador digital.

A lógica simbólica pode ser estudada sob diversas óticas, como por exemplo, a filosófica e a matemática. Neste capítulo o interesse está na aplicação da lógica simbólica, isto é, pretende-se usar a lógica simbólica para representar problemas e obter sua solução.

Uma vez que a lógica simbólica ainda não foi apresentada ao leitor, pede-se que o mesmo confie em sua intuição para acompanhar o início deste texto, ao menos por alguns instantes. Assuma as sentenças do exemplo a seguir:

S1 : Se está quente e está úmido, então vai chover.

S2 : Se está úmido, então está quente.

S3 : Está úmido agora.

Pergunta : Vai chover?

Estas sentenças estão representadas em português, porém pode-se empregar símbolos na sua representação. Assuma que Q , U , C representam, respectivamente, as sentenças “Está quente”, “Está úmido” e “Vai chover”. Também serão necessários alguns conectivos para fazer a junção das sentenças. O “e” será representado pelo

\wedge , e o sentido de implicação das sentenças do tipo “Se ... Então” será descrito por \rightarrow . Assim o conjunto de sentenças em português pode ser reescrito como:

$$S1: Q \wedge U \rightarrow C$$

$$S2: U \rightarrow Q$$

$$S3: U$$

$$Pergunta: C$$

Ao reescrever as sentenças através de símbolos e conectivos aqui propostos, foram criadas o que se chamam *fórmulas lógicas*. No decorrer deste capítulo, o leitor irá observar que se $S1$, $S2$ e $S3$ forem verdadeiros então a fórmula associada à pergunta também será verdadeira, isto é, vai chover. Neste sentido diz-se que a pergunta é *logicamente dedutível* de $S1$, $S2$ e $S3$.

No exemplo, foi preciso mostrar que uma fórmula pôde ser obtida a partir de outras, e chama-se esta fórmula obtida de *teorema*. Para que a prova de um teorema seja verdadeira, é preciso que a fórmula seja logicamente dedutível a partir de outras. Sob esta ótica, o problema relacionado com provadores automáticos de teoremas está associado com a busca por métodos sistemáticos para demonstração destes teoremas.

4.1 Lógica Proposicional

Na lógica proposicional se está interessado em sentenças que podem assumir valores verdadeiro ou falso, conhecidos como *valores verdade*, e tradicionalmente representados por 1 e 0 respectivamente. A estas sentenças dá-se o nome de *proposições*. Como exemplo de proposições verdadeiras podemos citar: “ $100 > 99$ ” e “Todo pássaro voa”. Proposições falsas podem ser ilustradas por “ $99 > 100$ ” e “Pássaros não voam”.

Entretanto, existem sentenças que não podem ser aceitas como proposições pois não é possível lhes atribuir valor verdade, como por exemplo, “Esta sentença é falsa”, “Nunca diga nunca” e “Eu não sou demonstrável”. Em alguns casos, proposições isoladas podem ser livres de contradição, mas quando onservadas em conjunto emerge uma contradição: “A afirmação a seguir é verdadeira” e “A afirmação

anterior é falsa”.

Além disto, também existem aquelas proposições as quais possuem valor verdade, mas não há concessão sobre qual é esse valor: “Monteiro Lobato foi um escritor mais importante que Jorge Amado” e “Flamengo é freguês do Fluminense. Vasco, coitado, é saco de pancada”.

As proposições que não usam conectivos são chamadas de *fórmulas atômicas*, ou *átomos*, caso contrário são chamadas “proposições compostas”. São exemplos de proposições compostas, com conectivos “e” e “se ... então”: “Chapeuzinho Vermelho foi à floresta e o Lobo Mau foi para a casa da Vovó” e “Se tenho olhos grandes, então é para te ver melhor”. Na lógica proposicional pode-se empregar cinco conectivos lógicos: \neg (não), \wedge (e), \vee (ou), \rightarrow (se ... então), \leftrightarrow (se somente se). A ordem de precedência decrescente entre estes conectivos é dada por: \leftrightarrow , \rightarrow , \wedge , \vee , \neg .

Definição 4.1 [Fórmulas bem fundadas] Fórmulas bem fundadas, ou simplesmente fórmulas, na lógica proposicional são definidas recursivamente conforme a seguir:

1. Um átomo é uma fórmula.
2. Se G é uma fórmula, então $\neg G$ também é uma fórmula.
3. Se G e H são fórmulas, então $(G \wedge H)$, $(G \vee H)$, $(G \rightarrow H)$ e $(G \leftrightarrow H)$ são fórmulas.
4. Todas as fórmulas são geradas a partir destas regras.

Sejam G e H duas fórmulas. De acordo com seus valores verdade tem-se que a sua combinação imediata com os conectivos é dada pela tabela verdade 4.1.

Tabela 4.1: Tabela verdade do \neg , \wedge , \vee , \rightarrow e \leftrightarrow .

G	H	$\neg G$	$G \wedge H$	$G \vee H$	$G \rightarrow H$	$G \leftrightarrow H$
1	1	0	1	1	1	1
1	0	0	0	1	0	0
0	1	1	0	1	1	0
0	0	1	0	0	1	1

Definição 4.2 [Interpretação] Dada a fórmula proposicional G , e sejam A_1, A_2, \dots, A_n átomos que ocorrem nesta fórmula. Uma interpretação de G é uma possibilidade de valores verdade, 0 ou 1, assumidos por A_1, A_2, \dots, A_n .

Considere a fórmula $G = ((P \rightarrow Q) \wedge P) \rightarrow Q$. Os valores verdade para cada uma das quatro interpretações desta fórmula são apresentados na Tabela 4.2. Observe ainda nesta tabela que os valores de G são sempre 1 quaisquer que sejam as interpretações. Quando isto ocorre, diz-se que a fórmula é uma *fórmula válida*, ou simplesmente uma *tautologia*.

Tabela 4.2: Tabela verdade de uma tautologia.

P	Q	$(P \rightarrow Q)$	$(P \rightarrow Q) \wedge P$	$((P \rightarrow Q) \wedge P) \rightarrow Q$
1	1	1	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	1

Analogamente, seja a fórmula $G = (P \rightarrow Q) \wedge (P \wedge \neg Q)$, cujos valores-verdade são apresentados na Tabela 4.3. Neste caso as quatro interpretações associadas à fórmula assumem o valor 0. Sob estas circunstâncias, diz-se que G é uma *fórmula inconsistente*, ou também, uma *contradição*.

Tabela 4.3: Tabela verdade de uma contradição.

P	Q	$\neg Q$	$(P \rightarrow Q)$	$(P \wedge \neg Q)$	$(P \rightarrow Q) \wedge (P \wedge \neg Q)$
1	1	0	1	0	0
1	0	1	0	1	0
0	1	0	1	0	0
0	0	1	1	0	0

4.2 Forma Normal na Lógica Proposicional

Durante a manipulação de fórmulas é comum ser necessário trocar determinadas fórmulas por outras, seja por motivos de simplicidade de representação, seja

por necessidade de torná-las mais claras. Para que esta troca possa ser realizada é necessário que a nova fórmula F seja equivalente à antiga G , isto é, que os valores-verdade de F e G sejam os mesmos em qualquer que seja a interpretação. A Tabela 4.4 ilustra que as fórmulas $P \rightarrow Q$ e $\neg P \vee Q$ são logicamente equivalentes, ou seja, $P \rightarrow Q \equiv \neg P \vee Q$.

Tabela 4.4: Tabela verdade de uma equivalência lógica.

P	Q	$\neg P$	$(P \rightarrow Q)$	$\neg P \vee Q$
1	1	0	1	1
1	0	1	0	0
0	1	0	1	1
0	0	1	1	1

O processo para a verificação da equivalência lógica exige a construção de tabelas verdade tal como a Tabela 4.4. Conforme o número de átomos cresce, a quantidade de interpretações cresce significativamente ($2^{\text{átomos}}$), tornando o processo demorado, tedioso e muitas vezes, inviável.

De modo a agilizar a verificação de equivalências, torna-se desejável o emprego de algumas regras, listadas na Tabela 4.5, que contribuem para a redução do tempo gasto nesse processo. Estas relações de equivalência lógica podem ser facilmente comprovadas através de tabelas-verdade.

Tabela 4.5: Regras de equivalência para normalização.

$r.1)$	$P \leftrightarrow Q \equiv (P \rightarrow Q) \wedge (Q \rightarrow P)$	
$r.2)$	$P \rightarrow Q \equiv \neg P \vee Q$	
$r.3a)$	$P \vee Q \equiv Q \vee P$	$r.3b)$ $P \wedge Q \equiv Q \wedge P$
$r.4a)$	$(P \vee Q) \vee R \equiv P \vee (Q \vee R)$	$r.4b)$ $(P \wedge Q) \wedge R \equiv P \wedge (Q \wedge R)$
$r.5a)$	$P \vee (Q \wedge R) \equiv (P \vee Q) \wedge (P \vee R)$	$r.5b)$ $P \wedge (Q \vee R) \equiv (P \wedge Q) \vee (P \wedge R)$
$r.6a)$	$P \vee \mathbf{0} \equiv P$	$r.6b)$ $P \wedge \mathbf{0} \equiv \mathbf{0}$
$r.7a)$	$P \vee \mathbf{1} \equiv \mathbf{1}$	$r.7b)$ $P \wedge \mathbf{1} \equiv P$
$r.8a)$	$P \vee \neg P \equiv \mathbf{1}$	$r.8b)$ $P \wedge \neg P \equiv \mathbf{0}$
$r.9)$	$\neg(\neg P) \equiv P$	
$r.10a)$	$\neg(P \vee Q) \equiv \neg P \wedge \neg Q$	$r.10b)$ $\neg(P \wedge Q) \equiv \neg P \vee \neg Q$

Observe que se $(P \vee Q) \vee R \equiv P \vee (Q \vee R)$ então pode-se afirmar que são desnecessários os parênteses. Assim pode-se escrever equivalentemente $P \vee Q \vee R$. Quando isto acontece, diz-se que se tem uma *disjunção* de P , Q e R . De forma análoga, $P \wedge Q \wedge R$ representa uma *conjunção* de P , Q e R .

Definição 4.3 [Literal] Um *literal* é um átomo, ou a negação de um átomo.

Definição 4.4 [Forma Normal Conjuntiva] Uma fórmula F é dita estar representada através de uma *forma normal conjuntiva* se e somente se $F \equiv F_1 \wedge F_2 \wedge \dots \wedge F_n$, onde $n \geq 1$ e F_1, F_2, \dots, F_n são literais disjuntos.

Definição 4.5 [Forma Normal Disjuntiva] Uma fórmula F é dita estar representada através de uma *forma normal disjuntiva* se e somente se $F \equiv F_1 \vee F_2 \vee \dots \vee F_n$, onde $n \geq 1$ e F_1, F_2, \dots, F_n são literais conjuntos.

Exemplo 4.1 Obtenha a forma normal disjuntiva de $(P \vee \neg Q) \rightarrow R$.

$$\begin{aligned} (P \vee \neg Q) \rightarrow R &\equiv \neg(P \vee \neg Q) \vee R && (r.2) \\ &\equiv (\neg P \wedge \neg \neg Q) \vee R && (r.10a) \\ &\equiv (\neg P \wedge Q) \vee R && (r.9) \end{aligned}$$

Exemplo 4.2 Obtenha a forma normal conjuntiva de $(P \wedge (Q \rightarrow R)) \rightarrow S$.

$$\begin{aligned} (P \wedge (Q \rightarrow R)) \rightarrow S &\equiv (P \wedge (\neg Q \vee R)) \rightarrow S && (r.2) \\ &\equiv \neg(P \wedge (\neg Q \vee R)) \vee S && (r.2) \\ &\equiv (\neg P \vee \neg(\neg Q \vee R)) \vee S && (r.10b) \\ &\equiv (\neg P \vee (\neg \neg Q \wedge \neg R)) \vee S && (r.10a) \\ &\equiv (\neg P \vee (Q \wedge \neg R)) \vee S && (r.9) \\ &\equiv ((\neg P \vee Q) \wedge (\neg P \vee \neg R)) \vee S && (r.5a) \\ &\equiv S \vee ((\neg P \vee Q) \wedge (\neg P \vee \neg R)) && (r.3a) \\ &\equiv (S \vee (\neg P \vee Q)) \wedge (S \vee (\neg P \vee \neg R)) && (r.5a) \\ &\equiv (S \vee \neg P \vee Q) \wedge (S \vee \neg P \vee \neg R) && (r.4a) \end{aligned}$$

Um desdobramento direto das formas normais é a questão da consequência lógica.

Definição 4.6 [Consequência Lógica] Dadas as fórmulas P_1, P_2, \dots, P_n e a fórmula Q , Q é dito ser uma *consequência lógica* de P_1, P_2, \dots, P_n se e somente se para cada

interpretação I em que $P_1 \wedge P_2 \wedge \dots \wedge P_n$ for verdadeira, Q também é verdadeira. Matematicamente, diz-se $P_1, P_2, \dots, P_n \models Q$. P_1, P_2, \dots, P_n são chamados *axiomas* (ou *postulados*, *premissas*) de Q .

Teorema 4.1 [Teorema da Dedução] Dadas as fórmulas F_1, \dots, F_n e a fórmula G , dizemos que G é uma consequência lógica de F_1, \dots, F_n se e somente se a fórmula $(F_1 \wedge \dots \wedge F_n) \rightarrow G$ é válida.

(\Rightarrow) Considere I como sendo uma interpretação arbitrária qualquer. Se I é verdadeira em F_1, \dots, F_n , então por definição de consequência lógica, I é verdadeira em G . Assim I também é verdadeira em $(F_1 \wedge \dots \wedge F_n) \rightarrow G$. Por outro lado, se I é falso em F_1, \dots, F_n então I continua sendo verdadeira em $(F_1 \wedge \dots \wedge F_n) \rightarrow G$. Assim $(F_1 \wedge \dots \wedge F_n) \rightarrow G$ é uma fórmula válida.

(\Leftarrow) Suponha que $(F_1 \wedge \dots \wedge F_n) \rightarrow G$ seja uma fórmula válida, Qualquer que seja uma interpretação I , se $(F_1 \wedge \dots \wedge F_n)$ é verdadeiro em I , então G também é verdadeiro em I . Deste modo G é uma consequência lógica de F_1, \dots, F_n .

Teorema 4.2 [Teorema da Dedução por Refutação] Dadas as fórmulas F_1, \dots, F_n e G , G é uma consequência lógica de F_1, \dots, F_n se e somente se a fórmula $(F_1 \wedge \dots \wedge F_n \wedge \neg G)$ é inconsistente.

Pelo Teorema 4.1, G é uma consequência lógica de F_1, \dots, F_n se e somente se $(F_1 \wedge \dots \wedge F_n) \rightarrow G$ é válida. Deste modo a negação $\neg((F_1 \wedge \dots \wedge F_n) \rightarrow G)$ precisa ser uma fórmula inconsistente.

$$\begin{aligned} \neg((F_1 \wedge \dots \wedge F_n) \rightarrow G) &\equiv \neg(\neg(F_1 \wedge \dots \wedge F_n) \vee G) \\ &\equiv \neg\neg(F_1 \wedge \dots \wedge F_n) \wedge \neg G \\ &\equiv (F_1 \wedge \dots \wedge F_n) \wedge \neg G \\ &\equiv F_1 \wedge \dots \wedge F_n \wedge \neg G \end{aligned}$$

de onde se conclui o Teorema da Dedução por Refutação.

Os Teoremas 4.1 (Dedução) e 4.2 (Dedução por Refutação) são importantes pois eles são utilizados diretamente na prova automática de teoremas. Para ilustrar

o poder de síntese destes teoremas consideremos as fórmulas $F_1 = P \rightarrow Q$, $F_2 = \neg Q$ e $G = \neg P$, e será mostrado através de três métodos que G é uma consequência lógica de F_1 e F_2 .

Método 1 Pode-se utilizar uma tabela verdade, tal como a Tabela 4.6 para mostrar que os valores de $F_1 \wedge F_2$ cujas interpretações são 1 possuem correspondentes em G com o mesmo valor verdade.

Tabela 4.6: Demonstração de consequência lógica utilizando tabela verdade.

P	Q	$P \rightarrow Q$	$\neg Q$	$(P \rightarrow Q) \wedge \neg Q$	$\neg P$
1	1	1	0	0	0
1	0	0	1	0	0
0	1	1	0	0	1
0	0	1	1	1	1

Método 2 Pode-se utilizar o Teorema 4.1 simplesmente estendendo a Tabela 4.6, tal como apresentado na Tabela 4.7

Tabela 4.7: Demonstração de consequência lógica utilizando tabela verdade estendida.

P	Q	$P \rightarrow Q$	$\neg Q$	$(P \rightarrow Q) \wedge \neg Q$	$\neg P$	$((P \rightarrow Q) \wedge \neg Q) \rightarrow \neg P$
1	1	1	0	0	0	1
1	0	0	1	0	0	1
0	1	1	0	0	1	1
0	0	1	1	1	1	1

Alternativamente, pode-se também avaliar a fórmula $((P \rightarrow Q) \wedge \neg Q) \rightarrow \neg P$ e observar que ela representa uma tautologia

$$\begin{aligned}
((P \rightarrow Q) \wedge \neg Q) \rightarrow \neg P &\equiv \neg((P \rightarrow Q) \wedge \neg Q) \vee \neg P & (r.2) \\
&\equiv \neg((\neg P \vee Q) \wedge \neg Q) \vee \neg P & (r.2) \\
&\equiv \neg((\neg P \wedge \neg Q) \vee (Q \wedge \neg Q)) \vee \neg P & (r.5b) \\
&\equiv \neg((\neg P \wedge \neg Q) \vee \mathbf{0}) \vee \neg P & (r.8b) \\
&\equiv \neg((\neg P \wedge \neg Q)) \vee \neg P & (r.6a) \\
&\equiv (P \vee Q) \vee \neg P & (r.10b) \\
&\equiv (Q \vee P) \vee \neg P & (r.3a) \\
&\equiv Q \vee (P \vee \neg P) & (r.4a) \\
&\equiv Q \vee \mathbf{1} & (r.8a) \\
&\equiv \mathbf{1} & (r.7a)
\end{aligned}$$

Método 3 Tal como no método 2, pode-se usar a tabela verdade para mostrar que $(P \rightarrow Q) \wedge \neg Q \wedge P$ é falsa para todas as interpretações, tal como na Tabela 4.8

Tabela 4.8: Demonstração de consequência lógica utilizando tabela verdade e contradição.

P	Q	$P \rightarrow Q$	$\neg Q$	$(P \rightarrow Q) \wedge \neg Q \wedge P$
1	1	1	0	0
1	0	0	1	0
0	1	1	0	0
0	0	1	1	0

Também pode-se provar a inconsistência de $(P \rightarrow Q) \wedge \neg Q \wedge P$ mostrando que trata-se de uma contradição.

$$\begin{aligned}
(P \rightarrow Q) \wedge \neg Q \wedge P &\equiv (\neg P \vee Q) \wedge \neg Q \wedge P & (r.2) \\
&\equiv (\neg P \wedge \neg Q \wedge P) \vee (Q \wedge \neg Q \wedge P) & (r.5b) \\
&\equiv \mathbf{0} \vee \mathbf{0} & (r.8b) \\
&\equiv \mathbf{0} & (r.6a)
\end{aligned}$$

4.3 Lógica de Primeira Ordem

Na lógica proposicional as primitivas básicas são chamadas de átomos que expressam valores 0 ou 1. Através dos átomos são escritas as fórmulas, e estas são empregadas para expressar idéias complexas. O átomo é tratado como uma unidade indivisível, sendo ignorados sua composição e estrutura. Entretanto muitas idéias não podem ser tratadas deste modo simplificado. Considere, por exemplo, as seguintes sentenças:

Todo homem é mortal.

Uma vez que João é homem, então João é mortal.

O raciocínio intuitivo leva o leitor a reconhecer como correta a conclusão acima. Contudo, a primeira sentença não pode ser representada sob a ótica da lógica proposicional, pois não é uma sentença meramente declarativa. Neste sentido, através da lógica de primeira ordem, serão usadas mais três noções lógicas, a saber: termo, predicado e quantificador.

Suponha que se deseja representar “João ama Maria”. Primeiro define-se um predicado $ama(x, y)$, e em seguida, é feita sua instanciação, $ama(João, Maria)$. Além disto, se João for pai de Fulano, pode-se reescrever o predicado ama como sendo $ama(pai - de(Fulano), Maria)$.

Informalmente, dizemos que João, Maria e Fulano são *símbolos individuais*, ou *constantes*. As constantes representam qualquer objeto sobre o qual se deseja descrever algo. A noção de constante é uma representação ampla, podendo ser dividida em constantes concretas (a praia, o carro, ...), constantes abstratas (a amizade, o amor, ...) e constantes fictícias (o Papai Noel, o Homem-Aranha, ...).

As variáveis x e y são chamadas de *símbolos variáveis*. Estes símbolos variáveis permitem estabelecer fatos a respeito de objetos do Universo, sem que haja necessidade de torná-los explícitos. Desta forma, as variáveis x e y podem assumir um valor *verdadeiro* ou *falso* de acordo com a quantificação e a substituição que será feita.

A função $pai - de$ e ama são chamados de *símbolos predicativos*. Eles são res-

ponsáveis por representar propriedades ou relações entre objeto do Universo. Deste modo, estas funções mapeiam a pessoa chamada *Fulano* sobre aquela que é seu pai. Além disto, observe que os símbolos predicativos recebem um certo número de argumentos. Assim, se um símbolo predicativo P possui n argumentos, chama-se P de símbolo predicativo de n -argumentos.

Definição 4.7 [Termo] Termo é definido recursivamente como:

- (i) Uma constante é um termo;
- (ii) Uma variável é um termo;
- (iii) Se P é um predicado de n -argumentos, e t_1, \dots, t_n são termos, então $P(t_1, \dots, t_n)$ é um termo;
- (iv) Todos os termos são obtidos a partir destas regras.

A constante *Fulano* é um termo, e o predicado *pai – de* é um símbolo predicativo de 1-argumento. Assim, *pai – de(Fulano)* é um termo, bem como *pai – de(pai – de(Fulano))*, denotando o avô de Fulano, também é um termo.

Definição 4.8 [Átomo] Se P é um símbolo predicativo de n -argumentos e t_1, \dots, t_n são termos, então $P(t_1, \dots, t_n)$ é um átomo.

Uma vez que um átomo se encontra definido, pode-se utilizar os mesmos conectivos lógicos da lógica proposicional. Além disto, dado que foram introduzidas variáveis à representação, também é possível utilizar o **quantificador universal** \forall e o **quantificador existencial** \exists . Se x for uma variável, então $(\forall x)$ é lido com "para todo x " ou "para cada x ", enquanto que $(\exists x)$ é dito "existe um x ", "para algum x ", "para ao menos um x ".

As sentenças listadas a seguir apresentam suas respectivas representações em lógica, chamadas *fórmulas*:

- (a) Todo número racional é um número real.

$$(\forall x) (\mathbb{Q}(x) \rightarrow \mathbb{R}(x))$$

(b) Existe um número que é primo.

$$(\exists x) \text{primo}(x)$$

(c) Para todo número x , existe um número y tal que $x < y$.

$$(\forall x)(\exists y) \text{menor}(x, y)$$

O escopo de um quantificador em uma fórmula corresponde ao trecho desta fórmula ao qual o quantificador se aplica. Por exemplo, o escopo de ambos os quantificadores universal e existencial na fórmula $(\forall x)(\exists y) \text{menor}(x, y)$ é $\text{menor}(x, y)$. O escopo para o quantificador universal na fórmula $(\forall x) (\mathbb{Q}(x) \rightarrow \mathbb{R}(x))$ é $\mathbb{Q}(x) \rightarrow \mathbb{R}(x)$. Observe que na primeira situação o escopo não está explicitamente definido pelos símbolos (e), enquanto na segunda situação este escopo encontra-se explícito.

Definição 4.9 Uma ocorrência de uma variável em x em uma fórmula é *ligada* se x é uma variável de um quantificador na fórmula, ou se x está no escopo de um quantificador $(\forall x)$ ou $(\exists x)$ na fórmula. Caso contrário, a ocorrência de x é *livre*.

Definição 4.10 [Fórmula Bem Formada] Uma fórmula bem formada, ou simplesmente fórmula, na lógica de primeira ordem é definida como:

- (i) Um átomo é uma fórmula;
- (ii) Se F e G são fórmulas, então $(\neg F)$, $(F \vee G)$, $(F \wedge G)$, $(F \rightarrow G)$ e $(F \leftrightarrow G)$ são fórmulas;
- (iii) Se F é uma fórmula e x é uma variável livre de F , então $(\forall x) F$ e $(\exists x) F$ são fórmulas;
- (iv) Fórmulas são geradas a partir de um número finito de aplicações de (i), (ii) e (iii).

Sempre que possível, e baseando-se na simplificação da representação da fórmula, pode-se omitir alguns parêntesis. Por exemplo, $((\exists x) A) \vee B$ pode ser simplificado para $(\exists x) A \vee B$.

A representação do conhecimento empregado predicados envolve o uso de quatro categorias de sentenças, chamadas *enunciados categóricos*:

- *Universal Afirmativo*: estabelece que um relacionamento é um subconjunto de outro, normalmente representado com uma fórmula similar a $(\forall x) [P(x) \rightarrow Q(x)]$.
Sentença: Todos os políticos são ladrões.
Representação: $(\forall x) [político(x) \rightarrow ladrão(x)]$
- *Universal Negativo*: estabelece uma relação de disjunção entre relacionamentos, normalmente representado com uma fórmula similar a $(\forall x) [P(x) \rightarrow \neg Q(x)]$.
Sentença: Nenhum político é honesto.
Representação: $(\forall x) [político(x) \rightarrow \neg honesto(x)]$
- *Particular Afirmativo*: estabelece que os relacionamentos P e Q têm uma intersecção não vazia, normalmente representado com uma fórmula similar a $(\exists x) [P(x) \wedge Q(x)]$.
Sentença: Alguns políticos são honestos.
Representação: $(\exists x) [político(x) \wedge honesto(x)]$
- *Particular Negativo*: estabelece que existem elementos do relacionamento P que não estão em Q , normalmente representado com uma fórmula similar a $(\exists x) [P(x) \wedge \neg Q(x)]$.
Sentença: Alguns políticos não são honestos.
Representação: $(\exists x) [político(x) \wedge \neg honesto(x)]$

Exemplo 4.3 Inicialmente, considere que se deseja traduzir as sentenças "Todo homem é mortal. João é homem. Deste modo, João é mortal". Denote " x é homem" por $homem(x)$, e " x é mortal" por $mortal(x)$. Assim, a sentença "todo homem é mortal" pode ser representada por $(\forall x)(homem(x) \rightarrow mortal(x))$. Observe que até esse ponto nada foi modelado sobre a constante João. Assim, "João é homem" é representado por $homem(João)$, e ainda, "João é mortal" por $mortal(João)$. Desta forma, a sentença final é $(\forall x)(homem(x) \rightarrow mortal(x)) \wedge homem(João) \rightarrow mortal(João)$.

Na lógica proposicional, uma interpretação é uma atribuição de valores verdade para um átomo. Em lógica de primeira ordem, uma vez que existem variáveis envolvidas, as interpretações não são tão imediatas. A idéia central é definir um domínio D sobre o qual a fórmula é aplicada. Com base nos valores verdade assu-

midos pelos predicados instanciados por elementos deste domínio D , verifica-se qual o resultado é obtido para toda a fórmula.

Exemplo 4.4 Considere as fórmulas $(\forall x)P(x)$ e $(\exists x)\neg P(x)$. Uma interpretação pode ser descrita como se segue:

Domínio: $D = 1, 2$

Valores assumidos por P : $P(1) = 1 \mid P(2) = 0$

O valor de $(\forall x)P(x)$ para esta interpretação é 0 porque $P(x)$ não é 1 para ambos $x = 1$ e $x = 2$. Por outro lado, uma vez que $\neg P(2) = 1$, então $(\exists x)\neg P(x)$ é 1 para esta interpretação.

Exemplo 4.5 Considere a fórmula $(\forall x)(\exists y)P(x, y)$. A interpretação é definida como se segue:

Domínio: $D = 1, 2$

Valores assumidos por P : $P(1, 1) = 1 \mid P(1, 2) = 0 \mid P(2, 1) = 0 \mid P(2, 2) = 1$

Se $x = 1$, então existe um y , no caso $y = 1$, que torna o predicado $P(1, y) = 1$.

Se $x = 2$, então também existe um y , neste caso $y = 2$, o que torna o predicado $P(2, y) = 1$.

Desta forma, para esta interpretação $(\forall x)(\exists y)P(x, y) = 1$.

Exemplo 4.6 Considere a fórmula $G : (\forall x)(P(x) \rightarrow Q(f(x), a))$. Nesta fórmula existe uma constante a , uma função f , e dois predicados P e Q . A interpretação I de G é avaliada conforme a seguir:

Domínio: $D = 1, 2$

Valores assumidos por a : $a = 1$

Valores assumidos por f : $f(1) = 2 \mid f(2) = 1$

Valores assumidos por P : $P(1) = 0 \mid P(2) = 1$

Valores assumidos por Q : $Q(1, 1) = 1 \mid Q(1, 2) = 1 \mid Q(2, 1) = 0 \mid Q(2, 2) = 1$

Se $x = 1$, então

$$\begin{aligned} P(x) \rightarrow Q(f(x), a) &= P(1) \rightarrow Q(f(1), a) \\ &= P(1) \rightarrow Q(2, 1) \\ &= 0 \rightarrow 0 = 1 \end{aligned}$$

Se $x = 2$, então

$$\begin{aligned} P(x) \rightarrow Q(f(x), a) &= P(2) \rightarrow Q(f(2), a) \\ &= P(2) \rightarrow Q(1, 1) \\ &= 1 \rightarrow 1 = 1 \end{aligned}$$

Uma vez que $P(x) \rightarrow Q(f(x), a)$ é 1 para todos os elementos x no domínio D , a fórmula $G = 1$ sob a interpretação I .

4.4 Forma Normal Prenex

Em lógica proposicional foram introduzidos os conceitos de formas normais conjuntivas e disjuntivas. Em lógica de primeira ordem também existe uma forma normal, neste caso conhecida como *forma normal prenex*. A razão principal para se considerar fórmulas escritas em forma normal prenex deve-se às tentativas de simplificar o processo de prova, conforme será visto mais adiante.

Definição 4.11 [Forma Normal Prenex] A fórmula F em lógica de primeira ordem é dita esta em forma normal prenex se e somente a fórmula F encontra-se segundo o padrão $(Q_1x_1) \dots (Q_nx_n)(M)$, onde (Q_ix_i) pode ser tanto $(\forall x_i)$ como $(\exists x_i)$, e M uma fórmula sem quantificador algum. $(Q_1x_1) \dots (Q_nx_n)$ são chamados de prefixo e M de matriz da fórmula F .

Deste modo, na Fórmula Normal Prenex, o escopo dos quantificadores deve ser a fórmula inteira. Assim, a fórmula $(\forall x)(\forall y)(P(x, y) \wedge Q(y))$ e a fórmula $(\forall x)(\forall y)(\exists z)(Q(x, y) \rightarrow R(z))$ encontram-se na forma normal prenex. As regras de equivalência para normalização previstas na Tabela 4.5 de lógica proposicional continuam válidas para a lógica de primeira ordem. Entretanto, para a lógica de primeira ordem também existem algumas outras regras de equivalência válidas, tal como apresentado na Tabela 4.9, cujas as demonstrações são simples e deixadas para o leitor. Na tabela, considere que G é uma fórmula que não contém a variável x , e ainda, Q pode ser tanto o quantificador \forall , como o \exists .

Neste momento, é importante observar que o quantificador universal \forall e o

Tabela 4.9: Regras de equivalência para normalização prenex.

$$\begin{aligned}
r.1a) \quad & (Qx)F[x] \vee G \equiv (Qx)(F[x] \vee G) \\
r.1b) \quad & (Qx)F[x] \wedge G \equiv (Qx)(F[x] \wedge G) \\
r.2a) \quad & \neg((\forall x)F[x]) \equiv (\exists x)(\neg F[x]) \\
r.2b) \quad & \neg((\exists x)F[x]) \equiv (\forall x)(\neg F[x]) \\
r.3a) \quad & (\forall x)F[x] \wedge (\forall x)H[x] \equiv (\forall x)(F[x] \wedge H[x]) \\
r.3b) \quad & (\exists x)F[x] \vee (\exists x)H[x] \equiv (\exists x)(F[x] \vee H[x])
\end{aligned}$$

quantificador existencial \exists não são distributivos sobre os conectivos \vee e \wedge , respectivamente. Isto é,

$$\begin{aligned}
(\forall x)F[x] \vee (\forall x)H[x] &\neq (\forall x)(F[x] \vee H[x]) \\
(\exists x)F[x] \wedge (\exists x)H[x] &\neq (\exists x)(F[x] \wedge H[x])
\end{aligned}$$

Para casos como estes é preciso usar de alguns artifícios. Uma vez que a variável ligada na fórmula pode ser considerada como uma variável qualquer (ordinária), toda variável x pode ser renomeada para z , e a fórmula $(\forall x)H[x]$ é alterada para $(\forall z)H[z]$. Suponha que foi escolhida uma variável z que não aparece em $F[x]$. Assim tem-se,

$$\begin{aligned}
(\forall x)F[x] \vee (\forall x)H[x] &\equiv (\forall x)F[x] \vee (\forall z)H[z] \\
&\equiv (\forall x)(\forall z)(F[x] \vee H[z])
\end{aligned}$$

Similarmente tem-se,

$$\begin{aligned}
(\exists x)F[x] \wedge (\exists x)H[x] &\equiv (\exists x)F[x] \wedge (\exists z)H[z] \\
&\equiv (\exists x)(\exists z)(F[x] \wedge H[z])
\end{aligned}$$

De modo geral, o processo pra transformar fórmulas quaisquer para uma representação na forma normal prenex segue um processo sistemático, tal como

descrito no Algoritmo 4.1

Data: Fórmula da lógica de 1ª ordem desnormalizada

Result: Fórmula da lógica de 1ª ordem na forma normal prenex

while *existem conectivos \rightarrow e \leftrightarrow* **do**

Substituir: $F \leftrightarrow G \equiv (F \rightarrow G) \wedge (G \rightarrow F)$;

Substituir: $F \rightarrow G \equiv \neg F \vee G$;

end

while \neg *não está o mais próximo possível dos átomos* **do**

Substituir: $\neg(\neg F) \equiv F$;

Substituir: $\neg(F \vee G) \equiv \neg F \wedge \neg G$;

Substituir: $\neg(F \wedge G) \equiv \neg F \vee \neg G$;

Substituir: $\neg((\forall x)F[x]) \equiv (\exists x)(\neg F[x])$;

Substituir: $\neg((\exists x)F[x]) \equiv (\forall x)(\neg F[x])$;

end

Renomear as variáveis ligadas se necessário;

while *os quantificadores não estiverem todos na esquerda da fórmula* **do**

Substituir: $(Qx)F[x] \vee G \equiv (Qx)(F[x] \vee G)$;

Substituir: $(Qx)F[x] \wedge G \equiv (Qx)(F[x] \wedge G)$;

Substituir: $(\forall x)F[x] \wedge (\forall x)H[x] \equiv (\forall x)(F[x] \wedge H[x])$;

Substituir: $(\exists x)F[x] \vee (\exists x)H[x] \equiv (\exists x)(F[x] \vee H[x])$;

Substituir: $(Q_1x)F[x] \vee (Q_2x)H[x] \equiv (Q_1x)(Q_2z)(F[x] \vee H[z])$;

Substituir: $(Q_3x)F[x] \wedge (Q_4x)H[x] \equiv (Q_3x)(Q_4z)(F[x] \wedge H[z])$;

end

Algoritmo 4.1: Transformação de fórmulas para a forma normal prenex

Exemplo 4.7 Transforme a fórmula $(\forall x)P(x) \rightarrow (\exists x)Q(x)$ para forma normal prenex.

$$\begin{aligned} G &= (\forall x)P(x) \rightarrow (\exists x)Q(x) \\ &\equiv \neg((\forall x)P(x)) \vee (\exists x)Q(x) \\ &\equiv (\exists x)(\neg P(x)) \vee (\exists x)Q(x) \\ &\equiv (\exists x)(\neg P(x) \vee Q(x)) \end{aligned}$$

Exemplo 4.8 Obtenha a forma normal prenex para a fórmula $(\forall x)(\forall y)((\exists z)P(x, z) \wedge$

$$P(y, z) \rightarrow (\exists u)Q(x, y, u).$$

$$\begin{aligned} G &= (\forall x)(\forall y)((\exists z)P(x, z) \wedge P(y, z)) \rightarrow (\exists u)Q(x, y, u) \\ G &\equiv (\forall x)(\forall y)(\neg((\exists z)P(x, z) \wedge P(y, z)) \vee (\exists u)Q(x, y, u)) \\ &\equiv (\forall x)(\forall y)((\forall z)(\neg P(x, z) \vee \neg P(y, z)) \vee (\exists u)Q(x, y, u)) \\ &\equiv (\forall x)(\forall y)(\forall z)(\exists u)(\neg P(x, z) \vee \neg P(y, z) \vee Q(x, y, u)) \end{aligned}$$

4.5 Forma de Normal de Skolen

Nas seções anteriores foi estudado como transformar uma fórmula na forma normal prenex, e também descrito como transformar a matriz em forma normal conjuntiva. Nesta seção será verificado como eliminar os quantificadores existenciais empregando a forma normal de Skolen¹. Observe que se G está na Forma Normal de Skolen se ela é oriunda de uma Prenex H , cujos quantificadores existenciais (\exists) foram retirados por Skolem. É importante observar que $G \not\equiv H$, mas H é insatisfatível se e somente se G também for, e esta é a razão principal para estudá-la.

Seja uma formula F já na forma normal prenex $(Q_1x_1) \dots (Q_nx_n)M$, onde M é uma forma normal conjuntiva. Suponha que Q_r seja um quantificador existencial no prefixo $(Q_1x_1) \dots (Q_nx_n)$, $1 \leq r \leq n$. Se nenhum quantificador universal estiver a esquerda de Q_r , escolhe-se uma nova constante c diferente das outras constantes de M , substitui-se por c todas as ocorrências de x_r em M e apaga-se Q_rx_r do prefixo. Se Q_{s1}, \dots, Q_{sm} são quantificadores universais que antecedem Q_r , $1 \leq s_1 < s_2 < \dots < s_m < r$, escolhe-se uma função de aridade m diferente dos outros símbolos funcionais, substitui-se todos os x_r em M por $f(x_{s1}, x_{s2}, \dots, x_{sm})$ e apaga-se (Q_rx_r) do prefixo. Quando este processo alcança todos os quantificadores existenciais do prefixo, diz-se que a fórmula F encontra-se na forma de Skolen. As funções e as constantes utilizadas na substituição dos quantificadores existenciais são chamadas de funções de skolenização.

Exemplo 4.9 Obtenha a forma de Skolen para a fórmula:

$$(\exists x)(\forall y)(\forall z)(\exists u)(\forall v)(\exists w)P(x, y, z, u, v, w)$$

¹Thoralf Albert Skolem(23 de maio de 1887 - 23 de março de 1963), matemático norueguês.

Nesta fórmula, $(\exists x)$ não é precedido por qualquer quantificador universal, $(\exists u)$ é precedido por $(\forall y)$ e $(\forall z)$, enquanto que $(\exists w)$ é precedido por $(\forall y)$, $(\forall z)$ e $(\forall v)$. Deste modo, substitui-se a variável existencial x pela constante a , u pela função de aridade dois $f(y, z)$ e w pela função de aridade três $g(y, z, v)$. Assim, obtém-se a seguinte fórmula: $(\forall y)(\forall z)(\forall v)P(a, y, z, f(y, z), v, g(y, z, v))$.

Exemplo 4.10 Obtenha a forma de Skolen para a fórmula:

$$(\forall x)(\exists y)(\exists z)((\neg P(x, y) \wedge Q(x, z)) \vee R(x, y, z))$$

Primeiro a matriz é transformada em forma normal conjuntiva:

$$(\forall x)(\exists y)(\exists z)((\neg P(x, y) \vee R(x, y, z)) \wedge (Q(x, z) \vee R(x, y, z)))$$

Enfim, uma vez que $(\exists y)$ e $(\exists z)$ são precedidos por $(\forall x)$, as variáveis existenciais y e z são substituídas, respectivamente pelas funções $f(x)$ e $g(x)$. Deste modo, obtém-se a seguinte forma de Skolen:

$$(\forall x)((\neg P(x, f(x)) \vee R(x, f(x), g(x))) \wedge (Q(x, g(x)) \vee R(x, f(x), g(x))))$$

Seja, por exemplo, a fórmula $(\forall x)(\exists y)\text{é_parente-de}(y, x)$, cujo entendimento é que toda pessoa x tem ao menos um parente y . Se simplesmente trocarmos para $(\forall x)\text{é_parente-de}(\text{João}, x)$ acabaria-se por fazer uma interpretação incorreta uma vez que esta fórmula indica que João é parente de todas as pessoas x . A interpretação correta é que a variável x deveria ser substituída por um "parente genérico" do domínio de y , sem ser uma variável. Desta forma, é razoável fazer $y = f(x)$, pois y depende de x . Logo a substituição correta é $(\forall x)\text{é_parente-de}(f(x), x)$.

Definição 4.12 [Cláusula] Uma cláusula é uma disjunção de literais.

As disjunções $\neg P(x, f(x)) \vee R(x, f(x), g(x))$ e $Q(x, g(x)) \vee R(x, f(x), g(x))$ na forma de skolen do Exemplo 4.10 são cláusulas. Um conjunto S de cláusulas é o conjunto de todas as cláusulas, onde cada variável em S é considerada como sendo quantificada por um quantificador universal. Deste modo, as formas de Skolen podem ser escritas como um conjunto de cláusulas. Por exemplo, a forma de Skolen do Exemplo

4.10 pode ser representada pelo conjunto $\{\neg P(x, f(x)) \vee R(x, f(x), g(x)), Q(x, g(x)) \vee R(x, f(x), g(x))\}$. O conjunto S de cláusulas será um ponto chave no procedimento para prova automática de teoremas.

4.6 Teorema de Herbrand

Até o presente momento, foi apresentado um ferramental de suporte para ser empregado na prova de teoremas. Nesta seção serão considerados os procedimentos desta prova. A tarefa de se encontrar um procedimento para averiguar a validade (consistência) de uma fórmula foi uma tarefa estudada no passado. Primeiramente foi tentado por Leibniz, e em seguida por Peano quase duzentos anos depois, próximo a virada para o século XX. Na década de 1920 foram dados os primeiros passos efetivos neste sentido por Herbrand, mas foi só em 1936, através de Church e Turing que isso se mostrou ser uma tarefa impossível. De forma independente, tanto Church como Turing mostraram que não existe um procedimento genérico para checar a validade de fórmulas em lógica de primeira ordem. Entretanto, para o caso específico no qual as fórmulas são de fato válidas, existem procedimentos que podem chegar a esta conclusão. Para o caso em que as fórmulas não são válidas, estes procedimentos nunca se encerram. Sob a ótica de Church e Turing, está é, na melhor das hipóteses, a situação limite que se pode atingir com estes procedimentos de prova.

Uma abordagem importante para a prova automática de teoremas foi dada por Herbrand² em 1931. Por definição, uma fórmula é válida se ela assumir valores 1 para todas as interpretações. Herbrand propôs um algoritmo para encontrar uma interpretação que refuta uma determinada fórmula. Todavia, se uma fórmula realmente é válida, tal interpretação não deverá existir, e seu algoritmo deverá abortar após um número finito de tentativas. Seu método era impraticável de se aplicar até a invenção do computador digital. Só após o artigo de Robinson [64, 65] em 1965, junto com o desenvolvimento do princípio da resolução, foi possível o desenvolvi-

²Jacques Herbrand (1908-1931) foi um matemático francês, cujo estudo principal foi a teoria de demonstração e as funções recursivas gerais intitulado "On the consistency of arithmetic". Em uma escalada dos Alpes franceses com dois amigos, caiu nas montanhas de granito do Massif des Écrins e morreu. "On the consistency of arithmetic" foi publicado postumamente.

mento dos provadores.

Os procedimentos de Hebrand não seguem a tradição de prova direta, mas sim busca a prova por refutação, demonstrando que a negação da fórmula é inconsistente. A idéia intuitiva por de trás deste procedimento é de que se a negação de um teorema for falsa então ele será verdadeiro (princípio do meio excluído).

Por definição, um conjunto S de cláusulas é insatisfatório se e somente se ele é falso (0) sob todas as interpretações sobre todos os domínios. Uma vez que é inconveniente e impossível considerar todas as interpretações sobre todos os domínios, seria interessante se fosse possível fixar um domínio especial H no qual S é insatisfatório se e somente se S é falso sob todas as interpretações sobre este domínio. Felizmente, existe tal conjunto, e ele é chamado de *universo de Herbrand* de S , sendo definido conforme se segue.

Definição 4.13 [Universo de Hebrand] Seja H_0 um conjunto de constantes que ocorrem em S . Se não existe constante em S , então H_0 consiste de uma única constante, $H_0 = \{a\}$. Para $i = 0, 1, 2, \dots$ seja H_{i+1} a união de H_i e o conjunto de todos os termos da forma $f^n(t_1, \dots, t_n)$ para todas as funções f^n que ocorrem em S , onde t_j , $j = 1, \dots, n$ pertencem ao conjunto H_i . Os H_i são chamados de conjuntos nível i de constantes de S , e H_∞ é chamado de *Universo de Herbrand* de S .

Exemplo 4.11 Seja $S = \{P(a), \neg P(x) \vee P(f(x))\}$

$$\begin{aligned} H_0 &= \{a\} \\ H_1 &= \{a, f(a)\} \\ H_2 &= \{a, f(a), f(f(a))\} \\ &\vdots \\ H_\infty &= \{a, f(a), f(f(a)), f(f(f(a))), \dots\} \end{aligned}$$

Exemplo 4.12 Seja $S = \{P(x) \vee Q(x), R(z), T(y) \vee \neg W(y)\}$. Uma vez que não existe constante em S , tem-se que $H_0 = \{a\}$. Além disto, também não existem funções em S , logo $H_0 = H_1 = \dots = H_\infty = \{a\}$.

Exemplo 4.13 Seja $S = \{P(f(x), a, g(x, y), b)\}$

$$\begin{aligned} H_0 &= \{a, b\} \\ H_1 &= \{a, b, f(a), f(b), g(a, a), g(a, b), g(b, a), g(b, b)\} \\ H_2 &= \{a, b, f(a), f(b), g(a, a), g(a, b), g(b, a), g(b, b), f(f(a)), f(f(b)), f(g(a, a)), \\ &\quad f(g(a, b)), f(g(b, a)), f(g(b, b)), \dots\} \\ &\vdots \end{aligned}$$

Definição 4.14 [Base de Herbrand] Seja S um conjunto de cláusulas. O conjunto de todas as fórmulas atômicas de S é chamado de Base de Herbrand, ou conjunto atômico de S .

Exemplo 4.14 Seja $S = \{P(x, a), \neg Q(x) \vee P(f(x), b)\}$ contendo duas constantes e um símbolo funcional. Assim,

$$\begin{aligned} H_0 &= \{a, b\} \\ H_1 &= \{a, b, f(a), f(b)\} \\ H_2 &= \{a, b, f(a), f(b), f(f(a)), f(f(b))\} \\ &\vdots \\ H_\infty &= \{a, b, f(a), f(b), f(f(a)), f(f(b)), \dots\} \end{aligned}$$

Como P e Q são símbolos predicativos, tem-se que a base de Hebrand é dada por $A = \{P(a, b), Q(a), Q(b), P(a, f(a)), P(a, f(b)), Q(f(a)), Q(f(b)), \dots\}$

Chama-se árvore semântica a árvore binária tal que os descendentes de cada nó são respectivamente elementos da base de Herbrand e sua negação (fórmulas atômicas complementares). A árvore semântica T associada ao conjunto de cláusulas do Exemplo 4.14 é apresentada na Figura 4.1. Nesta representação, optou-se por explicitar os valores verdades assumidos pelas fórmulas durante a transição de um nó para outro.

Seja a fórmula $(\forall x)(\forall y)((P(x) \rightarrow Q(x)) \wedge P(f(y)) \wedge \neg Q(f(y)))$. Assim, tem-se que a fórma de skolen é dada por

$$\begin{aligned} G &\equiv (\forall x)(\forall y)((P(x) \rightarrow Q(x)) \wedge P(f(y)) \wedge \neg Q(f(y))) \\ &\equiv (\forall x)(\forall y)((\neg P(x) \vee Q(x)) \wedge P(f(y)) \wedge \neg Q(f(y))) \end{aligned}$$

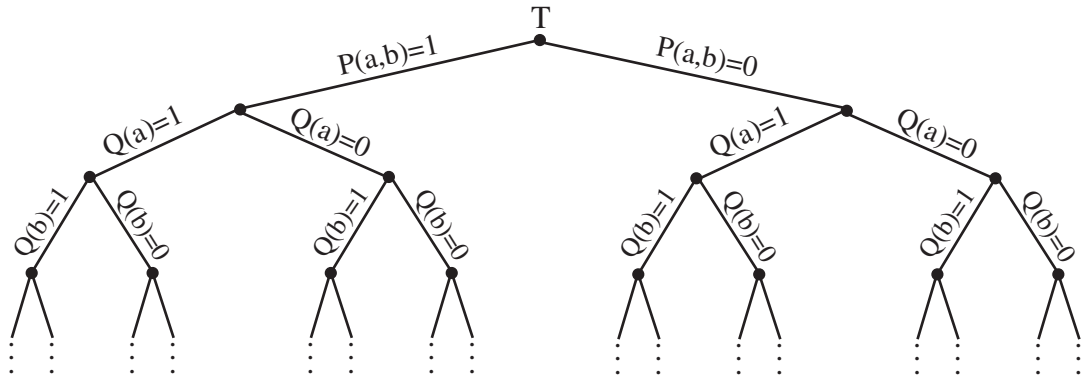


Figura 4.1: Árvore semântica para $S = \{P(x, a), \neg Q(x) \vee P(f(x), b)\}$.

Logo, o conjunto de cláusulas é dado por $S = \{\neg P(x) \vee Q(x), P(f(y)), \neg Q(f(y))\}$.

Deste modo tem-se:

$$H_0 = \{a\}$$

$$H_1 = \{a, f(a)\}$$

$$H_2 = \{a, f(a), f(f(a))\}$$

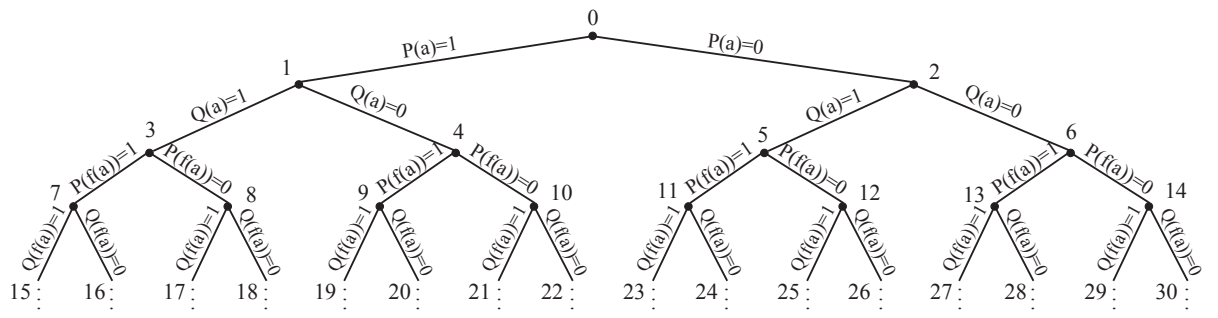
\vdots

$$H_\infty = \{a, f(a), f(f(a)), \dots\}$$

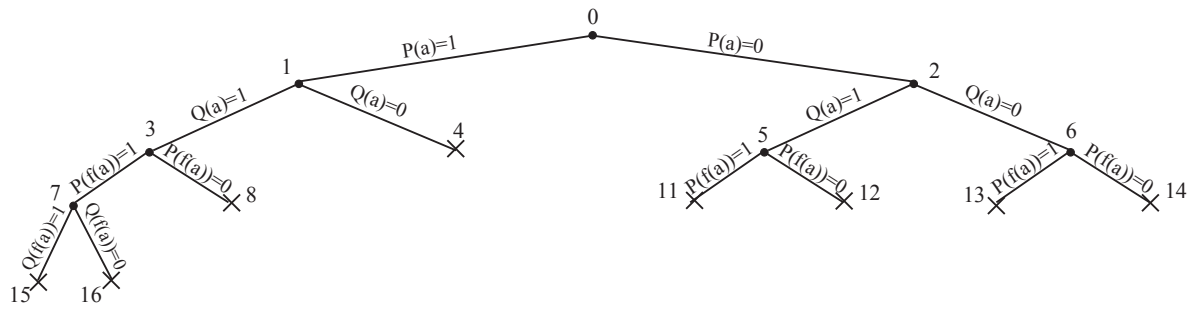
Como se tem símbolos predicativos P e Q , tem-se que a base de Herbrand é dada por $A = \{P(a), Q(a), P(f(a)), Q(f(a)), \dots\}$. A árvore semântica cheia associada a este conjunto de cláusulas é apresentada na Figura 4.2(a).

Os caminhos existentes em uma árvore semântica oferecem interpretações para o conjunto de cláusulas S que representa, podendo satisfazê-lo ou não. Sob esta ótica, chama-se a atenção para o fato de que é comum que as árvores semânticas cheias possuam uma altura infinita, dado que a própria base de Herbrand pode ser um conjunto infinito. Consequentemente, o conjunto de interpretações também é infinito, o que dificulta de sobremaneira uma solução computacional.

Se o conjunto de cláusula S não for passível de ser satisfeito, então todas as interpretações irão falhar em tentar assumir um valor verdade 1 para S . Observe que quando uma interpretação I , isto é, um caminho na árvore semântica falsifica S , pode-se ignorar as demais ramificações abaixo deste caminho. Desta forma, pode-se efetuar as definições a seguir.



(a)



(b)

Figura 4.2: Árvore semântica para $S = \{\neg P(x) \vee Q(x), P(f(y)), \neg Q(f(y))\}$: (a) cheia, (b) fechada.

Definição 4.15 [Nó de Falha] Um nó N é um nó de falha se a interpretação $I(N)$ associada ao caminho desde a raiz da árvore semântica até o nó N em questão falsifica ao menos uma das cláusulas de S . Os nós desdobrados após o nó de falha são ignorados, isto é, são podados da árvore diminuindo a altura daquele ramo.

Definição 4.16 [Árvore Semântica Fechada] Uma árvore semântica é dita ser fechada se e somente se todos os ramos da árvore se encerram em nós de falha.

Definição 4.17 [Nó de Inferência] Um nó N de uma árvore semântica fechada é chamado de nó de inferência se todos os nós descendentes imediatos de N são nós de falha.

Neste sentido, retornando ao exemplo da Figura 4.2 cujo conjunto de cláusulas é $S = \{\neg P(x) \vee Q(x), P(f(y)), \neg Q(f(y))\}$ pode-se concluir que a árvore da Figura 4.2(b) corresponde à sua árvore semântica fechada. A descrição do processo de poda, viabilizado pela identificação dos nós de falha é detalhado a seguir.

De modo a facilitar que o leitor acompanhe o processo de poda da árvore, as cláusulas de S serão chamadas de $C1$, $C2$ e $C3$, onde $C1 = \neg P(x) \vee Q(x)$, $C2 = P(f(y))$ e $C3 = \neg Q(f(y))$. Além disto, observe que os nós da árvore semântica (ver Figura 4.2) encontram-se numerados segundo um percurso em nível. As implementações de provadores automáticos de teoremas não utilizam necessariamente este percurso. Nós o fazemos apenas por uma questão didática.

- Caminho 0 – 1: $P(a) = 1$ não torna falsa nenhuma das cláusulas. O valor verdade deste caminho pode vir a tornar a cláusula $C1$ falsa, mas isto dependerá de $Q(x)$. Nada pode ser afirmado sobre as demais cláusulas.
- Caminho 0 – 2: $P(a) = 0$ não torna falsa nenhuma das cláusulas.
- Caminho 0 – 1 – 3: $P(a) = 1$, $Q(a) = 1$ não torna falsa nenhuma das cláusulas.
- Caminho 0 – 1 – 4: $P(a) = 1$, $Q(a) = 0$ falsifica a cláusula $C1$, logo trata-se de um nó de falha. Os demais desdobramentos deste nó são removidos.
- Caminho 0 – 2 – 5: $P(a) = 0$, $Q(a) = 1$ não torna falsa nenhuma das cláusulas.

- Caminho 0–2–6: $P(a) = 0, Q(a) = 0$ não torna falsa nenhuma das cláusulas.
- Caminho 0–1–3–7: $P(a) = 1, Q(a) = 1, P(f(a)) = 1$ não torna falsa nenhuma das cláusulas. Chama-se a atenção para o início da utilização do elemento de H_∞ chamado de $f(a)$. Isto significa que, a partir deste ponto, devem ser consideradas todas as substituições envolvendo a e $f(a)$.
- Caminho 0–1–3–8: $P(a) = 1, Q(a) = 1, P(f(a)) = 0$ falsifica a cláusula $C2$, logo trata-se de um nó de falha.
- Caminho 0–2–5–11: $P(a) = 0, Q(a) = 1, P(f(a)) = 1$ não provocam nenhuma falsidade quando procedemos a substituição de $x = a$ e $y = a$. Entretanto, a substituição de $f(y) = a$ falsifica a cláusula $C2$, de onde se conclui que trata-se de um nó de falha.
- Caminho 0–2–5–12: $P(a) = 0, Q(a) = 1, P(f(a)) = 0$ falsifica a cláusula $C2$, logo trata-se de um nó de falha.
- Caminho 0–2–6–13: $P(a) = 0, Q(a) = 0, P(f(a)) = 1$ não provocam nenhuma falsidade quando procedemos a substituição de $x = a$ e $y = a$. Entretanto, a substituição de $f(y) = a$ falsifica a cláusula $C2$, de onde se conclui que trata-se de um nó de falha.
- Caminho 0–2–6–14: $P(a) = 0, Q(a) = 0, P(f(a)) = 0$ falsifica a cláusula $C2$, logo trata-se de um nó de falha.
- Caminho 0–1–3–7–15: $P(a) = 1, Q(a) = 1, P(f(a)) = 1, Q(f(a)) = 1$ falsifica a cláusula $C3$, logo trata-se de um nó de falha.
- Caminho 0–1–3–7–16: $P(a) = 1, Q(a) = 1, P(f(a)) = 1, Q(f(a)) = 0$ não provocam nenhuma falsidade quando procedemos a substituição de $x = a$ e $y = a$. Entretanto, a substituição de $f(y) = a$ falsifica a cláusula $C3$, de onde se conclui que trata-se de um nó de falha.

Teorema 4.3 [Teorema de Herbrand] Um conjunto de cláusulas S é insatisfatível se e somente se para sua árvore semântica completa existe uma árvore semântica fechada finita.

Deste modo, um conjunto de cláusulas S é insatisfatível se existe um conjunto finito de instâncias de S que é insatisfatível. Sob esta ótica, o conjunto $S = \{\neg P(x) \vee Q(x), P(f(y)), \neg Q(f(y))\}$, cujas árvores semânticas foram ilustradas na Figura 4.2 é insatisfatível. O teorema, do modo como é colocado, sugere a implementação proposta no Algoritmo 4.2.

Data: Conjunto de cláusulas S

Result: S é satisfatível, ou insatisfatível

$B = \emptyset$;

while B é satisfatível **do**

$b = \text{nova-instância}(S)$;

$B = B \cup \{b\}$;

end

Algoritmo 4.2: Algoritmo imediato para a implementação do Teorema de Herbrand

Observe que este algoritmo não garante sempre obter uma resposta dado que o loop somente se encerra no momento em que B passa a apresentar um conjunto de cláusulas insatisfatíveis. Neste sentido torna-se importante definir uma estratégia de produção de cláusulas capazes de abreviar a execução do programa. Entre estas estratégias pode-se destacar como mais significativas o método de Gilmore [66], o método de Davis-Putman [67] e finalmente o método da resolução de Robinson [64, 65].

4.7 O Princípio da Resolução

O Teorema de Herbrand permite decidir se um conjunto de cláusulas dadas é insatisfatível. Porém, como a árvore semântica cresce exponencialmente, dependendo das cláusulas em questão, a quantidade de elementos envolvidos pode superar a capacidade de armazenamento dos maiores computadores existentes, o que a torna impraticável.

Para evitar esta excessiva geração de elementos, Robinson estabeleceu em 1965 outro algoritmo que pode ser aplicado diretamente sobre qualquer conjunto S de cláusulas para verificar sua insatisfabilidade. Sua idéia essencial é verificar se S

possui a cláusula vazia, ou se não, se esta pode ser derivada de S . Caso isto ocorra, S é insatisfatível.

Para isto é importante entender a argumentação da dedução do *modus tollendo ponens* (literalmente, o modo no qual, negando, se afirma) que atualmente é conhecido como silogismo disjuntivo. Basicamente a regra é que:

$$((p \vee q) + (\neg p)) \Rightarrow q$$

ou seja, considere as seguintes afirmações: “Ou ele está mentindo, ou eu estou doido”, mas “Eu não estou doido” logo “Ele está mentindo”.

Assim o *modus tollendo ponens* pode ser visto como uma regra de inferência que pode ser usada para gerar novas cláusulas de S . Acrescendo à S estas cláusulas, alguns nós da árvore original tornam-se nós de falha. Então, o número de nós pode ser reduzido e a cláusula vazia pode eventualmente ser derivada, explicitando a contradição existente.

Estendendo a idéia para qualquer par de cláusulas (não necessariamente unitárias), pode-se enunciar o Princípio da Resolução da seguinte forma: para qualquer par de cláusulas $C1$ e $C2$, se existir um literal $L1$ em $C1$ que for complementar de outro literal $L2$ de $C2$, então remova $L1$ de $C1$ e $L2$ de $C2$ e construa a disjunção dos remanescentes nas cláusulas. A nova cláusula construída é o *resolvente* de $C1$ e $C2$, ou seja:

$$\left(\underbrace{\overbrace{(p)}^{L1} \vee q}_{C1} + \underbrace{\overbrace{(\neg p)}^{L2}}_{C2} \right) \Rightarrow \underbrace{q}_{\text{resolvente}}$$

Assim, o procedimento de prova por resolução usa sentenças na forma de cláusulas. Inicialmente os axiomas e a negação do teorema que se deseja demonstrar são convertidos em cláusulas. Então, novas cláusulas, os resolventes, são deduzidas pela regra de inferência do *modus tollendo ponens*. O teorema principal do procedimento estabelece que se um resolvente não for satisfeito, então nenhum de seus antecedentes o será, o que inclui a negação do teorema inicial. Seu objetivo é obter a cláusula vazia, uma contradição explícita.

Seja o conjunto de cláusulas $S = \{P \vee Q, \neg P \vee Q, P \vee \neg Q, \neg P \vee \neg Q\}$. Logo

as cláusulas são dadas por:

$$\begin{aligned} C1 &= P \vee Q \\ C2 &= \neg P \vee Q \\ C3 &= P \vee \neg Q \\ C4 &= \neg P \vee \neg Q \end{aligned}$$

Não há como produzir a cláusula vazia Λ , mas é possível produzir o resolvente $C5 = Q$ a partir de $C1$ e $C2$. Desta forma a nova coleção de cláusulas passa a ser:

$$\begin{aligned} C1 &= P \vee Q \\ C2 &= \neg P \vee Q \\ C3 &= P \vee \neg Q \\ C4 &= \neg P \vee \neg Q \\ C5 &= Q \end{aligned}$$

Mesmo com a adição da cláusula $C5$, ainda não há como produzir a cláusula vazia Λ , mas combinando $C3$ com $C4$ chega-se ao resolvente $C6 = \neg Q$. Desta vez, a nova coleção é:

$$\begin{aligned} C1 &= P \vee Q \\ C2 &= \neg P \vee Q \\ C3 &= P \vee \neg Q \\ C4 &= \neg P \vee \neg Q \\ C5 &= Q \\ C6 &= \neg Q \end{aligned}$$

Desta vez a combinação $C5$ com $C6$ permite derivar a cláusula vazia Λ , logo S é insatisfável.

Para a lógica de primeira ordem, no entanto, nem sempre os literais complementares são evidentes, como por exemplo, nas cláusulas:

$$\begin{aligned} C1 &= P(x) \vee Q(x) \\ C2 &= \neg P(f(x)) \vee R(x) \end{aligned}$$

Eles somente aparecem se x for substituído por $f(a)$ em $C1$ e x por a em $C2$, resultando

$$\begin{aligned} C1 &= P(f(a)) \vee Q(f(a)) \quad \rightsquigarrow x = f(a) \\ C2 &= \neg P(f(a)) \vee R(a) \quad \rightsquigarrow x = a \end{aligned}$$

que permite chegar no seguinte resolvente:

$$C3 = Q(f(a)) \vee R(a)$$

No entanto, se x for substituído por $f(x)$ em $C1$ tem-se um novo resolvente $C3' = Q(f(x)) \vee R(x)$. Note que $C3$ é uma instância de $C3'$, no sentido que todas as demais cláusulas que podem ser produzidas por este processo são instâncias dela. Desta forma, quando aplicado à lógica de primeira ordem, o princípio da resolução requer que se verifique a situação onde dois predicados são unificáveis, isto é, podem ser feitos "idênticos", antes de realizarem o cancelamento para gerar a cláusula resolvente.

Com isso chega-se ao processo de unificação, cuja idéia básica é encontrar um conjunto minimal de substituições que torna duas fórmulas idênticas a fim de que se possa usar resolução. Por exemplo, os literais:

amigos(Luis, Pedro)

amigos(Luis, pai(Luis))

Podem ser unificados. De fato, para unificar os dois literais, deve-se primeiro verificar se o predicado é o mesmo. Já o caso dos literais:

amigos(Luis, Pedro)

parentes(Luis, Pedro)

não podem ser unificados, pois os predicados são diferentes (*amigos* e *parentes*). Se os predicados combinarem, devem-se testar os respectivos argumentos. Se o primeiro combina, continua-se com o segundo e assim por diante. Um modo de implementar este teste é chamar, para cada um dos argumentos, o procedimento de unificação recursivamente: funções, predicados ou constantes só podem combinar se forem idênticas. Uma variável pode se combinar com outra, ou com uma constante, ou com uma função ou com uma expressão predicativa, com a exceção de que estas duas últimas não devem ter qualquer instância das variáveis sendo comparadas.

Definição 4.18 [Substituição] Uma substituição é um conjunto finito da forma $\theta = \{t_1/v_1, t_2/v_2, \dots, t_n/v_n\}$, onde v_i é a variável a ser substituída e t_i um termo substituto diferente de v_i . Desta forma, t_i é o substituto de v_i .

Exemplo 4.15 $\{f(z)/x, y/z\}$ e $\{a/x, g(y)/y, f(g(b))/z\}$ são substituições quais-

quer.

A expressão $E\theta$ é chamada de instância de E , e é obtida a partir de E realizando a troca simultânea de cada ocorrência da variável v_i , $1 \leq i \leq n$, em E pelos termos de t_i . Por exemplo, se $\theta = \{a/x, f(b)/y, c/z\}$ e $E = P(x, y, z)$ então $E\theta = P(a, f(b), c)$.

Definição 4.19 [Composição] Seja $\theta = \{t_1/x_1, \dots, t_n/x_n\}$ e $\lambda = \{u_1/y_1, \dots, u_m/y_m\}$ substituições quaisquer. Uma composição $\theta \circ \lambda$ é a substituição dada pelo conjunto $\{(t_1/\lambda)/x_1, \dots, (t_n/\lambda)/x_n, u_1/y_1, \dots, u_m/y_m\}$ excluídos os elementos $(t_i/\lambda)/x_i$ nos quais $t_i/\lambda = x_i$ e qualquer elemento u_j/y_j tal que y_j esteja entre $\{x_1, x_2, \dots, x_n\}$

Seja:

$$\begin{aligned}\theta &= \{t_1/v_1, t_2/v_2\} &= \{f(y)/x, z/y\} \\ \lambda &= \{u_1/y_1, u_2/y_2, u_3/y_3\} &= \{a/x, b/y, y/z\}\end{aligned}$$

Assim:

$$\begin{aligned}\lambda \circ \theta &= \{(f(y)/\lambda)/x, (z/\lambda)/y, a/x, b/y, y/z\} \\ &= \{f(b)/x, y/y, a/x, b/y, y/z\} \dots \text{substituindo } \lambda \text{ por } t \\ &= \{f(b)/x, a/x, b/y, y/z\} \dots \text{removendo a redundância } y/y \\ &= \{f(b)/x, b/y, y/z\} \dots \text{removendo } a/x \text{ porque } u_i/y_i, y_i = x \in \{x, y\} \\ &= \{f(b)/x, y/z\} \dots \text{removendo } b/y \text{ porque } u_i/y_i, y_i = y \in \{x, y\}\end{aligned}$$

Definição 4.20 [Unificador] A substituição θ é chamada unificador σ para um conjunto de expressões $\{E_1, \dots, E_k\}$, se e somente se, $E_1\theta = E_2\theta = \dots = E_k\theta$.

Definição 4.21 [Unificador Mais Geral - UMG] Um unificador σ para um conjunto de expressões $\{E_1, \dots, E_k\}$ é o unificador mais geral, se e somente se, cada unificador θ para o conjunto, existe uma substituição λ tal que $\theta = \sigma \circ \lambda$.

Definição 4.22 [Conjunto de Discordância] O conjunto de discordância de um conjunto não vazio W de expressões é dado pelos diferentes símbolos que ocupam a mesma posição relativa de argumento da expressão E .

Por exemplo, para o conjunto $W = \{P(x, \underline{f(y, z)}), P(x, \underline{a}), P(x, \underline{g(h(k(x)))})\}$, o conjunto de discordância é dados por $\{f(y, z), a, g(h(k(x)))\}$.

Desta forma, tomando por base as definições apresentadas, pode-se afirmar que o algoritmo de unificação é dados é representado pelo Algoritmo 4.3:

Data: Conjunto W de expressões provenientes das cláusulas da Base de Herbrand

Result: Unificador mais geral UMG

$k = 0$;

$W_k = W$;

$\sigma_k = \epsilon$;

while W_k não é único **do**

if $\exists v_k$ **then**

 Encontre o conjunto de discordância D_k ;

if $\exists v_k, t_k \in D_k$ onde v_k não ocorre em t_k **then**

$\sigma_{k+1} = \sigma \circ \{t_k/v_k\}$;

$W_{k+1} = W_k \{t_k/v_k\}$;

else

W não é unificável ;

$UMG = Nulo$;

 Satura k ;

end

end

$k = k + 1$;

end

$UMG = \sigma_k$ é o unificador mais geral;

Algoritmo 4.3: Unificação de expressões

Para ilustrar o funcionamento do algoritmo 4.3, considere o conjunto $W = \{P(a, x, f(g(y))), P(z, f(z), f(u))\}$. O processo para determinar o unificador mais geral é dado a seguir:

$$\begin{aligned}
\sigma_0 &= \epsilon \\
W_0 &= W \\
W_0 &\quad \text{n\~ao \acute{e} \acute{u}nico (1 s\~o predicado?)} \\
D_0 &= \{a, z\} \\
v_0 &= z \quad \text{n\~ao ocorre em } t_0 = a \\
\sigma_1 &= \sigma_0 \circ \{t_0/v_0\} \\
&= \epsilon \circ \{a/z\} \\
&= \{a/z\} \\
W_1 &= W_0 \{t_0/v_0\} \\
&= \{P(a, x, f(g(y))), P(z, f(z), f(u))\} \{a/z\} \\
&= \{P(a, x, f(g(y))), P(a, f(a), f(u))\} \\
W_1 &\quad \text{n\~ao \acute{e} \acute{u}nico (1 s\~o predicado?)} \\
D_1 &= \{x, f(a)\} \\
v_1 &= x \quad \text{n\~ao ocorre em } t_1 = f(a) \\
\sigma_2 &= \sigma_1 \circ \{t_1/v_1\} \\
&= \{a/z\} \circ \{f(a)/x\} \\
&= \{a/z, f(a)/x\} \\
W_2 &= W_1 \{t_1/v_1\} \\
&= \{P(a, x, f(g(y))), P(a, f(a), f(u))\} \{f(a), x\} \\
&= \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\} \\
W_2 &\quad \text{n\~ao \acute{e} \acute{u}nico (1 s\~o predicado?)} \\
D_2 &= \{g(y), u\} \\
v_2 &= u \quad \text{n\~ao ocorre em } t_2 = g(y) \\
\sigma_3 &= \sigma_2 \circ \{t_2/v_2\} \\
&= \{a/z, f(a)/x\} \circ \{g(y)/u\} \\
&= \{a/z, f(a)/x, g(y)/u\} \\
W_3 &= W_2 \{t_2/v_2\} \\
&= \{P(a, f(a), f(g(y))), P(a, f(a), f(u))\} \{g(y)/u\} \\
&= \{P(a, f(a), f(g(y))), P(a, f(a), f(g(y)))\} \\
&= \{P(a, f(a), f(g(y)))\} \\
W_3 &\quad \text{\acute{e} \acute{u}nico (1 s\~o predicado?)} \\
UMG &= \sigma_3 \\
&= \{a/z, f(a)/x, g(y)/u\}
\end{aligned}$$

Uma vez que o cálculo de unificador mais geral já é conhecido, pode-se retornar ao problema inicial que é o cálculo de resolventes.

Definição 4.23 [Resolvente de Primeira Ordem] Sejam $C1$ e $C2$ duas cláusulas sem variáveis em comum. Sejam $L1$ e $L2$ dois literais das cláusulas $C1$ e $C2$, respectivamente. Se o cálculo do unificador mais geral entre $L1$ e $\neg L2$ é dado por σ , então o resolvente entre $C1$ e $C2$ é dado pela cláusula $(C1\sigma - L1\sigma) \cup (C2\sigma - L2\sigma)$

Desta forma, seja um conjunto de cláusulas $S = \{P(x) \vee Q(x), \neg P(a) \vee R(x)\}$, onde $C1 = P(x) \vee Q(x)$ e $C2 = \neg P(a) \vee R(x)$. As cláusulas $C1$ e $C2$ são candidatas a serem unificadas devido aos literais $L1 = P(x)$ e $L2 = \neg P(a)$.

Para calcular o unificador mais geral de modo a ser aplicada a definição de resolvente de primeira ordem, deve-se tomar $W = \{L1, \neg L2\} = \{P(x), P(a)\}$. Assim, tem-se que o unificador mais geral é dado por:

$$\begin{aligned}
W &= \{P(x), P(a)\} \\
\sigma_0 &= \epsilon \\
W_0 &= W \\
W_0 &\text{ não é único (1 só predicado?)} \\
D_0 &= \{a, x\} \\
v_0 &= x \text{ não ocorre em } t_0 = a \\
\sigma_1 &= \sigma_0 \circ \{t_0/v_0\} \\
&= \epsilon \circ \{a/x\} \\
&= \{a/x\} \\
W_1 &= W_0 \{t_0/v_0\} \\
&= \{P(x), P(a)\} \{a/x\} \\
&= \{P(a), P(a)\} \\
&= \{P(a)\} \\
W_1 &\text{ é único (1 só predicado?)} \\
UMG &= \sigma_1 \\
&= \{a/x\}
\end{aligned}$$

De posse do unificador mais geral, pode-se então determinar o resolvente de $C1$ e $C2$.

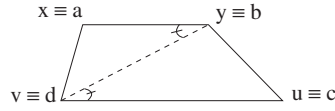


Figura 4.3: Os ângulos interiores alternados formados pela diagonal de um trapezóide são iguais.

Observe que x aparece em ambas as cláusulas $C1$ e $C2$. Contudo, esta variável x de $C1$ é apenas lexicamente igual à variável x de $C2$. Sob a ótica da semântica, estas variáveis são distintas, e por este motivo, uma delas será substituída por y para evitar equívocos, ou seja, $C2 = \neg P(a) \vee R(y)$.

Uma vez resolvida esta questão, o resolvente $C3$ de $C1$ e $C2$ é dado por:

$$\begin{aligned}
 C3 &= (C1\sigma - L1\sigma) \cup (C2\sigma - L2\sigma) \\
 &= (\{P(a), Q(a)\} - \{P(a)\}) \cup (\{\neg P(a), R(y)\} - \{\neg P(a)\}) \\
 &= \{Q(a)\} \cup \{R(y)\} \\
 &= \{Q(a), R(y)\} \\
 &= Q(a) \vee R(y)
 \end{aligned}$$

Neste momento, torna-se importante compreender todo o processo de prova automática de teoremas, a fim de ter uma clara percepção do processo como um todo. Neste sentido, será apresentado a seguir um exemplo completo de dedução. Seja, por exemplo, um trapezóide qualquer $xyuv$, no qual se deseja mostrar que são iguais os ângulos interiores alternados formados pela diagonal do mesmo. A Figura 4.3 apresenta de forma complementar e ilustrativa o que está sendo proposto.

Primeiro deve-se axiomatizar o teorema:

- Seja $T(x, y, u, v)$ um trapezóide com vértice superior esquerdo x , superior direito y , inferior direito u e inferior esquerdo v ;
- Seja $P(x, y, u, v)$ o paralelismo entre o segmento de reta xy e o segmento de reta uv ;
- Seja $E(x, y, z, u, v, w)$ a igualdade entre o ângulo \widehat{xyz} e o ângulo \widehat{uvw} .

Assim, os axiomas do teorema proposto são:

$A_1 = (\forall x)(\forall y)(\forall u)(\forall v) [T(x, y, u, v) \rightarrow P(x, y, u, v)]$, que é a definição de trapezóide.

$A_2 = (\forall x)(\forall y)(\forall u)(\forall v) [P(x, y, u, v) \rightarrow E(x, y, v, u, v, y)]$, isto é, os ângulos interiores de uma reta que cruza duas outras retas paralelas são iguais.

$A_3 = T(a, b, c, d)$, ou seja, uma instância do modelo abstrato de trapezóide que se tem na mente e que foi materializada em um pedaço de papel conforme ilustrado na Figura 4.3.

A partir destes axiomas, deve-se ser capaz de concluir que $E(a, b, d, c, d, b)$ é verdadeiro, isto é, que a fórmula a seguir é válida:

$$A_1 \wedge A_2 \wedge A_3 \rightarrow E(a, b, d, c, d, b)$$

Uma vez que se deseja realizar a prova conforme propôs Herbrand, nega-se a conclusão de modo que $A_1 \wedge A_2 \wedge A_3 \wedge \neg E(a, b, d, c, d, b)$ é uma fórmula insatisfatível.

Para construir o conjunto de cláusulas é preciso escrever a fórmula na forma normal prenex, e em seguida, skolenizá-la. Assim, a fórmula normal prenex para $A_1 \wedge A_2 \wedge A_3 \wedge \neg E(a, b, d, c, d, b)$ são:

$$\begin{aligned} A_1 &= (\forall x)(\forall y)(\forall u)(\forall v) [\neg T(x, y, u, v) \vee P(x, y, u, v)] \\ A_2 &= (\forall x)(\forall y)(\forall u)(\forall v) [\neg P(x, y, u, v) \vee E(x, y, v, u, v, y)] \\ A_3 &= T(a, b, c, d) \end{aligned}$$

A fórmula normal prenex já é uma fórmula conjuntiva, ainda que neste exemplo tenha-se uma degeneração das conjunções, haja vista que cada expressão forma um único literal. A forma de Skolen, por sua vez, exige a retirada dos quantificadores existenciais. Novamente, no caso particular deste exemplo, tais quantificadores não existem, logo:

$$S = \{\neg T(x, y, u, v) \vee P(x, y, u, v), \neg P(x, y, u, v) \vee E(x, y, v, u, v, y), T(a, b, c, d), \neg E(a, b, d, c, d, b)\}$$

Assim,

$$\begin{aligned} C1 &= \neg T(x, y, u, v) \vee P(x, y, u, v) \\ C2 &= \neg P(x, y, u, v) \vee E(x, y, v, u, v, y) \\ C3 &= T(a, b, c, d) \\ C4 &= \neg E(a, b, d, c, d, b) \end{aligned}$$

Inicialmente calcula-se a unificação de $C2$ e $C4$. Fazendo $L1 = E(x, y, v, u, v, y)$ e $L2 = \neg E(a, b, d, c, d, b)$ tem-se que $W = \{L1, \neg L2\}$. Calculando o unificador mais geral tem-se:

$$\begin{aligned}
W &= \{E(x, y, v, u, v, y), E(a, b, d, c, d, b)\} \\
\sigma_0 &= \epsilon \\
W_0 &= W \\
W_0 &\text{ não é único (1 só predicado?)} \\
D_0 &= \{a, x\} \\
v_0 &= x \text{ não ocorre em } t_0 = a \\
\sigma_1 &= \sigma_0 \circ \{t_0/v_0\} \\
&= \epsilon \circ \{a/x\} \\
&= \{a/x\} \\
W_1 &= W_0 \{t_0/v_0\} \\
&= \{E(x, y, v, u, v, y), E(a, b, d, c, d, b)\} \{a/x\} \\
&= \{E(a, y, v, u, v, y), E(a, b, d, c, d, b)\} \\
W_1 &\text{ não é único (1 só predicado?)} \\
D_1 &= \{b, y\} \\
v_1 &= y \text{ não ocorre em } t_1 = b \\
\sigma_2 &= \sigma_1 \circ \{t_1/v_1\} \\
&= \{a/x\} \circ \{b/y\} \\
&= \{a/x, b/y\} \\
W_2 &= W_1 \{t_1/v_1\} \\
&= \{E(a, y, v, u, v, y), E(a, b, d, c, d, b)\} \{a/x, b/y\} \\
&= \{E(a, b, v, u, v, b), E(a, b, d, c, d, b)\} \\
W_2 &\text{ não é único (1 só predicado?)} \\
D_2 &= \{d, v\} \\
v_2 &= v \text{ não ocorre em } t_2 = d \\
\sigma_3 &= \sigma_2 \circ \{t_2/v_2\} \\
&= \{a/x, b/y\} \circ \{d/v\} \\
&= \{a/x, b/y, d/v\}
\end{aligned}$$

$$\begin{aligned}
W_3 &= W_2 \{t_2/v_2\} \\
&= \{E(a, b, v, u, v, b), E(a, b, d, c, d, b)\} \{a/x, b/y, d/v\} \\
&= \{E(a, b, d, u, d, b), E(a, b, d, c, d, b)\} \\
W_3 &\text{ não é único (1 só predicado?)} \\
D_3 &= \{c, u\} \\
v_3 &= u \text{ não ocorre em } t_3 = c \\
\sigma_4 &= \sigma_3 \circ \{t_3/v_3\} \\
&= \{a/x, b/y, d/v\} \circ \{c/u\} \\
&= \{a/x, b/y, d/v, c/u\} \\
W_4 &= W_3 \{t_3/v_3\} \\
&= \{E(a, b, d, u, d, b), E(a, b, d, c, d, b)\} \{a/x, b/y, d/v, c/u\} \\
&= \{E(a, b, d, c, d, b), E(a, b, d, c, d, b)\} \\
&= \{E(a, b, d, c, d, b)\} \\
W_4 &\text{ é único (1 só predicado?)} \\
UMG &= \sigma_4 \\
&= \{a/x, b/y, d/v, c/u\}
\end{aligned}$$

De posse do unificador mais geral, o cálculo do resolvente $C5$, de $C2$ e $C4$ é dado por:

$$\begin{aligned}
C5 &= (C2\sigma - L1\sigma) \cup (C4\sigma - L2\sigma) \\
&= (\{\neg P(a, b, c, d), E(a, b, d, c, d, b)\} - \{E(a, b, d, c, d, b)\}) \cup \\
&\quad (\{\neg E(a, b, d, c, d, b)\} - \{\neg E(a, b, d, c, d, b)\}) \\
&= \{\neg P(a, b, c, d)\} \cup \emptyset \\
&= \neg P(a, b, c, d)
\end{aligned}$$

Deste modo tem-se que a nova coleção de clausulas é dada por:

$$\begin{aligned}
C1 &= \neg T(x, y, u, v) \vee P(x, y, u, v) \\
C2 &= \neg P(x, y, u, v) \vee E(x, y, v, u, v, y) \\
C3 &= T(a, b, c, d) \\
C4 &= \neg E(a, b, d, c, d, b) \\
C5 &= \neg P(a, b, c, d)
\end{aligned}$$

O passo seguinte é prosseguir com a resolução, desta vez unificando $C1$ e $C5$. Nesta etapa, cria-se um $L3 = P(x, y, u, v)$ e $L4 = \neg P(a, b, c, d)$ e tem-se outro conjunto $W = \{L3, \neg L4\}$. Proceda-se o cálculo do unificador mais geral, tal como

realizado anteriormente com $W = \{L1, \neg L2\}$. O cálculo do unificador mais geral fornece que $UMG = \{a/x, b/y, c/u, d/v\}$ (deixa-se esta demonstração para o leitor). O resolvente $C6$ é dado por:

$$\begin{aligned}
C6 &= (C1\sigma - L3\sigma) \cup (C5\sigma - L4\sigma) \\
&= (\{\neg T(a, b, c, d), P(a, b, c, d)\} - \{P(a, b, c, d)\}) \cup \\
&\quad (\{\neg P(a, b, c, d)\} - \{\neg P(a, b, c, d)\}) \\
&= \{\neg T(a, b, c, d)\} \cup \emptyset \\
&= \neg T(a, b, c, d)
\end{aligned}$$

Neste sentido, a nova coleção de cláusulas passa ser:

$$\begin{aligned}
C1 &= \neg T(x, y, u, v) \vee P(x, y, u, v) \\
C2 &= \neg P(x, y, u, v) \vee E(x, y, v, u, v, y) \\
C3 &= T(a, b, c, d) \\
C4 &= \neg E(a, b, d, c, d, b) \\
C5 &= \neg P(a, b, c, d) \\
C6 &= \neg T(a, b, c, d)
\end{aligned}$$

A partir desta coleção, seguindo com a resolução, faz-se a unificação de $C3$ e $C6$. Os literais criados para compor o conjunto W são $L5 = T(a, b, c, d)$ e $L6 = \neg T(a, b, c, d)$, isto é, $W = \{L5, \neg L6\} == \{T(a, b, c, d), T(a, b, c, d)\}$. Observe que não há substituição a ser feita e o $UMG = \epsilon$. O resolvente, então, é dado por:

$$\begin{aligned}
C7 &= (C3\sigma - L5\sigma) \cup (C6\sigma - L6\sigma) \\
&= (\{T(a, b, c, d)\} - \{T(a, b, c, d)\}) \cup (\{\neg T(a, b, c, d)\} - \{\neg T(a, b, c, d)\}) \\
&= \emptyset \cup \emptyset \\
&= \Lambda
\end{aligned}$$

Uma vez que se obteve a cláusula vazia em $C7$, realizando derivações de S , conclui-se que S é insatisfatível. Ora, se S é insatisfatível, é porque $A_1 \wedge A_2 \wedge A_3 \rightarrow E(a, b, d, c, d, b)$ é uma fórmula válida.

4.8 Conclusões e leituras recomendadas

Este capítulo foi uma compilação dos tópicos mais relevantes sobre lógica simbólica e prova automática de teoremas de Chang e Lee [68]. Para um estudo

mais aprofundado sobre lógica proposicional e de primeira ordem recomenda-se Kleene [69] e [70]. Um bom livro sobre lógica, com viés voltado para lingüística é McCawley [71]. Uma abordagem bastante leve e introdutória de lógica é encontrada nos capítulos iniciais de Suppes [72], recomendado para leitores sem familiaridade com o assunto. Os conceitos de natureza filosófica para melhor compreensão da lógica podem ser encontrados em Russell [73, 74]. Uma literatura um pouco mais pesada, mas de igual importância a de Russell são os trabalhos de Popper [75], Lakatos e Musgrave [76], e principalmente Wang [77]. Com relação à técnica de solução de problemas, cerne das máquinas de inferência, o capítulo 2 de Green [78] apresenta um resumo rigoroso. Para estudos mais aprofundados existem as excelentes publicações de Chang e Lee [68] e Loveland [79].

4.9 Exercícios

4.1 Para cada uma das fórmulas a seguir, verifique se elas são tautologias, contradições, ou nenhuma das duas opções.

- (a) $\neg(\neg P) \rightarrow P$
- (b) $P \rightarrow (P \wedge Q)$
- (c) $\neg(P \vee Q) \vee \neg Q$
- (d) $(P \rightarrow Q) \rightarrow (\neg Q \rightarrow \neg P)$
- (e) $P \vee (P \rightarrow Q)$
- (f) $(P \wedge (Q \rightarrow P)) \rightarrow P$
- (g) $P \vee (Q \rightarrow \neg P)$
- (h) $P \rightarrow \neg P$
- (i) $\neg P \rightarrow P$

4.2 Escreva as fórmulas a seguir na forma normal disjuntiva.

- (a) $(\neg P \wedge Q) \rightarrow R$

- (b) $P \rightarrow ((Q \wedge R) \rightarrow S)$
- (c) $\neg(P \vee \neg Q) \wedge (S \rightarrow T)$
- (d) $(P \rightarrow Q) \rightarrow R$
- (e) $\neg(P \wedge Q) \wedge (P \vee Q)$

4.3 Escreva as fórmulas a seguir na forma normal conjuntiva.

- (a) $P \vee (\neg P \wedge Q \wedge R)$
- (b) $\neg(P \rightarrow Q)$
- (c) $(\neg P \wedge Q) \vee (P \wedge \neg Q)$
- (d) $\neg(P \rightarrow Q) \vee (P \vee Q)$
- (e) $(P \rightarrow Q) \rightarrow R$

4.4 Prove que $(\neg Q \rightarrow \neg P)$ é uma consequência lógica de $(P \rightarrow Q)$.

4.5 Mostre que Q é uma consequência lógica de $(P \rightarrow Q)$ e P . Esta afirmação é conhecida como regra de *modus ponens*.

4.6 Mostre que R é uma consequência lógica de $P \rightarrow Q$, $\neg P \rightarrow R$, $Q \rightarrow S$, $\neg S$.

4.7 Formaliza matematicamente as sentenças a seguir:

- (a) Toda alma será salva.
- (b) Tudo que sobe, desce.
- (c) Alguns times são ruins.
- (d) Nenhum time será desclassificado.
- (e) Há pessoas não confiáveis.
- (f) Existem ladrões que não são políticos.
- (g) Não existe elefante azul.

- (h) Idéias boas são raras.
- (i) Ninguém gosta de sinal fechado.
- (j) Cariocas não gostam de sinal fechado.
- (k) Toda unanimidade é burra.
- (l) Todo papa-léguas que diz "beep-beep" é esperto.

4.8 Seja $C(x)$ a representação de "x é um vendedor de carros" e $H(x)$, "x é honesto". Traduza as seguintes fórmulas para o Português:

- (a) $(\exists x)C(x)$
- (b) $(\exists x)H(x)$
- (c) $(\forall x)(C(x) \rightarrow \neg H(x))$
- (d) $(\exists x)(C(x) \wedge H(x))$
- (e) $(\exists x)(H(x) \rightarrow C(x))$

4.9 Seja $P(x)$, $L(x)$, $R(x, y, z)$ e $I(x, y)$ átomos representando "x é um ponto", "x é uma linha", "z passa por x e y" e "x é igual a y", respectivamente. Construa a seguinte fórmula: Para todos dois pontos, existe uma e somente uma linha que passa por ambos os pontos.

4.10 Considere a interpretação $D = a, b$, e os valores assumidos por P : $P(a, a) = 1$ | $P(a, b) = 0$ | $P(b, a) = 0$ | $P(b, b) = 1$. Determine os valores verdade para as seguintes fórmulas:

- (a) $(\forall x)(\exists y)P(x, y)$
- (b) $(\forall x)(\forall y)P(x, y)$
- (c) $(\exists x)(\forall y)P(x, y)$
- (d) $(\exists y)\neg P(a, y)$
- (e) $(\forall x)(\forall y)(P(x, y) \rightarrow P(y, x))$

(f) $(\forall x)P(x, x)$

4.11 Considere a fórmula $A : (\exists x)P(x) \rightarrow (\forall x)P(x)$. Prove que esta fórmula é sempre 1 para um domínio qualquer contendo apenas um único elemento.

4.12 Considere a interpretação $D = 1, 2$, os valores de constantes são $a = 1$ e $b = 2$, com uma função f assumindo os valores $f(1) = 2 \mid f(2) = 1$, e os valores assumidos por P : $P(1, 1) = 1 \mid P(1, 2) = 1 \mid P(2, 1) = 0 \mid P(2, 2) = 0$. Determine os valores verdade para as seguintes fórmulas:

(a) $P(a, f(a)) \wedge P(b, f(b))$

(b) $(\forall x)(\exists y)P(y, x)$

(c) $(\forall x)(\forall y)(P(x, y) \rightarrow P(f(x), f(y)))$

4.13 Transforme as fórmulas a seguir para a forma normal prenex.

a) $(\forall x)(P(x) \rightarrow (\exists y)Q(x, y))$

b) $(\exists x)(\neg((\exists y)P(x, y)) \rightarrow ((\exists z)Q(z) \rightarrow R(x)))$

c) $(\forall x)(\forall y)((\exists z)P(x, y, z) \wedge ((\exists u)Q(x, u) \rightarrow (\exists v)Q(y, v)))$

d) $\neg((\forall x)(A(x) \rightarrow ((\forall y)B(x, y))))$

e) $(\forall x)(\forall y)((\exists z)(A(x, z) \wedge B(y, z)) \rightarrow (\exists u)C(x, y, u))$

f) $(\forall x)A(x) \rightarrow \neg(\exists z)(B(w, z) \rightarrow (\forall y)C(w, y))$

g) $A(x, y) \wedge ((\exists x)B(x) \rightarrow (\forall y)(\forall z)C(x, y, z))$

4.14 Efetue a skolenização das fórmulas a seguir.

a) $(\exists x)(\exists y)(\forall z)(P(x, y) \vee \neg Q(z) \vee R(x))$

b) $(\forall x)[P(x) \rightarrow \{(\forall y)(P(y) \rightarrow P(x)) \wedge \neg(\forall y)(Q(x, y) \rightarrow P(y))\}]$

c) $((\forall x)P(x)) \rightarrow ((\exists x)Q(x) \wedge R(x))$

d) $(\exists x)(P(x, y) \wedge ((\exists y)\neg Q(y) \rightarrow (\exists z)R(z)))$

- e) $(\exists x)(\forall y)(\forall z)(\exists u)(\forall v)(\exists w)(P(x, y, z) \wedge Q(u, v) \wedge \neg R(w))$
- f) $(\forall x)(\exists y)(\exists z)((\neg P(x, y) \wedge Q(x, z)) \vee R(x, y, z))$
- g) $(\forall x)((A(x, y) \rightarrow (\exists y)P(x, y, z)) \rightarrow \neg(\forall z)Q(x, z))$
- h) $(\forall x)((\neg P(x, a) \rightarrow (\exists y)(P(y, g(x))) \wedge (\forall z)(P(z, g(x)) \rightarrow P(y, z))))$
- i) $(\forall x)((P(x) \rightarrow \neg(\forall y)(Q(x, y) \rightarrow (\exists z)P(z))) \wedge (\forall t)(Q(x, y) \rightarrow R(t)))$
- j) $(\forall x)P(x) \wedge ((\forall x)Q(x) \rightarrow (\exists y)R(x, y, z))$

4.15 Determine o universo de Herbrant para o conjunto de cláusulas

$$S = \{P(a), \neg P(x) \vee Q(f(x), a)\}.$$

4.16 Determine o universo de Herbrant para o conjunto de cláusulas

$$S = \{P(a), Q(b, c) \vee P(d)\}.$$

4.17 Determine o universo e a base de Herbrant para o conjunto de cláusulas

$$S = \{P(x, y) \vee Q(f(y)), R(x, y, z)\}.$$

4.18 Determine o universo e a base de Herbrant para o conjunto de cláusulas

$$S = \{P(a, b) \vee Q(f(y)) \vee R(x)\}.$$

4.19 Seja $S = \{P, \neg P \vee Q, \neg Q\}$. Determine a a árvore semântica fechada de S .

4.20 Considere $S = \{P(x), \neg P(x) \vee Q(x, a), \neg Q(y, a)\}$. Determine: (a) o conjunto atômico de S ; (b) a árvore semântica completa de S ; (c) a árvore fechada de S .

4.21 Seja $S = \{P(x), Q(x), \neg Q(x), \neg Q(x) \vee P(x)\}$. Determine a a árvore semântica fechada de S .

4.22 Seja $S = \{P(x), Q(x) \vee R(x), \neg P(x) \vee \neg Q(x), \neg P(x) \vee \neg R(x)\}$. Determine a a árvore semântica fechada de S .

4.23 Resolva o seguinte problema utilizando a prova de Herbrand:

Toda mulher é gentil.

Maria é mulher.

Logo, Maria é gentil.

4.24 Resolva o seguinte problema utilizando a prova de Herbrand:

Todo homem é mortal.

O pai de João é homem.

Logo, o pai de João é mortal.

4.25 Sejam as fórmulas $F_1 = P \rightarrow (\neg Q \vee (R \wedge S))$, $F_2 = P$, $F_3 = \neg S$. Utilize o princípio da resolução para verificar que $G = \neg Q$ é consequência lógica de F_1 , F_2 e F_3 .

4.26 Sejam as fórmulas a seguir. Utilizando o princípio da resolução, mostre a relação de consequência lógica desejada.

a) $F_1 = P \rightarrow S$

$$F_2 = S \rightarrow U$$

$$F_3 = P$$

$$G = U$$

$$F_1, F_2, F_3 \models G$$

b) $F_1 = P \rightarrow (\neg Q \vee (R \wedge S))$

$$F_2 = P$$

$$F_3 = \neg S$$

$$G = \neg Q$$

$$F_1, F_2, F_3 \models G$$

c) $F = (\forall x)(\forall y)(P(x, f(y)) \vee P(y, f(x)))$

$$G = (\exists u)(\exists v)(P(u, f(v)) \vee P(v, f(u)))$$

$$F \models G$$

4.27 Zico é carioca. Zé é carioca. Romário também é carioca. Todo carioca é brasileiro. Construa uma base de conhecimento que modele o cenário aqui descrito. Construa um programa capaz de verificar se determinada interpretação é válida para o predicado `é_brasileiro`, sob a ótica da base de conhecimento. Em seguida crie outro programa que apresente todas as interpretações válidas para `é_brasileiro`.

4.28 João VI é pai de Pedro I e de Miguel de Bragança. Pedro I, por sua vez, é pai de Pedro II e Maria da Glória. Pedro II é pai de Princesa Isabel. Sabe-se que o

pai do pai de um indivíduo também é avô deste indivíduo. Construa um programa capaz de verificar se determinada interpretação é válida para o predicado `é_avô-de`, sob a ótica da base de conhecimento. Em seguida crie outro programa que apresente todas as interpretações válidas para `é_avô-de`.

4.29 O resfriado é uma infecção leve das vias aéreas superiores atingindo principalmente nariz e garganta. Seus sintomas são coriza, febre alta ou baixa e eventualmente espirros. Neste caso deve ser ministrado repouso, líquido e boa alimentação. Se necessário podem ser administrados analgésicos e anti-térmicos. A gripe é uma doença muito contagiosa que ataca as vias respiratórias, nariz, garganta e pulmões. Os pacientes com esta enfermidade apresentam febre alta, dores musculares e nas articulações, dor de cabeça e eventualmente inflamação dos olhos. Não existe remédio para a gripe, porém deve-se prescrever ao paciente repouso, boa alimentação, analgésicos, anti-térmicos e descongestionantes. A pneumonia, por sua vez, é uma infecção aguda que pode atingir os pulmões inteiros ou em partes. Certas pneumonias pioram rapidamente e requerem hospitalização do paciente. Nesta situação os pacientes apresentam tosse, dor no peito, febre alta, calafrios e sudorese. Em muitos casos a sudorese e os calafrios podem não estar presentes no quadro patológico do paciente. O tratamento exige anti-bióticos e oxigênio. Um paciente chegou no hospital com febre alta, dor de cabeça, tosse e dor no peito. Após o atendimento foi ministrado antibiótico. Esta combinação de medicamentos irá tratar sua enfermidade (faça um programa que forneça esta decisão)?

4.30 Alonzo, Kurt, Rudolf e Willard são artistas criativos de grande talento, um é dançarino, um é pintor, um é cantor e um é escritor, não necessariamente nesta ordem. Alonzo e Rudolf estavam na audiência na noite em que o cantor fez seu debut no palco. O escritor publicou a biografia de Willard e pretende escrever a biografia de Alonzo. Rudolf não conhece Alonzo. Rudolf e o escritor ambos tiveram seus retratos pintados ao vivo pelo pintor. Quais são as profissões de Kurt, Willard, Rudolf e Alonzo? (Implemente um programa para fornecer estas respostas).

4.31 Circula na Internet um teste de QI supostamente atribuído a Albert Einstein sobre o qual, supostamente, afirmou que apenas 2% das pessoas são capazes de resolvê-lo. Ainda que a veracidade destas informações possa ser questionada, o teste

por si só é bastante interessante, principalmente se este suposto teste de QI possa ser implementado em uma máquina, completamente desprovida de inteligência. Deste modo, implemente um programa de computador capaz de resolver o problema a seguir:

Há cinco casas de diferentes cores. Em cada casa mora uma pessoa de uma diferente nacionalidade. Esses cinco proprietários bebem diferentes bebidas, fumam diferentes tipos de cigarros e têm um certo animal de estimação. Nenhum deles têm o mesmo animal, fumam o mesmo cigarro ou bebem a mesma bebida (isto é, são conjuntos excludentes).

1. O Inglês vive na casa Vermelha.
2. O Sueco tem Cachorros como animais de estimação.
3. O Dinamarquês bebe Chá.
4. A casa Verde fica do lado esquerdo da casa Branca.
5. O homem que vive na casa Verde bebe Café.
6. O homem que fuma Pall Mall cria Pássaros.
7. O homem que vive na casa Amarela fuma Dunhill.
8. O homem que vive na casa do meio bebe Leite.
9. O Norueguês vive na primeira casa.
10. O homem que fuma Blends vive ao lado do que tem Gatos.
11. O homem que cria Cavalos vive ao lado do que fuma Dunhill.
12. O homem que fuma BlueMaster bebe Cerveja.
13. O Alemão fuma Prince.
14. O Norueguês vive ao lado da casa Azul.
15. O homem que fuma Blends é vizinho do que bebe Água.

Pergunta-se, quem tem um peixe como animal de estimação?

4.32 Verificar se $W = \{Q(f(a), g(x)), Q(y, y)\}$ é unificável.

4.33 Verificar se $W = \{P(x), P(f(x))\}$ é unificável.

4.34 Verificar se $W = \{P(a, f(x)), P(y, z)\}$ é unificável.

4.35 Verificar se $W = \{P(a, x, f(g(y))), P(z, h(z, w), f(w))\}$ é unificável.

Capítulo 5

Sistemas Formais

A Informática é a ciência do artificial por excelência e, mais ainda, é a ciência do formal: não existe na história da humanidade nenhum profissional que tenha se comportado de modo mais formal. Os computadores seguem fielmente regras, e não admitem exceções; é preciso especificar, codificar, digitar, depurar e, mesmo assim, o programa pode não funcionar simplesmente porque foi trocado um ‘0’ (zero) por ‘O’ , ou ‘a’ por ‘A’. Depois de semanas buscando erros lógicos, alguém de fora, por trás dos ombros do programador diz: **“Claro que não funciona: você usou ‘1’ e não ‘l’.”**, e então descobre-se que o engano era de natureza apenas formal. O grande objetivo dos sistemas amigáveis é a eliminação dos erros formais.

Explicar ou definir formal é uma tarefa difícil e, se for buscado o seu significado em um dicionário, encontra-se que:

- (0) Formal - do latim formale - relativo a forma.
- (1) Forma - do latim forma - Configuração exterior dos corpos; disposição das partes de um corpo, aparência; feitio de um objeto; modelo; norma.
- (2) Forma - do latim forma - molde dentro do qual ou sobre o qual se forma qualquer coisa que toma o feitio desse molde.

Depois desta consulta, descobre-se que “formal” é definido através de “forma” e o problema é saber qual a escolha a ser feita: se (1) ou (2), notando-se que há di-

ferença fonética, ainda que não gráfica entre elas: em (1) a letra “o” é pronunciada aberta e em (2) fechada.

Então pode-se usar (1) e (2) para melhorar o entendimento de formal. Pois o próprio conceito definido em (1) deve se encaixar com a “forma” (com “o” fechado) de um grupo social.

A matemática é considerada como a mais formal das ciências sendo a linguagem formal utilizada pelas outras, pois todos os resultados são baseados em regras e apresentados por fórmulas. No entanto os formalistas são apenas um grupo dentre os matemáticos, tendo existido sérias controvérsias com respeito à validade do enfoque formal. Na realidade, a maioria dos matemáticos desenvolve seus resultados dentro de um espírito informal e intuitivo, mais geométrico que algébrico; e quando algébrico, se analisado de forma mais rigorosa, pouco formal. De qualquer modo, a mais formal das correntes em matemática tem feito e faz concessões ao informal. No que se segue será apresentado que o conceito de formal a ser adotado é extremamente exigente; pode-se dizer que o formal de que se necessita é também mecânico, completamente dissociado de qualquer intuição ou fatores de natureza cognitiva.

5.1 Sistemas Formais

Nesta seção será definido o que se entende por um sistema formal. Para isto é preciso esclarecer o que significa alfabeto e palavra, pois o conceito de formal - e, particularmente sistemas formais - depende da definição e conhecimentos básicos sobre representação gráfica de símbolos e a fixação de critérios particulares de sua aceitação ou definição (recomenda-se ao leitor uma releitura da Seção 2.7). Dá-se o nome de letra a todo sinal gráfico satisfazendo os seguintes critérios:

1. As letras devem possuir uma estrutura espacial que facilite sua reprodução e reconhecimento. Por exemplo: $\square, \triangle, |, *$. Como contra-exemplo o leitor pode imaginar figuras complicadas como rubricas pessoais, tais como $b\ddot{q}$.
2. As letras devem possuir uma estrutura que impossibilite decomposições horizontais. Assim, “ $||$ ” não seria uma escolha apropriada, pois é composta

horizontalmente de dois sinais iguais (“|” e “|”); no entanto, “-” e “==” seriam escolhas adequadas - apesar de poderem ser decompostas verticalmente.

3. Para a construção de alguns sistemas formais, existe a necessidade de um suprimento infinito de letras, assim deve-se exigir que elas possam ser produzidas de modo uniforme.

Os elementos de $\Sigma_1 = \{*, |\}$ e $\Sigma_2 = \{\square, \triangle, \odot\}$ satisfazem os critérios (a), (b) e (c) recém descritos. *Alfabetos* são conjuntos recursivos de letras.

*Expressões, palavras*¹ ou *cadeias* são seqüências de letras justapostas horizontalmente, tendo seus limites claramente identificados por interespaço separador com mesma função que o espaço em branco na escrita convencional. Assim, $\square\triangle$ e $\square\odot\square\triangle$ são expressões no alfabeto Σ_2 .

Alguns autores definem uma cadeia u em um alfabeto Σ como uma função $u : \{1 \cdots n\} \rightarrow \Sigma$, onde n é o comprimento de u . Assim, uma cadeia u pode ser escrita como $a_1a_2 \cdots a_n$, onde a_i é a i -ésima letra. A cadeia de comprimento 0 é chamada **cadeia nula** e denotada por Λ .

A justaposição de duas cadeias é chamada concatenação. Sejam $u : \{1 \cdots m\} \rightarrow \Sigma_1$ e $v : \{1 \cdots n\} \rightarrow \Sigma_2$, então $u \frown v$ (ou simplesmente uv) é definida como:

$$u \frown v : \{1 \cdots m + n\} \rightarrow \Sigma_1 \cup \Sigma_2$$

$$u \frown v = \begin{cases} u(i) & \text{se } 1 \leq i \leq m \\ v(i - m) & \text{se } m + 1 \leq i \leq m + n \end{cases}$$

O produto de dois alfabetos é dado pela combinação de pares feita entre seus elementos. Por exemplo, se $\Sigma_1 = \{*, |\}$ e $\Sigma_2 = \{\square, \triangle, \odot\}$, tem-se que $\Sigma_1 \times \Sigma_2 = \{*\square, *\triangle, *\odot, |\square, |\triangle, |\odot\}$, e ainda que $\Sigma_1 \times \Sigma_2 \neq \Sigma_2 \times \Sigma_1$.

¹Alguns autores mais rigorosos consideram que palavras são expressões que respeitam determinados critérios explícitos para sua formação. Desta forma pode-se explicitar que o critério de formação para as palavras em Σ_1 seja: *as únicas palavras são as expressões onde não apareçam mais que duas ocorrências sucessivas de ‘*’ e não menos que duas de ‘|’ em seqüência*. Utilizando-se este critério, conclui-se (informalmente) que, $**||*$ é uma palavra, e que $**|*$ não é, pois possui menos que duas ocorrências sucessivas de “|”.

A exponenciação de um alfabeto é dada por:

$$\begin{aligned}\Sigma^0 &= \{\Lambda\} \\ \Sigma^1 &= \Sigma \\ \Sigma^n &= \Sigma^{n-1} \times \Sigma\end{aligned}$$

observe que Σ^n é o conjunto de todas as cadeias de símbolos de Σ com comprimento n .

Por fim, se Σ for um alfabeto, denota-se por Σ^* o conjunto de todas as cadeias de Σ , incluindo a cadeia nula, isto é, $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots \cup \Sigma^n \cup \dots$. Segue-se que uma linguagem em Σ é qualquer subconjunto de Σ^* . Desta forma, se Σ é um alfabeto, então qualquer conjunto de expressões em Σ é uma *linguagem* em Σ . Assim, por exemplo, $\mathcal{L} = \{\square, \square\Delta, \square\odot, \Delta\}$ é uma linguagem em Σ .

Definição 5.1 Um sistema formal \mathcal{F} é uma quádrupla $\langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$, onde:

Σ – é um alfabeto.

\mathcal{L} – é um conjunto recursivo em Σ , chamado de linguagem do sistema formal.

\mathcal{A} – é um subconjunto recursivo de \mathcal{L} , chamado de *axiomas*

\mathcal{R} – é um conjunto recursivo de relações em \mathcal{L}

Exemplo 5.1 Seja um sistema formal, onde o alfabeto, as palavras, os axiomas e as relações estejam definidas a seguir:

$$\begin{aligned}\Sigma &= \{ |, * \}, \mathcal{L} = \{ \Sigma^* \}, \mathcal{A} = \{ |, * \}, \mathcal{R} = \{ r_1, r_2 \}, \text{ onde :} \\ r_1 &= \{ \langle x |, x* \rangle \mid x \in \Sigma^* \} \\ r_2 &= \{ \{ \langle x | *, x* \rangle \mid x \in \Sigma^* \} \cup \\ &\quad \{ \langle x | **, x* \rangle \mid x \in \Sigma^* \} \cup \\ &\quad \{ \langle x*, x \rangle \mid x \in \Sigma^* \} \}\end{aligned}$$

Definição 5.2 Seja $\mathcal{F} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ um sistema formal, $\Gamma \subseteq \mathcal{L}$. Uma dedução de α a partir de Γ em \mathcal{F} é uma seqüência $\alpha_1, \alpha_2, \dots, \alpha_n$ de palavras de \mathcal{L} , tal que:

1. α_n é α ; e

2. Para todo $j, 1 \leq j < n, \alpha_j \in \Gamma \cup \mathcal{A}$, ou existem $\alpha_{j_1}, \dots, \alpha_{j_k}, j_i \in \{1, \dots, j-1\}, 1 \leq i \leq k$ tais que $\langle \alpha_{j_1}, \dots, \alpha_{j_k}, \alpha_j \rangle \in r$ com $r \in \mathcal{R}$.

Se existir uma dedução de α a partir de Γ diz-se que α é dedutível a partir de Γ em \mathcal{F} . Isto é denotado por $\Gamma \vdash_{\mathcal{F}} \alpha$.

Exemplo 5.2 No sistema formal do exemplo 5.1 uma dedução de $*|$ é:

$$\begin{array}{l} | \quad (\in \mathcal{A}) \\ * \quad (\langle |, * \rangle \in r_1) \\ || \quad (\langle *, || \rangle \in r_2) \\ |* \quad (\langle ||, |* \rangle \in r_1) \\ *| \quad (\langle |*, *| \rangle \in r_2) \end{array}$$

portanto a seqüência $*, ||, |*, *|$ é uma dedução de $*|$ onde $\Gamma = \emptyset$, assim $\emptyset \vdash *|$.

Existem várias considerações que devem ser feitas com respeito às componentes de um sistema formal.

Σ — Os alfabetos dos sistemas formais podem conter vários tipos de letras, assim em geral devem ser especificados tais tipos de modo não ambíguo. Os tipos de letras correspondem a modos de construir as palavras da linguagem \mathcal{L} . Assim na definição de Σ pode-se ter:

$$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \dots \cup \Sigma_k$$

Por exemplo, nos dialetos formalizáveis oriundos de simplificações das linguagens naturais tem-se letras para representar as diversas categorias gramaticais. É frequente se incluir no alfabeto letras que são auxiliares (variáveis sintáticas) para representar subclasses da linguagem \mathcal{L} .

\mathcal{L} — A definição da linguagem pode já necessitar de um outro sistema formal, ou notações informais para sua especificação detalhada. Note que a exigência de ser a linguagem um conjunto recursivo não é suficiente quando o sistema formal visa aplicações. As operações que são utilizadas para os critérios de construção dos componentes da linguagem devem ser de baixa complexidade.

- \mathcal{A} – Existem várias maneiras de se especificar os axiomas de um sistema formal. Se o conjunto for finito basta uma simples enumeração, no caso de um conjunto infinito pode-se utilizar um outro sistema formal para sua especificação. É frequente na literatura a utilização de *axiomas esquemas* que são bastante difíceis de serem efetivos nas aplicações computacionais.
- \mathcal{R} – A definição das regras de inferência é a parte mais complicada de um sistema formal. As regras de inferência devem ser relações recursivas definidas na linguagem \mathcal{L} podendo, portanto, ser bastante complexas para definir e para serem efetivamente utilizadas. Nos exemplos que serão apresentados se discutirá a complexidade de tais regras.

Estas observações devem ser levadas em conta quando se deseja ter um sistema formal que possa ser efetivamente utilizado. O desenvolvimento de critérios para a escolha de um sistema formal entre alternativas de formalização é uma necessidade pouco estudada. Deve-se acrescentar que os sistemas formais foram inventados com o objetivo de se *mecanizar* a solução de problemas, principalmente na matemática tradicional, e que o *agente* executante das deduções era inicialmente imaginado como um ser humano treinado como matemático. Hoje deve-se dirigir os esforços na construção de sistemas formais cujos *agentes* são de outra natureza, muitas vezes uma máquina ou sistema de programação.

Na especificação de regras de inferência escreve-se:

$$\frac{\alpha_1, \alpha_2, \dots, \alpha_l}{\beta}, \text{ ou } \alpha_1, \alpha_2, \dots, \alpha_l \rightarrow \beta$$

significando que $\langle \alpha_1, \alpha_2, \dots, \alpha_l, \beta \rangle \in \mathcal{R}$.

No caso em que $\Gamma \subseteq \mathcal{L} = \emptyset$, diz-se que α é um teorema em \mathcal{F} . O conjunto de teoremas de um sistema formal \mathcal{F} é denotado por $\mathcal{T}_{\mathcal{F}}$, ou simplesmente por \mathcal{T} , quando é claro a que sistema formal pertencem os teoremas. O conjunto de teoremas de um sistema formal inclui os axiomas, ou seja, todo axioma é um teorema. Em geral, o conjunto de teoremas é maior que o conjunto de axiomas, mas é claro que as regras de um sistema formal podem não ser aplicáveis, e neste caso o conjunto de teoremas é composto apenas dos axiomas. O conjunto de teoremas pode ser igual à

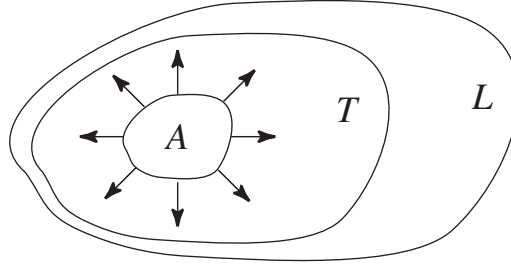


Figura 5.1: Derivação de Teoremas.

linguagem, e neste caso dizemos que o sistema formal é inconsistente². A figura 5.1 mostra a relação entre axiomas, teoremas e linguagem.

Existem três tipos básicos de sistemas formais: geradores, reconhecedores e transdutores. Os geradores são sistemas formais cujo objetivo é produzir cadeias, somente a partir dos axiomas. Os reconhecedores admitem outras cadeias como entradas e verificam se tais cadeias são adequadas, sem produzir cadeias de saída. Por fim, os transdutores transformam cadeias de entrada em cadeias de saída.

A seguir serão apresentadas algumas variações dos tipos básicos, que serão importantes para o entendimento de questões a serem discutidas nos próximos capítulos.

5.2 Sistemas de Produções de Post

Definição 5.3 Um sistema de produção de Post (*SPP*) \mathcal{S} é um sistema formal $\langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$, onde:

Σ É um alfabeto consistindo de dois subconjuntos disjuntos N e T , chamados de alfabeto não terminal e terminal, respectivamente, onde $N = \{\square | i > 0\}$

\mathcal{L} É o conjunto Σ^*

\mathcal{A} É um sub-conjunto de Σ^* , e é dito ser o conjunto de palavras de partida.

²Existem várias concepções da noção de inconsistência. Esta, em particular, é chamada de inconsistência absoluta [80]

\mathcal{R} É um conjunto de relações binárias em \mathcal{L} , que são chamadas de regras de produção.

Cada regra é da forma:

$$x_0 \boxed{i_1} x_1 \boxed{i_2} \cdots x_{n-1} \boxed{i_n} x_n \rightarrow y_0 \boxed{j_1} y_1 \boxed{j_2} \cdots y_{k-1} \boxed{j_k} y_k$$

,onde $i_1, i_2, \dots, i_n \in \mathbb{N}$, e $j_1, j_2, \dots, j_k \in \{i_1, i_2, \dots, i_n\}$

e $x_0, x_1, \dots, x_n, y_0, \dots, y_k \in (\Sigma - N)^*$

Definição 5.4 Seja $\mathcal{S} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ um *SPP* e $\alpha \in \mathcal{L}$ uma palavra.

(a) Diz-se que uma regra:

$$x_0 \boxed{i_1} x_1 \boxed{i_2} \cdots x_{n-1} \boxed{i_n} x_n \rightarrow y_0 \boxed{j_1} y_1 \boxed{j_2} \cdots y_{k-1} \boxed{j_k} y_k$$

é aplicável a α se e somente se:

$$\alpha = x_0 z_{i_1} x_1 z_{i_2} \cdots x_{n-1} z_{i_n} x_n$$

,onde $z_{i_1}, z_{i_2}, \dots, z_{i_n} \in \Sigma^*$ e se $i_t = i_s$ então $z_{i_t} = z_{i_s}$

(b) Se uma regra:

$$x_0 \boxed{i_1} x_1 \boxed{i_2} \cdots x_{n-1} \boxed{i_n} x_n \rightarrow y_0 \boxed{j_1} y_1 \boxed{j_2} \cdots y_{k-1} \boxed{j_k} y_k$$

é aplicável a uma palavra $\alpha = x_0 z_{i_1} x_1 z_{i_2} \cdots x_{n-1} z_{i_n} x_n$ então o resultado da aplicação é a palavra

$$y_0 z_{j_1} y_1 z_{j_2} y_2 \cdots y_{k-1} z_{j_k} y_k$$

(c) Diz-se $\alpha \Rightarrow_{\mathcal{S}} \beta$ se β foi obtida a partir da palavra α pela aplicação de uma regra $r \in \mathcal{R}$.

(d) Diz-se que β é derivável em \mathcal{S} a partir de α , o que denota-se por $\alpha \Rightarrow_{\mathcal{S}}^* \beta$, se e somente se $\beta = \alpha$ ou existem $\beta_1, \beta_2, \dots, \beta_l$ tais que $\beta_l = \beta$ e para todo i , $1 \leq i < l$ $\beta_i \Rightarrow_{\mathcal{S}} \beta_{i+1}$

(e) A linguagem gerada por \mathcal{S} é definida por:

$$\mathcal{L}(\mathcal{S}) = \{x \in (\Sigma - N)^* \mid \alpha \Rightarrow_{\mathcal{S}}^* x, \alpha \in \mathcal{A}\}$$

Intuitivamente as caixas $\boxed{i_m}$ capturam as palavras z_{i_m} quando a regra é aplicada e reproduzem as palavras capturadas por estas caixas na cadeia do lado direito da regra.

Exemplo 5.3 A regra $aa\boxed{1}b\boxed{2}a \rightarrow a\boxed{2}ba\boxed{1}$ aplicada à palavra $aaabbaa$ coloca ab na caixa numerada com 1 e a na caixa numerada por 2 e gera a palavra $aabaab$. Note que outra solução seria colocar a na caixa 1 e ba na caixa 2 e o resultado neste caso seria $ababaa$.

Exemplo 5.4 Seja S o SPP no qual

$$\begin{aligned}\Sigma &= \{\square, \diamond, \underline{suc}, S, , \} \\ N &= \{S, \underline{suc}, , \} \\ \mathcal{A} &= \{\underline{suc}x | x \in (\Sigma - N)^*\} \\ \mathcal{R} &= \left\{ \begin{array}{l} \underline{suc}\boxed{1} \rightarrow S\boxed{1}, \\ S, \boxed{1} \rightarrow \square\boxed{1} \\ S\boxed{1}\square, \boxed{2} \rightarrow \boxed{1} \diamond \boxed{2} \\ S\boxed{1} \diamond, \boxed{2} \rightarrow S\boxed{1}, \square\boxed{2} \end{array} \right.\end{aligned}$$

A seguir serão exibidas algumas derivações em S :

Axioma	Geração	Axioma	Geração
\underline{suc}	$\underline{suc} \Rightarrow S,$ $S, \Rightarrow \square$	$\underline{suc}\diamond$	$\underline{suc}\diamond \Rightarrow S\diamond,$ $S\diamond, \Rightarrow S, \square$ $S, \square \Rightarrow \square\square$
$\underline{suc}\square$	$\underline{suc}\square \Rightarrow S\square,$ $S\square, \Rightarrow \diamond$	$\underline{suc}\square\square$	$\underline{suc}\square\square \Rightarrow S\square\square,$ $S\square\square, \Rightarrow \square\diamond$

Pode-se interpretar este SPP como gerando a partir de um axioma \underline{suc} , onde n é a representação diádica de um natural no alfabeto $\{\square, \diamond\}$, a representação $n + 1$.

As regras de inferência de um *SPP* podem ser bastante complexas. As operações básicas são de concatenação, casamento de padrões e substituição. Tais operações não fazem parte do sistema formal e são de difícil execução, mesmo quando o agente é um ser humano bem treinado. O leitor familiarizado com os processos de análise sintática para a construção de compiladores pode apreciar tais dificuldades. A complexidade da linguagem \mathcal{L} é um fator preponderante tanto para o uso do casamento de padrões como para a construção das regras de inferência. O leitor interessado pode consultar Book e Otto [81] que trata dos chamados sistemas de reescrita.

5.3 Linguagens Formais e Gramáticas

O aprendizado de uma língua qualquer envolve o estudo da estrutura das sentenças. A grosso modo esta estrutura é chamada de gramática para a língua, e sua apresentação usual é como um conjunto de critérios chamados de regras gramaticais que definem a correteza das sentenças. As regras gramáticas são sistemas formais geradores de linguagens, portanto deve-se esperar que uma gramática construa a língua.

Em geral, uma linguagem natural tende a ser muito extensa e complexa. Duas das atividades dos linguistas são definir com precisão as sentenças válidas de linguagens e encontrar formas estruturadas de representá-las. Entretanto, há uma dificuldade para definir completamente estas linguagens. Considere a seguinte sentença do português: *O menino louco pintava o lindo quadro*.

Pode-se dizer que a sentença é constituída de sujeito “O menino louco- e predicado - “pintava o lindo quadro”. Estes elementos podem ser mais analisados. O predicado é constituído de verbo “pintava” e objeto “lindo quadro”. A análise desta sentença é apresentada na Figura 5.2 e descreve uma representação conhecida como árvore de derivação sintática. Quando uma sentença produz uma árvore de derivação válida, diz-se que esta sentença é válida. Contudo, para uma árvore de derivação ser válida é preciso que ela seja especificada, uma tarefa pouco trivial quando se trata de uma linguagem natural. Uma tentativa incompleta de formalização das regras

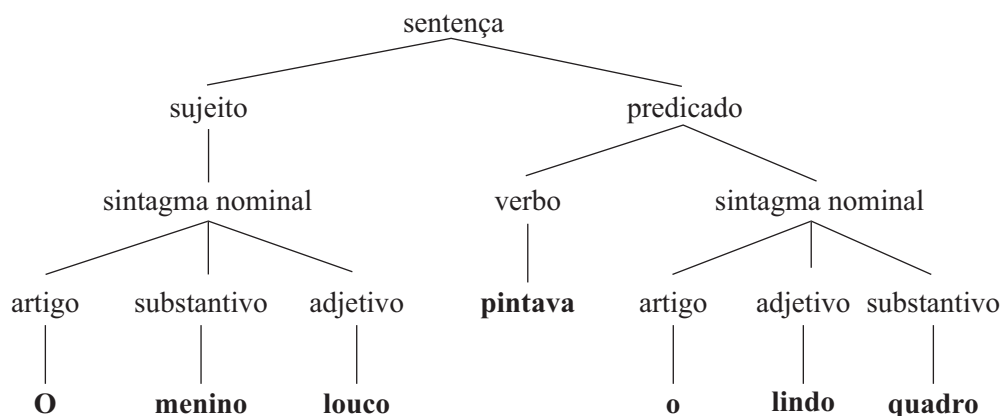


Figura 5.2: Árvore de Derivação Sintática de uma sentença em Português.

do Português pode ser descrita como se segue:

$\langle \textit{sentenca} \rangle \rightarrow \langle \textit{sujeito} \rangle \langle \textit{predicado} \rangle$
 $\langle \textit{sujeito} \rangle \rightarrow \langle \textit{sintagma nominal} \rangle$
 $\langle \textit{predicado} \rangle \rightarrow \langle \textit{verbo} \rangle \langle \textit{sintagma nominal} \rangle$
 $\langle \textit{sintagma nominal} \rangle \rightarrow \langle \textit{artigo} \rangle \langle \textit{substantivo} \rangle \langle \textit{adjetivo} \rangle$
 $\langle \textit{sintagma nominal} \rangle \rightarrow \langle \textit{artigo} \rangle \langle \textit{adjetivo} \rangle \langle \textit{substantivo} \rangle$
 $\langle \textit{verbo} \rangle \rightarrow \textit{pintava}$
 $\langle \textit{artigo} \rangle \rightarrow \textit{o}$
 $\langle \textit{substantivo} \rangle \rightarrow \textit{menino}$
 $\langle \textit{substantivo} \rangle \rightarrow \textit{quadro}$
 $\langle \textit{adjetivo} \rangle \rightarrow \textit{louco}$
 $\langle \textit{adjetivo} \rangle \rightarrow \textit{lindo}$

Estas regras indicam que os elementos à esquerda da seta podem ser transformados nos elementos à direita da seta. Os símbolos $\langle s \rangle$ indicam categorias gramaticais genéricas, chamados *símbolos não-terminais*, e os demais símbolos são palavras do Português, chamados *símbolos terminais*.

As regras podem estabelecer que a sentença dada é gramaticalmente correta, bastando constatar que esta é gerável a partir daquelas. Além disso, estas regras podem ser utilizadas para construir outras sentenças corretas do Português como “O menino lindo pintava o quadro louco”. Além disso, também podem ser produzidas

outras sentenças sem valor semântico como “O quadro louco pintava o lindo menino”.

Até o presente momento, não existe um conjunto de regras gramaticais e de palavras que possibilitem a formalização de uma linguagem natural, dada a explosão combinatorial que palavras e regras de uma linguagem natural produz. Entretanto, o mesmo conceito funciona para linguagens de programação pois estas podem ser definidas por uma sintaxe rígida e uma semântica bem determinada, possibilitando o processamento computacional.

Para especificar a sintaxe de uma linguagem de programação de modo sistemático é preciso uma gramática que possibilite:

- *Geração*: um método sistemático de especificação para a construção de programas corretos;
- *Reconhecimento*: um método de análise de um programa a fim de verificar se ele está sintaticamente correto.

A definição seguinte torna precisa a noção de gramática como um sistema formal.

Definição 5.5 Uma gramática \mathcal{G} (Gramática de Chomsky³) é um sistema formal $\langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$, onde:

Σ É um alfabeto consistindo de dois subconjuntos disjuntos N e T , chamados de alfabeto não terminal e terminal, respectivamente.

\mathcal{L} É o conjunto Σ^*

\mathcal{A} É o conjunto unitário $\{S\}$, sendo $S \in N$. S é dito ser o símbolo de partida e possui o nome da categorial sintática principal.

\mathcal{R} É um conjunto de relações binárias em \mathcal{L} , chamadas de regras de produção.

³Avram Noam Chomsky (1928-) é responsável pela gramática gerativa transformacional, abordagem que revolucionou os estudos no domínio da linguística teórica. É também o autor de trabalhos fundamentais sobre as propriedades matemáticas das linguagens formais, sendo o seu nome associado à chamada Hierarquia de Chomsky.

Por convenção, os símbolos não terminais são representados por letras maiúsculas e os símbolos terminais, por letras minúsculas. Seja a gramática $G1$ com $N = \{S, A\}$, $T = \{a, b\}$, $\mathcal{A} = \{S\}$ e $\mathcal{R} = \{S \rightarrow aA, A \rightarrow b\}$. Neste caso, $S \rightarrow aA \rightarrow ab$ que é a única cadeia gerada por $G1$.

Definição 5.6 Seja $\mathcal{G} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ uma gramática, $x, y \in \mathcal{L}$. Diz-se que:

(a) y é diretamente derivável em \mathcal{G} a partir de x , o que é denotado por $x \Rightarrow_{\mathcal{G}} y$ se e somente se

$$x = x_1 \alpha x_2 \text{ e } y = x_1 \beta x_2 \text{ e } \alpha \rightarrow \beta \in \mathcal{R}$$

(b) y é derivável em \mathcal{G} a partir de x , o que é denotado por $x \Rightarrow_{\mathcal{G}}^* y$ se e somente se $y = x$ ou existem $\beta_1, \beta_2, \dots, \beta_l$ tais que $\beta_l = \beta$ e para todo i , $1 \leq i < l$ $\beta_i \Rightarrow_{\mathcal{G}} \beta_{i+1}$

Uma gramática pode ser considerada como um SPP , cujas regras de produção da forma $\alpha \rightarrow \beta$ são representadas como $\boxed{1}\alpha\boxed{2} \rightarrow \boxed{1}\beta\boxed{2}$. Seja $G2$ com $N = \{A, B\}$, $T = \{a, b, c\}$, $\mathcal{A} = \{A\}$ e $\mathcal{R} = \{A \rightarrow aB, B \rightarrow bB, B \rightarrow c\}$. A sentença $x = abbbc$ é uma produção de $G2$ porque ela é derivável da seguinte forma: $A \xRightarrow{1} aB \xRightarrow{2} abB \xRightarrow{2} abbB \xRightarrow{2} abbbB \xRightarrow{3} abbbc$.

Definição 5.7 Seja $\mathcal{G} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ uma gramática. A linguagem formal gerada por \mathcal{G} , denotada por $L(\mathcal{G})$ é o conjunto de teoremas de \mathcal{G} contendo apenas símbolos terminais de \mathcal{G} . Assim $L(\mathcal{G}) = \{x \in T^* | S \Rightarrow_{\mathcal{G}}^* x\}$.

Exemplo 5.5 Seja a gramática $G2$ definida anteriormente. Pode-se verificar que $L(G2) = \{x \in T^* | x = ab^n c, n \geq 0\}$.

Exemplo 5.6 Seja a gramática $G3$ com $N = \{S\}$, $T = \{a, b, c\}$, $\mathcal{A} = \{S\}$ e $\mathcal{R} = \{S \rightarrow aSa, S \rightarrow bSb, S \rightarrow c\}$. Então é fácil verificar que:

$$L(\mathcal{G}) = \{c, aca, aacaa, bcb, bacab, bbcbb, \dots\}$$

Exemplo 5.7 Seja a gramática $G4$ com $N = \{E, T, F\}$, $T = \{+, *, (,), x\}$, $\mathcal{A} = \{E\}$ e $\mathcal{R} = \{E \rightarrow E+T, E \rightarrow T, T \rightarrow T*F, T \rightarrow F, F \rightarrow (E), F \rightarrow x\}$. Um exemplo de cadeia derivável da gramática é a expressão conforme descrito na derivação $E \xRightarrow{2}$

$$T \xrightarrow{3} T + F \xrightarrow{5} T * (E) \xrightarrow{5} F * (E) \xrightarrow{1} F * (E + T) \xrightarrow{2} F * (T + T) \xrightarrow{4} F * (F + F) \xrightarrow{6} x * (x + x).$$

Quando se trata de linguagem de programação, um impedimento importante é a impossibilidade de uma cadeia $w \in L(G)$ possuir mais de um entendimento. Assim, linguagens de programação não devem ser ambíguas, ou ao menos, devem permitir que eventuais ambiguidades possam ser facilmente evitadas. O mais notório caso de ambiguidade é conhecido como o *else pendente* (dangling else).

Seja um gramática \mathcal{G} com o seguinte conjunto de regras:

$$BLK \rightarrow \text{if } a \text{ then } BLK \text{ else } BLK$$

$$BLK \rightarrow \text{if } a \text{ then } BLK$$

$$BLK \rightarrow b$$

A gramática \mathcal{G} é ambígua uma vez que a cadeia “if a then if a then b else b” pode ser interpretada como “if a then (if a then b else b)” ou como “if a then (if a then b) else b”. Alguns compiladores não aceitam este tipo de construção, e outros, quando aceitam, escolhem a interpretação que associa o *else* ao *if* mais próximo, isto é, “if a then (if a then b else b)”.

Em alguns casos, é possível perceber esta ambiguidade como algo inerente à gramática e nestas situações, pode-se reescrever a mesma a fim de eliminar estas ambiguidades. Seja a gramática $G5$ com $N = \{E, T, F, A\}$, $T = \{+, *, 0, 1, 2, \dots, 9\}$, $\mathcal{A} = \{E\}$ e \mathcal{R} dado por:

$$E \rightarrow E + T$$

$$E \rightarrow T$$

$$T \rightarrow T * F$$

$$T \rightarrow F$$

$$F \rightarrow 0 \dots 9$$

que é utilizada para gerar expressões aritméticas com $+$ e $*$ para números inteiros. Este tipo de construção é bastante empregada em linguagens de programação. Além disto, seja também a gramática $G6$ com $N = \{E, A\}$, $T = \{+, *, 0, 1, 2, \dots, 9\}$, $\mathcal{A} = \{E\}$ e \mathcal{R} dado por:

$$E \rightarrow E + E$$

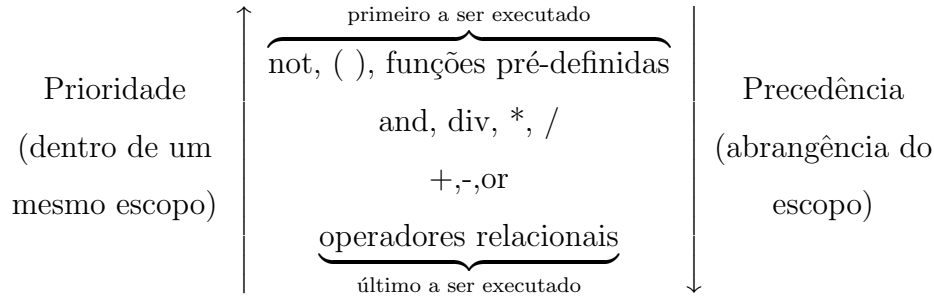
$$E \rightarrow E * E$$

$$E \rightarrow F$$

$$F \rightarrow 0 \dots 9$$

que não só é equivalente a $G5$, como também possui uma quantidade menor de produções e símbolos. Entretanto, $G6$ é ambígua enquanto que $G5$, não. Por exemplo, a cadeia $2 + 4 * 6$ possui dois entendimentos na gramática $G6$, “ $2+(4*6)$ ” e “ $(2+4)*6$ ”. Já na gramática $G5$ só há o entendimento “ $2+(4*6)$ ”.

Para dirimir a ambiguidade é preciso compreender as propriedades de precedência e de associatividade. A precedência permite decidir qual o correto entendimento das expressões, isto é, quanto maior a precedência de um operador, maior será o escopo contemplado por este mesmo operador. Já a associatividade permite decidir o correto entendimento de operações contendo operadores de mesma precedência. Em geral, a relação de prioridade e precedência entre operadores segue o esquema a seguir.



Por exemplo, na gramática $G5$ o operador $+$ tem maior precedência que $*$ porque está definido mais próximo das regras de produção que se iniciam por axiomas. Além disso, $*$ tem maior prioridade que $+$ pois é resolvido primeiro que $+$. O exemplo a seguir demonstra a precedência de operadores da gramática $G5$.

Exemplo 5.8 Precedência de operadores da gramática $G5$:

$$2 + 4 * 6 = +(2, *(4, 6))$$

$$2 * 4 + 6 * 8 = +(*(2, 4), *(6, 8))$$

$$2 + 4 + 6 = +(+(2, 4), 6)$$

Assim, o truque é aumentar a precedência dos operadores, colocando suas regras cada vez mais distantes do axioma \mathcal{A} . No exemplo a seguir, a primeira

gramática é inicialmente ambígua. Na segunda gramática, a precedência é resolvida pela associatividade que é colocada em um nível mais baixo da árvore sintática. Contudo, este procedimento delega a quem monta a expressão a responsabilidade de determinar a precedência entre as operações utilizando a associatividade pois os operadores de $+$ e $*$ têm a mesma precedência. Por fim, a última gramática resolve este problema colocando o operador $*$ com maior precedência, isto é, colocando o operador $*$ em um nível mais baixo da árvore sintática que o operador $+$.

1ª GRAMÁTICA	2ª GRAMÁTICA	3ª GRAMÁTICA
	Ambiguidade removida com associatividade, $+$ e $*$ tem a mesma	
Gramática ambígua	precedência	Precedência convencional
$E \rightarrow E + E$	$E \rightarrow E + T$	$E \rightarrow E + T$
$E \rightarrow E * E$	$E \rightarrow E * T$	$E \rightarrow T$
$E \rightarrow (E)$	$E \rightarrow T$	$T = T * F$
$E \rightarrow a$	$T \rightarrow (E)$	$T \rightarrow F$
	$T \rightarrow a$	$F \rightarrow (E)$
		$F \rightarrow a$

5.4 Hierarquia de Chomsky

Formas normais impõem restrições às formações de regras de uma gramática a fim de assegurar que determinadas propriedades sejam alcançadas. Existem alguns tipos de abordagens para essa construção das restrições, sendo duas de maior destaque: a forma normal de Chomsky e a de Greibach. Nesta seção será usada a abordagem de Chomsky. Sob esta ótica, uma gramática $\mathcal{G} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ pode ser classificada pela forma de suas regras de produção em quatro tipos:

Tipo 0 Nenhuma restrição é imposta na forma das regras de produção. Estas gramáticas são chamadas de Turing reconhecíveis. A definição 5.5 refere-se a este tipo.

Tipo 1 Se $\alpha \rightarrow \beta \in \mathcal{R}$ tem-se que $|\alpha| \leq |\beta|$, onde $|\alpha|$ e $|\beta|$ representam os com-

primos de α e β , respectivamente, e excetuando a regra $\alpha \rightarrow \Lambda$, cujo α não pode estar do lado direito de qualquer produção. Gramáticas do tipo 1 são também chamadas *sensíveis ao contexto* pois há restrição de que as regras sejam da forma $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ com α_1, α_2 e $\beta \in \Sigma^*$, $\beta \neq \Lambda$ e $A \in N$. Esta restrição é equivalente à anterior, e pode ser lida como “ A pode ser substituído por β no contexto de α_1 e α_2 ”. As gramáticas dos exemplos a seguir são do tipo 1.

Exemplo 5.9 Seja a gramática \mathcal{G} com $N = \{S\}$, $T = \{0, 1\}$ e $\mathcal{R} = \{S \rightarrow 0S1, S \rightarrow 01\}$. Então é fácil verificar que $L(\mathcal{G}) = \{0^n 1^n | n \geq 1\}$.

Exemplo 5.10 Seja a gramática \mathcal{G} com $N = \{S, B, C\}$ e $T = \{a, b, c\}$ e

$$\mathcal{R} = \left\{ \begin{array}{ll} 1. S \rightarrow aSBC & 5. bB \rightarrow bb \\ 2. S \rightarrow aBC & 6. bC \rightarrow bc \\ 3. CB \rightarrow BC & 7. cC \rightarrow cc \\ 4. aB \rightarrow ab \end{array} \right\}$$

Neste caso não é tão imediato verificar que $L(\mathcal{G}) = \{a^n b^n c^n | n \geq 1\}$. Deixe-se como exercício para o leitor (um arrazoado completo para este exemplo encontra-se em Hopcroft e Velman [82], seção 2.2).

Tipo 2 Se $\alpha \rightarrow \beta \in \mathcal{R}$ tem-se que $\alpha \in N$. Em contraste com as gramáticas do tipo 1, estas são chamadas “livres do contexto”. A gramática do exemplo a seguir é do tipo 2.

Exemplo 5.11 Seja a gramática \mathcal{G} com $N = \{S, A, B\}$ e $T = \{a, b\}$ e

$$\mathcal{R} = \left\{ \begin{array}{ll} 1. S \rightarrow aB & 5. A \rightarrow bAA \\ 2. S \rightarrow bA & 6. B \rightarrow b \\ 3. A \rightarrow a & 7. B \rightarrow bS \\ 4. A \rightarrow aS & 8. B \rightarrow aBB \end{array} \right\}$$

A linguagem $L(\mathcal{G})$ consiste das cadeias em $T^* - \{\Lambda\}$ com mesmo número de símbolos a e b .

A gramática do exemplo 5.9 também é do tipo 2.

Tipo 3 Também chamadas de gramáticas regulares, nestas gramáticas toda regra de produção é de uma das formas a seguir:

$$A \rightarrow a, \quad a \in T \text{ e } A \in N$$

$$A \rightarrow aB, \quad a \in T \text{ e } A, B \in N$$

Exemplo 5.12 Seja a gramática \mathcal{G} com $N = \{S, A, B\}$ e $T = \{0, 1\}$ e

$$\mathcal{R} = \left\{ \begin{array}{ll} 1. \ S \rightarrow 0A & 6. \ B \rightarrow 1B \\ 2. \ S \rightarrow 1B & 7. \ B \rightarrow 1 \\ 3. \ A \rightarrow 0A & 8. \ B \rightarrow 0 \\ 4. \ A \rightarrow 0S & 9. \ S \rightarrow 0 \\ 5. \ A \rightarrow 1B & \end{array} \right\}$$

Naturalmente as gramáticas do tipo 3 são do tipo 2, 1 e 0; as do tipo 2 são também 1 e 0 e as do tipo 1 são também 0.

Existe uma *caracterização* sintática do poder gerativo e representacional com respeito ao *tipo* de linguagem gerada, e uma correspondência efetiva entre tais gramáticas e certos sistemas formais conhecidos como *reconhecedores*.

Os reconhecedores são sistemas de *validação* que não produzem *saídas*. Em geral, admitem certos *estados iniciais* e quando *param* é verificado em que *estado final* se encontram. Em geral possuem apenas dois estados finais *aceitação* e *rejeição*. No entanto existem sistemas que possuem um número ilimitado de estados finais.

As gramáticas de tipo 3 são as menos complexas e as linguagens que elas geram são chamadas linguagens regulares: seus elementos podem ser reconhecidos em tempo real por autômatos finitos. O estudo dos autômatos finitos não será feito neste capítulo, mas eles surgem naturalmente a partir das regras de produção das gramáticas tipo 3.

Exemplo 5.13 Seja a gramática tipo 3 com

$$\Sigma = \{a, b, S, A, B\}$$

$$N = \{S, A, B\}$$

$$\mathcal{A} = \{S\}$$

$$\mathcal{R} = \{S \rightarrow aA, S \rightarrow bB, A \rightarrow bB, B \rightarrow aA, A \rightarrow a\}$$

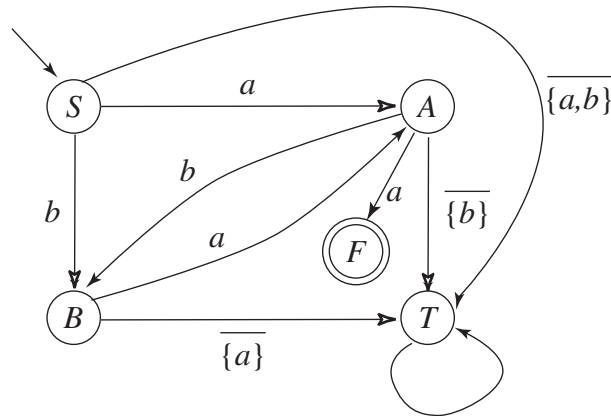


Figura 5.3: Autômato finito.

tem como reconhecedor o autômato finito da Figura 5.3

O estado S é o estado inicial e o estado F é o estado final, dada uma palavra, por exemplo $ababa$. Esta é lida da esquerda para a direita, símbolo a símbolo. Quando lê a a máquina vai para o estado A , quando lê b vai para o estado B , lê a vai para o estado A , lê b vai para B e finalmente lê a e termina no estado A . T é uma armadilha.

A organização das linguagens através desta caracterização sintática é a chamada **Hierarquia de Chomsky**, que é apresentada esquematicamente pela Figura 5.4.

5.5 Gramáticas Sensíveis ao Contexto

Quando Chomsky introduziu o conceito de gramáticas sensíveis ao contexto, ele queria capturar a idéia de que em linguagens naturais, certas palavras podem ser, ou não ser, apropriadas em determinadas posições, de acordo com o contexto onde elas se encontram, isto é, os strings vizinhos a estas palavras. Assim a substituição de um símbolo terminal depende de um string de contexto do seu lado esquerdo e de outro string de contexto do seu lado direito.

Estas mesmas gramáticas sensíveis ao contexto são suficientes para descrever

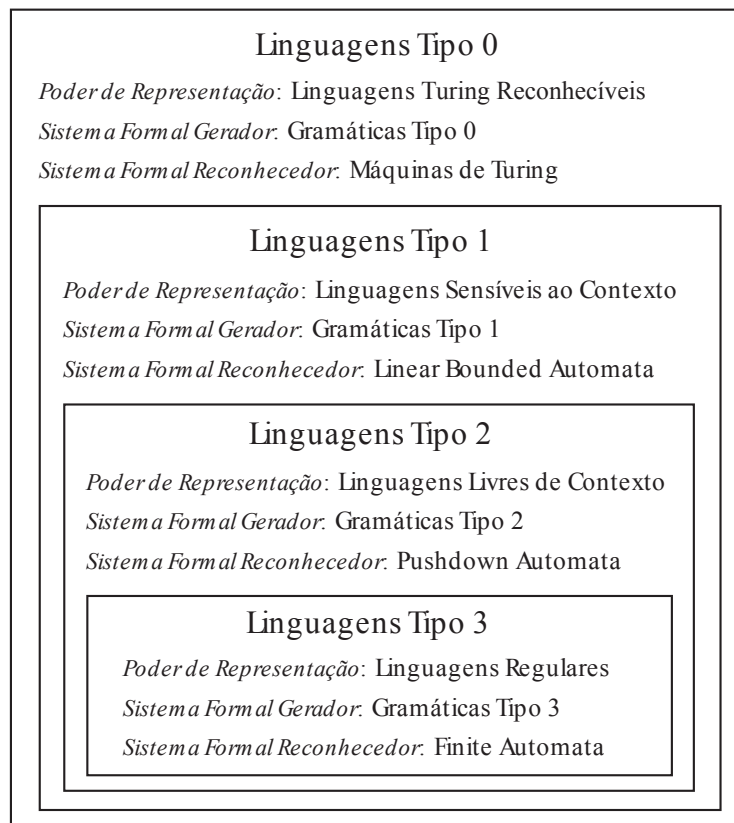


Figura 5.4: Hierarquia de Chomsky.

uma linguagem de programação. Entretanto, na prática elas não são empregadas porque é melhor usar as gramáticas livres de contexto acrescidas de alguns complementos, tais como regras para tipagem, para escopo e para mecanismos de restrição de acesso (métodos `public` \times `private`).

Na Seção 5.4 foi apresentado que gramáticas sensíveis ao contexto $\mathcal{G} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ são aquelas que possuem como característica possuir regras $\alpha \rightarrow \beta \in \mathcal{R}$ tais que $|\alpha| \leq |\beta|$, onde $|\alpha|$ e $|\beta|$ representam os comprimentos de α e β , respectivamente. Existe uma exceção a esta definição que são as regras do tipo $\alpha \rightarrow \Lambda$, cujo α não pode estar do lado direito de qualquer produção. As linguagens produzidas por este tipo de gramática são reconhecidas por Autômatos Linearmente Limitados. Nesta seção serão abordados alguns teoremas importantes das gramáticas sensíveis ao contexto.

Teorema 5.1 Toda gramática sensível ao contexto é decidível.

Seja $w \in \Sigma^*$ e $|w| = n$. Em qualquer derivação $\alpha_i, i \leq n$ pode-se afirmar com segurança que $\alpha_i \neq \alpha_j$ para qualquer $i < j$. Observe que $|\alpha_i| \leq n$, o que significa que existe uma quantidade finita de derivações, e ainda, que estas seqüências podem ser geradas por funções recursivas primitivas. Por fim, basta verificar dentre todas estas produções qual é aquela que é igual a w . Por outro lado, note que a quantidade de derivações cresce de modo incrivelmente exponencial, o que torna impraticável esta determinação na realidade.

O fechamento de uma operação sobre um conjunto indica que elementos de um conjunto operados por este operador resultam em elementos que também pertencem ao conjunto. Desta forma, torna-se importante conhecer aquelas operações que asseguram as propriedades de elementos de um conjunto tal como as gramáticas de modo geral.

Teorema 5.2 As gramáticas sensíveis ao contexto são fechadas para a união, interseção, complemento, concatenação, estrela de Kleene e reverso.

Uma linguagem $L(G)$ é definida por todas as sentenças que podem ser derivadas por uma combinação de passos começando a partir dos axiomas \mathcal{A} , tal como $\mathcal{A} = \{S\}$. Suponha também que $\mathcal{R} = \{S \rightarrow S\}$. Começando com S e aplicando a re-

gra de produção, obtém-se S . Aplicando a regra duas vezes, novamente obtém-se S . Por indução, aplicando um número finito de vezes a regra de produção, continua-se obtendo S . Desta forma, como nenhuma sentença é obtida, tem-se que $L(G) = \{\}$.

Observe que não é um requisito de uma linguagem que ela termine. Entretanto, quando se trata de uma produção feita por máquina, torna-se importante saber que a produção em algum momento irá parar. Uma forma alternativa para caracterizar que a construção de uma linguagem termina em algum momento é dizer que uma gramática produz linguagens não vazias. O teorema a seguir está inserido neste cenário.

Teorema 5.3 O problema de saber se uma gramática sensível ao contexto gera uma linguagem vazia é indecidível.

5.6 Gramáticas Livres de Contexto

Na Seção 5.4 foi apresentado que gramáticas livres de contexto $\mathcal{G} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ são aquelas onde $\alpha \rightarrow \beta \in \mathcal{R}$ tal que $\alpha \in N$, além de possuírem as propriedades de gramáticas tipo 1. Nesta seção serão observados alguns aspectos importantes sobre estas gramáticas.

Teorema 5.4 O problema de decidir se uma gramática livre de contexto é ambígua é insolúvel.

Tal como foi discutido na seção 5.5, é importante conhecer as propriedades de fechamento para as gramáticas livres de contexto. Neste sentido, os dois teoremas a seguir apontam para as operações que são fechadas em gramáticas livres de contexto, e as operações que não são fechadas.

Teorema 5.5 As linguagens livres de contexto são fechadas em relação às operações de união, concatenação e estrela de Kleene.

Teorema 5.6 As linguagens livres de contexto não são fechadas em relação às operações de interseção e complemento.

Algumas vezes é necessário ser capaz de avaliar se uma linguagem não é livre de contexto, algo que pode ser feito através do pumping lemma (lema do bombeamento). Este lema atesta que toda linguagem livre de contexto possui um valor especial, chamado de pumping length, tal que todas as cadeias com comprimentos maiores que este valor, podem ser obtidas através de uma espécie de operação de bombeamento. Uma cadeia é dividida em cinco partes (tipicamente $uvxyz$), de tal modo que a segunda (x) e a quarta (y) partes podem ser repetidas quantas vezes se desejar, resultando em uma cadeia que ainda pertence à linguagem.

Lema 5.1 [Pumping Lemma para Linguagens Livres de Contexto] Se $L(G)$ é uma linguagem livre de contexto, então existe um valor p (pumping length) onde, $w \in L(G)$, com $|w| \geq p$, então w pode ser dividido em cinco partes $w = uvxyz$ satisfazendo as condições:

- (a) para $i \geq 0$, $uv^i xy^i z \in L(G)$
- (b) $|vy| > 0$
- (c) $|vxy| \leq p$

Para entender este teorema, observe que o símbolo inicial S de uma gramática G , cujas regras são do tipo $\alpha \rightarrow \beta$, irá derivar uma cadeia com comprimento máximo $|\beta|$, isto é:

$$S \Rightarrow^1 \underbrace{\square}_{|\beta|}$$

Duas derivações a partir do símbolo inicial S irão produzir uma cadeia com comprimento máximo $|\beta|^2$, isto é:

$$S \Rightarrow^2 \underbrace{\underbrace{\square}_{|\beta|} \underbrace{\square}_{|\beta|} \dots \underbrace{\square}_{|\beta|}}_{|\beta| \text{ vezes}}$$

Por indução, h derivações a partir do símbolo inicial S irão produzir uma cadeia com comprimento máximo $|\beta|^h$. Desta forma, se uma cadeia possui comprimento $|\beta|^h + 1$ é porque ela foi derivada no passo $h + 1$.

Seja $\#(N)$ o número de símbolos não terminais de G . Além disso, assumamos um valor conhecido com comprimento de bombeamento dado por $p = |\beta|^{\#(N)+1}$.

Se $w \in L(G)$, $|w| \geq p$, então a quantidade de passos para produzir w tem que ser maior que $\#(N) + 1$. Assim, w é derivado com pelo menos $\#(N) + 2$ passos. A produção começa com o símbolo inicial S , seguido de pelo menos $\#(N) + 2$ cadeias. A última cadeia é w e deve conter apenas símbolos terminais. Todas as demais $\#(N) + 1$ cadeias possuem ao menos um símbolo não terminal, caso contrário, não seria possível realizar a derivação da cadeia do passo seguinte. Ora, se G possui $\#(N)$ símbolos não terminais e existem $\#(N) + 1$ produções, então algum símbolo não terminal R tem que ser repetido nas produções de w ⁴.

Se o símbolo não terminal R será repetido é porque o conjunto \mathcal{R} de regras da gramática G precisa ter regras do tipo:

$$\begin{aligned} S &\rightarrow uRz \\ R &\rightarrow vRy \\ R &\rightarrow x \end{aligned}$$

A primeira regra de produção corresponde ao caso mais genérico possível de cadeia derivada, onde o símbolo R pode estar cercado por cadeias quaisquer u e z . Em seguida, se R deve ser repetido, então a regra mais genérica de produção é derivar uma nova cadeia, novamente com R cercado por duas outras cadeias, neste caso por v e y . Por fim, deve haver finalmente uma substituição de R por uma cadeia x . Assim, w deve ser da forma $w = uv^i xy^i z$, $i \geq 0$, como estabelece a condição (a) do lema.

Para obter a condição (b) é necessário que v e y não sejam Λ . Se eles fossem Λ , isso significaria que a produção de w não repetiu R . Isto é uma contradição visto que a construção de w , por definição, exige ao menos uma chamada à regra $R \rightarrow vRy$.

Por fim, observe que a quantidade de passos em que R produz vxy é no máximo $\#(N) + 1$ vezes. Logo, o comprimento máximo da cadeia produzido por essas derivações é no máximo $|\beta|^{\#(N)+1} = p$ e portanto $|vxy| \leq p$, o que satisfaz a condição (c).

⁴Este raciocínio utiliza o *pigeonhole principle* (casa de pombos), que em sua formulação mais simples diz que de $n + 1$ pombos ocupam uma casa de pombos com n buracos, então algum buraco possui ao menos dois pombos.

Exemplo 5.14 Seja $L(G) = \{w = 0^i 1^i 2^i | i \geq 0\}$ e seja p o comprimento de bombeamento. Tomando $w = 0^p 1^p 2^p$, $|w| \geq p$, divide-se w em cinco partes, isto é, $w = uvxyz$. É necessário que $|vxy| \leq p$, logo vxy não pode conter 0 e 2 ao mesmo tempo.

Supondo que o 0 não ocorra em vxy (o caso com o símbolo 2 é análogo) e fazendo $i = 0$, tem-se que o número de 1's e 2's em $w_0 = uv^0xy^0z = uxz$ é menor que $2p$. Além disso, o número de 0's em w_0 tem que ser p , logo $w_0 \notin L(G)$ e L não é livre de contexto.

Exemplo 5.15 Seja $L(G) = \{w = 0^i 10^i 10^i | i \geq 1\}$ e seja p o comprimento de bombeamento. Tomando $w = 0^p 10^p 10^p$, $|w| \geq p$, divide-se w em cinco partes, isto é, $w = uvxyz$. É necessário que $|vxy| \leq p$ e que $|vy| > 0$. Existe dois casos a serem analisados: (1) quando vy não possui 0's; (2) quando vy possui pelo menos um 0.

No primeiro caso, se vy não possui 0's e $|vy| > 0$, então ou $v = 1$, ou $y = 1$. Assumindo que $v = 1$ (para $y = 1$ a análise é análoga), então o bombeamento produziria 1's contíguos, fazendo com que $w \notin L(G)$.

No segundo caso quando existe ao menos um 0 em vy e $|vxy| \leq p$. Assim, vxy é pequeno demais para acomodar todos os três agrupamentos de 0's presentes em $0^p 10^p 10^p$. Assim, $w = uv^0xy^0z = uxz$ possui pelo menos um bloco com p 0's e pelo menos um bloco com menos de p 0's. Logo, $w \notin L(G)$.

5.7 Gramáticas Regulares

Na Seção 5.4 foi definido que gramáticas regulares $\mathcal{G} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ são aquelas cujas regras de produção são da forma $A \rightarrow a$, $a \in T$ e $A \in N$ e $A \rightarrow aB$, $a \in T$ e $A, B \in N$, além de possuírem as propriedades de gramáticas tipo 2.

Desta forma, pode-se contruir uma gramática regular $G3 = \langle \Sigma_3, \mathcal{L}_3, \mathcal{A}_3, \mathcal{R}_3 \rangle$ obtida a partir da concatenação das gramáticas $G1 = \langle \Sigma_1, \mathcal{L}_1, \mathcal{A}_1, \mathcal{R}_1 \rangle$ e $G2 = \langle \Sigma_2, \mathcal{L}_2, \mathcal{A}_2, \mathcal{R}_2 \rangle$ tal que $L(G3) = \{ab | a \in L(G1), b \in L(G2)\}$. Nesta gramática tem-se que:

- $N_3 = N_1 \cup N_2$
- $T_3 = T_1 \cup T_2$
- $\mathcal{A}_3 = \mathcal{A}_1$
- \mathcal{R}_3 construído da seguinte forma:
 1. Se $A \rightarrow aB \in \mathcal{R}_1$, então $A \rightarrow aB \in \mathcal{R}_3$
 2. Se $A \rightarrow a \in \mathcal{R}_1$, então $A \rightarrow a\mathcal{A}_2 \in \mathcal{R}_3$
 3. Todas as produções de \mathcal{R}_2 estão em \mathcal{R}_3

Exemplo 5.16 Seja uma gramática G_1 , com $\mathcal{A}_1 = \{S_1\}$ cujas regras de produção são dadas por:

$$S_1 \rightarrow 0A$$

$$S_1 \rightarrow 1B$$

$$A \rightarrow 1$$

$$B \rightarrow 2$$

e G_2 , com $\mathcal{A}_2 = \{S_2\}$ cujas regras de produção são dadas por:

$$S_2 \rightarrow a$$

$$S_2 \rightarrow bC$$

$$S_2 \rightarrow aS_2$$

$$C \rightarrow a$$

As linguagem produzidas por estas gramáticas são $L(G_1) = \{01, 12\}$ e $L(G_2) = \{a, aa, aaa, \dots, ba, aba, aaba, \dots\}$. A concatenação $L(G_3) = L(G_2).L(G_2)$ resulta em $\{01a, 01aa, \dots, 01ba, 01aba, \dots, 12a, 12aa, \dots, 12ba, 12aba, \dots\}$. Para que isso aconteça, as regras de produção passam a ser:

$$S_1 \rightarrow 0A$$

$$S_1 \rightarrow 1B$$

$$A \rightarrow 1S_2$$

$$B \rightarrow 2S_2$$

$$S_2 \rightarrow a$$

$$S_2 \rightarrow bC$$

$$S_2 \rightarrow aS_2$$

$$C \rightarrow a$$

De modo análogo, pode-se contruir uma gramática regular $G3 = \langle \Sigma_3, \mathcal{L}_3, \mathcal{A}_3, \mathcal{R}_3 \rangle$ obtida a partir da união das gramáticas $G1 = \langle \Sigma_1, \mathcal{L}_1, \mathcal{A}_1, \mathcal{R}_1 \rangle$ e $G2 = \langle \Sigma_2, \mathcal{L}_2, \mathcal{A}_2, \mathcal{R}_2 \rangle$ tal que $L(G3) = L(G1) \cup L(G2) = \{w | w \in L(G1) \vee w \in L(G2)\}$. Nesta gramática tem-se que:

- $N_3 = N_1 \cup N_2 \cup \mathcal{A}_3$
- $T_3 = T_1 \cup T_2$
- $\mathcal{A}_3 = \{S_3\}$, onde S_3 é um novo símbolo não terminal
- \mathcal{R}_3 construído da seguinte forma:
 1. $\mathcal{R}_1 \in \mathcal{R}_3$ substituindo todos os S_1 do lado esquerdo da regra por S_3
 2. $\mathcal{R}_2 \in \mathcal{R}_3$ substituindo todos os S_2 do lado esquerdo da regra por S_3
 3. Remove-se as produções $S \rightarrow \Lambda$
 4. Somente se repetem as regras que contém S_1 e S_2 se estes símbolos aparecerem em algum momento no lado direito das produções

Exemplo 5.17 Utilizando as gramáticas G_1 e G_2 definidas no exemplo 5.16 tem-se

que $G_3 = G_1 \cup G_2$ é dada por:

$$\begin{aligned}
S &\rightarrow 0A \\
S &\rightarrow 1B \\
A &\rightarrow 1 \\
B &\rightarrow 2 \\
S &\rightarrow a \\
S &\rightarrow bC \\
S &\rightarrow aS_2 \\
C &\rightarrow a \\
S_2 &\rightarrow a \\
S_2 &\rightarrow bC \\
S_2 &\rightarrow aS_2
\end{aligned}$$

Por fim, se for considerada a estrela de Kleene, tem-se que $L^* = \{\Lambda\} \cup L \cup L.L \cup L.L.L \cup \dots$, isto é se $w \in L^*$ então w é obtido a partir da concatenação de zero ou mais cadeias de L , algo parecido com a definição de Σ^* . Assim, o próximo passo é definir uma gramática $G = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ a partir de $G_1 = \langle \Sigma_1, \mathcal{L}_1, \mathcal{A}_1, \mathcal{R}_1 \rangle$ tal que $L(G) = L(G_1)^*$. Nesta gramática tem-se que:

- $N = N_1 \cup \mathcal{A}$
- $T = T_1$
- $\mathcal{A} = \{S\}$
- \mathcal{R} construído da seguinte forma, com $A, B \neq S_1$:
 1. Se $S_1 \rightarrow \alpha \in \mathcal{R}_1$ então $S \rightarrow \alpha \in \mathcal{R}$
 2. Se $A \rightarrow aB \in \mathcal{R}_1$ então $A \rightarrow aB \in \mathcal{R}$
 3. Se $A \rightarrow a \in \mathcal{R}_1$ então $A \rightarrow aS \in \mathcal{R}$ e $A \rightarrow a \in \mathcal{R}$
 4. $S \rightarrow \Lambda \in \mathcal{R}$

Exemplo 5.18 Seja $L(G_1) = \{a^n bc | n \geq 0\}$ construída pelo seguinte conjunto de produções:

$$S_1 \rightarrow aS_1$$

$$S_1 \rightarrow bA$$

$$A \rightarrow c$$

$L(G) = L(G_1)^*$ é dada por:

$$S \rightarrow aS$$

$$S \rightarrow bA$$

$$A \rightarrow cS$$

$$A \rightarrow c$$

$$S \rightarrow \Lambda$$

Definição 5.8 Um conjunto regular sobre um alfabeto finito Σ é construído da seguinte forma:

1. O conjunto \emptyset é um conjunto regular sobre o alfabeto Σ ;
2. O conjunto $\{\Lambda\}$ com a cadeia vazia é um conjunto regular sobre o alfabeto Σ ;
3. O conjunto $\{\sigma\}$ com $\sigma \in \Sigma$ é um conjunto regular sobre o alfabeto Σ ;
4. Se A e B são conjuntos regulares sobre o alfabeto Σ então $A \cup B$, $A.B$ e A^* são conjuntos regulares sobre o alfabeto Σ ;

O modo de representação de conjuntos regulares é conhecido como Expressão Regular, e é definido como se segue.

Definição 5.9 Uma expressão regular sobre um alfabeto finito Σ é definida como:

1. \emptyset é uma expressão regular denotando o conjunto regular \emptyset ;
2. Λ é uma expressão regular denotando o conjunto regular $\{\Lambda\}$;
3. σ é uma expressão regular denotando o conjunto regular $\{\sigma\}$;
4. Se a e b são expressões regulares denotando os conjuntos regulares A e B , então $(a + b)$ é uma expressão regular denotando $A \cup B$, (ab) é uma expressão regular denotando $A.B$, e $(a)^*$ é uma expressão regular denotando A^* ;⁵

⁵Na literatura, pode-se encontrar também a notação a^+ para denotar a expressão regular pp^* , porém esta notação não será empregada neste livro.

Em alguns casos, quando não há ambiguidade, é possível remover os parênteses. Para isto é importante ter em mente que a ordem crescente de precedência (abrangência do escopo) entre os operadores é $*$ \prec $.$ \prec $+$. Desta forma, a expressão $(0 + (1(0^*)))$ pode ser reescrita como sendo $0 + 10^*$.

Exemplo 5.19 A seguir são apresentados alguns significados de expressões regulares:

1. ab denota a linguagem regular $L(G) = \{ab\}$;
2. a^* denota a linguagem regular $L(G) = \{\Lambda, 0, 00, 000, \dots\}$;
3. $(a + b)$ denota a linguagem regular $L(G) = \{a, b\}$;
4. a^*b^* denota o conjunto de todas as cadeias de a 's seguidas de todas as cadeias de b 's e a cadeia nula;
5. $(a^*b^*)^*$ também denota o conjunto de todas as cadeias de a 's e b 's, incluindo a cadeia nula;
6. $(a + b)^*$ denota o conjunto de todas as cadeias de a 's e b 's, incluindo a cadeia nula;
7. $(a + b)^*aab$ denota o conjunto de todas as cadeias de a 's e b 's, terminando com o string aab ;
8. $(a + b)(\clubsuit + \diamondsuit + \heartsuit + \spadesuit)^*$ denota o conjunto de todas as cadeias de $\{\clubsuit, \diamondsuit, \heartsuit, \spadesuit\}^*$, começando com a ou b ;

O processo de determinação da gramática correspondente a uma expressão regular consiste em um problema de dividir para conquistar. Inicialmente divide-se a expressão regular em fragmentos para os quais seja facilmente determinada a gramática associada. Em seguida, vai se construindo a expressão original, operador por operador ($+$, $.$, $*$), e refletindo estas operações em operações entre gramáticas (\cup , $.$, $*$, respectivamente). Por exemplo, seja a expressão regular $(01 + 2)$. Esta expressão pode ser dividida em dois fragmentos, 01 e 2 , cada qual com sua gramática

associada:

$$\underbrace{\underbrace{(01)}_{G_1} + \underbrace{2}_{G_2}}_{G_1 \cup G_2}$$

onde as regras destas gramáticas são dadas por:

$$\begin{aligned} L(G_1) & : S_1 \rightarrow 0A \\ & A \rightarrow 1 \end{aligned}$$

$$L(G_2) : S_2 \rightarrow 2$$

$$\begin{aligned} L(G_1) \cup L(G_2) & : S \rightarrow 0A \\ & A \rightarrow 1 \\ & S \rightarrow 2 \end{aligned}$$

Exemplo 5.20 Determine as regras de produção associadas a expressão regular $(0 + 1 + 2)^*$.

Para a determinação desta gramática, serão criados os fragmentos:

$$\underbrace{\underbrace{\underbrace{0}_{G_1} + \underbrace{1}_{G_2}}_{G_4} + \underbrace{2}_{G_3}}_{G_5}^*$$

$$\underbrace{\hspace{10em}}_{G_6}$$

onde as regras destas gramáticas são dadas por:

$$\begin{aligned} L(G_1) & : S_1 \rightarrow 0 & L(G_5) & : S_5 \rightarrow 0 \\ & & & S_5 \rightarrow 1 \\ L(G_2) & : S_2 \rightarrow 1 & & S_5 \rightarrow 2 \\ \\ L(G_3) & : S_3 \rightarrow 2 & L(G_6) & : S \rightarrow 0S \\ & & & S \rightarrow 0 \\ L(G_4) & : S_4 \rightarrow 0 & & S \rightarrow 1S \\ & S_4 \rightarrow 1 & & S \rightarrow 1 \\ & & & S \rightarrow 2S \\ & & & S \rightarrow 2 \\ & & & S \rightarrow \Lambda \end{aligned}$$

Exemplo 5.21 Determine as regras de produção associadas a expressão regular $(00^* + 1)2$.

Para a determinação desta gramática, serão criados os fragmentos:

$$\begin{array}{c}
 \underbrace{\underbrace{0}_{G_1} \underbrace{0^*}_{G_2} + \underbrace{1}_{G_3}}_{G_5} 2 \\
 \underbrace{\hspace{10em}}_{G_6} \\
 \underbrace{\hspace{10em}}_{G_7}
 \end{array}$$

onde as regras destas gramáticas são dadas por:

$$\begin{array}{ll}
 L(G_1) : S_1 \rightarrow 0 & L(G_6) : S_6 \rightarrow 1 \\
 & S_6 \rightarrow 0S_2 \\
 L(G_2) : S_2 \rightarrow 0S_2 & S_2 \rightarrow 0S_2 \\
 S_2 \rightarrow \Lambda & \\
 & L(G_7) : S \rightarrow 1S_4 \\
 L(G_3) : S_3 \rightarrow 1 & S \rightarrow 0S_2 \\
 & S_2 \rightarrow 0S_2 \\
 L(G_4) : S_4 \rightarrow 2 & S_4 \rightarrow 2 \\
 & \\
 L(G_5) : S_5 \rightarrow 0S_2 & \\
 S_2 \rightarrow 0S_2 & \\
 S_2 \rightarrow \Lambda &
 \end{array}$$

O processo inverso de determinação de uma expressão regular a partir de um conjunto de regras de uma gramática envolve algumas operações de uma álgebra chamada Álgebra de Kleene. Assim, sejam α, β, γ expressões regulares, tem-se as seguintes propriedades:

1. $\alpha + (\beta + \gamma) = (\alpha + \beta) + \gamma$
2. $\alpha.(\beta.\gamma) = (\alpha.\beta).\gamma$
3. $\alpha + \beta = \beta + \alpha$
4. $\alpha + \alpha = \alpha$
5. $\alpha.(\beta + \gamma) = (\alpha.\beta) + (\alpha.\gamma)$

$$6. (\beta + \gamma).\alpha = (\beta.\alpha) + (\gamma.\alpha)$$

$$7. \Lambda.\alpha = \alpha = \alpha.\Lambda$$

$$8. \alpha^* = \Lambda + \alpha.\alpha^*$$

$$9. \alpha^* = (\Lambda + \alpha)^*$$

Deste modo, para construir um expressão regular que denota $L(G)$ a partir de regras de produção, deve-se inicialmente associar uma linguagem a cada regra de produção. Por exemplo, as regras de produção a seguir seriam mapeadas pelas seguintes linguagens:

$$S \rightarrow aS \rightsquigarrow L(S) = \{a\}.L(S)$$

$$S \rightarrow bR \rightsquigarrow L(S) = \{b\}.L(R)$$

$$R \rightarrow aS \rightsquigarrow L(R) = \{a\}.L(S)$$

Em seguida, realiza-se a união das linguagens com mesma simbologia, obtendo-se:

$$L(S) = \{a\}.L(S) \cup \{b\}.L(R)$$

$$L(R) = \{a\}.L(S)$$

O passo seguinte é uma simplificação notacional que já remete a notação das expressões regulares. Os sinais de $\{ \}$ são omitidos, e cada linguagem $L(x)$ é substituída por uma variável X . Por fim, a operação de \cup é substituída pela operação regular homomorfica a mesma, isto é, $+$. Assim, tem-se:

$$S = aS + bR$$

$$R = aS$$

A etapa seguinte é realizar as substituições das expressões a fim se compor uma única expressão. Neste momento, começa-se a aplicar a álgebra de Kleene para compor esta única expressão. Neste caso, substituindo o valor de R e S tem-se:

$$S = aS + baS = (a + ba)S$$

Para prosseguir com o passo seguinte é importante observar uma propriedade das expressões denotativas de linguagens. Seja a expressão $X = \alpha X + \beta$, está

associada às regras de produção:

$$\begin{aligned} X &\rightarrow \beta \\ X &\rightarrow \alpha X \end{aligned}$$

que produz a linguagem $L(X) = \{\beta, \alpha\beta, \alpha\alpha\beta, \alpha\alpha\alpha\beta, \dots\} = \alpha^*\beta$. Esta observação permite concluir que $X = \alpha X + \beta$ por ser substituído por $X = \alpha^*\beta$. Esta propriedade é conhecida como lema de Arden.

Lema 5.2 [Lema de Arden] Sejam A e B linguagens de um alfabeto Σ , $\Lambda \notin A$ e uma linguagem X descrita por ela mesma. O subconjunto de Σ^* que satisfaz $X = AX + B$ é $X = A^*B$.

Desta forma, voltando à expressão $S = (a + ba)S$, e tomando $\alpha = (a + ba)$ e $\beta = \Lambda$, tem-se que $S = \alpha S + \beta = \alpha^*\beta = (a + ba)^*\Lambda = (a + ba)^*$. Como o símbolo S é o símbolo inicial da gramática G , tem-se que $L(G) = L(S)$. Portanto, $L(G) = (a + ba)^*$.

De forma análoga ao que aconteceu na seção 5.6, algumas vezes é necessário ser capaz de avaliar se uma linguagem não é regular. Nesta seção será utilizado novamente o *pumping lemma* (lema do bombeamento) para a execução desta atividade, desta vez adaptado para linguagens regulares.

Lema 5.3 [Pumping Lemma para Linguagens Regulares] Se $L(G)$ é uma linguagem regular, então existe um valor p (*pumping length*) onde, $w \in L(G)$, com $|w| \geq p$, então w pode ser dividido em três partes $w = xyz$ satisfazendo as condições:

- (a) para $i \geq 0$, $xy^iz \in L(G)$
- (b) $|y| > 0$
- (c) $|xy| \leq p$

As regras em uma gramática regular à direita são do tipo $A \rightarrow aB$, onde $A, B \in N$ e $a \in T$ (para gramáticas regulares à esquerda o raciocínio é análogo). Assim, para produzir uma cadeia w , com $|w| \geq p$ são necessárias p derivações. Seja $\#(\mathcal{R})$ o número de regras existentes em G e assumamos o comprimento de bombeamento

dado por $p = \#(\mathcal{R}) + 1$. Desta forma, para produzir w com $|w| \geq p$ serão necessárias ao menos p derivações, isto é, pelo menos $\#(\mathcal{R}) + 1$ derivações. Ora, se existem $\#(\mathcal{R})$ regras em G é porque ao menos uma regra R foi repetida (*pigeonhole principle*).

A derivação de w , com a regra R sendo repetida, é realizada como se segue:

$$\underbrace{w_1 \Rightarrow \cdots \Rightarrow w_{j-1}}_x \xRightarrow{R} \underbrace{w_j \Rightarrow \cdots \Rightarrow w_{k-1}}_y \xRightarrow{R} \underbrace{w_k \Rightarrow \cdots \Rightarrow w_n}_z$$

Assim, a cadeia que pode ser repetida fica cercada por duas outras cadeias, produzindo a condição (a), xy^iz , $i \geq 0$. A condição (b) é facilmente satisfeita uma vez que $j \neq k$ e portanto $|y| > 0$. Além disso, como w_k é a cadeia imediatamente antes da regra R ser aplicada novamente, tem-se que $|w_k| = |xy| \leq \#(\mathcal{R}) + 1$, ou seja, $|xy| \leq p$, atendendo a condição (c).

Exemplo 5.22 Seja $L(G) = \{w = 0^n 1^n | n > 0\}$ e tomando o comprimento de bombeamento p , tem-se que $w = 0^p 1^p$ e $|w| \geq p$. Dividindo w em três partes xyz , tem-se que considerar três hipóteses distintas sobre y : (1) y só possui 0's; (2) y só possui 1's; (3) y possui 0's e 1's.

No primeiro caso onde y só possui 0's, então x também só possui 0's. Desta forma xy^2z tem que possuir mais 0's do que 1's, e portanto, $xy^2z \notin L(G)$. O caso onde y só possui 1's é análoga a situação ora descrita.

No caso em que y possui 0's e 1's, então xy^2z vai possuir uma sequência $0^i 1^k 0^i 1^k$, com $j > 0$ e $k > 0$. Logo $xy^2z \notin L(G)$.

Exemplo 5.23 Seja $L(G) = \{w = 0^i 1^j | i > j\}$ e tomando o comprimento de bombeamento p , pode-se escolher $w = 0^{p+1} 1^p$ e $|w| \geq p$. Em seguida, divide-se w em três partes xyz . Como uma das condições do pumping lemma é que $|xy| \leq p$, tem-se que y é composto somente por 0's. Fazendo o bombeamento de y tem-se que a cadeia xy^2z também possuirá mais 0's do que 1's, o que em princípio não contradiz a linguagem $L(G)$.

Contudo $xy^iz \in L(G)$, até mesmo para $i = 0$. Assim, fazendo $w_0 = xy^0z = xz$ se reduz a quantidade de 0's em w_0 . Lembrando que w possui apenas um 0 a mais que 1, então xz não pode ter mais 0's que 1's, e portanto $w_0 \notin L(G)$.

5.8 Aritmética Rudimentar

Definição 5.10 Define-se o sistema formal da aritmética rudimentar (AR) , como sendo o sistema formal

$$R = \langle \Sigma_R, \mathcal{L}_R, \mathcal{A}_R, \mathcal{R}_R \rangle$$

onde

Σ_R é o alfabeto $\{0, ', =\}$

\mathcal{L}_R é a linguagem constituída de fórmulas, definidas da seguinte maneira:

- (a) 0 é um termo.
- (b) Se x é um termo, então x' é um termo.
- (c) Se x e y são termos, então $x = y$ é uma fórmula

\mathcal{A}_R é o conjunto $\{0 = 0\}$.

\mathcal{R}_R é o conjunto $\{x = y \rightarrow x' = y'\}$, escreve-se também esta única regra de inferência na forma:

$$\frac{x = y}{x' = y'}$$

É fácil ver que os únicos teoremas de R são as fórmulas:

$$0 = 0, 0' = 0', 0'' = 0'', 0''' = 0''', \dots$$

O sistema AR é bastante simples e a única operação extra-sistema formal é de *afixação* de $'$ à direita de suas palavras. Os termos da linguagem \mathcal{L}_R podem ser definidos por uma gramática regular com regras:

$$T \rightarrow 0$$

$$T \rightarrow 0V$$

$$V \rightarrow'$$

$$V \rightarrow' V$$

e as fórmula pela regra $S \rightarrow T = T$.

5.9 Conclusões e leituras recomendadas

Um sistema formal produz, reconhece ou transforma palavras de um alfabeto, independentemente do significado de tais palavras. No entanto, tais sistemas não ocorrem de modo arbitrário, salvo na explicação ou conceituação de sua natureza. Sistemas formais procuram retratar ou reproduzir formal e mecanicamente certos estados de coisas ou situações. Eles devem capturar o nosso *entendimento de uma parte da realidade* e, precedendo a formulação ou construção dos sistemas formais, temos um entendimento analítico através de um sistema semântico. O estudo dos sistemas semânticos será feito posteriormente.

A grande maioria do conhecimento humano é registrado através de descrições informais e seu entendimento depende de fatores de acultramento e adestramento. O saber é algo de natureza individual e depende de interpretações adequadas para que de modo efetivo possa ser utilizado.

Existem sistemas formais, como as gramáticas, que podem ser utilizados para capturar aspectos estruturais do conhecimento, e com isto se tornarem ferramentas de uso geral, pois possuem recursos de representação capazes de *capturar* parte do significado semântico do mesmo. A generalidade de tais sistemas formais é também devido a habilidade de poderem ser transformados, através de procedimentos efetivos, em geradores, transdutores ou reconhecedores.

O estudo detalhado das gramáticas é muito importante para a computação, principalmente para a construção de compiladores, seja para linguagens de programação ou naturais. O leitor interessado em uma abordagem mais abrangente deve consultar Hopcroft [82]. Para um tratamento completo de gramáticas recomenda-se Hopcroft e Ullman [83]. Para Sistemas de Post Brainerd [38]. Para sistemas formais em geral, Curry [84].

5.10 Exercícios

5.1 Seja um sistema formal F onde o alfabeto, a linguagem, os axiomas e as relações estão definidas a seguir:

$$\begin{aligned}
\Sigma &= \{+, *\} \\
\mathcal{L} &= \{\Sigma^*\} \\
\mathcal{A} &= \{+, *\} \\
\mathcal{R} &= \{r_1, r_2, r_3\} \\
r_1 &= \{\langle x+, x* \rangle \mid x \in \Sigma^*\} \\
r_2 &= \{\langle x+*, x*+ \rangle \mid x \in \Sigma^*\} \\
r_3 &= \{\langle x*, x++ \rangle \mid x \in \Sigma^*\}
\end{aligned}$$

Realize a dedução de $*+$.

5.2 Seja um sistema formal F do Exercício 5.1. Pede-se para deduzir $++*$.

5.3 Definir a linguagem do sistema formal do exemplo ??.

5.4 Você é um pesquisador do Projeto Genoma, dedicado ao mapeamento do código genético humano. Através de pesquisas concluiu-se que o processo de produção de cadeias genéticas respeita o sistema formal $F = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$, onde:

$$\begin{aligned}
\Sigma &= \{A(\text{adenina}), T(\text{timina}), C(\text{citosina}), G(\text{guanina})\} \\
\mathcal{L} &= \{\Sigma^*\} \\
\mathcal{A} &= \{A, T, C, G\} \\
\mathcal{R} &= \{r_1, r_2, r_3, r_4\} \\
r_1 &= \{\langle xC, TxC \rangle \mid x \in \Sigma^*\} \\
r_2 &= \{\langle xCGy, xCGCGAy \rangle \mid x, y \in \Sigma^*\} \\
r_3 &= \{\langle Tx, TCxA \rangle \mid x \in \Sigma^*\} \\
r_4 &= \{\langle xCy, xGTy \rangle \mid x, y \in \Sigma^*\}
\end{aligned}$$

A síndrome *alumni nervosus* é um problema que se manifesta em um indivíduo durante os primeiros minutos da execução de exames acadêmicos. Você descobriu que este problema está diretamente relacionado com o código genético dado por *TCGCGATA*. Mostre que é possível deduzir o código genético da *alumni nervosus* a partir do sistema formal F . Desta forma você estará provando que a síndrome é uma doença genética.

5.5 Seja o sistema de Post $\mathcal{P} = \langle N \cup T, (N \cup T)^*, \mathcal{A}, \mathcal{R} \rangle$. Descreva a linguagem $\mathcal{L}(\mathcal{P})$ para cada um dos casos a seguir.

$$\begin{aligned} \text{a)} \quad \mathcal{A} &= \{11\} \\ \mathcal{R} &= \{\boxed{1} \rightarrow \boxed{1}11\} \end{aligned}$$

$$\begin{aligned} \mathcal{A} &= \{+ =\} \\ \text{b)} \quad \mathcal{R} &= \{\boxed{1} + \boxed{2} = \boxed{3} \rightarrow \boxed{1}1 + \boxed{2} = \boxed{3}1, \\ &\quad \boxed{1} + \boxed{2} = \boxed{3} \rightarrow \boxed{2} + \boxed{1} = \boxed{3}\} \end{aligned}$$

$$\begin{aligned} \mathcal{A} &= \{||\} \\ \text{c)} \quad \mathcal{R} &= \{\boxed{1}|\boxed{2}|\boxed{3} \rightarrow \boxed{1}a|\boxed{2}b|d\boxed{3}, \\ &\quad \boxed{1}|\boxed{2}|\boxed{3} \rightarrow \boxed{1}\boxed{2}\boxed{3}\} \end{aligned}$$

5.6 Construa um sistema de produção de Post \mathcal{P} com 3 símbolos terminais que:

- a) dada uma cadeia qualquer, produza a cadeia reversa (como por exemplo $abc \Rightarrow_p^* cba$);
- b) $\mathcal{L}(\mathcal{P}) = \{a^p b^{2p}\}$, $p \geq 0$.

5.7 Determine a linguagem gerada pela gramática do exemplo 5.12.

5.8 Exercícios de gramáticas do Hopcroft, pag 24.

5.9 Contruir uma gramática que gere todas as expressões booleanas válidas, com a seguinte ordem crescente de precedência entre operadores: $\equiv, \vee, \wedge, \neg$. Os operandos são a, b, c e a gramática não deve gerar nenhuma sentença com uma quantidade supérflua de parênteses.

Exemplo de sentença aceita: $\neg(a \wedge b) \equiv \neg a \vee \neg b$

Exemplo de sentença rejeitada: $(a \vee b) \equiv c$

5.10 Contrua uma gramática livre de contexto capaz de efetuar a análise léxica e sintática do programa a seguir escrito na linguagem C. Justifique porque a gramática proposta é livre de contexto.


```

void main(void){
    //Comentário: Início do Programa
    int a ;
    int b ;
    float c ;
    //Outro comentário
    a=1 ;
    b=2 ;
    c=a+b ;
    b=b-a ;
}

```

Algoritmo 5.1: Programa ilustrativo para construção de uma gramática livre de contexto proposto no Exercício 5.10

5.11 Determine qual é a classificação das gramáticas a seguir, de acordo com a hierarquia de Chomsky.

a) $A \rightarrow 0A1$

$$A \rightarrow B$$

$$B \rightarrow \#$$

b) $A \rightarrow A1$

$$A \rightarrow BA0$$

$$B \rightarrow 10$$

5.12 Considere a gramática sensível ao contexto sobre o alfabeto $\{a, b\}$ com símbolo de partida S e as seguintes regras de produção:

$$S \rightarrow ABS|\Lambda$$

$$AB \rightarrow BA$$

$$BA \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow b$$

a) Derivar as cadeias *abba* e *babaabbbbaa* usando esta gramática

- b) Prove que qualquer palavra gerada por esta gramática um número igual de a's e b's.
- c) Prove que todas as palavras com um número igual de a's e b's podem ser geradas por esta gramática.

5.13 Considere a gramática sensível ao contexto sobre o alfabeto $\{a\}$ com símbolo de partida S e as seguintes regras de produção:

$$S \rightarrow a|DSF$$

$$Da \rightarrow aaD$$

$$DF \rightarrow \Lambda$$

- a) Derivar as cadeias aa e $aaaa$ usando esta gramática
- b) Esta gramática gera cadeias da forma a^n , em que n é uma potência de 2 com $n \neq 0$. Justifique esta afirmação, explicando o significado das variáveis D e F .

5.14 Desenvolver gramáticas livres de contexto que gerem as seguintes linguagens:

- a) $L_1 = \{a, b\}^*$
- b) $L_2 = \{w \in \{a, b\}^* | w \text{ é palíndromo}\}$
- c) $L_3 = \{w \in \{a, b\}^* | w = ss^R\}$
- d) $L_4 = \{w \in \{a, b, c\}^* | w = a^i b^j c^k, i = j \text{ ou } j = k \text{ e } i, j, k > 0\}$

5.15 Determine a gramática regular associada às seguintes expressões regulares:

- a) 1^*0
- b) $1^*0(0)^*$
- c) $111 + 001$
- d) $(1 + 00)^*$
- e) $(0(0)^*1)^*$
- f) $(0 + 1)(0 + 1)^*00$

5.16 Seja o sistema formal proposicional $\mathcal{F} = \langle \Sigma, \mathcal{L}, \mathcal{A}, \mathcal{R} \rangle$ definido na seção ??.

- a) Mostre que em qualquer fórmula α de \mathcal{L} o número de '(' é igual ao número de ')'.
b) A função $sub : \mathcal{L} \rightarrow 2^{\mathcal{L}}$ que associa a cada fórmula α o conjunto $sub(\alpha)$ de subfórmulas de α é definida por:

$$\begin{aligned} i. \quad sub(p_i) &= \{p_i\} \\ ii. \quad sub(\neg\alpha) &= sub(\alpha) \cup \{\neg\alpha\} \\ iii. \quad sub((\alpha \Rightarrow \beta)) &= sub(\alpha) \cup sub(\beta) \cup \{(\alpha \Rightarrow \beta)\} \end{aligned}$$

Prove que, se uma fórmula tem n conectivos (\Rightarrow ou \neg), então $sub(\alpha)$ tem no máximo $2n + 1$ fórmulas.

Capítulo 6

Autômatos e Linguagens

O entendimento dos autômatos trata da definição e das propriedades de modelos matemáticos de computação. Estes modelos possuem um papel importante em várias áreas aplicadas da ciência da computação. Um modelo, chamado de autômato finito, é empregado em processadores de texto, compiladores e desenvolvimento de hardware. Outro modelo, chamado de gramáticas livres de contexto, é utilizado em linguagens de programação e inteligência artificial. As teorias de computabilidade e de complexidade exigem a definição precisa do que é um computador. O estudo dos autômatos possibilita adquirir prática com as definições formais de computação introduzindo conceitos importantes neste contexto.

Um autômato finito, também chamado de máquina de estados finitos, é um formalismo que representa de forma inequívoca um processo qualquer composto por um conjunto de estados e de transições entre esses estados. O termo finito é empregado para caracterizar que o autômato pode conter apenas uma quantidade finita e limitada de informação em um dado momento. Esta condição de contorno impõe uma limitação significativa, pois limita seu emprego às linguagens regulares. Os autômatos finitos, conforme será observado nas seções a seguir, podem ser classificados como determinísticos ou não-determinísticos.

A Figura 6.1 apresenta de forma ilustrativa a representação clássica de um autômato finito como sendo uma máquina de estados finitos alimentada por um fita de entrada. Modernamente, pode-se também fazer uma analogia com uma CPU

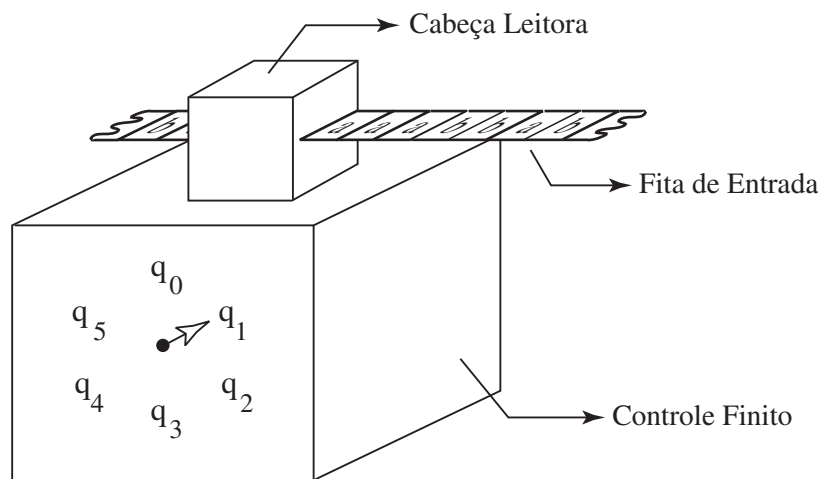


Figura 6.1: Autômato Finito ou Máquina de Estados Finita.

(Unidade Central de Processamento) de um único núcleo com uma só entrada e uma saída. No caso da máquina de fita, ela recebe como entrada uma string através de uma fita de entrada. Cada *slot* desta fita possui um símbolo de um alfabeto qualquer. Esta mesma máquina não produz saída alguma, exceto um indicativo de que a entrada foi aceita ou rejeitada. Além disto, atribui-se a máquina uma ausência permanente de memória volátil.

Inicialmente, o controle finito encontra-se configurado em um estado final chamado q_0 . A cabeça leitora lê um símbolo da fita de entrada, desloca a fita para o próximo *slot*, e o controle finito muda seu indicativo de estado de acordo com o estado atual e o símbolo que acabou de ser lido. Por fim, a cabeça leitora chega ao final da string (fim da fita) e então indica sua aceitação, ou reprovação, quanto ao que leu. Se o estado de parada faz parte do conjunto de estados considerados como estados de aceitação, então se diz que a máquina aceitou a string.

6.1 Autômato Finito Determinístico - DFA

Definição 6.1 Um autômato finito determinístico (*Deterministic Finite Automaton* - DFA) é uma quintupla $M = \langle k, \Sigma, \delta, s, F \rangle$ onde:

- k é o conjunto finito de ESTADOS;
- Σ é um ALFABETO;
- $s \in k$ é o ESTADO INICIAL;
- $F \subseteq k$ é o conjunto de ESTADOS FINAIS;
- δ é uma FUNÇÃO DE TRANSIÇÃO de $k \times \Sigma$.

O termo determinístico é um indicativo de que a função de transição δ determina de forma inequívoca o próximo estado q' a ser assumido quando a máquina M encontra-se em um estado q e lê o símbolo a . Dizemos que $(q, w) \vdash_M (q', w')$, isto é, que a máquina M no estado q e com uma string w produz um resultado onde o novo estado é q' sendo a sobra de string dada por w' se e somente se $w = aw'$ para algum símbolo $a \in \Sigma$ e $\delta(q, a) = q'$. Uma produção (q, Λ) significa que M consumiu todas as entradas, chegando ao fim da string a ser processada, e portanto sua produção acaba neste ponto.

Definição 6.2 Uma string $w \in \Sigma^*$ é aceita por M , se e somente se, há um estado $q \in F$ tal que $(s, w) \vdash_M^* (q, \Lambda)$. Uma linguagem aceita por M , $L(M)$ é o conjunto de todas as strings aceitas por M .

Exemplo 6.1 Suponha que M seja o autômato finito determinístico $\langle k, \Sigma, \delta, s, F \rangle$ onde:

	q	σ	$\delta(q, \sigma)$
$k = \{q_0, q_1\}$	q_0	a	q_0
$\Sigma = \{a, b\}$	q_0	b	q_1
$s = q_0$	q_1	a	q_1
$F = \{q_0\}$	q_1	b	q_0

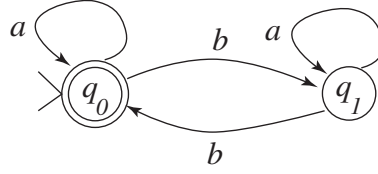


Figura 6.2: Diagrama de estados do Exemplo 6.1.

Seja $w = aabba$ uma entrada para M . Assim, tem-se que:

$$\begin{aligned}
 (q_0, \underline{a}abba) &\vdash_M (q_0, \underline{a}bba) \\
 &\vdash_M (q_0, \underline{b}ba) \\
 &\vdash_M (q_1, \underline{b}a) \\
 &\vdash_M (q_0, \underline{a}) \\
 &\vdash_M (q_0, \Lambda)
 \end{aligned}$$

Portanto, $(q_0, aabba) \vdash_M^* (q_0, \Lambda)$, e assim a string $aabba$ é aceita pela máquina M .

A representação tabular da função de transição não é a mais adequada descrição de uma máquina uma vez que a quantidade de transições pode crescer significativamente. Além disto, esta mesma representação contribui fracamente para o entendimento imediato da tarefa a qual o autômato M se destina. Deste modo, geralmente é empregada a representação gráfica chamada de diagrama de estados. A Figura 6.2 apresenta o diagrama representativo da mesma máquina M do Exemplo 6.1.

A Figura 6.2 apresenta algumas peças importantes na construção deste tipo de representação. Os estados q_0 e q_1 são representados por círculos, enquanto que a transição de um estado para outro, mediante a leitura de um símbolo, é indicada pelas setas etiquetadas com um símbolo correspondente. O estado final é caracterizado por dois círculos concêntricos e o estado inicial apresenta o indicativo $>$. No caso do exemplo em questão, o estado q_0 coincide como sendo o estado inicial e o único estado final.

Observe que todas as transições propostas pela função de transição do Exemplo 6.1 são contempladas no diagrama da Figura 6.2. Através desta figura, é mais

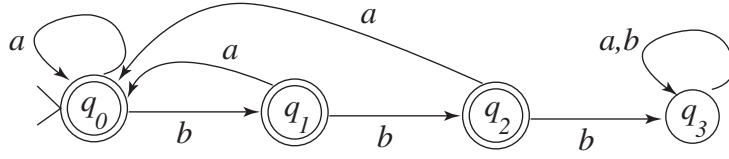


Figura 6.3: Autômato finito determinístico reconhecedor de strings que não contenham 3 b 's consecutivos.

fácil afirmar que a máquina pára no estado final q_0 quando houver um número par de b 's.

Exemplo 6.2 Projetar um autômato finito determinístico M que reconheça a linguagem $L(M) = \{w \in \{a, b\}^* \mid w \text{ não contém 3 } b\text{'s consecutivos}\}$

$$k = \{q_0, q_1, q_2, q_3\}$$

$$\Sigma = \{a, b\}$$

$$s = q_0$$

$$F = \{q_0, q_1, q_2\}$$

A Figura 6.3 apresenta o autômato correspondente.

Definição 6.3 Uma linguagem aceita por um DFA é uma linguagem regular.

6.2 Autômato Finito Não Determinístico - NFA

Na seção anterior foi observado que os DFA's apresenta como característica mais marcante a transição inequívoca de um estado para outro, isto é, $\delta(a, q) = q'$. Os autômatos finitos não determinísticos (NFA), abordados nesta seção, apresentam uma transição que não pode ser definida de modo assertivo. De fato, a função de transição desta espécie de autômato pode apresentar mais de uma opção de estado de destino, ou seja, $\delta(a, q) = q'$ ou q'' ou $q''' \dots$.

Sob esta ótica, pode-se afirmar que o conjunto de todos os autômatos finitos não determinísticos é um super conjunto dos autômatos finitos determinísticos.

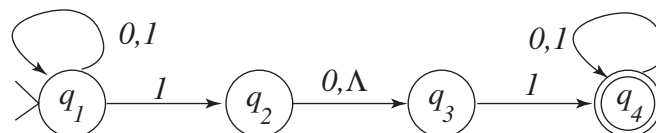


Figura 6.4: Exemplo de autômato finito não determinístico.

Além disto, também é possível afirmar que para todo NFA, existe um DFA semelhante.

O exemplo da Figura 6.4 apresenta a representação gráfica de um NFA. Este autômato guarda muita semelhança com os DFA's já estudados, porém ele apresenta um símbolo novo, Λ , que representa um *símbolo vazio*. Isto quer dizer que a transição do estado q_2 para o estado q_3 pode acontecer por dois motivos: (1) uma transição tradicional que é realizada sempre que for lido um símbolo 0; (2) uma transição imediata que pode ser feita a qualquer momento, bastando que a máquina esteja no estado q_2 , independente se leu ou não um símbolo. Observe que o não determinismo do autômato reside justamente nesta situação. Quando o autômato está no estado q_2 e surge o símbolo 0, ele é obrigado a ir para o estado q_3 . Entretanto, se ele está neste mesmo estado q_2 e surge qualquer outro símbolo diferente de 0, então ele pode, ou não, ir para o estado q_3 .

Os NFA's não foram concebidos para serem utilizados por modelos realistas computacionais, eles são generalizações notacionais dos autômatos finitos. Os NFA's são, em geral, muito menores que os DFA's, além de simplificarem significativamente a descrição dos autômatos. Em um NFA, uma cadeia é aceita se existir alguma maneira de levar o dispositivo de um estado inicial para um estado final, percorrendo as setas rotuladas. Utilizando o exemplo da Figura 6.4, observe o esquema da Figura 6.5 contendo as possíveis configurações que o autômato pode assumir, dado uma string de entrada 010110 (sugere-se que sejam testadas também as strings 010, 101, 0110). Uma análise deste autômato permite afirmar que a linguagem aceita por esta máquina é dada pelo conjunto de strings que contenham as substrings 101 ou 11.

Observe ainda que existem dois percursos que permitem atingir o único estado final desta máquina, no caso o q_4 . Isto significa que existem dois comportamentos

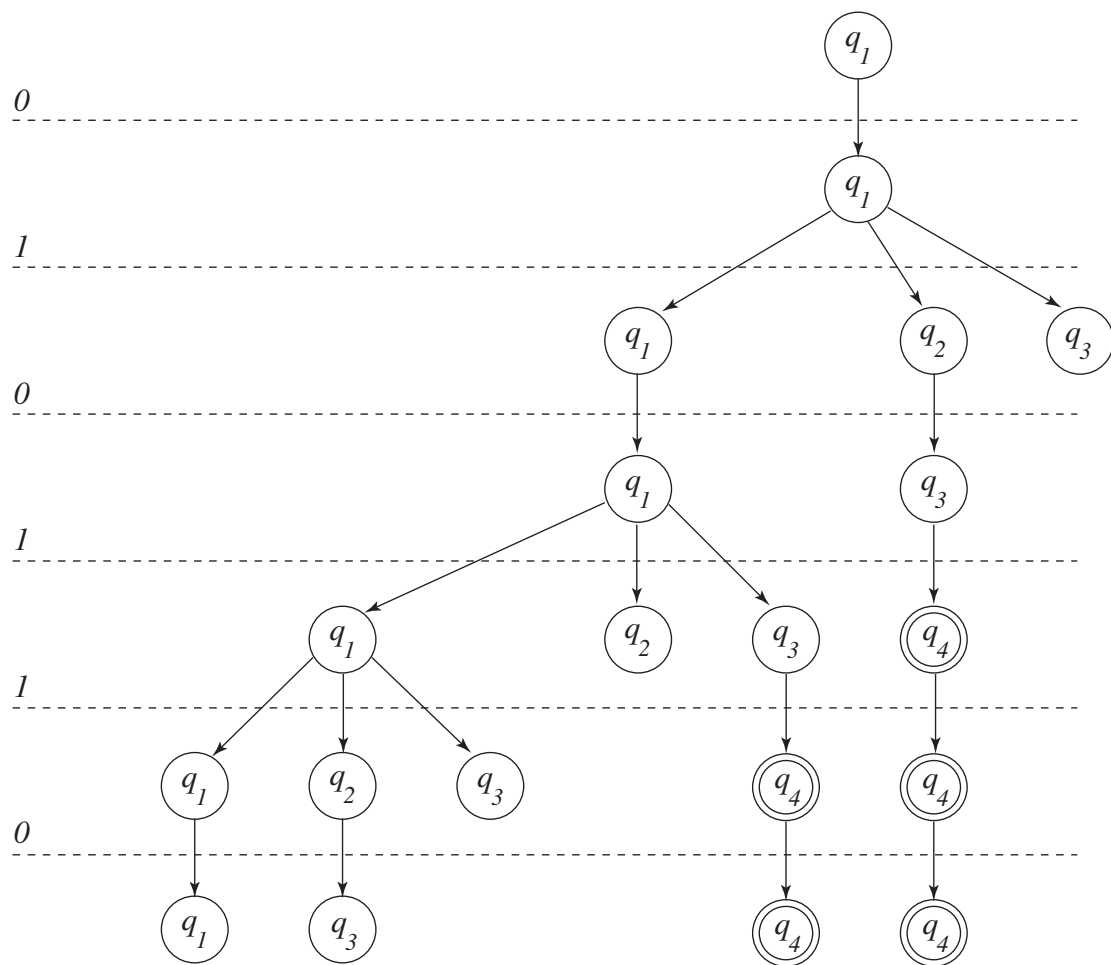


Figura 6.5: Possíveis configurações para o autômato da Figura 6.4.

distintos para a máquina que permitem a aceitação da string 010110. Além disto, também se pode constatar que uma vez o autômato configurado no estado q_1 e lendo um símbolo 1, a máquina pode: (a) ficar em q_1 , devido ao laço existente no próprio estado; (b) seguir para q_2 através da transição existente entre q_1 e q_2 ; (c) ir para q_3 , pois uma vez em q_2 , o autômato pode avançar para q_3 sem consumir uma entrada devido a existência de uma transição para o símbolo Λ .

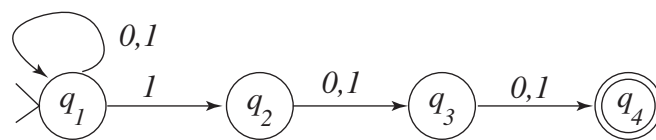
Definição 6.4 Um autômato finito não determinístico deve possuir ao menos uma das seguintes características:

- i) Mais de uma seta rotulada com um mesmo símbolo σ saindo de q ;
- ii) Para um estado e uma determinada entrada pode não haver movimento possível, isto é, $\exists \delta(q, \sigma)$ mas $\nexists \delta(q, \sigma')$;
- iii) Podem existir setas rotuladas Λ indicando que a máquina pode ir para outro estado sem precisar consumir uma entrada.

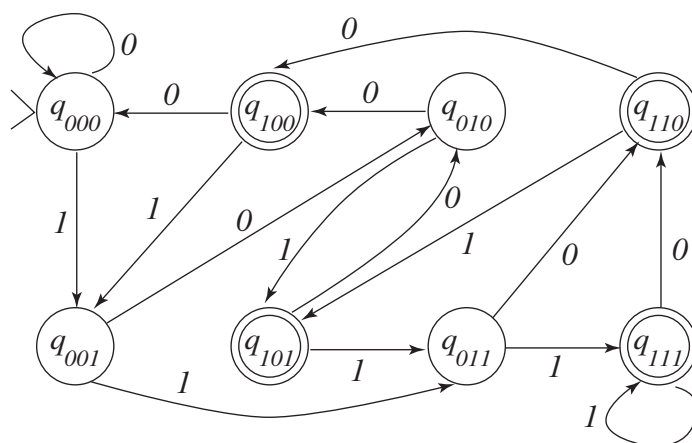
Considere a linguagem formada por strings que contenham o símbolo 1 na terceira posição de trás para frente da string. Existem dois possíveis autômatos para a representação desta linguagem. O autômato não determinístico é bastante simples, sendo ilustrado na Figura 6.6a. A idéia central deste autômato é que ele permanece em q_1 até que adivinhe que faltam apenas três símbolos para o término da string.

O leitor deve se sentir desconfortável com o uso desta adivinhação, haja vista que este procedimento não é passível de ser computável. Aqui, fica evidente a noção de que um NFA é uma generalização notacional, e não um modelo realista computacional. Entretanto, este autômato é mais agradável para ser compreendido do que seu dual determinístico, apresentado na Figura 6.6b. Para eliminar este comportamento não computável, o DFA está a todo momento se preparando para caracterizar um símbolo 1 na anti-penúltima posição, sem necessariamente saber se ainda existem símbolos na fita, ou não.

Sob esta ótica, uma tarefa que se torna interessante, e também desejável, é a transformação de algo cuja notação seja mais simples, para algo que possa ser



(a)



(b)

Figura 6.6: Autômato reconhecedor de strings com o símbolo 1 na 3ª posição de trás para frente da string: (a) NFA; (b) DFA.

exeqüível em sistemas computadorizados. A seção a seguir trata justamente desta conversão.

6.3 Conversão de NFA para DFA

Um NFA, ao adivinhar o caminho a ser seguido durante uma execução, exige de um hardware qualquer desempenhar atividades de difícil sistematização. Apesar de serem de mais fácil compreensão e com notação menos densa, os NFAs não são apropriados para serem transformados em programas de computadores. Por este motivo, existe uma necessidade de converter um autômato não determinístico em um determinístico.

O segredo desta conversão é que cada estado de um NFA corresponde a um conjunto de estados do DFA correspondente. Assim, existem cinco propriedades que devem ser seguidas:

1. Se o NFA possui os estados $k = \{q_0, q_1, q_2, \dots\}$ então o conjunto de estados do DFA é dado pelo conjunto potência de k , isto é:
 $k' = \{\emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}, \dots\}$. Desta forma, a complexidade de pior caso para a construção destes estados é dada por $O(2^n)$, onde n é a quantidade de estados do NFA.
2. O estado inicial $s = q_0$ do NFA corresponde a um estado inicial $s' = \{q_0\}$ no DFA.
3. Se q_j é um estado final do NFA então todos os conjuntos de estados pertencentes à k' que contém q_j serão estados finais no DFA.
4. A função de transição passa a ser uma função de transição estendida δ^* onde

$$\delta^*(q, \sigma) = \left\{ \text{todos os estados atingíveis partindo de } q \text{ e lendo } \sigma \right\}$$
5. Estados não atingíveis do DFA podem ser ignorados.

Considere o exemplo da Figura 6.7. Neste exemplo sugere-se um autômato não determinístico que é capaz de aceitar strings que contenham em seu término a

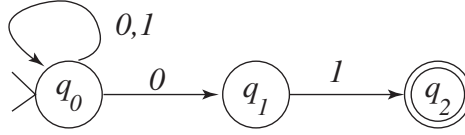


Figura 6.7: Autômato não determinístico reconhecedor de strings contendo 01 no final.

seqüência 01. Uma vez que o NFA possui apenas 3 estados (q_0, q_1, q_2) , temos que o DFA irá possuir $2^3 = 8$ estados, conforme apresentado na representação da função de transição estendida a seguir.

q	0	1
\emptyset	\emptyset	\emptyset
q_0	q_0, q_1	q_0
q_1	\emptyset	q_2
q_2	\emptyset	\emptyset
q_0, q_1	q_0, q_1	q_0, q_2
q_0, q_2	q_0, q_1	q_0
q_1, q_2	\emptyset	q_2
q_0, q_1, q_2	q_0, q_1	q_0, q_2

Segundo que foi apresentado na quarta propriedade da conversão de NFA para DFA, a função de transição estendida δ^* , onde $\delta^*(q, \sigma)$ é dada por todos estados que são factíveis de serem atingidos no NFA, estando no estado q e lendo um símbolo σ , que neste caso pode ser 0 ou 1. Consideremos o caso trivial, o estado \emptyset . Não importa se o autômato da Figura 6.7 lê 0 ou 1, este estado não é previsto, logo concluímos que o melhor a ser feito é manter a máquina configurada no próprio estado \emptyset . Em seguida, considere o caso seguinte, onde a máquina está no estado q_0 . Nesta situação, ao ler um símbolo 0 a máquina tem duas opções: continuar em q_0 ou ir para q_1 . Desta forma dizemos a máquina irá para a junção destes dois estados, isto é, $\{q_0, q_1\}$. Por outro lado, ainda considerando que a máquina esteja em q_0 , e que ela leia o símbolo 1, então observamos que a única opção é se manter em $\{q_0\}$.

Um caso mais interessante é quando o autômato encontra-se no estado $\{q_0, q_1\}$. A montagem da função de transição estendida se dá considerando os estados q_0

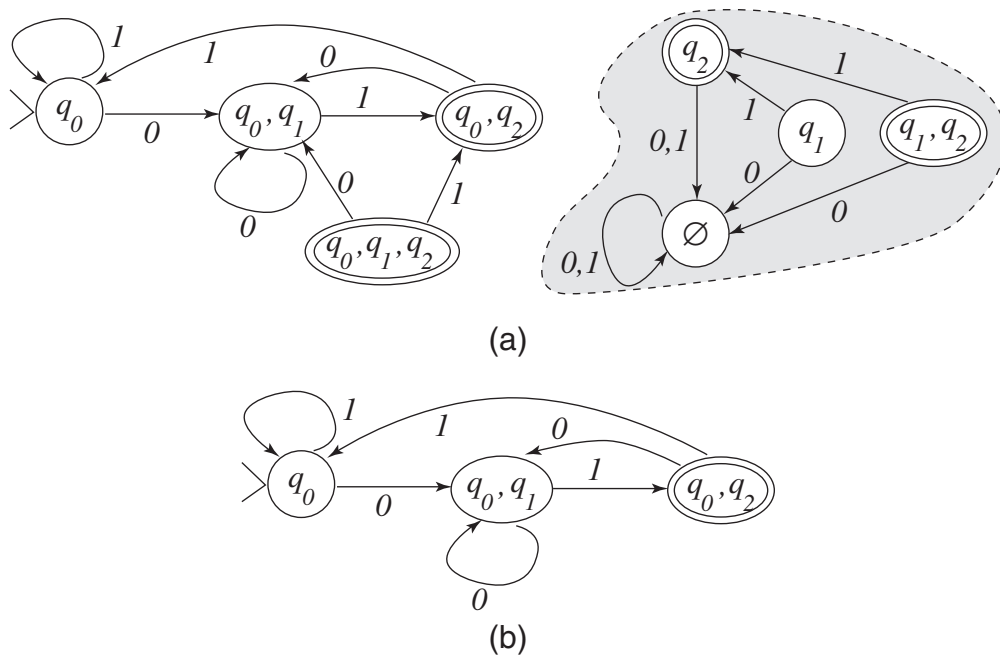


Figura 6.8: Autômato determinístico reconhecedor de strings contendo 01 no final: (a) representação com estados inatingíveis; (b) representação final.

e q_1 separadamente. De acordo com o que já foi constatado para q_0 , temos que $\delta^*(q_0, 0) = \{q_0, q_1\}$. Analogamente podemos dizer que estando em q_1 e lendo o símbolo 0 não há nada o que ser feito a não ser ficar em q_1 . Assim, a união de todos estes possíveis estados atingíveis é dada por $\{q_0, q_1\} \cup \{q_1\} = \{q_0, q_1\}$.

A Figura 6.8(a) apresenta a função de transição estendida representada de modo gráfico, que já representa um esboço do DFA desejado. Tal como descrito nas propriedades da conversão de um NFA em DFA, o estado inicial do autômato continua sendo q_0 , enquanto que os estados finais são todos aqueles que contêm q_2 , isto é, $\{q_2\}$, $\{q_0, q_2\}$, $\{q_1, q_2\}$ e $\{q_0, q_1, q_2\}$.

Nesta forma de visualização também se pode executar a tarefa de ignorar os estados não atingíveis do DFA. Observe que se o autômato se inicia no estado q_0 , não há como alcançar o trecho em evidência dado pelos estados $\{\emptyset\}$, $\{q_1\}$, $\{q_2\}$ e $\{q_1, q_2\}$. Sob esta ótica, o trecho em questão será eliminado. Além disto, o estado $\{q_0, q_1, q_2\}$ também é um estado que nunca pode ser alcançado, dado que só existem setas de transição saindo dele. Assim, este estado também será eliminado. A Figura

6.8(b) apresenta o autômato finito determinístico em sua versão final.

6.4 Minimização de Autômatos

Neste momento, parece sugestível que não há um algoritmo que permita a construção de um autômato qualquer que implemente a linguagem L a ser representada. A construção de autômatos envolve um processo eminentemente criativo, que tende a ser facilitado com a prática e o estudo. Durante o processo de criação, o projetista do autômato, cria um arranjo de estados e transições, sem que necessariamente esteja preocupado com a quantidade dos mesmos. Em geral, problemas desta natureza, objetivando uma otimização, são deixados para uma etapa posterior.

Na seção 6.3 foi apresentado o processo de conversão de um NFA em um DFA. Neste processo, observa-se que também não há uma preocupação formal com a redução da quantidade de estados. Existe apenas um ensaio de otimização quando se efetua a verificação de estados inatingíveis e posteriormente realiza-se a eliminação.

Poder calcular o autômato mínimo é uma questão importante na construção de ferramentas baseadas em autômatos finitos tais como: protocolos de comunicação, processamento de texto, análise de imagens, lingüística computacional, entre outras. Isto reduz a complexidade da engenharia do software, e em última medida, torna o processamento da aplicação mais leve, e seu custo financeiro, menor.

Um autômato com m estados e que aceite uma linguagem L é dito mínimo quando, para todo DFA de n estados que aceite a mesma linguagem L , se a relação entre estas quantidades de estados é dada por $m < n$. Dois estados q e q' são ditos equivalentes (\equiv) se $\forall \sigma, \delta(q, \sigma) \in F \Leftrightarrow \delta(q', \sigma) \in F$. Alternativamente, diz-se que dois estados são distintos ($\not\equiv$) se $\exists \sigma, \delta(q, \sigma) \in F \wedge \delta(q', \sigma) \notin F$, ou vice-versa.

Observe o autômato da Figura 6.9(a), cuja linguagem aceita é dada por $(a, b)(a, b)^*$. Analisando este autômato, percebe-se que a transição de q_0 para q_2 aponta para a construção de strings do tipo $a(a, b)^*$. Ainda nesta mesma figura, as transições que passam pelo estado q_1 permitem a construção de strings do tipo $b(a, b)^*$ de um modo pouco convencional. Isto ocorre porque a passagem por q_1

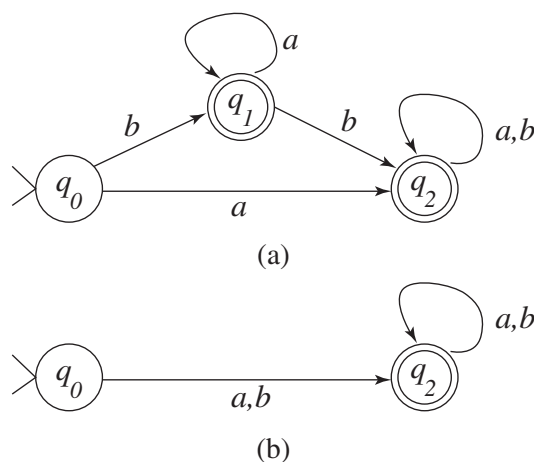


Figura 6.9: Exemplo de minimização: (a) Autômato finito determinístico não minimizado; (b) DFA minimizado.

força a existência de um símbolo b no início da string. Depois disso, é possível fazer um a^*b , e depois em q_2 se pode combinar a ou b como se bem entender. Note que existem dois comportamentos distintos nesta passagem: (1) quando são repetidos os vários símbolos a ; (2) quando não se usa o símbolo a e efetua-se a transição de q_1 para q_2 , através do símbolo b . O efeito poderia ser reproduzido se fosse criada a transição $\delta(q_0, b) = q_2$, combinada com a transição já existente de q_0 para q_2 . Desta forma, toda a parte de cima do autômato poderia ser descartada, produzindo um autômato menor, conforme a Figura 6.9(b).

A minimização descrita para a Figura 6.9 torna-se proibitiva quando os autômatos a serem minimizados ficam maiores, pois a tarefa de determinação do que minimizar passa por um processo de escolha às vezes pouco sistemático, e dependente da experiência do projetista. O processo informal de minimização pode ser descrito em dois passos: (1) identificar todos os estados não equivalentes, chamados de distintos; (2) combinar os estados restantes, os estados equivalentes, respeitando as transições necessárias para que a linguagem L seja preservada.

Sob esta ótica, dois estados equivalentes serão aqueles que executam tarefas semelhantes. Mais especificamente, dois estados q e q' são distintos se, e somente se, estando no estado q e recebendo um símbolo σ , o DFA efetua a transição para um estado de aceitação (estado final), enquanto que estando no estado q' , e recebendo

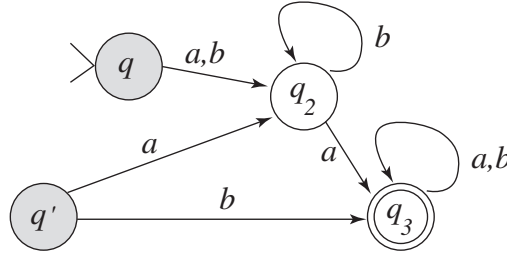


Figura 6.10: Exemplo de não equivalência entre estados.

o mesmo símbolo σ , o DFA efetua a transição para um estado de não aceitação. Observe o exemplo da Figura 6.10, o símbolo a não permite fazer a distinção entre os estados q e q' pois ambos conduzem o autômato para um estado de não aceitação, isto é, que não pertence a F . Entretanto, o recebimento de um símbolo b , estando no estado q conduz para o estado de não aceitação q_2 , enquanto que a partir do estado q' atinge-se o estado final q_3 . Esta distinção entre os estados de chegada torna os estados q e q' distintos.

Outro conceito importante é o de particionamento de um conjunto, que permite dividir um conjunto em subconjuntos sem sobreposição. Suponha um conjunto de estados $k = \{q_0, q_1, q_2, q_3, q_4, q_5\}$. Uma partição hipotética deste conjunto pode ser dada por $P_i = \{\{q_0, q_1\}, \{q_2, q_3, q_4\}, \{q_5\}\}$. Esta partição, por sua vez, pode ser operada por um operador de refinamento que divide os subconjuntos de P_i em novos subconjuntos, também sem sobreposição, como por exemplo $P_{i+1} = \{\{q_0\}, \{q_1\}, \{q_2, q_3, q_4\}, \{q_5\}\}$. Observe que não é possível efetuar o particionamento de um conjunto unitário, pois o conjunto $\{\}$ não é considerado como um grupo de partição.

O particionamento inicial de todos conjunto de estados é dado por $P_0 = \{k - F, F\}$. Desta forma, na Figura 6.11(a) são apresentados cinco estado, cujo particionamento inicial é dado por $P_0 = \{\{q_0, q_1, q_4\}, \{q_2, q_3\}\}$, onde $G_1 = \{q_0, q_1, q_4\}$

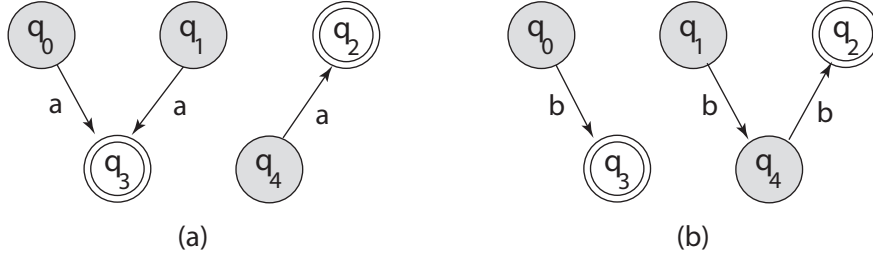


Figura 6.11: Exemplo de particionamento.

e $G_2 = \{q_2, q_3\}$. Considere o grupo $G_1 = \{q_0, q_1, q_4\}$ sendo submetido a entrada a :

$$\delta(q_0, a) = q_3 \in G_2$$

$$\delta(q_1, a) = q_3 \in G_2$$

$$\delta(q_4, a) = q_2 \in G_2$$

para a entrada a , todos os estados pertencentes a G_1 são encaminhados para outros estados pertencentes a um mesmo grupo (G_2 no caso), e por isso não é possível fazer uma distinção entre eles. Entretanto, considerando a Figura 6.11(b) tem-se:

$$\delta(q_0, b) = q_3 \in G_2$$

$$\delta(q_1, b) = q_4 \in G_1$$

$$\delta(q_4, b) = q_2 \in G_2$$

o estado q_1 é distinto dos estados q_0 e q_4 já que, cada qual são conduzidos a grupos distintos. Desta forma a partição P_0 pode ser refinada para $P_1 = \{\{q_0, q_4\}, \{q_1\}, \{q_2, q_3\}\}$.

Existem duas grandes famílias de algoritmos para minimização. A primeira utiliza uma série de refinamentos das partições do conjunto de estados, e a segunda utiliza uma seqüência de fusões de estados. Os algoritmos de Hopcroft e de Moore pertencem a primeira família, enquanto que o algoritmo de Revuz pertence a segunda. Berstel *et alli* [86] fornecem uma excelente comparação entre os algoritmos de Hopcroft e Moore.

A seguir é descrito o algoritmo de Hopcroft, que é um dos algoritmos mais velozes para efetuar minimização ($O(m n \log n)$, m é o número de símbolos do alfabeto e n a quantidade de estados) [1] em casos gerais. Este algoritmo foi estendido por

Béal e Crochemore [87], Valmari e Lehtinen [88], e otimizado para casos específicos. A seguir é apresentado o algoritmo 6.1 conceitualmente na sua versão original, mas adaptado para tornar-se mais legível.

Data: DFA $M = \langle k, \Sigma, \delta, s, F \rangle$ a ser minimizado

Result: Particionamento P correspondente a minimização

```

/* Cria a partição inicial                                     */
1  $P = \{F, k - F\};$ 
2 while true do
    /* Início do refinamento da partição  $P_{novo}$              */
3      $P_{novo} = P;$ 
4     foreach grupo  $G$  em  $P_{novo}$  do
        // Quebrar  $G$  em subgrupos
5         foreach  $\sigma \in \Sigma$  do
6             Determinar a que grupo pertence  $\delta(q_i^G, \sigma);$ 
7             Reagrupar os estados  $q_i^G$  associados ao mesmo grupo de
                chegada ;
8             Dar novas etiquetas para os grupos no reagrupamento
                ( $G', G'', \dots$ );
9             Remover o grupo  $G$  do conjunto  $P_{novo};$ 
10            Adicionar os novos grupos  $G', G'', \dots$  ao conjunto  $P_{novo}$  e
                interromper o loop de  $\sigma \in \Sigma;$ 
11        end
12    end
    /* Fim do refinamento da partição  $P_{novo}$                  */
13    if  $P_{novo} == P$  then
14        break;
15    end
16     $P = P_{novo};$ 
17 end

```

Algoritmo 6.1: Minimização de um DFA $M = \langle k, \Sigma, \delta, s, F \rangle$, segundo Hopcroft [1].

A proposta do algoritmo de minimização 6.1 é realizar sucessivos refinamen-

tos do particionamento P até o processo de refinamento não efetue mais nenhuma modificação do conjunto particionado. Quando isto ocorre o algoritmo é finalizado (linhas 13 e 14).

Ao término do algoritmo, e caso seja cabível a minimização, o conjunto P será composto por novos grupos. Cada um destes grupos corresponderá a um novo estado do autômato minimizado, sendo que o grupo que contiver um estado inicial corresponderá ao estado inicial do novo autômato, e os grupos que contiverem estados finais serão convertidos em estados finais no novo autômato. Dentro de cada grupo, deve-se escolher um dos estados para ser mantido, enquanto os restantes são descartados. As arestas de entrada e saída deste estado devem ser preservadas, eventualmente sendo necessário adaptá-las para um laço de loop. Em seguida devem ser removidos quais estados inatingíveis a partir do estado inicial.

Para exemplificar o funcionamento do algoritmo, retome o exemplo da Figura 6.9(a). O passo a passo do algoritmo é apresentado a seguir:

linha	:	operação
1	:	$P = \{\{q_1, q_2\}, \{q_0\}\}$
3	:	$P_{novo} = \left\{ \underbrace{\{q_1, q_2\}}_{G_1}, \underbrace{\{q_0\}}_{G_2} \right\}$
4	:	$G_1 = \{q_1, q_2\}$
5	:	$\sigma = a$
6	:	$\delta(q_1, a) = q_1 \in G_1$
	:	$\delta(q_2, a) = q_2 \in G_1$
7	:	Não há reagrupamento a ser feito
8, 9, 10	:	Nada a fazer
5	:	$\sigma = b$
6	:	$\delta(q_1, b) = q_2 \in G_1$
	:	$\delta(q_2, b) = q_2 \in G_1$
7	:	Não há reagrupamento a ser feito
8, 9, 10	:	Nada a fazer

Neste momento chega ao fim o loop referente a linha 5 do algoritmo. O próximo

passo a ser executado é a nova iteração do loop da linha 4, onde toma-se como novo grupo o $G_2 = \{q_0\}$. Observe que o trecho de código entre as linhas 3 e 12 se propõe a fazer um refinamento da partição P_{novo} . Isto, em última análise, significa particionar o grupo G_2 . Entretanto, este mesmo grupo G_2 é um conjunto unitário, e não é possível mais particioná-lo, sugerindo que as próximas operações são desnecessárias. Contudo, por completude do exemplo proposto, e para demonstrar a insensibilidade desta situação por parte do algoritmo, será dada continuidade ao passo a passo do algoritmo.

linha	:	operação
4	:	$G_2 = \{q_0\}$
5	:	$\sigma = a$
6	:	$\delta(q_0, a) = q_2 \in G_1$
7	:	Não há reagrupamento a ser feito
8, 9, 10	:	Nada a fazer
5	:	$\sigma = b$
6	:	$\delta(q_0, b) = q_1 \in G_1$
7	:	Não há reagrupamento a ser feito
8, 9, 10	:	Nada a fazer

De fato, esta última iteração não tinha como acrescentar nenhuma modificação ao P_{novo} . Por fim, o algoritmo verifica que de fato $P_{novo} == P$ (linha 13) e interrompe sua execução. Como resultado, é obtido o conjunto de particionamento $P = \{\{q_1, q_2\}, \{q_0\}\}$, indicando que q_1 e q_2 são equivalentes.

Dando continuidade a etapa posterior ao algoritmo, observa-se que não há estado a ser escolhido no grupo $G_2 = \{q_0\}$ porque a única escolha possível, é o único elemento do grupo, isto é, q_0 . Já no grupo $G_1 = \{q_1, q_2\}$ deve-se escolher um dos estados para ser mantido no autômato minimizado, enquanto os restantes devem ser descartados. Neste caso, optou-se por escolher, aleatoriamente, o estado q_0 e descartar o estado q_1 . As transições que precisam ser adaptadas são aquelas que saem de q_0 e q_1 e seguem para q_2 . Por fim, ao analisar o autômato restante, percebe-se que não existem estados inatigíveis, resultando na minimização da Figura

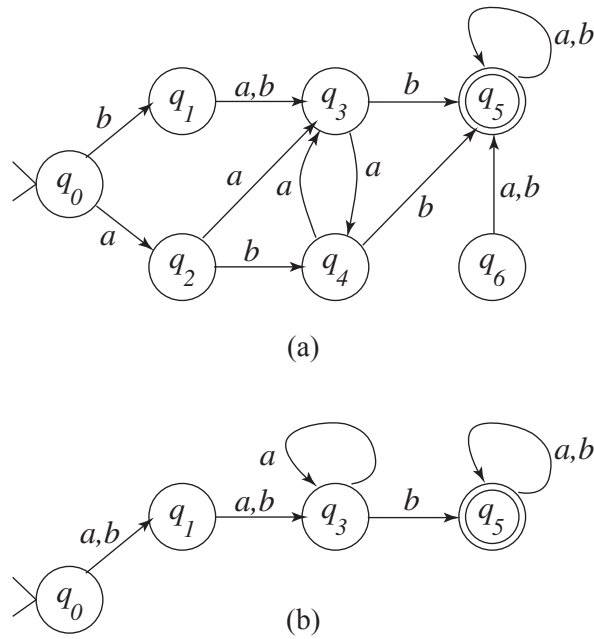


Figura 6.12: Minimização de DFA: (a) autômato original; (b) autômato minimizado.

6.9(b).

O objetivo do exemplo apresentado foi familiarizar o leitor com a mecânica do funcionamento do Algoritmo 6.1. A Figura 6.12(a) apresenta um outro DFA que será usado como ilustração adicional do processo de minimização. Trata-se de um exemplo um pouco mais longo, onde irão surgir algumas situações que ficaram escondidas no exemplo anterior. Neste sentido, um primeiro momento da execução do algoritmo pode ser descrito como se segue:

linha : operação

$$1 : P = \{\{q_0, q_1, q_2, q_3, q_4, q_6\}, \{q_5\}\}$$

$$3 : P_{novo} = \left\{ \underbrace{\{q_0, q_1, q_2, q_3, q_4, q_6\}}_{G_1}, \underbrace{\{q_5\}}_{G_2} \right\}$$

$$4 : G_1 = \{q_0, q_1, q_2, q_3, q_4, q_6\}$$

$$5 : \sigma = a$$

$$6 : \delta(q_0, a) = q_2 \in G_1$$

$$\delta(q_1, a) = q_3 \in G_1$$

$$\delta(q_2, a) = q_3 \in G_1$$

$$\delta(q_3, a) = q_4 \in G_1$$

$$\delta(q_4, a) = q_3 \in G_1$$

$$\delta(q_6, a) = q_5 \in G_2$$

$$7, 8 : G_3 = \{q_0, q_1, q_2, q_3, q_4\}, G_4 = \{q_6\}$$

$$9, 10 : P_{novo} = \left\{ \underbrace{\{q_0, q_1, q_2, q_3, q_4\}}_{G_3}, \underbrace{\{q_6\}}_{G_4}, \underbrace{q_5}_{G_2} \right\}$$

Observe que nesta etapa do algoritmo foi necessário substituir o grupo G_1 pelo grupos G_3 e G_4 . Isso provocou uma alteração no conjunto de particionamento P_{novo} , o que vai obrigar que o loop da linha 4 recomece a varredura sobre P_{novo} . Deste

modo, o próximo grupo a ser testado será G_3 .

linha : operação

$$4 : G_3 = \{q_0, q_1, q_2, q_3, q_4\}$$

$$5 : \sigma = a$$

$$6 : \delta(q_0, a) = q_2 \in G_3$$

$$\delta(q_1, a) = q_3 \in G_3$$

$$\delta(q_2, a) = q_3 \in G_3$$

$$\delta(q_3, a) = q_4 \in G_3$$

$$\delta(q_4, a) = q_3 \in G_3$$

$$7 : \text{Não há reagrupamento a ser feito}$$

$$8, 9, 10 : \text{Nada a fazer}$$

$$5 : \sigma = b$$

$$6 : \delta(q_0, b) = q_1 \in G_3$$

$$\delta(q_1, b) = q_3 \in G_3$$

$$\delta(q_2, b) = q_4 \in G_3$$

$$\delta(q_3, b) = q_5 \in G_2$$

$$\delta(q_4, b) = q_5 \in G_2$$

$$7, 8 : G_5 = \{q_0, q_1, q_2\}, G_6 = \{q_3, q_4\}$$

$$9, 10 : P_{\text{nov}} = \left\{ \underbrace{\{q_0, q_1, q_2\}}_{G_5}, \underbrace{\{q_3, q_4\}}_{G_6}, \underbrace{\{q_6\}}_{G_4}, \underbrace{\{q_5\}}_{G_2} \right\}$$

Novamente houve uma alteração no conjunto de particionamento P_{nov} , o que vai

provocar um recomeço da varredura sobre P_{novo} .

linha : operação

$$4 : G_5 = \{q_0, q_1, q_2\}$$

$$5 : \sigma = a$$

$$6 : \delta(q_0, a) = q_2 \in G_5$$

$$\delta(q_1, a) = q_3 \in G_6$$

$$\delta(q_2, a) = q_3 \in G_6$$

$$7, 8 : G_7 = \{q_0\}, G_8 = \{q_1, q_2\}$$

$$9, 10 : P_{novo} = \left\{ \underbrace{\{q_0\}}_{G_7}, \underbrace{\{q_1, q_2\}}_{G_8}, \underbrace{\{q_3, q_4\}}_{G_6}, \underbrace{\{q_6\}}_{G_4}, \underbrace{\{q_5\}}_{G_2} \right\}$$

Nesta iteração, ocorre outra alteração no conjunto de particionamento P_{novo} , fazendo com que o loop da linha 4 recomece uma nova varredura sobre P_{novo} .

linha : operação

$$4 : G_7 = \{q_0\}$$

$$5 : \sigma = a$$

$$6 : \delta(q_0, a) = q_2 \in G_8$$

7 : Não há reagrupamento a ser feito

8, 9, 10 : Nada a fazer

$$5 : \sigma = b$$

$$6 : \delta(q_0, b) = q_1 \in G_8$$

7 : Não há reagrupamento a ser feito

8, 9, 10 : Nada a fazer

O grupo a ser avaliado foi o G_7 . Nessa avaliação já era esperado que nada acontecesse

pois o conjunto é unitário. Em seguida, o próximo grupo a ser testado é o G_8 .

linha	: operação
4	: $G_8 = \{q_1, q_2\}$
5	: $\sigma = a$
6	: $\delta(q_1, a) = q_3 \in G_6$
	: $\delta(q_2, a) = q_3 \in G_6$
7	: Não há reagrupamento a ser feito
8, 9, 10	: Nada a fazer
5	: $\sigma = b$
6	: $\delta(q_1, b) = q_3 \in G_6$
	: $\delta(q_2, b) = q_4 \in G_6$
7	: Não há reagrupamento a ser feito
8, 9, 10	: Nada a fazer

O grupo G_8 era candidato a sofrer alteração, porém isto não aconteceu. Em seguida, o terceiro grupo a ser avaliado é o G_6 .

linha	: operação
4	: $G_6 = \{q_3, q_4\}$
5	: $\sigma = a$
6	: $\delta(q_3, a) = q_4 \in G_6$
	: $\delta(q_4, a) = q_3 \in G_6$
7	: Não há reagrupamento a ser feito
8, 9, 10	: Nada a fazer
5	: $\sigma = b$
6	: $\delta(q_3, b) = q_5 \in G_2$
	: $\delta(q_4, b) = q_5 \in G_2$
7	: Não há reagrupamento a ser feito
8, 9, 10	: Nada a fazer

A seguir o grupo G_4 , tal como aconteceu anteriormente, é um conjunto unitário, e

portanto é esperado que não haja alteração.

linha	:	operação
4	:	$G_4 = \{q_0\}$
5	:	$\sigma = a$
6	:	$\delta(q_0, a) = q_2 \in G_8$
7	:	Não há reagrupamento a ser feito
8, 9, 10	:	Nada a fazer
5	:	$\sigma = b$
6	:	$\delta(q_0, b) = q_1 \in G_8$
7	:	Não há reagrupamento a ser feito
8, 9, 10	:	Nada a fazer

Analogamente, o grupo G_2 também é um conjunto unitário e portanto não haverá alteração.

linha	:	operação
4	:	$G_2 = \{q_5\}$
5	:	$\sigma = a$
6	:	$\delta(q_5, a) = q_5 \in G_2$
7	:	Não há reagrupamento a ser feito
8, 9, 10	:	Nada a fazer
5	:	$\sigma = b$
6	:	$\delta(q_5, b) = q_5 \in G_2$
7	:	Não há reagrupamento a ser feito
8, 9, 10	:	Nada a fazer

Neste momento, o algoritmo segue um fluxo de execução, que até então não havia sido percorrido. Trata-se da verificação da condição de parada do algoritmo.

linha	:	operação
13	:	P_{novo} é diferente de P
14	:	$P = \{\{q_0\}, \{q_1, q_2\}, \{q_3, q_4\}, \{q_6\}, \{q_5\}\}$

A condição de parada não foi satisfeita, e em seguida o algoritmo retorna para a linha 3 de forma a fazer uma nova iteração de refinamento. Nesta nova tentativa, P_{novo} não será mais refinado e consequentemente não seria modificado (deixa-se esta verificação a cargo do leitor). Assim, quando é novamente testada a equivalência entre P e P_{novo} , o resultado é verdadeiro e o algoritmo encerra (linha 14).

Desta forma, ao término da execução do algoritmo é obtido o particionamento $P = \{\{q_0\}, \{q_1, q_2\}, \{q_3, q_4\}, \{q_6\}, \{q_5\}\}$, indicando que $q_1 \equiv q_2$ e $q_3 \equiv q_4$. Estas equivalências permitem remodelar o DFA da Figura 6.12(a) para aquele apresentado na Figura 6.12(b). Observe ainda que a eliminação do estado q_6 não foi resultado da minimização, mas sim da percepção de que ele é um estado inatingível.

6.5 Conclusões e leituras recomendadas

Turing propôs um modelo abstrato de computação, conhecido como máquina de Turing, com o objetivo de explorar os limites da capacidade de expressar soluções de problemas. Sob esta ótica, a máquina de Turing é uma proposta de definição formal da noção intuitiva de algoritmo. Por conseguinte, trata-se de uma tentativa de responder ao questionamento sobre o que pode, e o que *não* pode ser computado.

Neste capítulo foram apresentados modelos matemáticos diferentes que executam tarefas neste contexto: decidir, semi-decidir, reconhecer, construir funções. A adição de certos aspectos (quantidade de fitas, leitoras e acesso aleatório) não aumenta o conjunto de tarefas que podem ser executadas por uma máquina simples de Turing. A capacidade de computação representada pela máquina de Turing é o limite máximo que pode ser atingido por qualquer dispositivo de computação. Qualquer outra forma de expressar algoritmos terá, no máximo, a mesma capacidade computacional da máquina de Turing.

6.6 Exercícios

6.1 Construir um autômato finito determinístico que reconheça as linguagens a seguir:.

- (a) $L(M) = \{w \in \{a, b\}^* \mid \text{cada } a \text{ em } w \text{ é imediatamente precedido por um } b\}$
- (b) $L(M) = \{w \in \{a, b\}^* \mid w \text{ tenha } abab \text{ como substring}\}$
- (c) $L(M) = \{w \in \{a, b\}^* \mid w \text{ tem dois } a\text{'s consecutivos ou dois } b\text{'s consecutivos}\}$
- (d) $L(M) = \{w \in \{a, b\}^* \mid w \text{ não tem } aa \text{ nem } bb \text{ como substrings}\}$
- (e) $L(M) = \{w \in \{a, b\}^* \mid w \text{ é uma string com número ímpar de } b\text{'s}\}$
- (f) $L(M) = \{w \in \{a, b\}^* \mid w \text{ contém um número ímpar de } a\text{'s e ímpar de } b\text{'s em qualquer ordem}\}$
- (g) $L(M) = \{w \in \{a, b\}^* \mid w \text{ tem } ab \text{ e } ba \text{ como substrings}\}$
- (h) $L(M) = \{w \in \{a, b\}^* \mid w \text{ contém um número ímpar de } a\text{'s e par de } b\text{'s em qualquer ordem}\}$
- (i) $L(M) = \{w \in \{a, b\}^* \mid w \text{ inicia e termina com o mesmo símbolo}\}$

6.2 Descreva a linguagem aceita por cada um dos autômatos da Figura 6.13.

6.3 Construa um autômato finito determinístico cujo alfabeto são os dígitos decimais $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ tal que os strings sejam divisíveis por três.

6.4 Construa um autômato finito determinístico cujo alfabeto são os dígitos decimais $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ tal que os strings sejam divisíveis por seis.

6.5 Construa um autômato finito determinístico que reconheça se um número binário é divisível por três. A leitura do número começa pelo bit mais significativo (MSB) e termina no bit menos significativo (LSB).

6.6 Construa um autômato finito determinístico que reconheça um número binário cujo divisão por cinco resulte em resto dois. A leitura do número começa pelo bit mais significativo (MSB) e termina no bit menos significativo (LSB).

6.7 Construa um autômato finito determinístico que reconheça se um número binário é divisível por cinco. A leitura do número começa pelo bit mais significativo (MSB) e termina no bit menos significativo (LSB).

6.8 Construa um autômato finito determinístico sobre o alfabeto $\Sigma = \{0, 1\}$ que aceite strings que terminem com 01.

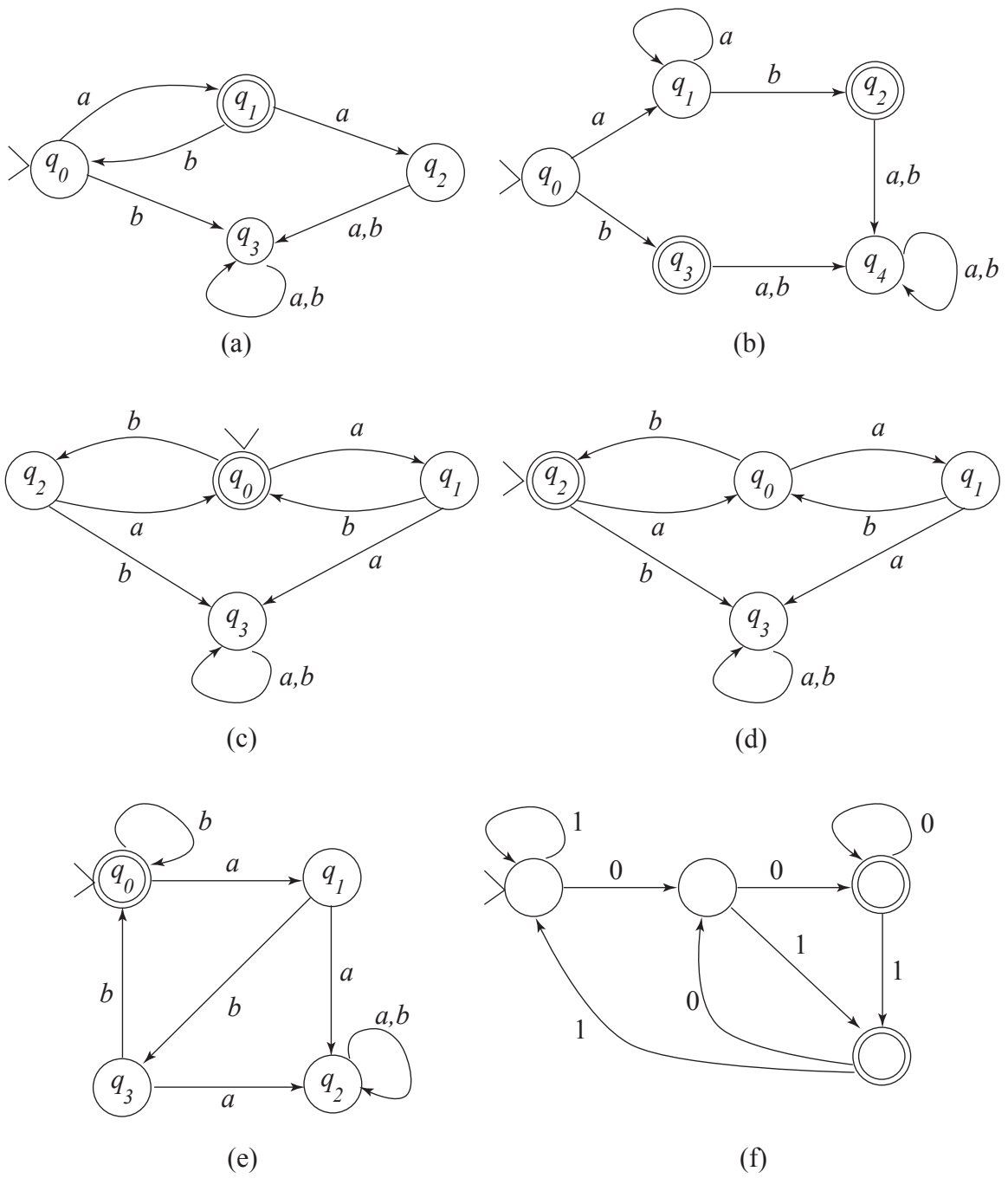


Figura 6.13: Autômatos do Exercício 6.2.

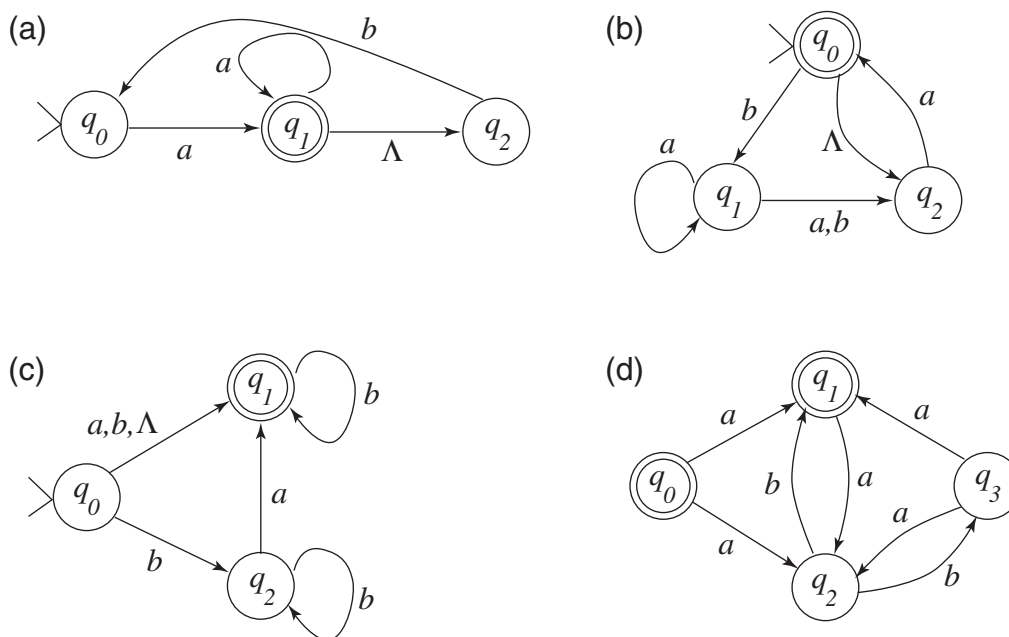


Figura 6.14: Autômatos do Exercício 6.13.

6.9 Construa um autômato finito determinístico sobre o alfabeto $\Sigma = \{0, 1\}$ que aceite strings que contenham uma quantidade igual de ocorrências de 01 e 10.

6.10 Construa um autômato finito determinístico sobre o alfabeto $\Sigma = \{0, 1\}$ que aceite strings que contenham pelo menos dois 0s e no máximo um 1.

6.11 Construa um autômato finito determinístico sobre o alfabeto $\Sigma = \{0, 1\}$ que aceite strings em que cada 0 do string seja imediatamente precedido e imediatamente seguido por 11.

6.12 Determine o DFA correspondente ao NFA da Figura 6.6(a).

6.13 Determine o DFA correspondente ao NFA da Figura 6.14.

6.14 Determine o DFA correspondente ao NFA da Figura 6.15.

6.15 Determine o DFA correspondente ao NFA da Figura 6.16.

6.16 Realize a minimização dos autômatos da Figura 6.17.

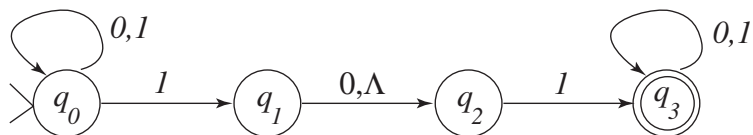


Figura 6.15: Autômato do Exercício 6.14.

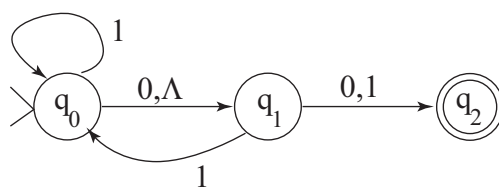


Figura 6.16: Autômato do Exercício 6.15.

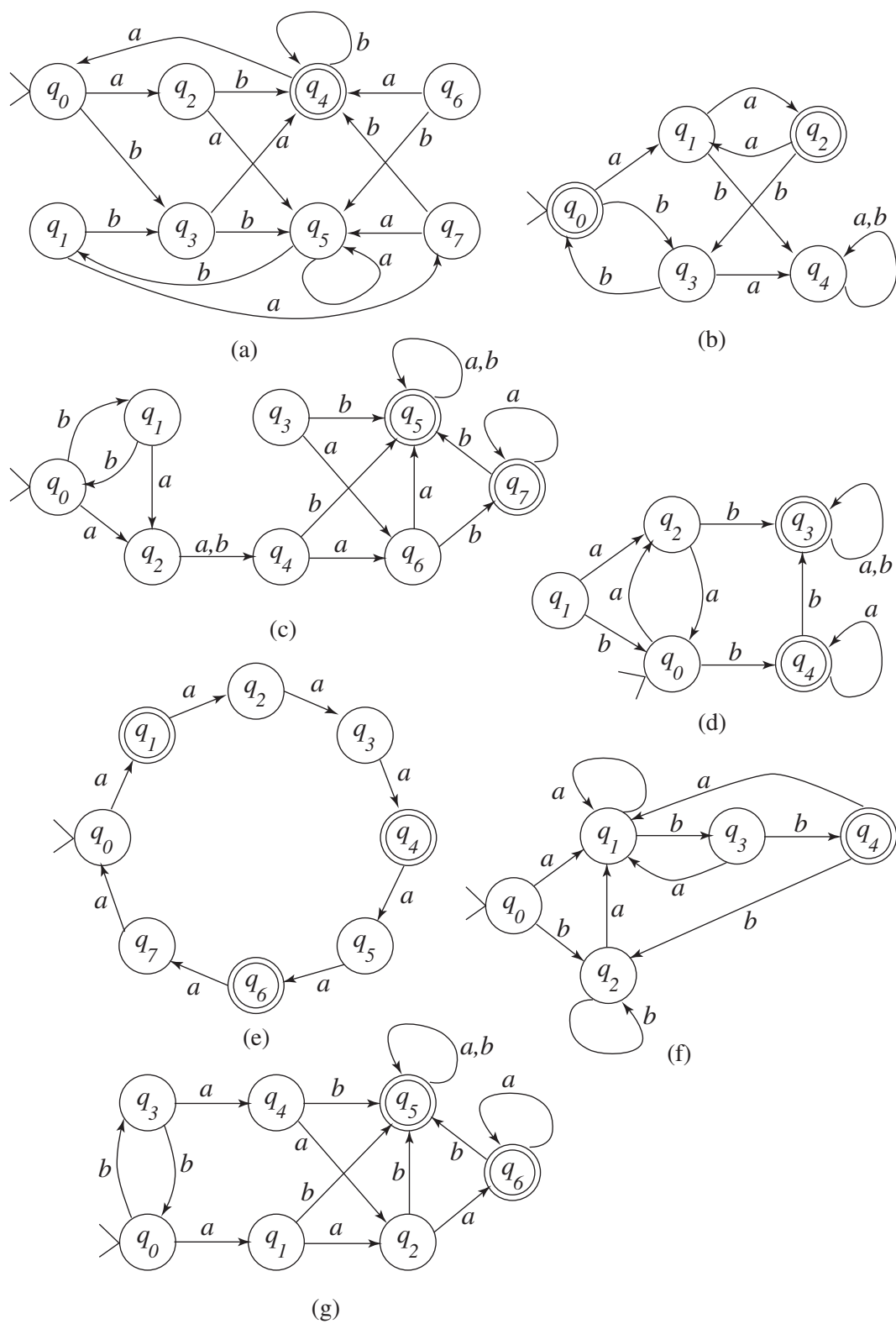


Figura 6.17: Autômato do Exercício 6.16.

Capítulo 7

Máquina de Turing

A Máquina de Turing é universalmente conhecida e aceita como a formalização de um algoritmo. Este mecanismo se assemelha em muito aos computadores atuais, embora tenha sido proposta por Alan Turing ¹ em 1936, muito antes dos primeiros computadores digitais. A Máquina de Turing possui o mesmo poder computacional de qualquer outro computador de propósito geral.

7.1 Máquina Clássica

Este mecanismo é uma evolução da máquina de estados finita, seja ela um DFA ou um NFA. A diferença dela para os autômatos estudados até o presente momento é que a Máquina de Turing pode não só ler uma fita de entrada, mas também pode escrever na mesma. A fita de entrada é limitada à esquerda por um símbolo \triangleright marcador de fim de fita, enquanto que se estende indefinidamente à direita. Sempre

¹Alan Mathison Turing, Ph.D. (1912-1954), inglês nascido no bairro de Paddington em Londres, é considerado um dos fundadores da ciência da computação. Durante a Segunda Guerra Mundial foi o responsável pela quebra da máquina de encriptação alemã chamada Enigma, contribuindo de forma significativa para o desfecho da guerra. Trabalhou com mecânica quântica, probabilidade, lógica, computabilidade, eletrônica, inteligência artificial, redes neurais e biologia. Foi cruelmente massacrado, preso e humilhado pela sociedade inglesa que o impedia de trabalhar devido a sua opção sexual. Um dos maiores vultos da computação suicidou-se em sua casa com uma maça envenenada com cianeto. Atualmente, a *Association for Computing Machinery*, ou ACM, concede anualmente o prêmio Nobel da Computação, conhecido merecidamente como Prêmio Turing.

que o símbolo \triangleright é lido, a máquina move a cabeça leitora obrigatoriamente para a direita. Estes movimentos da cabeça leitora para a direita e esquerda são representados respectivamente por \rightarrow e \leftarrow . Além disto, a fita de entrada pode não ter todos os seus campos preenchidos com símbolos, isto é, ela pode ter espaços em branco. Um espaço em branco, por conveniência, é denotado como sendo o símbolo \sqcup .

Definição 7.1 Uma Máquina de Turing é uma quintupla $M = \langle k, \Sigma, \delta, s, F \rangle$ onde:

- k é o conjunto finito de ESTADOS;
- Σ é um ALFABETO que contém os símbolos \triangleright e \sqcup , mas não contém \rightarrow e \leftarrow ;
- $s \in k$ é o ESTADO INICIAL;
- $F \subseteq k$ é o conjunto de ESTADOS DE PARADA;
- δ é uma FUNÇÃO DE TRANSIÇÃO de $k \times \Sigma$ onde:
 - (a) para todos os $q \in (k - F)$, se $\delta(q, \triangleright) = (p, b)$, então $b = \rightarrow$;
 - (b) para todos os $q \in (k - F)$ e $a \in \Sigma$, $a \neq \triangleright$, se $\delta(q, a) = (p, b)$, então $b \neq \triangleright$.

Definição 7.2 Seja M uma Máquina de Turing. Dizemos que a *configuração* C_1 resulta em C_2 se $C_1 \vdash_M C_2$. Uma *computação* por M é uma seqüência de configurações C_0, C_1, \dots, C_n , para algum $n \geq 0$, tal que $C_0 \vdash_M C_1 \vdash_M \dots \vdash_M C_n$ e dizemos que a computação é de comprimento n , ou que possui n *passos*. Neste caso, escrevemos $C_0 \vdash_M^n C_n$.

Para entender seu funcionamento, considere a Máquina de Turing $M = \langle k, \Sigma, \delta, s, F \rangle$, onde $k = \{q_0, q_1, h\}$, $\Sigma = \{a, \sqcup, \triangleright\}$, $s = q_0$, $F = \{h\}$ e δ é dada conforme a seguir:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_1, \sqcup)
q_0	\sqcup	(h, \sqcup)
q_0	\triangleright	(q_0, \rightarrow)
q_1	a	(q_0, a)
q_1	\sqcup	(q_0, \rightarrow)
q_1	\triangleright	(q_1, \rightarrow)

Considere a fita de entrada $\triangleright a a \square \square \square \square \square \square \square \square \square \square \dots$ um exemplo de alimentação da máquina M . Pode-se afirmar que o processamento desta fita se dará conforme a seguir.

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline \Downarrow & a & a & \square & \square & \square & \square & \square & \square & \square & \square \\ \hline \end{array} \dots, \Downarrow = q_0$$

$$\delta(q_0, \triangleright) = (q_0, \rightarrow)$$

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline \triangleright & \Downarrow & a & \square & \square & \square & \square & \square & \square & \square & \square \\ \hline \end{array} \dots, \Downarrow = q_0$$

$$\delta(q_0, a) = (q_1, \sqcup)$$

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline \triangleright & \Downarrow & \square & a & \square & \square & \square & \square & \square & \square & \square \\ \hline \end{array} \dots, \Downarrow = q_1$$

$$\delta(q_1, \sqcup) = (q_0, \rightarrow)$$

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline \triangleright & \square & \Downarrow & a & \square & \square & \square & \square & \square & \square & \square \\ \hline \end{array} \dots, \Downarrow = q_0$$

$$\delta(q_0, a) = (q_1, \sqcup)$$

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline \triangleright & \square & \Downarrow & \square & \square & \square & \square & \square & \square & \square & \square \\ \hline \end{array} \dots, \Downarrow = q_1$$

$$\delta(q_1, \sqcup) = (q_0, \rightarrow)$$

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline \triangleright & \square & \square & \Downarrow & \square & \square & \square & \square & \square & \square & \square \\ \hline \end{array} \dots, \Downarrow = q_0$$

$$\delta(q_0, \sqcup) = (h, \sqcup)$$

$$\begin{array}{|c|c|c|c|c|c|c|c|c|c|c|} \hline \triangleright & \square & \square & \Downarrow & \square & \square & \square & \square & \square & \square & \square \\ \hline \end{array} \dots, \Downarrow = h$$

Observe que a máquina é iniciada no estado q_0 , com a cabeça leitora \Downarrow na posição de leitura do símbolo \triangleright . Em seguida, de acordo com os símbolos lidos e a função de transição δ , a máquina prossegue com seu processamento até que em algum momento atinge o estado h , um estado de parada.

A representação gráfica utilizada nesta descrição de processamento pode ser substituída por uma representação menos trabalhosa (*estado, situação da fita*). A representação do estágio inicial da máquina do exemplo anterior seria dada por $(q_0, \underline{\triangleright} aa \square \square \square \square \square \square \square \square \dots)$, onde $\underline{\quad}$ representa a posição da cabeça leitora. Esta forma de explicitar um determinado momento da máquina será usada em detrimento da forma de representação gráfica.

Desta forma, a computação da máquina M possui seis passos é dada como se segue:

$$\begin{aligned}
(q_0, \triangleright aa \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup) &\vdash_M (q_0, \triangleright \underline{a} a \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup) \\
&\vdash_M (q_1, \triangleright \sqcup \underline{a} \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup) \\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \underline{a} \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup) \\
&\vdash_M (q_1, \triangleright \sqcup \sqcup \sqcup \underline{a} \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup) \\
&\vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \sqcup \underline{a} \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup) \\
&\vdash_M (h, \triangleright \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup \sqcup)
\end{aligned}$$

Um problema de decisão é uma questão sobre um sistema formal, tal como uma Máquina de Turing, no qual se obtém uma resposta do tipo sim-ou-não, ou ainda, do tipo aceita-rejeita, ou pára-não pára. Observe que no exemplo proposto a máquina M percorre toda a fita de entrada até que finalmente atinge o estado h , definido como um estado final. Ora, se a máquina parou em um estado final, é porque ela aceitou a string a qual ela foi submetida. Neste sentido dizemos que a máquina m efetuou um *decisão*.

Entretanto, não é sempre que uma Máquina de Turing consegue realizar uma decisão. Considere uma nova máquina $M = \langle k, \Sigma, \delta, s, F \rangle$, onde $k = \{q_0, q_1, q_2, h\}$, $\Sigma = \{a, \sqcup, \triangleright\}$, $s = q_0$, $F = \{h\}$ e δ é dada conforme a seguir:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_1, \leftarrow)
q_0	\sqcup	(q_0, \sqcup)
q_0	\triangleright	(q_0, \rightarrow)
q_1	a	(q_2, \sqcup)
q_1	\sqcup	(h, \sqcup)
q_1	\triangleright	(q_1, \rightarrow)
q_2	a	(q_2, a)
q_2	\sqcup	(q_0, \leftarrow)
q_2	\triangleright	(q_2, \rightarrow)

Suponha que $n \geq 0$. Podemos tentar descrever o que M faz quando iniciada na configuração $(q_0, \triangleright \sqcup a^n \underline{a})$. Observe as iterações para alguns valores de n .

$$\begin{aligned}
& \text{Para } n = 0 \rightsquigarrow (q_0, \triangleright \sqcup \underline{a}) \vdash_M (q_1, \triangleright \sqcup \underline{a}) \\
& \vdash_M (h, \triangleright \sqcup a) \leftarrow \text{Pára} \\
& \text{Para } n = 1 \rightsquigarrow (q_0, \triangleright \sqcup \underline{aa}) \vdash_M (q_1, \triangleright \sqcup \underline{aa}) \\
& \vdash_M (q_2, \triangleright \sqcup \sqcup \underline{a}) \\
& \vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup a) \leftarrow \text{Loop} \\
& \text{Para } n = 2 \rightsquigarrow (q_0, \triangleright \sqcup \underline{aaa}) \vdash_M (q_1, \triangleright \sqcup \underline{aaa}) \\
& \vdash_M (q_2, \triangleright \sqcup \sqcup \underline{a}) \\
& \vdash_M (q_0, \triangleright \sqcup \sqcup \underline{a} \sqcup a) \\
& \vdash_M (q_1, \triangleright \sqcup \sqcup \underline{a} \sqcup a) \\
& \vdash_M (h, \triangleright \sqcup \sqcup a \sqcup a) \leftarrow \text{Pára} \\
& \text{Para } n = 3 \rightsquigarrow (q_0, \triangleright \sqcup \underline{aaaa}) \vdash_M (q_1, \triangleright \sqcup \underline{aaaa}) \\
& \vdash_M (q_2, \triangleright \sqcup \sqcup \underline{aa}) \\
& \vdash_M (q_0, \triangleright \sqcup \sqcup \underline{aa} \sqcup a) \\
& \vdash_M (q_1, \triangleright \sqcup \sqcup \underline{aa} \sqcup a) \\
& \vdash_M (q_2, \triangleright \sqcup \sqcup \sqcup \underline{a} \sqcup a) \\
& \vdash_M (q_0, \triangleright \sqcup \sqcup \sqcup \sqcup a \sqcup a) \leftarrow \text{Loop}
\end{aligned}$$

Pode-se concluir que a máquina M aceita strings com n par, e que esta mesma máquina não pára, isto é, rejeita strings com n ímpar. Neste caso em que a máquina ora apresenta um comportamento de decisão, ora outro de não decisão, diz que ela faz uma *semi-decisão*. Este assunto será ainda aprofundado na seção 7.5.

As Máquinas de Turing também podem ser representadas através de diagramas de estados, com apenas algumas alterações em relação à representação dos autômatos finitos. Observe a Figura 7.1 que apresenta uma máquina segundo este tipo de representação, sendo seu estado inicial o q_0 . Inicialmente, os círculos que antes continham as etiquetas descritivas dos estados, agora estão com estas marcações fora deles, como por exemplo o estado q_0 . Em geral, estas marcações são descartadas, sendo utilizadas neste exemplo apenas para efeito didático. Ainda descrevendo os estados, observa-se que no interior deles existe um símbolo R , ou L . Estes símbolos

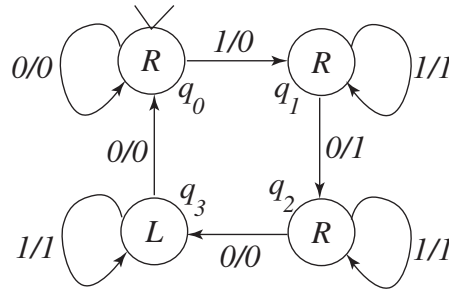


Figura 7.1: Máquina de Turing representada através de um diagrama de estados: *shiftadora*.

são os indicativos de qual movimento será realizado na cabeça leitora da máquina, isto é, direita (*R - right*) ou esquerda (*L - left*), respectivamente.

Outra questão importante são os eventos que ocorrem nas setas de transição. Nos autômatos finitos que foram estudados até o presente momento, apenas era indicado o símbolo lido associado a transição. Como as Máquinas de Turing podem não somente ler um símbolo, mas também podem escrever na fita de entrada, tem-se que esta operação mais sofisticada é representada através de uma marcação do tipo "símbolo lido, símbolo escrito". Assim 1/0 significa que o símbolo lido foi o 1, e que após esta leitura será escrito o símbolo 0 na mesma posição, alterando a fita de entrada.

Por fim, a dinâmica do processamento da Máquina de Turing se dá na seguinte ordem:

1. lê um símbolo da fita;
2. escreve um símbolo na mesma posição da leitura;
3. entra no estado indicado pela seta correspondente;
4. efetua um movimento da cabeça leitora para a direita ou para a esquerda, ditado pelo estado no qual se acabou de entrar.

Desta forma, o início do processamento da máquina descrita na Figura 7.1 pode ser explanado conforme a seguir. A máquina começa no estado inicial q_0 , aguardando

saber qual símbolo será lido. Se o símbolo lido for o 0, então ela escreverá 0 na fita, desloca-se para q_0 e deslocará a cabeça leitora para a próxima posição à direita. O leitor pode pensar que sair de q_0 e entrar novamente em q_0 significa o mesmo que permanecer em q_0 . Entretanto, para auxiliar no entendimento do movimento da cabeça leitora, sugere-se esta forma de leitura do que está acontecendo. Se o símbolo lido for 1, então ela irá escrever um 0 na mesma posição, irá entrar no estado q_1 , e irá efetuar um movimento da cabeça leitora para a direita.

Suponha a configuração $(q_0, \dots \underline{0}1000 \dots)$ para a máquina da Figura 7.1. A computação desta máquina é apresentada a seguir:

$$\begin{aligned}
(q_0, \dots \underline{0}1000 \dots) &\vdash_M (q_0, \dots 0\underline{1}000 \dots) \\
&\vdash_M (q_1, \dots 000\underline{0}0 \dots) \\
&\vdash_M (q_2, \dots 001\underline{0}0 \dots) \\
&\vdash_M (q_3, \dots 00\underline{1}00 \dots) \\
&\vdash_M (q_3, \dots 00\underline{0}100 \dots) \\
&\vdash_M (q_0, \dots 00\underline{1}00 \dots) \\
&\vdash_M (q_1, \dots 000\underline{0}0 \dots) \\
&\vdash_M (q_2, \dots 0001\underline{0} \dots) \\
&\vdash_M (q_3, \dots 000\underline{1}0 \dots) \\
&\vdash_M (q_3, \dots 000\underline{0}10 \dots) \\
&\vdash_M (q_0, \dots 000\underline{1}0 \dots) \dots
\end{aligned}$$

Agora suponha a computação realizada para a um outra configuração.

$$\begin{aligned}
(q_0, \dots \underline{0}111000 \dots) &\vdash_M (q_0, \dots 0\underline{1}11000 \dots) \\
&\vdash_M (q_1, \dots 00\underline{1}1000 \dots) \\
&\vdash_M (q_1, \dots 001\underline{1}000 \dots) \\
&\vdash_M (q_1, \dots 0011\underline{0}00 \dots) \\
&\vdash_M (q_2, \dots 00111\underline{0}0 \dots) \\
&\vdash_M (q_3, \dots 00111\underline{0}0 \dots) \\
&\vdash_M (q_3, \dots 001\underline{1}100 \dots) \\
&\vdash_M (q_3, \dots 00\underline{1}1100 \dots) \\
&\vdash_M (q_3, \dots 0\underline{0}11100 \dots) \\
&\vdash_M (q_0, \dots 00\underline{1}1100 \dots) \\
&\vdash_M (q_1, \dots 000\underline{1}100 \dots) \\
&\vdash_M (q_1, \dots 0001\underline{1}00 \dots) \\
&\vdash_M (q_1, \dots 00011\underline{0}0 \dots) \\
&\vdash_M (q_2, \dots 000111\underline{0} \dots) \\
&\vdash_M (q_3, \dots 000111\underline{0} \dots) \\
&\vdash_M (q_3, \dots 0001\underline{1}10 \dots) \\
&\vdash_M (q_3, \dots 000\underline{1}110 \dots) \\
&\vdash_M (q_3, \dots 000\underline{1}110 \dots) \\
&\vdash_M (q_3, \dots 000\underline{1}110 \dots) \\
&\vdash_M (q_0, \dots 000\underline{1}110 \dots) \dots
\end{aligned}$$

Observando as duas computações que foram realizadas é possível concluir que o que está acontecendo é que a máquina pega o primeiro símbolo 1 a aparecer, e o desloca para a posição do primeiro 0, após uma cadeia de 1's. O efeito visual que ocorre é o do deslocamento à direita da seqüência de 1's, uma operação conhecida como shift à direita. Como não há estado final, a máquina permanece em loop realizando os descolamentos infinitamente.

Exemplo 7.1 A máquina da Figura 7.2 representa uma máquina copiadora. Esta máquina copia a seqüência contígua de 1's à direita do símbolo A para a direita do símbolo B . (Sugere-se utilizar a configuração $\dots 0\underline{A}111000B0000 \dots$ para teste).

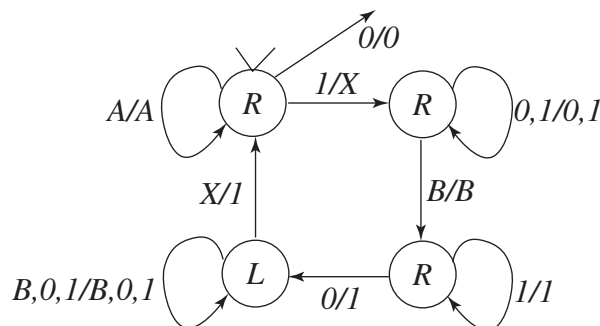


Figura 7.2: Máquina de copiadora do Exemplo 7.1.

A máquina da Figura 7.2 apresenta alguns itens de representação novos. O primeiro item, e mais importante, é a presença de um estado final. Diferente de como era realizado na representação dos autômatos finitos, nas Máquinas de Turing não é comum indicar o estado final explicitamente. Este estado é presumido a partir da seta de transição que sai do estado inicial, com uma etiqueta $0/0$. Outro item é uma generalização das próprias etiquetas das setas de transição, como por exemplo a $0, 1/0, 1$. Esta etiqueta é equivalente a duas etiquetas do tipo $0/0$ e $1/1$. Para tornar o desenho menos denso, opta-se por desenhar apenas uma seta de transição, na condição de que a leitura e escrita dos símbolos são determinadas pela ordem com que eles aparecem na etiqueta.

Uma característica interessante é a possibilidade de uso de tantos símbolos quanto os que forem desejados. Observe que a fita na configuração $\cdots 0A111000B0000\cdots$ é possível graças a um alfabeto $\Sigma = \{0, 1, A, B\}$. Os símbolos A e B tem a função nítida de servirem como marcadores para dar suporte ao procedimento de cópia, isto é, A indica a posição a partir da qual os símbolos devem ser copiados, e B a posição a partir da qual deve ser feita a cópia.

O projetista desta máquina aproveitou o conceito de marcação para introduzir um novo símbolo X . No movimento executado para a direita, X marca o símbolo do qual está sendo feita a cópia. No retorno da cabeça leitora, quando ele avança para a esquerda, este X indica para a máquina qual foi o símbolo copiado, e assim, ela é capaz de inferir o próximo símbolo a sofrer uma cópia. A introdução deste marcador serve como se fosse uma variável temporária na qual se armazena a posição do

elemento que está sendo copiado. Tal como uma variável temporária, ao término do processamento ela deve ser descartada pelo gerenciador de memória do compilador. Assim, se antes um símbolo 1 foi sobrescrito por um X , agora este mesmo X deve ser removido, restaurando o símbolo original. Desta forma, ao término da computação, a Máquina de Turing irá apresentar somente os símbolos pertencentes ao alfabeto Σ .

7.2 Máquinas de Turing Combinadas

A combinação de Máquinas de Turing permite que sejam criadas máquinas mais complexas a partir de máquinas mais simples. Muitas vezes é difícil conceber uma máquina que execute uma tarefa qualquer simplesmente por conta da inabilidade de quem a constrói em lidar com tantos estados e transições. Nestes casos, a estratégia de dividir para conquistar torna-se uma opção valiosa. Divide-se o problema inicial em partes menores mais simples e se realiza a construção de máquinas para cada uma destas partes individualmente, algo que a Engenharia de Software chama de análise. Logo após, realiza-se a síntese, isto é, a combinação destas partes para que juntas provejam a solução do problema inicial.

Sob esta ótica, torna-se desejável entender como combinar estas máquinas. Considere dois grupos de Máquinas de Turing com alfabeto $\Sigma = \{a, b, \dots, \sqcup\}$. O primeiro grupo é composto por máquinas que escrevem na fita um determinado símbolo, sobre outro já existente, e depois pára no estado h . Por exemplo, a função de transição da máquina M_a que escreve o símbolo a nestas condições (e similarmente M_b, \dots, M_{\sqcup}) é definida como:

q	σ	$\delta(q, \sigma)$
q_0	a	(h, a)
q_0	b	(h, a)
q_0	\vdots	(h, a)
q_0	\sqcup	(h, a)

O segundo grupo é composto por máquinas que movem a cabeça leitora uma unidade à esquerda (M_L) ou uma unidade à direita (M_R) e em seguida pára no estado h . Por

exemplo, a função de transição da máquina M_R que desloca a cabeça leitora para a esquerda é definida como:

q	σ	$\delta(q, \sigma)$
q_0	a	(h, \rightarrow)
q_0	b	(h, \rightarrow)
q_0	\vdots	(h, \rightarrow)
q_0	\sqcup	(h, \rightarrow)

Agora, imagine duas Máquinas de Turing M_1 e M_2 que executam tarefas quaisquer. A máquina resultante da sequenciação de M_1 seguida de M_2 é uma máquina que primeiro se comporta como M_1 e depois como M_2 . Esta composição, denotada por $> M_1 \rightarrow M_2$ é realizada da seguinte forma:

1. Altere o nome de todas as etiquetas dos estados de M_2 de tal forma que nenhuma etiqueta coincida com as etiquetas dos estados de M_1 .
2. Mude todos os estados de parada de M_1 para o estado inicial de M_2 , já com a nova etiqueta.
3. Anexe todas as transições de M_2 ao final da tabela de função de transição de M_1 .

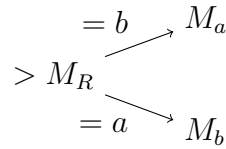
Por exemplo, $> M_a \rightarrow M_R$ é definida como:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_1, a)
q_0	b	(q_1, a)
q_0	\vdots	(q_1, a)
q_0	\sqcup	(q_1, a)
q_1	a	(h, \rightarrow)
q_1	b	(h, \rightarrow)
q_1	\vdots	(h, \rightarrow)
q_1	\sqcup	(h, \rightarrow)

Outra combinação ilustrativa é $> M_a \rightarrow M_R \rightarrow M_\sqcup \rightarrow M_L \rightarrow M_b$ (deixamos para o leitor a construção da função de transição).

Observe ainda que a conexão entre duas Máquinas de Turing não precisa ser incondicional, como até o momento. Essa conexão pode depender do símbolo sob a cabeça de leitura no ponto onde a primeira máquina pára. Esta dependência é representada por um teste na conexão entre as duas máquinas, isto é, $M_1 \xrightarrow{\text{teste}} M_2$. Por exemplo, $> M_R \xrightarrow{=b} M_\sqcup$ primeiro move o cabeçote da máquina para a direita. Em seguida, se existe um símbolo b sob a cabeça leitora, então ela sobrescreve um \sqcup e pára. Se nesta posição existisse qualquer outro símbolo diferente de b , a máquina pararia imediatamente. Não existe uma regra rígida sobre como indicar a semântica deste teste, por isso também são válidas estas outras conexões: $M_R \xrightarrow{\in\{a,b\}} M_\sqcup$, $M_R \xrightarrow{\neq a} M_\sqcup$, $M_R \xrightarrow{\notin\{a,b\}} M_\sqcup$.

A partir desse momento, as conexões podem ser realizadas segundo condições de contorno, por isso uma demanda natural que se segue é a possibilidade de realizar múltiplas conexões (múltiplas setas), desde que os testes sejam mutuamente exclusivos. Como por exemplo:



Inicialmente, esta máquina move a cabeça leitora para a direita. Depois, se existe um a sob a cabeça leitora, então ela escreve um b e pára. Se existe um b sob a cabeça leitora, então um a é escrito e a máquina pára. Se qualquer outro símbolo surgir, a máquina pára imediatamente após a cabeça leitora ter se deslocado para a direita.

A tabela desta função de transição é escrita da seguinte forma:

1. Renomear todos os estados de M_a e M_b para evitar duplicatas, não só entre si mas também com M_R .
2. Alterar os estados de parada da máquina M_R . Todos os estados de parada associados ao símbolo a são renomeados para coincidirem com o estado inicial de M_b . Já os estados de parada associados ao símbolo b são alterados para o

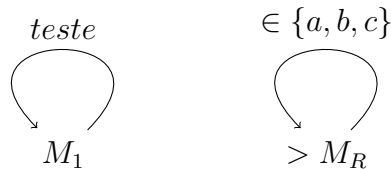
estado de partida de M_a .

3. Anexe todas as tabelas em uma única tabela.

Assim, a tabela fica definida como:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_2, \rightarrow)
q_0	b	(q_1, \rightarrow)
q_0	\vdots	(h, \rightarrow)
q_0	\sqcup	(h, \rightarrow)
q_1	a	(h, a)
q_1	b	(h, a)
q_1	\vdots	(h, a)
q_1	\sqcup	(h, a)
q_2	a	(h, b)
q_2	b	(h, b)
q_2	\vdots	(h, b)
q_2	\sqcup	(h, b)

Por fim, uma vez de posse dos desvios condicionais, resta a hipótese de que a conexão da primeira máquina seja feita com a entrada dela mesma, isto é, que seja feita uma retroalimentação, gerando um loop. Assim, a máquina M_1 é executada, até o ponto quando normalmente ela pararia mas a avaliação de um teste, caso se mostre verdadeiro, faz com que ela retorne para o estado inicial. No exemplo a seguir, a máquina move-se repetidamente para a direita enquanto existe um a , b ou c sob a cabeça leitora. Quando o símbolo sob esta cabeça leitora é diferente de a , b ou c , a máquina pára.



Nos casos com loop, é importante a inserção de um teste condicional para evitar que a máquina entre em loop eterno. A tabela da função de transição para esta máquina é construída da seguinte forma:

1. Altere os estados de parada da máquina M_1 . Qualquer estado de parada h associado à leitura de um símbolo envolvido no teste deve ser alterado para o estado inicial.

No caso da máquina M_R tem-se:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_0, \rightarrow)
q_0	b	(q_0, \rightarrow)
q_0	\vdots	(h, \rightarrow)
q_0	\sqcup	(h, \rightarrow)

7.3 Variações da Máquina de Turing

É comum se encontrar diversas aplicações que requerem cálculos da ordem de bilhões de operações por segundo, tais como o processamento de sinais e imagens, cálculos para renderização, simulações numéricas, criptografia, entre outras. Uma solução natural para estas demandas, oferecida pelos fabricantes de hardware, é o desenvolvimento de processadores específicos dedicados para um determinado tipo de atividade, tais como os Processadores de Sinais Digitais (*Digital Signal Processor* - DSPs) e *Field-Programmable Gate Arrays* (FPGAs). Entretanto, esta abordagem possui um custo e um esforço de desenvolvimento elevado [89, 90], além do que o produto final carece de flexibilidade, pois não se adapta com facilidade à evolução dos algoritmos e dos aplicativos. Os projetistas de computadores sempre buscaram construir máquinas mais rápidas. Entretanto, existem vários limites físicos que impedem o aumento indiscriminado desta velocidade de hardware, a fim de atingir computadores mais velozes, tais como temperatura de operação, miniaturização do hardware, entre outros. Uma abordagem alternativa, que tem encontrado muitos adeptos, é a construção de máquinas paralelas que usam a simultaneidade de execução de tarefas, como meio para acelerar um processamento. Segundo Hwang [91], o processamento paralelo pode ser definido como uma forma eficiente do processamento de dados, com ênfase na exploração de eventos concorrentes do processo computacional. Neste sentido, uma das soluções é o emprego de *Stream Processors*

programáveis [92], que exploram a baixa localidade de referência, e a pequena necessidade de concorrência de alguns algoritmos. Estes dispositivos passaram a exigir linguagens de programação que permitissem seu uso com mais facilidade. Estas linguagens, por sua vez, evoluíram concomitantemente com o progresso tecnológico das *Graphical Processing Units* (GPUs), criando uma disputa acirrada pelo mercado de computação de alto desempenho, através das *General Purpose Computing on Graphics Processing Units* (GPGPUs), uma evolução dos *Stream Processors* programáveis. A proposta das GPGPUs é utilizar as GPUs não somente para aplicações gráficas, mas também para realizar computação em um sentido mais amplo, tarefa esta que normalmente está associada as *Central Processing Units* (CPUs). Trata-se de um rompimento com a estrutura clássica de processadores programáveis, conhecida como arquitetura de von Neumann [93]. Esse paradigma é uma quebra, que no final da década de 1970, já havia sido apontada por Backus como necessária [94]. As arquiteturas modernas de GPUs apresentam características importantes como [95]: (1) uma largura de banda elevada; (2) uma capacidade de processamento significativa; (3) são programáveis; (4) e possuem uma relação custo/benefício vantajosa. O paradigma da construção de um algoritmo paralelo, resolutor de um determinado problema, depende significativamente da maneira pela qual a carga de processamento é dividida entre os processadores disponíveis. Este particionamento do problema está intimamente ligado, não só a sua natureza, mas também à arquitetura paralela disponível na máquina. Esse cenário se contrapõe a uma primeira impressão de que tudo pode ser paralelizado, e de que o paralelismo pode aumentar indefinidamente a velocidade de processamento, desde que haja recursos computacionais disponíveis. A Lei de Amdahl [96] e a Lei de Gustafson-Barsis [96] fazem a relação de como os trechos não paralelizáveis de um programa comprometem o aumento de velocidade do paralelismo, a Conjectura de Minsky [97] estabelece como ocorre a queda de desempenho de processadores paralelos devido ao conflito de acesso aos dados e a Lei de Grosch [98] postula sobre o custo financeiro dos MFLOPS (*Mega Floating-point Operations Per Second*). Estes princípios, por exemplo, são limites clássicos das condições de contorno associadas ao processamento paralelo, como será apresentado nos capítulos a seguir. Os programas e os dispositivos de processamento podem ser classificados de acordo com a quantidade de fluxo de instruções,

combinada com a quantidade de dados usada por tais instruções. Essa rotulação é conhecida como taxonomia de Flynn, e classifica os dispositivos e programas segundo quatro grandes grupos, a saber [93]:

- a) SISD (*Single Instruction, Single Data*): uma Única Instrução e um Único Dado. A grande parte dos computadores da atualidade segue este modelo de arquitetura, que está diretamente associada à arquitetura de von Neumann;
- b) SIMD (*Single Instruction, Multiple Data*): uma Única Instrução e Múltiplos Dados. São os computadores vetoriais, que aplicam as mesmas operações matemáticas em um grande volume de dados;
- c) MISD (*Multiple Instruction, Single Data*): Múltiplas Instruções e Dados Únicos. Não é uma representação muito utilizada, sendo conhecida como Vetores Sistólicos², apresentando com certa frequência, uma predisposição a gerar ociosidade dos processadores;
- d) MIMD (*Multiple Instruction, Multiple Data*): Múltiplas Instruções e Múltiplos Dados. Através de diversas unidades de processamento, manipula-se conjuntos de dados distintos, como por exemplo as GPGPUs.

Apesar do determinismo desta taxonomia, a prática mostra que existe uma dualidade combinatória entre as classificações. Os *Stream Processors* programáveis podem ser considerados como uma combinação de SIMD/MIMD, de acordo com a abordagem adotada na solução do problema a ser paralelizado. Deste modo, as GPGPUs podem ser consideradas, ora como pertencentes à taxonomia de SIMD, ora como MIMD, conforme seu emprego.

No início da seção anterior, foi mencionado que uma Máquina de Turing é um mecanismo que se assemelha em muito aos computadores modernos, e ainda que possui o mesmo poder computacional de qualquer outro computador de propósito geral. Isto não quer dizer que a máquina de Turing seja mais rápida, ou mesmo igualmente rápida, quando comparada com os computadores atuais. Afinal, uma

²Caracterizam-se pelo emprego de processadores simples (células), responsáveis por operações elementares, que são aplicadas a um fluxo de dados, “bombeado” através do sistema.

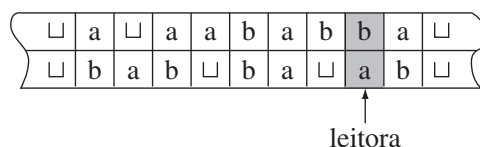


Figura 7.3: Máquina de fita *multitrack*: exemplo com duas trilhas.

máquina de fita seqüencial, que lê e escreve um único dado em uma fita, tem que ser mais lenta que as atuais tecnologias. O que é proposto ao afirmar sobre o mesmo poder computacional é que a Máquina de Turing pode computar exatamente as mesmas coisas que o computador mais rápido da atualidade, seja qual for a tecnologia que ele emprega, isto é, vários núcleos de processamento, computação quântica, entre outros. A Máquina de Turing pode efetuar uma computação muito mais lenta, mas certamente chegará aos mesmos resultados de qualquer outro computador, e por este motivo é considerada como a máquina universal de computação. Esta é a beleza da criação de Turing, que concebeu um modelo trivial para algo que nunca perdeu sua atualidade. Sob esta ótica, esta seção tem por objetivo mostrar que diversas variações de máquinas possuem uma Máquina de Turing equivalente, que através de uma única cabeça leitora realiza a leitura e escrita de um único símbolo.

Uma primeira variação é a máquina de fitas *multitrack*, uma representação análoga à categoria SIMD. O leitor poder também imaginar um processador que faz a leitura e escrita simultânea em um barramento com 32, 64, ou até mesmo mais bits. Uma ilustração esquemática deste tipo de situação é apresentada na Figura 7.3, cuja principal característica é a quantidade de símbolos que são lidos e escritos ao mesmo tempo. No exemplo da figura em questão a máquina de fita *multitrack* possui, por simplicidade, apenas duas trilhas e conseqüentemente lê e escreve dois símbolos simultaneamente. Esta máquina M poderia estar sujeita a seguinte função de transição δ :

M com fita de duas trilhas

q	σ	$\delta(q, \sigma)$
q_0	$\begin{smallmatrix} a \\ a \end{smallmatrix}$	$(q_0, \begin{smallmatrix} a \\ b \end{smallmatrix})$
q_0	$\begin{smallmatrix} a \\ b \end{smallmatrix}$	$(q_1, \begin{smallmatrix} a \\ b \end{smallmatrix})$
q_0	$\begin{smallmatrix} b \\ a \end{smallmatrix}$	$(q_1, \begin{smallmatrix} b \\ b \end{smallmatrix})$
q_0	$\begin{smallmatrix} b \\ b \end{smallmatrix}$	$(q_1, \begin{smallmatrix} a \\ a \end{smallmatrix})$
q_1	$\begin{smallmatrix} a \\ a \end{smallmatrix}$	$(q_1, \begin{smallmatrix} b \\ a \end{smallmatrix})$
q_1	$\begin{smallmatrix} a \\ b \end{smallmatrix}$	$(h, \begin{smallmatrix} a \\ b \end{smallmatrix})$
q_1	$\begin{smallmatrix} b \\ a \end{smallmatrix}$	$(q_0, \begin{smallmatrix} b \\ a \end{smallmatrix})$
q_1	$\begin{smallmatrix} b \\ b \end{smallmatrix}$	$(q_1, \begin{smallmatrix} a \\ a \end{smallmatrix})$

Observe que a representação do tipo $\sigma = \begin{smallmatrix} b \\ a \end{smallmatrix}$ indica que é lido o símbolo b da primeira trilha, e o símbolo a da segunda trilha (analogamente no caso da escrita). Note que esta função de transição δ utiliza um alfabeto $\Sigma = \{a, b\}$. Uma vez que a composição da fita com a cabeça leitora permite a leitura e escrita de símbolos simultaneamente, e ainda, que existem apenas dois símbolos no alfabeto, temos então que todas as possibilidades de combinação de símbolos nesta fita de duas trilhas é dada por $\begin{smallmatrix} a \\ a \end{smallmatrix}, \begin{smallmatrix} a \\ b \end{smallmatrix}, \begin{smallmatrix} b \\ a \end{smallmatrix}, \begin{smallmatrix} b \\ b \end{smallmatrix}$. Este resultado é importante porque indica que o conjunto de duplas de símbolos que podem ser lidos e escritos pela máquina M é fixo e limitado. Assim, pode-se fazer uso de um artifício matemático onde cada dupla fixa de símbolos é substituída por uma nova representação, e assim se obtém:

$$\begin{smallmatrix} a \\ a \end{smallmatrix} = 0 \qquad \begin{smallmatrix} a \\ b \end{smallmatrix} = 1 \qquad \begin{smallmatrix} b \\ a \end{smallmatrix} = 2 \qquad \begin{smallmatrix} b \\ b \end{smallmatrix} = 3$$

Este artifício permite realizar uma substituição direta dos símbolos existentes na função de transição δ da máquina M com fita de duas trilhas, resultando na função de transição δ' que representará a simulação da mesma máquina M em uma fita simples. Esta função δ' pode ser facilmente aplicada em uma Máquina de Turing com fita simples que ao invés de ter um alfabeto de dois símbolos, possui um alfabeto de quatro símbolos dado por $\{0, 1, 2, 3\}$. O leitor terá facilidade para entender que se considerarmos uma fita *multitrack* de k trilhas e um alfabeto de n símbolos, sempre poderemos convertê-la em uma Máquina de Turing de fita simples com n^k símbolos.

Simulação com fita simples

q	σ	$\delta'(q, \sigma)$
q_0	0	$(q_0, 1)$
q_0	1	$(q_1, 1)$
q_0	2	$(q_1, 3)$
q_0	3	$(q_1, 0)$
q_1	0	$(q_1, 2)$
q_1	1	$(h, 1)$
q_1	2	$(q_0, 2)$
q_1	3	$(q_1, 0)$

Deste modo, um computador atual contendo um barramento de 64 bits, e ainda sabendo que os níveis de quantização são apenas dois (0 e 1), temos que existe uma Máquina de Turing de fita simples análoga ao computador, a qual está associado um alfabeto de 2^{64} símbolos. Sob a ótica da Engenharia, este número gigante de símbolos é desconfortável de ser manipulado, o que tornaria a construção de tal máquina muito trabalhosa. Entretanto, desconsiderando esta questão, tanto a máquina como o computador são capazes de manipular a mesma quantidade de dados.

A segunda variação de Máquina de Turing é a máquina com várias fitas, uma representação análoga à categoria MIMD. Esta máquina possui uma única unidade de controle, porém apresenta mais de um par de fitas com cabeças leitoras. A Figura 7.4 apresenta uma máquina de duas fitas e duas cabeças. Nesta máquina a leitura e a escrita na fita é realizada de modo síncrono, isto é, símbolos são lidos e escritos simultaneamente em suas respectivas fitas. Por sua vez, o movimento das fitas é independente, pois estas não precisam obrigatoriamente avançar para o mesmo lado.

Na parte superior direita da Figura 7.4 são apresentadas de forma esquemática as duas fitas, bem como o posicionamento da cabeça leitora, denotado por \uparrow . Neste caso, a proposta é imitar o comportamento da cabeça leitora através de uma fita auxiliar. Nesta fita auxiliar este posicionamento da cabeça leitora é indicado pela inscrição do símbolo 1 na posição correspondente. Assim, no exemplo proposto

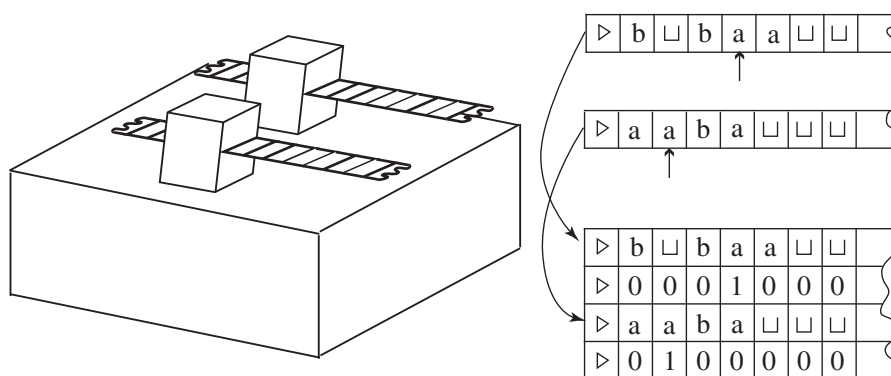


Figura 7.4: Máquina de várias fitas.

transforma-se a máquina em questão em uma máquina de fita *multitrack* de quatro trilhas. Em seguida, converte-se esta fita de quatro trilhas para uma fita simples com um alfabeto mais abrangente.

A máquina de fitas multidimensionais também é uma variação da máquina de fita simples. Nesta situação, a fita multidimensional é transformada em *multitrack*, que em seguida é transformada em fita simples. Já uma máquina multiprocessada na qual existam várias cabeças leitoras para uma única fita, faz-se uma abordagem similar à máquina de várias fitas, marcando-se em uma fita auxiliar o posicionamento das cabeças leitoras com símbolos diferentes para cada uma delas. Casos mais complexos são reduzidos sucessivamente a casos mais simples, até que se obtenha o mapeamento adequado para a máquina de fita simples.

7.4 Máquina Universal de Turing

As Máquinas de Turing desenvolvidas até o momento, são dispositivos construídos para tratar especificamente um determinado problema, isto é, elas são configuradas para uma determinada tarefa, ou de forma simplista, são computadores com um programa armazenado estaticamente. Quando uma nova tarefa precisa ser computada, uma construção nova de máquina precisa ser realizada. Assim, a Máquina de Turing é construída conforme o problema que se deseja resolver, enquanto que os computadores atuais são máquinas de propósito geral podendo ser programadas.

Contudo, isto está equivocado, pois as Máquinas de Turing podem ser programadas. O formalismo da Máquina de Turing é nada mais que um Sistema Formal. Por isso ele pode ser representado por uma string, que em última análise, é um programa que descreve o comportamento da máquina. Essa string pode ser colocada na fita de outra Máquina de Turing de propósito geral que então simula o comportamento da máquina original codificada na fita. Em seguida, também se coloca na fita o string de entrada da máquina original que se deseja processar, provendo então a emulação da máquina e a execução da tarefa. Tal Máquina de Turing de propósito geral é conhecida como Máquina de Turing Universal. O grande desafio é codificar a tabela da função de transição, e nesta seção, será seguida a abordagem de Lewis e Papadimitriou [99].

Inicialmente, será necessário criar uma linguagem em que as strings sejam representações fidedignas da Máquinas de Turing. Será utilizado um alfabeto que dê suporte às seguintes convenções: (1) a cadeia que representa um estado da Máquina de Turing será da forma $q\{0,1\}^*$, isto é, a letra q seguida por uma cadeia binária indicativa do número do estado; (2) os símbolos da fita serão representados por uma cadeia da forma $a\{0,1\}^*$.

Seja $M = \langle k, \Sigma, \delta, s, F \rangle$ uma Máquina de Turing. Sejam i e j valores inteiros tais que $2^i \geq \#(k)$ e $2^j \geq \#(\Sigma) + 2$. Desta forma, cada estado do conjunto k é representado pelo símbolo q seguido de uma cadeia binária de comprimento i . Cada símbolo do conjunto Σ é representado pelo símbolo a seguido de uma cadeia de j bits. Observe que os símbolos \leftarrow e \rightarrow representando os movimentos da cabeça leitora também precisam ser codificados como símbolos na nova fita, o que exige aumentar a aridade de Σ em duas unidades. Por convenção, os símbolos \sqcup , \triangleright , \leftarrow e \rightarrow serão representados respectivamente por $a0^j$, $a0^{j-1}1$, $a0^{j-2}10$ e $a0^{j-2}11$. O estado inicial da máquina é representado por $q0^i$.

Para melhor compreender esta codificação, considere uma nova máquina $M = \langle k, \Sigma, \delta, s, F \rangle$, onde $k = \{q_0, q_1, h\}$, $\Sigma = \{a, \sqcup, \triangleright\}$, $s = q_0$, $F = \{h\}$ e δ definido como a seguir. Simultaneamente, também é apresentada a tabela de codificação dos estados e símbolos. Como $2^i \geq \#(k) = 3$ então $i = 2$, e como $2^j \geq \#(\Sigma) + 2 = 5$

então $j = 3$.

			estado/símbolo	codificação
q	σ	$\delta(q, \sigma)$		
q_0	a	(q_1, \sqcup)	q_0	$q00$
q_0	\sqcup	(h, \sqcup)	q_1	$q01$
q_0	\triangleright	(q_0, \rightarrow)	h	$q11$
q_1	a	(q_0, a)	\sqcup	$a000$
q_1	\sqcup	(q_0, \rightarrow)	\triangleright	$a001$
q_1	\triangleright	(q_1, \rightarrow)	\leftarrow	$a010$
			\rightarrow	$a011$
			a	$a100$

A representação hipotética de uma fita de M dada por $w = \triangleright aa\sqcup$ a será a string $a001a100a100a000a100$. Já a representação de M será da seguinte forma:

$$\begin{aligned}
\langle M \rangle &= \delta(q_0, a) = (q_1, \sqcup); \delta(q_0, \sqcup) = (h, \sqcup); \delta(q_0, \triangleright) = (q_0, \rightarrow); \\
&\quad \delta(q_1, a) = (q_0, a); \delta(q_1, \sqcup) = (q_0, \rightarrow); \delta(q_1, \triangleright) = (q_1, \rightarrow) \\
&= \langle q_0, a, q_1, \sqcup \rangle; \langle q_0, \sqcup, h, \sqcup \rangle; \langle q_0, \triangleright, q_0, \rightarrow \rangle; \\
&\quad \langle q_1, a, q_0, a \rangle; \langle q_1, \sqcup, q_0, \rightarrow \rangle; \langle q_1, \triangleright, q_1, \rightarrow \rangle \\
&= (q00, a100, q01, a000); (q00, a000, q11, a000); (q00, a001, q00, a011); \\
&\quad (q01, a100, q00, a011); (q01, a001, q00, q011); (q01, a001, q01, a011) \\
&= q00a100q01a000q00a000q11a000q00a001q00a011 \quad \text{continua ...} \\
&\quad q01a100q00a011q01a001q00q011q01a001q01a011
\end{aligned}$$

Tomando por base uma máquina M de uma fita, tem-se uma máquina universal M_U com três fitas e cabeçotes distintos. A primeira fita de M_U contém a codificação do conteúdo da fita de M , a segunda fita contém a codificação da função de transição da própria máquina M e a terceira fita possui a codificação do estado de M no ponto corrente da computação. A computação de M_U ocorre da seguinte forma:

1. A máquina M_U é inicializada com uma string composta pelas codificações de M e de w em sua primeira fita, enquanto que as outras são preenchidas com espaços em branco.
2. M_U move a codificação de M para a segunda fita e desloca w para a esquerda deixando a primeira fita com o formato $\triangleright \sqcup w$.

3. M_U escreve na terceira fita a codificação do estado inicial de M que é sempre $q0^i$.
4. M_U passa a simular a computação de M . M_U mantém as cabeças leitoras da segunda e terceira fitas em suas extremidades esquerdas enquanto o cabeçote da primeira fita posiciona-se sobre o símbolo a associado à versão codificada do símbolo que M estaria lendo no momento.
5. M_U emula então um passo da operação de M percorrendo a segunda fita buscando uma quádrupla em que o primeiro componente corresponda ao estado codificado gravado na terceira fita, e o segundo componente ao símbolo codificado indicado na primeira fita.
6. Localizando esta a quádrupla, M_U altera o estado corrente da terceira fita substituindo pelo terceiro componente desta quádrupla e realiza na primeira fita a ação especificada pelo quarto componente da quádrupla.
7. Se em algum passo a combinação estado-símbolo não for encontrada na segunda fita, tem-se que o estado corrente é um estado de parada e M_U pára.

Observe que o funcionamento desta máquina não é simples, mas se for preciso especificar melhor sua operação, pode-se utilizar o artifício de conexão de Máquinas de Turing simples apresentado na seção 7.2. Desta forma, são definidas as seguintes máquinas triviais:

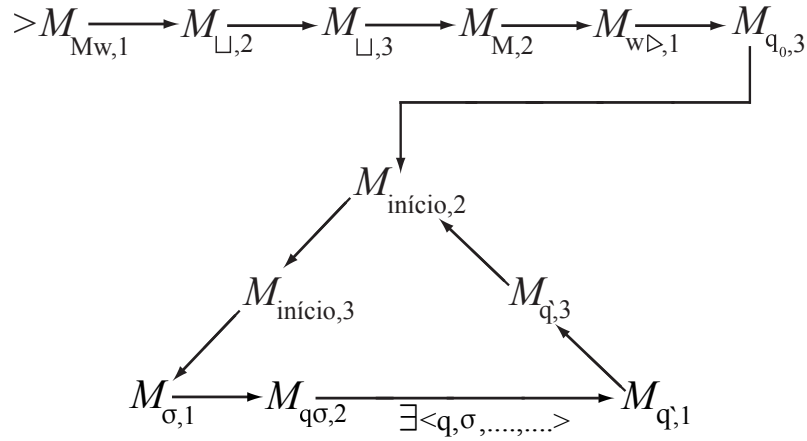


Figura 7.5: Máquina Universal de Turing.

- $M_{Mw,1}$: Copia a string codificada $\langle M \rangle$ e w para a fita 1
- $M_{\sqcup,2}$: Preenche a fita 2 com \sqcup
- $M_{\sqcup,3}$: Preenche a fita 3 com \sqcup
- $M_{M,2}$: Copia a codificação de M para a fita 2
- $M_{w\triangleright,1}$: Faz $\triangleright \sqcup w$ na fita 1
- $M_{q_0,3}$: Escreve q_0 na fita 3
- $M_{início,2}$: Coloca cabeçote na posição inicial da fita 2
- $M_{início,3}$: Coloca cabeçote na posição inicial da fita 3
- $M_{\sigma,1}$: Coloca o cabeçote sobre o símbolo corrente na fita 1
- $M_{q\sigma,2}$: Busca a quádrupla com 1º argumento q e 2º argumento σ na fita 2
- $M_{q',3}$: Escreve o novo estado corrente na fita 3
- $M_{\sigma',1}$: Realiza a operação de escrita na fita 1

A figura 7.5 representa graficamente a Máquina de Turing Universal M_U composta por conexões de máquina triviais. A máquina inicia sua operação com um conjunto de tarefas seqüenciais de preparação para a simulação de $M(w)$. Em seguida entra em um loop que realiza de fato o processamento desejado e que só é interrompido caso não seja encontrada uma quádrupla $\langle q, \sigma, q', \sigma' \rangle$, onde q e σ são os parâmetros de busca. Esta construção representa o algoritmo 7.1.

Data: M e w codificados

Result: w' resultante de $M(w)$

```
/* Inicialização da Máquina  $M_U$  */
1 Fita 1: Copia a string codificada  $\langle M \rangle$  e  $w$  para a fita ;
2 Fita 2: Preenche a fita com  $\sqcup$  ;
3 Fita 3: Preenche a fita com  $\sqcup$  ;
4 Fita 2: Copia a codificação de  $M$  da fita 1 para a fita ;
5 Fita 1: Faz  $\triangleright \sqcup w$  na fita ;
6 Fita 3: Escreve  $q_0$  na fita ;
7  $Pare = falso$  ;
8 repeat
    /* Simulação de uma computação por  $M$  */
    9 Fita 2: Coloca cabeçote na posição inicial da fita ;
    10 Fita 3: Coloca cabeçote na posição inicial da fita ;
    11 Fita 1: Coloca o cabeçote sobre o símbolo corrente na fita ;
    12 if Fita 2: Busca a quádrupla com 1º argumento  $q$  e 2º argumento  $\sigma$ 
        then
            13 Fita 3: Escreve o novo estado corrente na fita ;
            14 Fita 1: Realiza a operação de escrita na fita ;
        else
            15  $Pare = verdadeiro$  ;
        end
    17 until  $Pare = verdadeiro$ ;
19 Pára ;
```

Algoritmo 7.1: Algoritmo de funcionamento da Máquina Universal de Turing.

Conseqüentemente, esta é a descrição de uma Máquina de Turing de propósito geral que pode ser programada. Esta máquina possui três cabeçotes e três fitas independentes, formato que pode ser convertido por um Máquina de Turing de fita simples conforme estudado na seção 7.3.

7.5 Decidibilidade da Máquina de Turing

Ao longo deste estudo, o leitor já se deparou com alguns problemas em que havia uma espécie de dificuldade decisória em alguns modelos de computação. Recorde a definição de conjunto recursivo (definição 3.36) e de conjunto recursivamente enumerável (definição 3.37). Neste contexto matemático existe um problema relacionado com uma avaliação de função que pode nunca terminar. Em seguida, reveja os exemplos de Máquinas de Turing da seção 7.1 onde algumas máquinas exibiam um comportamento de semi-decisão, isto é, ora aceitavam uma determinada, ora entravam em loop. Esta seção abordará tais questões detalhadamente. A idéia é encontrar exemplos concretos de problemas que não podem ser resolvidos por computador e apresentar a técnica de redução como uma ferramenta útil na determinação de problema solúveis e insolúveis.

Um passo importante é revisitar a Hierarquia de Chomsky (figura 5.4) e questionar se existem linguagens que não são Turing reconhecíveis, isto é, se as linguagens tipo 0 representam o conjunto de todas as linguagens do universo ou se esse universo é mais amplo que as linguagens tipo 0. Para responder a esta questão são necessárias algumas observações que remetem muito ao estudo apresentado no capítulo 2.

Lema 7.1 O conjunto Σ^* de todas as strings de um alfabeto Σ é contável.

Note que os elementos de Σ^* são strings de tamanho finito. É possível enumerar tais strings ordenando os mesmos segundo seus tamanhos n lexicográficos, com $n = 0, 1, 2, \dots$ em ordem crescente. Assim Σ^* é contável. \square

Lema 7.2 A coleção de todas as Máquinas de Turing é contável.

Uma vez que uma Máquina de Turing pode ser codificada em uma string $\langle M \rangle$, pelo lema anterior esta coleção é contável. \square

Lema 7.3 O conjunto B de todas as strings binárias infinitas é incontável.

Demonstração por contradição

Assuma que B seja contável. Então deve existir uma bijeção f do conjunto dos números naturais \mathbb{N} para B . Construa uma string binária infinita b tal que o n -ésimo símbolo difere daquele de $f(n)$ para $n = 1, 2, \dots$. Assim, não existe um k tal

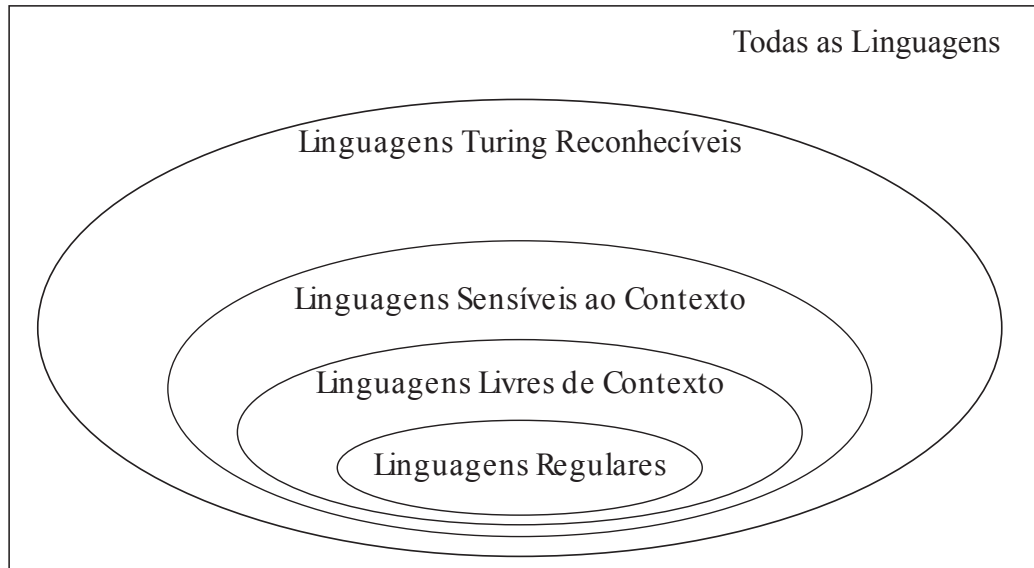


Figura 7.6: Hierarquia de Chomsky complementada.

que $f(k) = b$, o que é uma contradição. Desse modo, B é incontável. \square

Lema 7.4 A coleção \mathcal{L} de todas as linguagens de uma alfabeto Σ é incontável. Para demonstrar este lema basta mostrar que existe um bijeção entre \mathcal{L} e B do lema anterior. Como Σ^* é contável, pode-se numerar seus elementos como s_1, s_2, s_3, \dots . Em seguida, pode-se definir uma bijeção $f : \mathcal{L} \rightarrow B$ tal como: para toda linguagem A , seja o i -ésimo símbolo de $f(A)$ é igual a 1 se $s_i \in A$ e 0 se $s_i \notin A$. \square

Teorema 7.1 Algumas linguagens não são Turing reconhecíveis. Pelos lemas anteriores, a coleção \mathcal{L} de todas as linguagens de uma alfabeto Σ é incontável, enquanto que a coleção de todas as Máquinas de Turing é contável. Assim, \mathcal{L} contém linguagens que não são Turing reconhecíveis. \square

Deste modo, fica demonstrada a existência de linguagens que não são Turing reconhecíveis e a Hierarquia de Chomsky pode ser complementada conforme a figura 7.6.

Além disso, é importante ter em mente, e de forma organizada, os cenários relevantes nos quais uma Máquina de Turing pode se encontrar. Este cenários são elencados a seguir e ilustrados na figura 7.7.

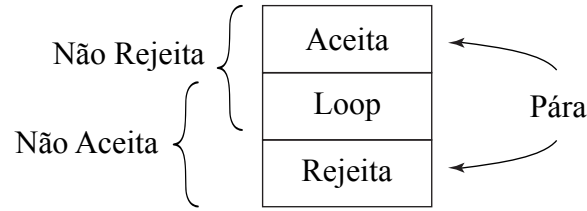


Figura 7.7: Cenários de aceitação de uma Máquina de Turing.

- M **aceita** um string w se ela se encontra em um estado final de aceitação
- M **rejeita** um string w se ela se encontra em um estado final de rejeição
- M entra em **loop infinito** para um string w se não acontece dela aceitar ou rejeitar uma string
- M **não aceita** uma string w se ela rejeita w ou se w provoca um loop infinito
- M **não rejeita** uma string w se ela aceita w ou se w provoca um loop infinito
- M **pára** se ela aceita ou rejeita uma string

Lembre que toda Máquina de Turing pode ser convertida em uma representação por string denotada por $\langle M \rangle$. A Máquina de Turing Universal M_U recebe uma codificação $\langle M, w \rangle$ da máquina M com entrada w e então simula a execução $M(w)$. A linguagem de U_M é denotada por A_M , $L(U_M) = A_M$, isto é:

$$A_M = \{ \langle M, w \rangle \mid M \text{ aceita } w \}$$

Ou de forma equivalente:

$$A_M = \{ \langle M, w \rangle \mid w \in L(M) \}$$

$$L(M) = \{ w \mid M \text{ aceita } w \}$$

Agora imagine se fosse desenhada uma tabela bem grande cujas as linhas fossem formadas pela coleção de todas as Máquinas de Turing M_1, M_2, M_3, \dots e cujas colunas são a codificação em strings destas máquinas $(\langle M_1 \rangle, \langle M_2 \rangle, \langle M_3 \rangle, \dots)$.

Em cada entrada indexada pelo par ordenado $\langle M_i, x_i \rangle$, coloca-se 1 se M_i aceita x_i , e 0 caso contrário. Em outras palavras:

$$\langle M_i, x \rangle = \begin{cases} 1 & \text{se } x \in L(M_i) \\ 0 & \text{se } x \notin L(M_i) \end{cases}$$

Por exemplo, o canto superior esquerdo desta tabela pode se parecer como a seguir:

	$\langle M_1 \rangle$	$\langle M_2 \rangle$	$\langle M_3 \rangle$	$\langle M_4 \rangle$	$\langle M_5 \rangle$	$\langle M_6 \rangle$	$\langle M_7 \rangle$	\dots
M_1	0	0	0	0	0	0	0	\dots
M_2	1	0	1	0	1	0	1	\dots
M_3	1	1	0	1	1	0	1	\dots
M_4	0	1	1	1	0	1	1	\dots
M_5	1	1	1	1	1	0	1	\dots
M_6	0	1	1	0	1	1	0	\dots
M_7	1	1	0	1	0	1	1	\dots
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\ddots

Deve-se deixar claro que as entradas desta tabela são hipotéticas, não foi utilizado nenhuma codificação de Máquinas de Turing e nem foram testadas as diferentes entradas. Contudo, poderia ser utilizado o esquema da seção 7.4 para produzir estas entradas.

Em seguida considere uma linha qualquer, por exemplo, a linha indexada pela máquina M_3 . Nesta linha existe uma seqüência infinita de 0's e 1's que determinam a linguagem aceita por M_3 . Usando estas entradas hipotéticas, pode-se observar que M_3 aceita $\langle M_1 \rangle$, $\langle M_2 \rangle$ e muitas outras máquinas representadas na tabela. Esta seqüência de 0's e 1's é chamada de *seqüência característica* da linguagem $L(M_3)$.

Deseja-se encontrar uma linguagem cuja a seqüência característica é diferente de todas as linhas da tabela. Se existe tal linguagem, então ela não pode ser Turing reconhecível porque as linhas da tabela já fornecem as seqüências características de todas as possíveis linguagens Turing reconhecíveis sobre o alfabeto Σ . Isto porque todas as possíveis máquinas M_1, M_2, M_3, \dots estão listadas ali e indexadas em uma das linhas da tabela.

Uma solução para a obtenção da seqüência desejada é tomar a seqüência

determinada pela diagonal principal da tabela. Deve-se notar, que este desenvolvimento é quase idêntico a demonstração de Cantor para o conjunto \mathbb{R} não ser enumerável (exemplo 2.10), e por isto a técnica que está sendo empregada neste desenvolvimento é a técnica da diagonalização. Se cada uma destas entradas for trocada por seu valor complementar, então é obtida uma seqüência infinita de 0's e 1's que difere de qualquer linha da tabela. No exemplo em questão, obtém-se a seqüência:

$$\overline{0001111\cdots} = 1110000\cdots$$

A razão pela qual esta seqüência difere de todas as linhas da tabela é:

1. A seqüência difere da primeira linha porque o primeiro símbolo da seqüência é diferente do primeiro símbolo da linha;
2. A seqüência difere da segunda linha porque o segundo símbolo da seqüência é diferente do segundo símbolo da linha;
3. Analogamente, este raciocínio pode ser levado para todas as linhas da tabela. A seqüência proposta difere da linha k sempre, e pelo menos, na k -ésima posição.

Assim, foi obtida uma seqüência característica de uma linguagem que não é Turing reconhecível porque é diferente de todas as linhas da tabela. Neste momento, deve-se perceber que a linguagem obtida pelo complemento das entradas da diagonal principal da tabela, conhecida como linguagem da diagonalização L_D , pode ser descrita como:

$$L_D = \{w \in \Sigma^* | w \notin L(M_i)\}$$

Observe que L_D depende da escolha feita como esquema de codificação das Máquinas de Turing. Isto significa que se o processo iniciar com uma diferente codificação, iria ser obtida uma linguagem diferente, mas sua propriedade de não ser Turing reconhecível não se alteraria. Todo este desenvolvimento é uma das duas formas de demonstração do teorema a seguir (a outra é por contradição).

Teorema 7.2 A linguagem L_D não é Turing reconhecível.

Demonstração por contradição

Assuma, por contradição, que a linguagem L_D é Turing reconhecível. Pela definição de Turing reconhecível tem-se que $L_D = L(M_i)$ para alguma Máquina de Turing M associada a um alfabeto Σ . A lista M_1, M_2, M_3, \dots inclui todas as Máquinas de Turing, de tal modo que é possível fazer $w \in \Sigma^*$ para ser a primeira string tal que $M_i = M$ e $L_D = L(M_i)$. Deve-se questionar sobre a possibilidade de tal string w estar, ou não, contida em L_D . Tem-se:

$$w \in L_D \xleftrightarrow{(a)} w \notin L(M_i) \xleftrightarrow{(b)} w \notin L_D$$

onde a equivalência (a) surge da definição prévia de que $L_D = \{w \in \Sigma^* | w \notin L(M_i)\}$ e a equivalência (b) do fato prévio de que $L_D = L(M_i)$.

Assim, isto prova que $w \in L_D \Leftrightarrow w \notin L_D$, o que é uma contradição. Assim a linguagem L_D não é Turing reconhecível. \square

Outro ponto importante sobre as linguagens Turing reconhecíveis é a questão sobre decidibilidade. Uma Máquina de Turing que pára em todas as suas entradas é chamada de máquina decisor. Já uma máquina sujeita à uma semi-decisão é chamada de reconhecedora. A figura 7.8 ilustra estas máquinas.

Definição 7.3 Uma linguagem \mathcal{L} é chamada de decidível (ou recursiva, ou computável, ou solúvel) se, e somente se, existe uma decisor M tal que $L(M) = \mathcal{L}$.

Portanto, as linguagens Turing decidíveis são problemas para os quais existe alguma máquina que produz aceita ou rejeita como resposta. Um problema é decidível exatamente quando existe um algoritmo que possa solucioná-lo. Assim, a decidibilidade é um modo de formalização de um algoritmo.

Seja R o conjunto de todas as linguagens recursivas e RE o conjunto de todas linguagens recursivamente enumeráveis. Uma vez que todas as decisoras são Máquinas de Turing, $R \subseteq RE$. Entretanto, o que ainda não está claro é se $R = RE$. Lembre que $A_M = \{\langle M, w \rangle | w \in L(M)\}$. Neste sentido, $A_M \in RE$ porque ela é a linguagem da Máquina de Turing Universal M_U .

Lema 7.5 Se $R = RE$, então A_M é decidível.

Assuma que $R = RE$. Como $A_M \in RE$ tem-se pela hipótese que $A_M \in R$. Assim, A_M é decidível. \square

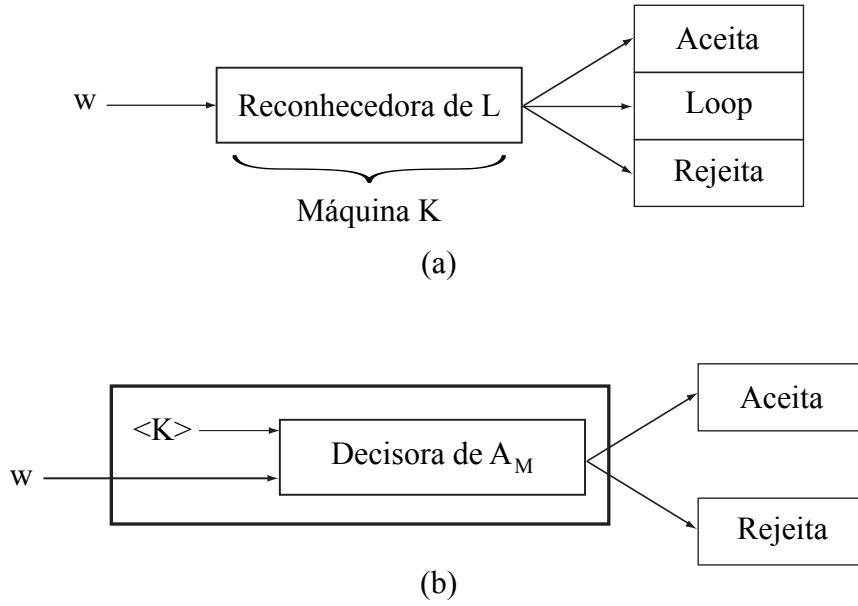


Figura 7.8: Máquinas: (a) reconhecedora; (b) decisora.

O passo seguinte é verificar o sentido contrário do lema, isto é, provar que se A_M é decidível, então $R = RE$. Assim, o que se deseja é mostrar que se A_M é decidível, então dada qualquer reconhecedora para uma linguagem L , é possível construir uma decisora para L . A figura 7.8 ilustra essa proposta, onde a descrição da máquina K é inserida em uma programação da máquina universal.

Lema 7.6 Se A_M é decidível, então $R = RE$.

Assuma que A_M é decidível, então existe uma decisora D tal que $L(D) = A_M$. Considere uma linguagem qualquer $\mathcal{L} \in RE$, logo deve existir uma reconhecedora de \mathcal{L} que será chamada de K . Em seguida, considere uma máquina M que aceita uma entrada w se D aceita $\langle K, w \rangle$; e que rejeita uma entrada w se D rejeita $\langle K, w \rangle$.

Acompanhe a figura 7.8 e observe que M aceita w se, e somente se, D aceita $\langle K, w \rangle$. Mas D aceita $\langle K, w \rangle$ se, e somente se, K aceita w . K aceita w se, e somente se, $w \in L(K) = \mathcal{L}$. Desta forma M aceita w se, e somente se, $w \in \mathcal{L}$, e portanto, $L(M) = \mathcal{L}$.

Agora considere o que acontece quando se executa a máquina M com uma entrada qualquer w . M constrói $\langle K, w \rangle$ e executa D sobre esta nova entrada. Como

D é uma decisora, ela eventualmente irá parar. Como M pára assim que D pára, tem-se que M pára qualquer que seja a entrada, e M é uma decisora também.

Juntando todas estas observações, veja que se M é uma decisora e $L(M) = \mathcal{L}$, então M é uma decisora para \mathcal{L} , e $\mathcal{L} \in R$. Isso mostra que $RE \subseteq R$. Combinando isto com o fato já observado que $R \subseteq RE$, tem-se que $R = RE$. \square

A construção matemática utilizada na demonstração do lema anterior é conhecida como redução. Para qualquer linguagem recursivamente enumerável \mathcal{L} , reduziu-se a pergunta ' $w \in \mathcal{L}$?' para a pergunta ' $\langle K, w \rangle \in A_M$?'. As reduções representam um papel importante nas análises de decidibilidade e reconhecibilidade de linguagens. Assumindo que uma redução pode ser realizada por uma Máquina de Turing, se A é redutível à B e B é decidível, então A também é decidível. Da mesma forma, se A é redutível à B e B é reconhecível, então A também é reconhecível.

Contudo, ainda não está claro é se $R = RE$, apenas que $R = RE$ se, e somente se, A_M é decidível. O teorema 7.3 resolve esta questão.

Teorema 7.3 A_M é indecidível.

Demonstração por contradição:

Suponha que A_M seja decidível e H uma máquina decisora para ela. Em seguida considere uma máquina D que entra em aceitação se a máquina H rejeita $\langle M, \langle M \rangle \rangle$, e em rejeição se H aceita $\langle M, \langle M \rangle \rangle$.

Uma vez que H é uma decisora para A_M , diz-se que H rejeita $\langle M, \langle M \rangle \rangle$ se, e somente se, $\langle M, \langle M \rangle \rangle \notin A_M$. Como $\langle M, \langle M \rangle \rangle \notin A_M$ e $\langle M, \langle M \rangle \rangle$ é uma codificação válida do par ordenado $\langle M, w \rangle$, pode-se concluir que $\langle M \rangle \notin L(M)$. Consequentemente, D aceita $\langle M \rangle$ se, e somente se, $\langle M \rangle \notin L(M)$. Assim, $L(D) = L_D$, a mesma linguagem de diagonalização L_D vista anteriormente nesta seção.

Como $L(D) = L_D$, pode-se concluir que $L_D \in RE$. Entretanto isto é impossível pois já foi demonstrado anteriormente (teorema 7.2) que $L_D \notin RE$. Uma vez que se obteve a contradição, tem-se que a hipótese está errada, logo A_M é indecidível. \square

As conclusões do teorema 7.2 sobre L_D , o teorema 7.3 sobre A_M , e a relação

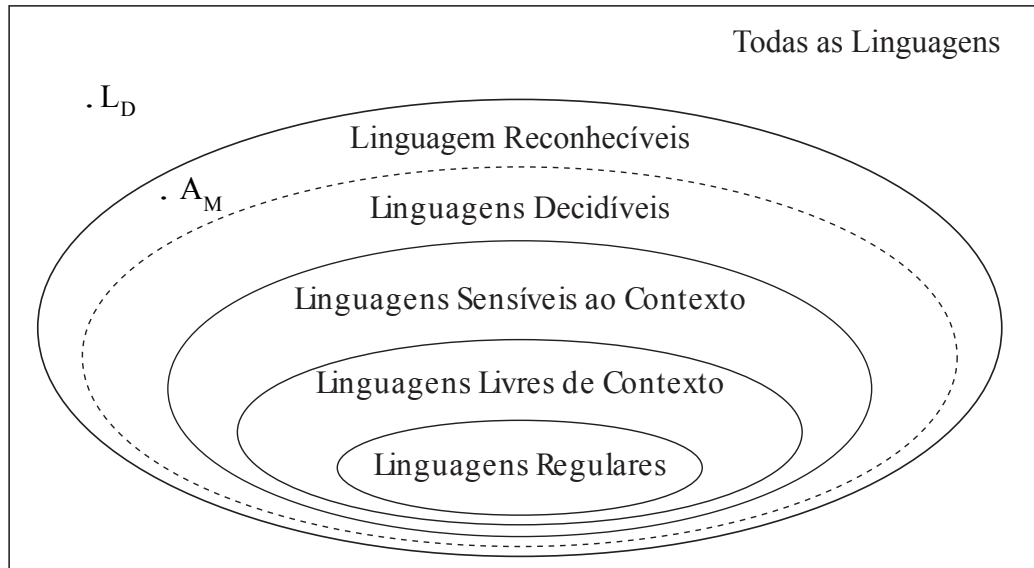


Figura 7.9: Limites da computabilidade.

$R \subseteq RE, R \neq RE \equiv R \subset RE$ também do teorema 7.3, são ilustradas na figura 7.9.

Por fim, chega-se finalmente a um dos problemas mais emblemáticos da computabilidade, o problema da parada.

Definição 7.4 Problema da Parada (HALT): dada uma Máquina de Turing M e uma string w , M pára ao processar w ? Formalmente:

$$HALT = \{\langle M, w \rangle | M \text{ pára quando processa } w\}$$

Note que M não precisa aceitar w , somente precisa parar quando processar w , isto é, não deve entrar em loop eterno. Também reveja a figura 7.7 e note como a relação entre A_M e $HALT$ é direta.

Dada uma Máquina de Turing qualquer M , considere uma máquina H , cuja entrada é $\langle M, w \rangle$ e que entra em estado de aceitação se M aceita/rejeita w . Observe que implicitamente está sendo utilizada uma Máquina de Turing Universal, e isso somente é possível porque A_M é reconhecível. Desta forma, H aceita $\langle M, w \rangle$ se, e somente se, M pára quando processa w . Logo, como $HALT$ é redutível à L_M , tem-se que $L(H) = HALT \in RE$, e que $HALT$ é indecidível.

Lembre também que se L e \bar{L} são RE , então $L \in R$. Observe também

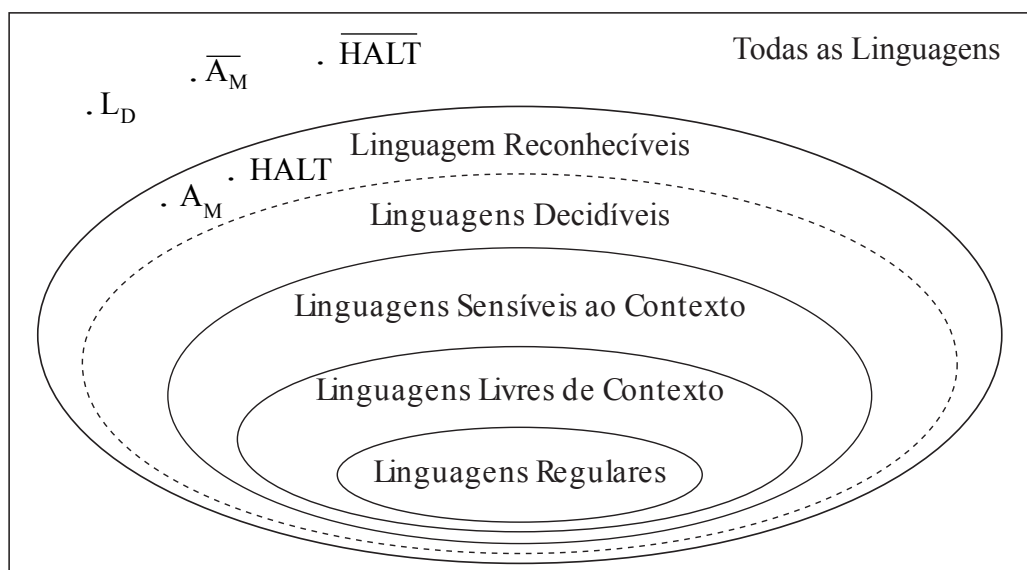


Figura 7.10: Limites da computabilidade e alguns problemas.

algumas manipulações lógicas:

Se $L \notin R$, então ao menos L ou \bar{L} não são RE

Se $L \notin R$ e $L \in RE$, então $\bar{L} \notin RE$

Assim, estes problemas podem ser representados no universo de todas as linguagens conforme a figura 7.10.

7.6 Reflexões de Fronteira

Esta é seção delicada, pois trata de um assunto no qual não há uma unanimidade. Deve ficar claro que o posicionamento do autor é favorável com relação a veracidade não só da Tese de Church-Turing, como também da Máquina de Turing ser o limite superior de tudo aquilo que uma máquina pode ser construída para computar. Entretanto, existe uma corrente de pesquisa que refuta estas ditas verdades. É importante conhecer estas abordagens que fazem parte da fronteira do conhecimento em computabilidade, e que existe certa discussão em torno do assunto. As palavras a seguir foram escolhidas com cuidado, portanto leia esta seção com moderação.

Nas seções anteriores ficou claro que a Máquina de Turing pode ser utilizada para resolver problemas complicados. Também foi apresentado que a adição de certos artifícios de hardware não estendem o poder de computação de uma Máquina de Turing, isto é, nenhum problema adicional torna-se resolvível por estas extensões. Por fim, foram colocadas certas situações onde a máquina não consegue resolver um problema. Estas são evidências de que a Máquina de Turing é o limite superior natural ao qual uma máquina de computar pode ser construída.

Também foram estudadas outras abordagens de computação (e suas variações) que se mostram serem equivalentes à Máquina de Turing, isto é, os conjuntos de problemas que podem ser resolvidos por estas abordagens e pela Máquina de Turing são os mesmos. Nos últimos 70 anos, foram propostos diversos modelos formais de computação. Existem modelos como gramáticas irrestritas, funções μ -recursivas, algoritmos de Markov, máquinas de Post, máquinas de acesso aleatório, máquinas com registros, λ -calculus, entre outros. Nenhuma destes modelos é mais poderoso que a Máquina de Turing, o melhor que eles podem fazer é terem um poder equivalente. Isto adiciona uma evidência de que foi encontrado o limite superior do que os computadores podem realizar.

Observe que esta consideração leva em conta extensões e modelos alternativos que estão alinhados conceitualmente com o mesmo contexto da Máquina de Turing. Pode-se chamar estas formalizações de modelos razoáveis de computação, ainda que a palavra razoável exija uma definição. Uma tentativa de enumerar as condições de contorno desta razoabilidade é apresentada a seguir:

- Existe um número finito de operações;
- Cada operação é simples, precisa e sistematizável, isto é, não exige inspiração, intuição ou introspecção;
- Cada operação deve poder ser realizada com um esforço finito a cada passo, como por exemplo, a máquina não pode responder a uma quantidade infinita de tarefas em um passo finito;
- Máquinas ou programas devem ser finitos;

- Máquinas ou programas devem ser uniformes, ou seja, para determinados problemas, uma máquina ou programa deve aceitar todos os tamanhos de entrada, isto é, não são utilizados máquinas ou programas diferentes de acordo com os diferentes tamanhos de entrada;
- Cada máquina ou programa, se operados corretamente, produzem seus resultados em um número finito de passos.

As equivalências entre os modelos razoáveis de computação sugerem a idéia de que conceitos distintos, apesar de serem bastante diferentes, levaram a conclusões equivalentes. Esta situação é corroborada pela Tese de Church-Turing. Entretanto, conforme colocado na seção 1.2, esta é apenas uma tese e não um teorema. Se fosse um teorema, deveria existir uma prova. Se fosse encontrada alguma proposta de modelo de computação alinhado com a idéia de razoabilidade e que resolvesse problemas não resolvíveis pela Máquina de Turing, então se teria uma refutação desta tese e toda discussão se encerraria.

É comum surgirem novas propostas de modelos de computação que se dizem serem capazes de resolver problemas que a Máquina de Turing não consegue resolver, chamados de super-Turing. Em geral, estes modelos têm se mostrado equivocados em sustentar certa incapacidade da Máquina de Turing, porém existem alguns casos em que isto é verdade. Existem modelos de computação que resolvem problemas que a Máquina de Turing não consegue resolver, e a hipercomputação é um exemplo destes modelos. A razão pela qual estes tipos de modelos não refutam definitivamente a Tese de Church-Turing é porque, nos casos em que estes modelos solucionam problemas que a máquina de Turing não resolve, não há a chamada razoabilidade. Em tais casos, o erro mais comum é comparar modelos não uniformes com modelos uniformes [100], ou comparar modelos que permitem uma quantidade infinita de processamento (hipercomputação) com modelos que não permitem (Máquinas de Turing). Contudo, o caminho para refutação da Tese de Church-Turing, se existir, é encontrar um modelo de computação razoável que resolva algo que a Máquina de Turing não é capaz.

Em outras situações, existem aqueles que proclamam ter encontrado maneiras de utilizar a Máquina de Turing para resolver problemas tidos como não

computáveis. Deste modo, parece que a Tese de Church-Turing foi sobrepujada de alguma forma. Entretanto, o que se observa nestas situações é a solução de casos particulares de um problema. O ponto principal aqui é que a Teoria da Computabilidade não afirma que casos particulares de problemas não computáveis não possam ser resolvidos. Por exemplo, não é porque o problema da parada é não computável que não possam ser criados programas que o resolvam em casos particulares.

Desta forma, deve-se refletir sobre tudo que foi estudado até o momento e evitar armadilhas conceituais. Só com procedimentos atentos à estas questões é que se pode confirmar ou refutar a Tese de Church-Turing. Aumentar a capacidade de computação é um problema fascinante. Um trabalho bastante interessante sobre tentativas de aumentar a capacidade de computação é o texto de Aaronson [101]. Neste material, ele apresenta informalmente uma série de possibilidades no mínimo instigante, como por exemplo, a de acelerar um computador até a velocidade da luz de modo que o tempo passe mais vagarosamente para ele do que para seu operador. Assim, aos olhos do operador, a tarefa é executada neste computador é terminada mais rapidamente pois gasta um tempo menor que a tarefa não acelerada.

7.7 Conclusões e leituras recomendadas

Turing propôs um modelo abstrato de computação, conhecido como máquina de Turing, com o objetivo de explorar os limites da capacidade de expressar soluções de problemas. Sob esta ótica, a máquina de Turing é uma proposta de definição formal da noção intuitiva de algoritmo. Por conseguinte, trata-se de uma tentativa de responder ao questionamento sobre o que pode, e o que *não* pode ser computado.

Neste capítulo foram apresentados modelos matemáticos diferentes que executam tarefas neste contexto: decidir, semi-decidir, reconhecer, construir funções. A adição de certos aspectos (quantidade de fitas, leitoras e acesso aleatório) não aumenta o conjunto de tarefas que podem ser executadas por uma máquina simples de Turing. A capacidade de computação representada pela máquina de Turing é o limite máximo que pode ser atingido por qualquer dispositivo de computação. Qualquer outra forma de expressar algoritmos terá, no máximo, a mesma capacidade

computacional da máquina de Turing.

7.8 Exercícios

7.1 Suponha que $M = \langle k, \Sigma, \delta, s, F \rangle$, onde $k = \{q_0, q_1, h\}$, $\Sigma = \{a, b, \sqcup, \triangleright\}$, $s = q_0$, $F = \{h\}$ e δ é dada conforme a seguir:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_1, b)
q_0	b	(q_1, a)
q_0	\sqcup	(h, \sqcup)
q_0	\triangleright	(q_0, \rightarrow)
q_1	a	(q_0, \rightarrow)
q_1	b	(q_0, \rightarrow)
q_1	\sqcup	(q_0, \rightarrow)
q_1	\triangleright	(q_1, \rightarrow)

(a) Descreva a computação de M , iniciando na configuração $(q_0, \triangleright \underline{a}abbba)$. (b) Descreva informalmente o que M faz quando iniciada com q_0 em qualquer quadro da fita de entrada.

7.2 Suponha que $M = \langle k, \Sigma, \delta, s, F \rangle$, onde $k = \{q_0, q_1, h\}$, $\Sigma = \{a, b, \sqcup, \triangleright\}$, $s = q_0$, $F = \{h\}$ e δ é dada conforme a seguir:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_1, \leftarrow)
q_0	b	(q_0, \rightarrow)
q_0	\sqcup	(q_0, \rightarrow)
q_1	a	(q_1, \leftarrow)
q_1	b	(q_2, \rightarrow)
q_1	\sqcup	(q_1, \leftarrow)
q_2	a	(q_2, \rightarrow)
q_2	b	(q_2, \rightarrow)
q_2	\sqcup	(h, \sqcup)

Observe que foram omitidas as transições $\delta(q, \triangleright) = (q, \rightarrow)$. Descreva a computação de M , iniciando na configuração $(q_0, \triangleright a \underline{b} b \sqcup b b \sqcup \sqcup \sqcup a b a)$.

7.3 Suponha que $M = \langle k, \Sigma, \delta, s, F \rangle$, onde $k = \{q_0, q_1, h\}$, $\Sigma = \{a, b, \sqcup, \triangleright\}$, $s = q_0$, $F = \{h\}$ e δ é dada conforme a seguir:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_2, \rightarrow)
q_0	b	(q_3, a)
q_0	\sqcup	(h, \sqcup)
q_0	\triangleright	(q_0, \rightarrow)
q_1	a	(q_2, \rightarrow)
q_1	b	(q_2, \rightarrow)
q_1	\sqcup	(q_2, \rightarrow)
q_1	\triangleright	(q_1, \rightarrow)
q_2	a	(q_1, b)
q_2	b	(q_3, a)
q_2	\sqcup	(h, \sqcup)
q_2	\triangleright	(q_2, \rightarrow)
q_3	a	(q_4, \rightarrow)
q_3	b	(q_4, \rightarrow)
q_3	\sqcup	(q_4, \rightarrow)
q_3	\triangleright	(q_3, \rightarrow)
q_4	a	(q_2, \rightarrow)
q_4	b	(q_4, \rightarrow)
q_4	\sqcup	(h, \sqcup)
q_4	\triangleright	(q_4, \rightarrow)

Descreva a computação de M , iniciando na configuração $(q_0, \triangleright a \underline{a} a b b b a a)$.

7.4 Sugira uma modificação na máquina do Exemplo 7.1, Figura 7.2, para que seja feita a cópia da string de 1's a direita de B para a posição à direita de A , a partir da configuração $\cdots 0A0000\underline{B}1110\cdots$.

7.5 Deseja-se contruir uma máquina comparadora de 1's a partir dos requisitos a seguir:

$$\dots 0 \underbrace{A 1111}_{\alpha} 00 \dots B \underbrace{111}_{\beta} 0 \dots$$

- (a) Pára em um estado q se $\alpha = \beta$
- (b) Pára em um estado q' se $\alpha \neq \beta$

7.6 Construa uma Máquina de Turing $M = \langle k, \Sigma, \delta, s, \{h\} \rangle$, que seja uma dobradora, isto é, dadas strings do tipo $0 \underbrace{111 \dots 1}_{n} 000 \dots$ ela construa strings do tipo $0 \underbrace{111 \dots 1}_{n} \underbrace{111 \dots 1}_{n} 000 \dots$.

7.7 Construa uma máquina dobradora tal como a do Exercício 7.6, mas que faça a dobra de modo ligeiramente diferente. Quando subetido uma string do tipo $0 \underbrace{111 \dots 1}_{n} 000 \dots$, ela constrói strings do tipo $0 \underbrace{111 \dots 1}_{n} 0 \underbrace{111 \dots 1}_{n} 000 \dots$.

7.8 Escrever o diagrama de estados de uma máquina de Turing capaz de reconhecer se dois string são palíndromes (o termo correto é capicua, mas palíndrome tornou-se consagrado pelo uso). Os strings são separados por um único símbolo 0, e a posição inicial da cabeça leitora é definida tal como se segue:

$\dots 000 \underline{a}bb0bba000 \dots \dashrightarrow$ Aceita, pois abb é o contrário de bba
 $\dots 000 \underline{a}ba0bba000 \dots \dashrightarrow$ Não aceita

7.9 Escrever o diagrama de estados de uma máquina de Turing capaz de somar 1 unidade a um número natural não nulo escrito em binário. Considere a posição da fita conforme descrito a seguir:

$$\begin{aligned} 1 &= 001 \dashrightarrow \dots \sqcup \underline{1} \sqcup \dots \\ 2 &= 010 \dashrightarrow \dots \sqcup \underline{1}0 \sqcup \dots \\ 3 &= 011 \dashrightarrow \dots \sqcup \underline{1}1 \sqcup \dots \\ 4 &= 100 \dashrightarrow \dots \sqcup \underline{1}00 \sqcup \dots \\ 5 &= 101 \dashrightarrow \dots \sqcup \underline{1}01 \sqcup \dots \\ 6 &= 110 \dashrightarrow \dots \sqcup \underline{1}10 \sqcup \dots \end{aligned}$$

7.10 Escrever o diagrama de estados de uma máquina de Turing capaz de multiplicar por dois um número binário, compreendido entre os símbolos A e B e com tamanho qualquer, escrevendo resultado a direita de B . A posição inicial da cabeça leitora é definida tal como se segue e \sqcup é o símbolo em branco da fita. $\dots \sqcup \underline{A}1011B \sqcup \dots$

7.11 Construa uma máquina de Turing que reconheça strings da linguagem $L(M) = \{w \in \{0, x, \sqcup\}^* \mid w = 0^{2^n} \text{ onde } n \geq 0\}$, com a cabeça leitora localizada na posição do 0 mais a esquerda da fita, e com espaços em branco indicados por \sqcup .

7.12 Construa uma Máquina de Turing que incremente em uma unidade um número binário qualquer. Em seguida, construa outra máquina que faça o incremento de duas unidades em um número binário qualquer. A posição inicial da cabeça leitora é a do bit mais significativo.

7.13 Uma empresa fabricante de GPUs (*Graphical Processing Unit*) lançou uma placa de vídeo com m núcleos de processamento paralelo que funcionam de forma síncrona e independente. Não há compartilhamento de memória entre os núcleos de processamento, e a lógica implementada é n -ária. Determine quantos símbolos são necessários para implementar uma máquina de Turing de fita simples que tenha o mesmo poder computacional que esta GPU (apresente o cálculo desta determinação).

Referências Bibliográficas

- [1] HOPCROFT, J. E., *Theory os Machines and Compuatations*, chapter An n log n algorithm for minimizing states in a finite automaton, Academic Press, pp. 189–196, 1971.
- [2] MONTEIRO, A. A., PAULO, J. D. S., *Aritmética Racional*. Lisboa, Livraria Avelar Machado, 1945.
- [3] IFRAH, G., *Os Números: a história de uma grande invenção*. São Paulo, Globo S.A., 2005.
- [4] DEDEKIND, R., *Was sind und was sollen die Zahlen*, v. 3, *Gesammelte Mathematische Werke*. New York, Chelsea Publishing Company, 1969. pp. 335-391.
- [5] GÖDEL, K., *Collected Works*, v. 2, *Gesammelte Mathematische Werke*. Oxford, Oxford University Press, 1990.
- [6] ISRAEL, D., “Reflections on Gödel’s and Gandy’s Reflection on Turing’s Thesis”, *Minds and Machines*, v. 12, n. 2, pp. 181–201, 2002.
- [7] SOBRINHO, J. Z., “Aspectos da Tese de Church-Turing”, *Revista Matemática Universitária - USP*, v. 1, n. 6, pp. 1–23, 1987.
- [8] MCDERMOTT, D., “Artificial Intelligence Meets Natural Stupidity”, *SI-GART Newsletter*, v. 57, pp. 4–9, April 1976.
- [9] SETTI, M. D. O. G., *O Processo de Discretiza çã o do Raciocínio Matemático na Tradu çã o para o Raciocínio Computacional*, Report, Universidade Federal do Paraná, 2009.

- [10] GUERREIRO, G., “A Vanguarda Matemática e os Limites da Razão”, *Scientific American Brasil*, v. 5, n. 12, pp. 39–56, 2007. Coleção Gênios da Ciência.
- [11] SMULLYAN, R., *Gödel’s Incompleteness Theorems*. Oxford University Press, 1992.
- [12] GÖDEL, K., *The Undecidable*, chapter On Formally Undecidable Propositions of Principia Mathematica and Related Systems, New York, Raven Press, pp. 5–38, 1965.
- [13] WHITEHEAD, A. N., RUSSELL, B., *Principia Mathematica*. Londres, Cambridge University Press, 1913.
- [14] NAGEL, E., NEWMAN, J. R., *Gödel’s Proof*. USA, Routledge, 1989.
- [15] GÖDEL, K., “Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme - On Formally Undecidable Propositions of Principia Mathematica and Related Systems”, *Kurt Gödel: Collected Works*, v. 1, n. 1, pp. 144–195, 1986. Tradução para o inglês por Martin Hirzel. www.research.ibm.com/people/h/hirzel/papers/canon00-goedel.pdf [capturado em 13 de agosto de 2011].
- [16] MELO, A. C. V. D., SILVA, F. S. C. D., *Modelos Clássicos de Computação*, Coleção Schaum. São Paulo, Thomson, 2006.
- [17] KUBRUSLY, R. D. S., *Uma viagem informal ao Teorema de Gödel ou (O preço da matemática é o eterno matemático)*, Report, Universidade Federal do Rio de Janeiro, 2007. IM/UFRJ.
- [18] SMULLYAN, R., *Recursion Theory for Metamathematics*. Oxford University Press, 1993.
- [19] SMULLYAN, R., *What’s the Name of This Book*. Penguin Books, 1978.
- [20] SMULLYAN, R., *Forever Undecided: A Puzzle Guide to Gödel*. Oxford University Press, 1987.
- [21] GOLDSTEIN, R., *Incompleteness: The Proof and Paradox of Kurt Gödel*. W. W. Norton Company, Inc., 2005.

- [22] HOFSTADTER, D. R., *Gödel, Escher e Bach: an Eternal Golden Braid*. Nova Iorque, Basic Books, 1979.
- [23] Rodríguez-Consuegra, F. A. (ed.), *Kurt Gödel - Unpublished Philosophical Essays*. Berlin, Birkhäuser Verlag, 1995.
- [24] Feferman, S., *et al.* (eds.), *Kurt Gödel - Collected Works*, v. I, II, III. New York, Oxford University Press, 1986.
- [25] WANG, H., *Reflections on Kurt Gödel*. Cambridge, Massachusetts, The MIT Press, 1988.
- [26] SUPPES, P., *Axiomatic Set Theory*. New York, Dover, 1972.
- [27] LIPSCHUTZ, S., *Teoria Elementar dos Conjuntos*, Cole çã o Schaum. Sã o Paulo, McGraw-Hill do Brasil, 1990.
- [28] SUPPES, P., *Axiomatic Theory Set*. USA, Van Nostrand Company Inc., 1960.
- [29] MELLO, F. L. D., CARVALHO, R. L. D., “Knowledge Geometry”, *Journal of Information and Knowledge Management*, v. 14, pp. 1550028, 2015.
- [30] STOLL, R. R., *Set Theory and Logic*. USA, Dover Publications Inc., 1961.
- [31] MIRAGLIA, F., *Teoria dos Conjuntos: um mínimo*. Brasil, Edusp - Editora da Universidade de São Paulo, 1992.
- [32] HALMOS, P., *Teoria Ingênua dos Conjuntos*. Brasil, Edusp - Editora da Universidade de São Paulo, 1970.
- [33] GRATZER, G., *Universal Algebra*. USA, Van Nostrand Company Inc., 1968.
- [34] COHN, P. M., *Universal Algebra*. USA, Harper and Row, 1965.
- [35] GALLIER, J. H., *Logic for Computer Science*. USA, John Wiley and Sons Inc., 1987.
- [36] PREPARATTA, F. P., YEH, R. T., *Introduction to Discreate Structures for Computer Science and Engineering*. USA, Addison-Wesley Publishing Company, 1973.

- [37] SHOENFIELD, J. R., *Degrees of Unsolvability*. North-Holland Publishing Company, 1971.
- [38] BRAINERD, W. S., LANDWEBER, L. H., *Theory of Computation*. USA, John Wiley and Sons, 1974.
- [39] EILENBERG, S., ELGOT, C., *Recursiveness*. New York: Academic Press, 1970.
- [40] CARVALHO, R. L. D., OLIVEIRA, C. M. G. M. D., *Modelos de Computação e Sistemas Formais*, 11^a Escola de Computação. Rio de Janeiro, Universidade Federal do Rio de Janeiro, 1998.
- [41] BRAINERD, W. S., LANDWEBER, L. H., *Theory of Computation*. John Wiley & Sons, 1974.
- [42] KLEENE, S. C., *Introduction to Metamathematics*. D. Van Nostrand Company, Inc., 1952.
- [43] Rogers Jr., H., *Theory of Recursive Functions and Effective Computability*. USA, McGraw-Hill Book Company, 1967.
- [44] DAVIS, M., *Computability and Unsolvability*. New York, Dover, 1983.
- [45] BOOLOS, G. S., JEFFREY, R. C., *Computability and Logic*. Cambridge University Press, 1974.
- [46] MARTIN DAVIS, R. S., WEYUKER, E. J., *Computability Complexity and Languages*. New York, Academic Press, 1994.
- [47] MALLOZZI, J. S., LILLO, N. J. D., *Computability with Pascal*. New Jersey, Prentice-Hall, Inc., 1984.
- [48] TURING, A. M., *The Undecidable*, chapter On Computable Numbers, with an Application to the Entscheidungsproblem, New York, Raven Press, pp. 115–151, 1965.
- [49] ELGOT, C. C., ROBINSON, A., “Random-access stored-program machines, an approach to programming languages”, *Journal of the ACM*, v. 11, pp. 365–399, 1964.

- [50] PREPARATTA, F. P., YEH, R. T., *Introduction to Discrete Structures for Computer Science and Engineering*. Addison-Wesley Publishing Company, 1973.
- [51] HENNIE, F., *Introduction to Computability*. Addison-Wesley Publishing Company, 1977.
- [52] NELSON, R. J., *Introduction to Automata*. USA, Jonh Wiley & Sons, Inc., 1968.
- [53] CARVALHO, R. L. D., *Máquinas, Programas e Algoritmos*, 2^a Escola de Computação. Campinas, Universidade Estadual de Campinas, 1981.
- [54] MENDELSON, E., *Introduction to Mathematical Logic*, Cole Mathematics Series. 3 ed. The Wadsworth and Brooks, 1987.
- [55] MANIN, Y. I., *A Course in Mathematical Logic*, Graduate Texts in Mathematics 53. 1 ed. Springer-Verlag, 1977.
- [56] HOMER, S., SELMAN, A. L., *Computability and Complexity Theory*, Texts in Computer Science. 2 ed. Springer, 2011.
- [57] CUTLAND, N. J., *Computability: An introduction to recursive function theory*. Cambridge University Press, 1980.
- [58] SIPSER, M., *Introduction to The Theory of Computation*, Course Technology Series. 2 ed. Thomson, 2006.
- [59] WALTER CARNIELLI, R. L. E., *Computability: computable functions, logic and the foundations of mathematics*. Belmont, Wadsworth and Brooks, 1989.
- [60] TARSKI, A., *Logic, semantics, metamathematics*, chapter Fundamental concepts of the methodology of the deductive sciences, London, Oxford at the Clarendon Press, pp. 60–109, 1969.
- [61] ADAM YOUNG, M. Y., *Malicious Cryptography: Exposing Cryptovirology*. John Wiley and Sons Inc., 2004.

- [62] BONFANTE, G., KACZMAREK, M., MARION, J.-Y., *A Classification of Viruses through Recursion Theorems*, volume 4497 of Lecture Notes in Computer Science. 2 ed. CiE 2007, 2007.
- [63] MACHTEY, M., YOUNG, P., *An Introduction to the General Theory of Algorithms*. New York, North Holland, 1978.
- [64] ROBINSON, J. A., “A machine oriented logic based on the resolution principle”, *J. Assoc. Comput.*, v. 12, pp. 23–41, 1965.
- [65] ROBINSON, J. A., “Automatic deduction with hyper-resolution”, *Internat. J. Comput. Math.*, v. 1, pp. 227–234, 1965.
- [66] GILMORE, P. C., “A proof method for quantification theory: Its justification and realization”, *IBM Journal of Research and Development*, v. 4, n. 1, pp. 28–35, 1960.
- [67] DAVIS, M., PUTNAM, H., “A computing procedure for quantification theory”, *Journal of the ACM*, v. 7, n. 3, pp. 201–215, 1960.
- [68] CHANG, C.-L., LEE, R. C.-T., *Symbolic Logic and Mechanical Theorem Proving*. Academic Press Inc, 1973.
- [69] KLEENE, S. C., *Mathematical Logic*. Wiley, 1967.
- [70] HILBERT, D., ACKERMANN, W., *Prinmciples of Mathematical Logic*. Chelsea, 1950.
- [71] MCCAWLEY, J. D., *Everything That Linguists Have Always Wanted To Know About Logic*. 2 ed. The University of Chicago Press, 1993.
- [72] SUPPES, P., *Introduction to Logic*. D. van Nostrand, 1966.
- [73] RUSSELL, B., *A Filosofia do Atomismo Lógico*, *Lógica e Conhecimento*. 1 ed. Abril Cultural, 1974. (Os Pensadores, 42).
- [74] RUSSELL, B., *Significado e Verdade*. 1 ed. Zahar, 1978.
- [75] POPPER, K. R., *A Lógica da Pesquisa Científica*. 2 ed. Cultrix, 1974.

- [76] LAKATOS, I., , MUSGRAVE, A., *A Crítica e o Desenvolvimento do Conhecimento*. 1 ed. EDUSP, Cultrix, 1979. Tradução: M. O. Caiado.
- [77] WANG, H., *From Mathematics to Philosophy*. 1 ed. Cultrix, 1974.
- [78] GREEN, C. C., *The Application of Theorem Proving to Question-Answering Systems*. Ph.D. dissertation, Stanford, June 1969. AI Project MEMO AI-96.
- [79] LOVELAND, D. W., *Automated Theorem Proving: a Logical Basis*. 1 ed. North Holland, 1978.
- [80] HUGHES, G. E., LONDEY, D. G., *The Elements of Formal Logic*. USA, Methuen and Co Ltd, 1965.
- [81] BOOK, R. V., OTTO, F., *String-Rewriting Systems*. USA, Springer-Verlag, 1993.
- [82] HOPCROFT, J. E., ULLMAN, J. D., *Introduction to Automata Theory, Language and Computation*. USA, Addison-Wesley Publishing Company, 1979.
- [83] HOPCROFT, J. E., ULLMAN, J. D., *Formal Languages and their Relation to Automata*. USA, Addison-Wesley Publishing Company, 1969.
- [84] CURRY, H. B., *Foundations of Mathematical Logic*. New York, Academic Press, 1977.
- [85] SMULLYAN, R., *Theory of Formal Systems*. USA, Princeton, 1961.
- [86] BERSTEL, J., BOASSON, L., CARTON, O., *et al.*, *Handbook of Automata: from Mathematics to Applications*, chapter Minimization of automata, European Mathematical Society, pp. 189–196, 2010.
- [87] BEAL, M. P., CROCHEMORE, M., “Minimizing incomplete automata”, *Workshop on Finite State Methods and Natural Language Processing*, , september 2008. Ispra.
- [88] VALMARI, A., LEHTINEN, P., “Efficient minimization of DFAs with partial transition”, *Proc. 25th Symp. Theoretical Aspects of Comp. Sci.*, v. 08001, pp. 645–656, 2008. S. Albers and P. Weil, editors.

- [89] PAPADONIKOLAKIS, M., BOUGANIS, C.-S., CONSTANTINIDES, G.,
“Performance comparison of GPU and FPGA architectures for the SVM training problem”, *IEEE International Conference on FieldProgrammable Technology*, pp. 388–391, 2009.
- [90] MU, S., WANG, C., LIU, M., *et al.*, “Evaluating the potential of graphics processors for high performance embedded computing”, *Proc. IEEE Design, Automation and Test in Europe Conference and Exhibition*, pp. 709–714, 2011.
- [91] KAI HWANG, F. A. B., *Computer Architecture and Parallel Processing*. McGraw-Hill, 1984.
- [92] AGARWAL, P., KRISHNAN, S., MUSTAFA, N., *et al.*, *Algorithms - ESA 2003, Lecture Notes in Computer Science*, chapter Streaming Geometric Optimization Using Graphics Hardware, Springer Berlin-Heidelberg, pp. 115–151, 2003.
- [93] TANENBAUM, A. S., *Organização Estruturada de Computadores*. 3 ed. Prentice Hall do Brasil, 1997.
- [94] BACKUS, J., “Can Programming be Liberated from von Neumann Style? A functional style and its algebra of program”, *ACM Turing Award Lecture, Communications of the ACM*, v. 21, n. 8, pp. 613–641, 1978.
- [95] OWENS, J. D., LUEBKE, D., GOVINDARAJU, N., *et al.*, “A Survey of General-Purpose Computing on Graphics Hardware”, *Eurographics 2005, State of the Art Reports*, pp. 21–51, 2005.
- [96] GUSTAFSON, J. L., “Reevaluating Amdahl’s law”, *Communications of the ACM*, v. 5, n. 31, pp. 532, 1988.
- [97] HANDLER, W., *Parallel Processing Systems, an advanced course*, chapter Innovative computer architecture - how to increase parallelism but not complexity, Cambridge University Press, pp. 1–41, 1982.
- [98] LOBUR, J., NULL, L., *The Essentials of Computer Organization And Architecture*. Jones and Bartlett Pub, 2006.

- [99] LEWIS, H. R., PAPADIMITRIOU, C. H., *Elements of the Theory of Computation*. 2 ed. New York, Prentice-Hall, 1998.
- [100] DUNNE, P., *Computability Theory: Concepts and Applications*. Ellis Horwood, 1991.
- [101] AARONSON, S., “NP-complete Problems and Physical Reality”, *ACM SIGACT News*, , march 2005. Complexity Theory Column 46.