Capítulo 1

Teoria da Computação: Uma Introdução à Complexidade e à Lógica Computacional

Celina M. H. de Figueiredo e Luís C. Lamb
Universidade Federal do Rio de Janeiro e Universidade Federal do Rio Grande do Sul

Abstract

Computing theory is an abstract and mathematical research area, motivated by the challenges of computer science practice. Perhaps the main objective of theoretical computer science is to explore and understand the nature of computing, contributing to development of efficient and effective methodologies. This course introduces the basics of two fundamental subareas of theoretical computer science: computational complexity and computational logic. Such areas have significantly contributed to the development of computer science, not only from a foundational, but also from an applied perspective.

Resumo

A teoria da computação é uma área de pesquisa abstrata e matemática, e é motivada pelos desafios da prática da computação. O objetivo da teoria da computação é explorar e entender a natureza da computação, de modo a oferecer metodologias eficientes e corretas. Este curso apresenta uma introdução à duas subáreas fundamentais da teoria da computação: complexidade e lógica computacional. Estas duas áreas têm contribuído significativamente para o desenvolvimento da Ciência da Computação, não somente do ponto de vista fundamental, mas também sob uma perspectiva de aplicações.

Sumário

1.1. Uma Introdução à Complexidade - Celina M. H. de Figueiredo

Abstract

" To solve or to verify?" It is a question worth a million dollars. In 2000, the Clay Institute for Mathematics distinguished seven problems considered central to the progress of mathematics, calling them The Millennium Problems. The solution of each problem corresponds to a prize of one million dollars. One of the seven selected problems is a Theory of Computation problem: Is there a question whose answer can be quickly verified but the answer requires a long time to be found? This Millennium Problem, known as P versus NP, is the central problem in the Computational Complexity area where we try to rank the difficulty of the problems according to the efficiency of possible solutions through computer algorithms. We discuss the P versus NP problem, by classifying challenging problems in Polynomial or NP-complete and defining classes of problems that admit a dichotomy, where each problem in the class is classified as polynomial or NP-complete.

Resumo

"Resolver ou Verificar?" é uma pergunta que vale um milhão de dólares. No ano 2000, o Instituto Clay para Matemática distinguiu sete problemas considerados centrais para o progresso da matemática, chamando-os de Os Problemas do Milênio. A solução de cada problema corresponde a um prêmio de um milhão de dólares. Um dos sete problemas selecionados é um problema de Teoria da Computação: existe pergunta cuja resposta pode ser verificada rapidamente mas cuja resposta requer muito tempo para ser encontrada? Esse Problema do Milênio, conhecido como P versus NP, é o problema central na área de Complexidade Computacional, onde tentamos classificar a dificuldade dos problemas de acordo com a eficiência das possíveis soluções através de algoritmos computacionais. Discutiremos o problema P versus NP, através da classificação de problemas desafiadores em polinomial ou NP-completo e na definição de classes de problemas que admitem um resultado de dicotomia, onde cada problema da classe é classificado como polinomial ou NP-completo.

1.1.1. O Problema do Milênio

O computador é um aliado imprescindível para resolver os complexos problemas que surgem em biologia, química, física, economia, áreas nas quais pesquisadores se dedicam à modelagem e à simulação de problemas de larga escala com uso de computadores. Porém, há problemas que têm resistido à habilidade dos programadores. À medida que resolvemos problemas cada vez maiores e mais complexos por meio de enorme poder computacional e algoritmos engenhosos, os problemas resistentes e desafiadores ganham destaque. Nesse sentido, a teoria que propõe o problema do milênio ajuda a entender as limitações computacionais fundamentais. Uma questão de interesse teórico explica a dificuldade prática de problemas formulados em toda a comunidade científica.

No ano 2000, o Instituto Clay para Matemática distinguiu sete problemas considerados centrais para o progresso da matemática, chamando-os de Os Problemas do Milênio. A solução de cada problema corresponde a um prêmio de um milhão de dólares. Um dos sete problemas selecionados é um problema de Teoria da Computação: existe pergunta cuja resposta pode ser verificada rapidamente mas cuja resposta requer muito tempo para ser encontrada? Esse Problema do Milênio, conhecido como P versus NP, é o problema central na área de Complexidade Computacional, onde tentamos classificar a dificuldade dos problemas de acordo com a eficiência das possíveis soluções através de algoritmos computacionais [22].

O problema P versus NP pode ser informalmente resumido na pergunta "Resolver ou Verificar?", que é ilustrada pela seguinte estória:

> Em 1903, o matemático americano Frank Cole provou que o número $2^{67} - 1 = 147573952589676412927$ não é primo exibindo a fatoração $193707721 \times 761838257287$. É simples (embora tedioso se feito manualmente) calcular $2^{67} - 1$, calcular o produto $193707721 \times 761838257287$ e verificar que dão o mesmo número. Já encontrar essa fatoração é difícil. Cole disse que ele levou três anos trabalhando aos domingos.

"Resolver ou Verificar?" é uma pergunta que vale um milhão de dólares.

Duas campanhas: vacinação e pavimentação

Imagine que uma campanha de vacinação no estado do Rio de Janeiro precisa visitar cada uma das capitais dos seus 50 municípios. Por restrições de custo, a equipe responsável, saindo da cidade do Rio de Janeiro, precisa visitar cada uma das outras 49 cidades e retornar à cidade de partida, realizando um circuito que usa as rodovias do estado e que visita cada capital de município exatamente uma vez. Será que tal circuito pode

ser realizado no estado do Rio de Janeiro? Este circuito corresponde ao problema do Ciclo Hamiltoniano, formulado por William Hamilton em 1856, e conhecido também como o problema do Caixeiro Viajante. Uma condição suficiente para que tal circuito do vacinador exista é que cada cidade tenha muitas rodovias do estado que passem por ela mas esta condição não é necessária. Uma condição necessária é a inexistência de cidade gargalo, isto é, uma cidade tal que todas as rotas que conectam outras duas cidades usando rodovias do estado precisam passar por esta cidade gargalo mas esta condição não é suficiente. No estado do Rio de Janeiro, o município de Cabo Frio é um gargalo que isola o município de Búzios e impossibilita um circuito do vacinador. O município de Angra é outro gargalo e isola o município de Paraty.

Imagine agora que, ao invés de um vacinador que saindo da cidade sede da campanha visita num circuito um conjunto de cidades passando por cada cidade visitada uma única vez e retornando à cidade de partida, você considera uma campanha para recuperar a pavimentação das rodovias do estado. Por restrições de custo, a equipe responsável, saindo da cidade sede da campanha, realiza um circuito que percorre cada rodovia exatamente uma vez e retorna à cidade sede de partida. Será que tal circuito pode ser realizado no estado do Rio de Janeiro? Observe que o circuito do vacinador visita cada cidade do estado exatamente uma vez enquanto que o circuito do pavimentador percorre cada rodovia do estado exatamente uma vez. Possivelmente o vacinador deixa de percorrer alguma rodovia do estado enquanto que possivelmente o pavimentador visita uma cidade do estado mais de uma vez. O circuito do vacinador corresponde a uma permutação do conjunto das cidades do estado enquanto que o circuito do pavimentador corresponde a uma permutação do conjunto das rodovias do estado. Ambos problemas buscam a solução, o circuito desejado, num espaço com um número enorme de circuitos viáveis, da ordem de 50! circuitos, onde consideramos todas as possíveis permutações. É impraticável resolver o problema através de um algoritmo de força bruta que testa cada uma das permutações candidatas. Buscamos uma propriedade matemática do problema que reduza drasticamente este espaço exponencial de busca. Leonhard Euler resolveu definitivamente o problema do pavimentador ao publicar em 1736 o que consideramos o primeiro artigo científico da área de Teoria dos Grafos. Neste artigo, foi apresentada uma propriedade que caracteriza, uma propriedade que é ao mesmo tempo necessária e suficiente, para a existência de tal circuito do pavimentador: o número de rodovias que passam por cada cidade é par. Euler reduziu drasticamente o espaço exponencial de busca constituído de todas as possíveis permutações das rodovias. A poderosa condição de Euler fornece a solução após a fácil verificação dos graus das cidades.

Em Matemática e em Computação, a Teoria dos Grafos é a área que estuda estruturas matemáticas que modelam relações binárias entre objetos de um conjunto. Um grafo é definido por um conjunto de objetos chamados vértices e um conjunto de arestas que ligam pares de vértices. Euler em 1736 modelou o problema do pavimentador

através de um problema em Teoria dos Grafos: cada cidade corresponde a um vértice e cada rodovia entre duas cidades é representada por uma aresta que liga os dois vértices correspondentes. No estado do Rio de Janeiro temos 50 cidades capitais de municípios correspondendo a 50 vértices e temos rodovias que correspondem às arestas conectando estes vértices. A cidade capital de Itaguaí tem 7 rodovias que passam por ela, e corresponde a um vértice de grau ímpar no grafo das cidades capitais e rodovias do estado, o que segundo o Teorema de Euler impossibilita a existência de um circuito do pavimentador no estado. Por outro lado, caso encontremos um outro estado onde todas as cidades tenham grau par, esta condição fácil e poderosa de Euler garante que tal estado admite um circuito do pavimentador.

Para o problema do pavimentador, podemos decidir se o circuito existe ou não através da condição simples encontrada por Euler: basta conferir a paridade de cada cidade. Para o problema do vacinador, temos algumas condições apenas necessárias, temos outras condições apenas suficientes, mas ainda não temos uma condição que caracterize e resolva o problema, e não sabemos se tal condição existe! Para o problema do vacinador, hoje nos resignamos a buscar a resposta examinando o enorme espaço exponencial de busca constituído de todas as possíveis permutações das cidades!

Por outro lado, caso alguém persistente ou com muita sorte apresente um circuito do vacinador, um circuito que visita cada cidade exatamente uma vez, é fácil verificar que este circuito satisfaz a restrição: basta conferir que cada cidade é visitada uma vez e que cidades consecutivas no circuito são de fato conectadas por uma rodovia.

Será que o problema do vacinador é mais difícil que o problema do pavimentador? Será que existem problemas cuja solução pode ser verificada facilmente mas cuja solução não pode ser encontrada facilmente? Este é o desafio central para a Teoria da Computação.

Problemas P, NP e NP-completo

Vejamos novo exemplo. Considere uma turma em um colégio, digamos com 40 alunos, cada qual com certo número de amigos dentro da turma. O problema do Emparelhamento procura dividir aqueles alunos em 20 pares, cada par formado por alunos que sejam amigos. Note que o número de conjuntos contendo 20 pares de alunos quaisquer, não necessariamente amigos, é igual a $40!/(20!2^{20})$, aproximadamente 2^{78}, ou $3,2 \times 10^{23}$, um número de 24 algarismos! Segundo essa estimativa, temos algo como 3 mil vezes mais maneiras distintas de dividir nossa turma em 20 pares de alunos do que grãos de areia no mundo. O problema do emparelhamento é um problema fundamental na área de complexidade computacional. É um problema cuja solução sugeriu a definição formal adotada para computação eficiente. Um algoritmo é eficiente se a sua

execução consome um número de passos que cresce como um polinômio no tamanho dos dados da entrada. O problema do emparelhamento admite algoritmo que o resolve em n^3 passos, para uma turma de n alunos. Para n igual a 40, observe a drástica redução do exponencial 2^{78}, muito maior que 2^{40}, para o polinomial 40^3, que dá um número de passos, ou operações básicas, próximo a 64 mil. Usamos o conceito de polinomial como sinônimo de tratável, viável, eficiente, em contraste com exponencial, sinônimo de inviável.

Considere algumas variações do problema do emparelhamento: o problema de se dividir a turma em grupos contendo 3 amigos mútuos em cada grupo; o problema de se dividir a turma em 3 grupos contendo apenas amigos mútuos em cada grupo; problema de organizar os alunos numa única mesa redonda de forma que apenas amigos sentem-se lado a lado (aqui reencontramos disfarçado o problema do vacinador, na turma de 40 alunos, existem 40! alocações possíveis numa mesa redonda); No caso de cada uma das variações do problema, ainda não conhecemos algoritmo que executa um número polinomial de passos. Cada uma das variações do problema define um problema matemático desafiador, tema de pesquisa avançada na área da Matemática chamada Combinatória. O desafio é encontrar alguma propriedade que corresponda a um atalho polinomial para buscar rapidamente no espaço exponencial das possíveis soluções. Estes problemas desafiadores compartilham uma propriedade: dada uma candidata a solução — um emparelhamento em grupos de três alunos compatíveis, uma alocação numa única mesa redonda, uma partição em três grupos de alunos compatíveis — podemos aprovar a candidata através de um algoritmo.

A questão central para Computação é: quão eficientemente um problema pode ser resolvido através de um algoritmo? Do ponto de vista computacional, distinguimos problemas fáceis e difíceis usando o conceito de algoritmo polinomial. Consideramos fáceis os problemas que podem ser resolvidos através de um algoritmo cujo número de passos cresce com uma potência fixa do número de símbolos usados para especificar a entrada do problema. Por outro lado, consideramos difíceis os problemas para os quais qualquer possível algoritmo consome um número extremamente grande de passos até retornar a resposta. Muitos problemas candidatos a problemas difícies compartilham a propriedade: encontrar a solução parece difícil mas verificar a solução é fácil. A intuição diz que encontrar a solução tem que ser mais difícil do que simplesmente verificar a solução mas nem sempre a intuição é um bom guia para a verdade. Para estes problemas candidatos a problemas difíceis, não conhecemos ainda algoritmos polinomiais e não sabemos provar a inexistência de algoritmo polinomial. A Teoria da Complexidade Computacional considera estes problemas desafiadores, estes problemas que resistem à classificação em fácil ou difícil, como o problema do vacinador, dentro de uma única classe contendo problemas igualmente difíceis, igualmente desafiadores.

Em Teoria da Computação, chamamos de P a classe dos problemas que podem

ser resolvidos através de um algoritmo polinomial, um algoritmo cujo número de passos seja uma potência fixa no tamanho dos dados do problema. E chamamos de NP a classe dos problemas onde toda candidata a solução pode ser aprovada ou rejeitada em tempo polinomial. A condição de verificar parece bem menos restritiva que a de encontrar a solução. A igualdade P = NP significaria que todo problema que tem solução que pode ser verificada rapidamente tem também solução que pode ser encontrada rapidamente. Para estudar a possível igualdade P = NP, os pesquisadores definiram o conjunto dos problemas igualmente difíceis entre si, e pelo menos tão difíceis quanto qualquer problema em NP. São os chamados problemas NP-completo. O problema do vacinador é um problema NP-completo. Caso alguém consiga um algoritmo de tempo polinomial para resolver o problema do vacinador, terá na verdade revolvido um problema que vale 1 milhão de dólares porque terá provado que P = NP.

Problemas desafiadores em Teoria dos Grafos

Um grafo é *perfeito* quando todo subgrafo induzido admite uma coloração própria de seus vértices (i.e., vértices adjacentes têm cores diferentes) que usa o mesmo número de cores que o número de vértices de uma de suas cliques (um conjunto de vértices mutuamente adjacentes). Os *grafos perfeitos* definem um rico tópico de pesquisa que pode ser visto como a interseção de três áreas: Teoria dos Grafos, Complexidade Computacional e Otimização Combinatória.

Claude Berge [5] introduziu a classe dos grafos perfeitos no início dos anos 1960, como uma classe de grafos interessante sob três aspectos, cada um com um problema aberto desafiador: uma classe definida a partir dos problemas de otimização combinatória NP-difíceis COLORAÇÃO MÍNIMA e CLIQUE MÁXIMA; uma classe cujo problema de reconhecimento (i.e., decidir se um dado grafo é perfeito) não se sabia sequer classificar como em NP; uma classe que sugeria uma caracterização por uma família autocomplementar infinita de subgrafos induzidos proibidos. Grötschel, Lovász e Schrijver [24] apresentaram em 1979 um algoritmo polinomial para a coloração mínima dos grafos perfeitos usando o método do elipsóide para programação linear, e deixaram em aberto o problema de encontrar um algoritmo polinomial de natureza puramente combinatória para colorir otimamente os grafos perfeitos. Johnson em 1981 na sua coluna *The NP-completeness – an ongoing guide* [28] propôs o problema de reconhecimento para a classe dos grafos perfeitos. Cameron [8] provou em 1982 que esse problema estava em CoNP. A prova de que a caracterização por subgrafos proibidos proposta por Berge estava correta e o algoritmo polinomial para o reconhecimento dos grafos perfeitos só apareceriam respectivamente em 2006 e em 2005 [12, 11]. O algoritmo polinomial de natureza puramente combinatória para colorir otimamente os grafos perfeitos ainda permanece como um desafio [37]. Descrevemos a seguir como se deu a classificação da

complexidade computacional de dois problemas centrais, abertos por décadas, um em polinomial e o outro em NP-completo.

Partição Assimétrica O problema da partição assimétrica foi proposto por Chvátal em 1985 no contexto de decomposições para grafos perfeitos [13]: dado um grafo G, o seu conjunto de vértices admite uma partição em quatro partes não vazias A, B, C, D tal que todo vértice em A é adjacente a todo vértice em B e todo vértice em C é não-adjacente a todo vértice em D? Chvátal propôs a partição assimétrica como uma generalização de decomposições bem estudadas: corte estrela, corte clique e conjunto homogêneo, e acertou ao afirmar que a partição assimétrica teria um papel determinante na prova da conjectura de Berge para grafos perfeitos. No grafo G, o conjunto de vértices $A \cup B$ é um corte cuja remoção desconecta as partes não vazias C e D, e no complemento do grafo G, o conjunto de vértices $C \cup D$ é um corte cuja remoção desconecta as partes não vazias A e B. Dessa propriedade auto-complementar, vem o nome partição assimétrica, o conjunto $A \cup B$ é chamado *corte assimétrico*, e a partição A, B, C, D é chamada *partição assimétrica*.

O primeiro algoritmo polinomial para testar se um grafo admite uma partição assimétrica, um algoritmo recursivo de complexidade $O(n^{100})$, na verdade resolve um problema mais geral, a partição assimétrica com listas [15]. A versão com listas de um problema de partição tem como entrada, além do grafo cujo conjunto de vértices será particionado, uma lista de partes permitidas, para cada um dos vértices. Na versão com listas, as partes podem ser vazias. Muitos problemas combinatórios em grafos podem ser formulados como um problema de partição dos vértices em subconjuntos de acordo com restrições internas ou externas em relação às adjacências: conjunto homogêneo, corte clique, coloração de vértices, grafo *split*, entre outros. Feder, Hell, Klein e Motwani descreveram uma dicotomia para partições em quatro partes na versão com listas, onde classificaram cada um dos problemas dessa classe em quasi-polinomial ou NP-completo [14]. Um algoritmo de tempo quasi-polinomial tem complexidade de pior caso $O(n^{c\log^t n})$, onde c e t são constantes positivas e n é o número de vértices do grafo de entrada. O problema da partição assimétrica foi classificado nessa ocasião como quasi-polinomial, uma indicação de que o problema era de fato polinomial. Posteriormente, o algoritmo polinomial que desenvolvemos para partição assimétrica estabeleceu a complexidade do problema de Chvátal. O nosso algoritmo recursivo para partição assimétrica com listas recursivamente define problemas de partição assimétrica com listas menores. O número de subproblemas $T(n)$ encontrados satisfaz recorrências da forma $T(n) \leq 4\,T(9n/10)$, o que fornece um tempo de execução $O(n^{100})$, um desafio para a noção aceita de que solúvel em tempo polinomial é o mesmo que solúvel eficientemente na prática.

Grafo Clique O problema de reconhecimento dos grafos clique foi proposto por Roberts e Spencer em 1971 no contexto de grafos de interseção [39]. Um *conjunto completo* de um grafo $H = (V, E)$ é um subconjunto de V que induz um subgrafo completo. Uma *clique* é um conjunto completo maximal. Um grafo é um *grafo clique* se ele é o grafo de interseção dos conjuntos completos maximais de algum grafo. Foram necessários quase 40 anos para que a prova de NP-completude fosse estabelecida [1]. Não é evidente que o problema de reconhecimento pertença a NP, já que o número de completos maximais em um grafo pode ser exponencial no seu número de vértices. Considere por exemplo o grafo que consiste de um emparelhamento induzido. O grafo contém $2^{n/2}$ conjuntos independentes maximais, e portanto o seu complemento é um grafo com $2^{n/2}$ conjuntos completos maximais. A propriedade de Helly tem sido muito estudada com o objetivo de classificar a complexidade do problema de reconhecimento de grafos clique. Uma família de conjuntos $\mathcal{F} = (F_i)_{i \in I}$ é *intersectante* se a interseção de cada dois de seus membros é não vazia. A *interseção total* de $\mathcal{F}$ é o conjunto $\bigcap \mathcal{F} = \bigcap_{i \in I} F_i$. A família $\mathcal{F}$ tem a *propriedade de Helly*, se qualquer subfamília intersectante tem interseção total não vazia. Roberts e Spencer nesse seu artigo fundamental caracterizaram os grafos clique em termos de uma cobertura das arestas do grafo por conjuntos completos que satisfaz a propriedade de Helly, o que forneceu uma prova de que o problema de reconhecimento dos grafos clique está em NP. Szwarcfiter caracterizou os grafos clique-Helly e apresentou um algoritmo polinomial para o seu reconhecimento [44]. O problema de reconhecimento dos grafos clique foi seguidamente proposto em vários livros [7, 32, 36, 45]. Após a nossa surpreendente prova de NP-completude [1], continuamos aprofundando o estudo do problema de reconhecimento dos grafos clique para a classe dos grafos *split*. A idéia era buscar um problema que separasse em termos de complexidade as classes cordal e *split*. É conhecido que um grafo cordal tem um número linear de conjuntos completos maximais. Embora a classe dos grafos *split* seja precisamente a interseção dos grafos cordais e dos complementos de grafos cordais, são raros os problemas que separam as classes cordal e *split*, i.e., problemas que são NP-completos para cordais mas polinomiais para *split*, segundo o *The NP-completeness – an ongoing guide* de Johnson [29] e o livro de Spinrad [42]. Estabelecemos recentemente que o reconhecimento dos grafos clique *split* é NP-completo [2].

1.1.2. O Guia para Problemas Desafiadores

Na sua famosa série de colunas sobre NP-completude, Johnson [28] atualiza o seu livro seminal *The NP-completeness – an ongoing guide* [23]. Nos referimos mais precisamente à coluna sobre *graph restrictions and their effect* [29]. O objetivo é identificar problemas interessantes e classes de grafos interessantes estabelecendo o conceito de separação de complexidade. Uma classe C é dita *classe de grafos separadora* quando C separa a complexidade de dois problemas π and σ: π é NP-completo quando consideramos como entrada para π grafos da classe C, mas σ é polinomial quando consideramos

classe de grafos	VERTEXCOL	EDGECOL	MAXCUT
perfect	P	N	N
chordal	P	O	N
split	P	O	N
strongly chordal	P	O	O
comparability	P	N	O
bipartite	P	P	P
permutation	P	O	O
cographs	P	O	P
proper interval	P	O	O
split-proper interval	P	P	P

Tabela 1.1. N: NP-completo, P: polinomial, O: aberto

como entrada para σ grafos da classe C. Um problema π é dito *problema separador de complexidade* quando π separa duas classes de grafos $\mathcal{A} \subset \mathcal{B}$: π é polinomial quando consideramos como entrada para π grafos da classe $\mathcal{A}$, mas π é NP-completo quando consideramos como entrada para π grafos da classe $\mathcal{B}$.

A coluna sobre *graph restrictions and their effect* [29] possui uma seção dedicada aos grafos perfeitos e aos grafos de interseção. A Tabela 1.1 é uma subtabela da tabela apresentada naquela coluna onde selecionamos alguma linhas correspondentes a classes de grafos e algumas colunas correspondentes a problemas, e mantivemos a mesma notação.

É bem conhecido na área de Teoria dos Grafos que os grafos perfeitos constituem uma classe de grafos onde o problema coloração de vértices é polinomial. Por outro lado, o problema coloração de arestas é NP-completo quando restrito à classe do grafos perfeitos. Na verdade, a subclasse dos grafos de comparabilidade é uma classe de grafos separadora [27] porque separa os problemas coloração de vértices e coloração de arestas. Além disso, coloração de arestas é um problema separador de complexidade já que separa a classe dos grafos de comparabilidade da classe dos grafos bipartidos. É notável que existam poucas classes de grafos onde a complexidade de coloração de arestas é estabelecida, e é surpreendente que, desde 1985, várias classes de grafos para as quais Johnson mencionou na sua coluna como "aparentemente aberto, mas possivelmente fácil de resolver", permanecem desafiadores, 30 anos depois. A classe dos cografos, uma classe de grafos muito restrita e estruturada definida ao proibirmos o caminho induzido por quatro vértices P_4, admite apenas uma solução parcial [40]. A classe de grafos split-proper interval merece atenção. Ortiz, Maculan, e Szwarcfiter [34] caracterizaram os grafos que são split e proper interval, e a caracterização permitiu que coloração de arestas [34] e corte máximo [6] sejam polinomiais para esta classe. Obser-

vamos que, dois problemas clássicos: coloração de arestas e corte máximo permanecem abertos para a classe dos grafos de intervalo, uma classe de grafos de interseção muito estudada [42]. Observe que existem classes de grafos onde coloração de vértices é NP-completo e coloração de arestas é polinomial, por exemplo, grafos com um vértice universal [35]. Possivelmente mais interessante é a existência de classes de grafos onde coloração de arestas é NP-completo, mas coloração total — onde colorimos todos os elementos do grafo, vértices e arestas — é polinomial [30].

split versus cordal O fato de que as classes split e cordal concordam em relação à complexidade dos três problemas: coloração de vértices, coloração de arestas e corte máximo, sugere um padrão que tem sido investigado na literatura.

Um grafo é cordal, quando todo ciclo com pelo menos quatro vértices admite uma corda, e um grafo é split quando o seu conjunto de vértices pode ser particionado em um conjunto independente e uma clique. Os grafos split constituem uma subclasse dos grafos cordais muito estruturada, já que são exatamente os grafos tais que tanto o grafo quanto o seu complemento são ambos cordais.

Johnson, na sua coluna sobre *graph restrictions and their effect* [29], declarou que não conhecia problema que separasse as classes split e cordal do aspecto de complexidade. Spinrad, em seu livro [42] vinte anos depois, ao relatar os resultados de complexidade para classes de grafos, atualiza os problemas separadores de complexidade em relação às classes split e cordal: clique máxima e coloração de vértices são polinomiais para ambos; enquanto conjunto dominante, corte máximo e ciclo hamiltoniano são NP-completos para ambos. Existem poucos problemas separadores de complexidade em relação às classes split e cordal: empacotamento de triângulos e pathwidth, para os quais o problema é NP-completo para cordal mas polinomial para split. Spinrad [42] informa que os grafos split estão no núcleo dos algoritmos e dos resultados de dificuldade para os grafos cordais. Problemas separadores de complexidade em relação às classes split e cordal são raros. Recentemente, tentamos o problema grafo clique, mas obtivemos que o problema permanece NP-completo para grafos split [1, 2]. Possivelmente, a classe dos grafos planares fornecerá uma classe onde o problema grafo clique seja polinomial [3]. Outro problema que continua aberto é coloração de arestas, e os resultados parciais para as classes cordal, intervalo e split sugerem que o problema coloração de arestas é desafiador mesmo restrito a classes muito estruturadas [16, 34].

grafos sem ciclos com corda única Trotignon e Vušković [46] estudaram a classe C dos grafos sem ciclos com corda única. A motivação principal para investigar a classe foi encontrar um teorema que revelasse a estrutura desses grafos, um tipo de resultado que não é muito frequente na literatura. Dada uma estrutura proibida, por exemplo, ciclo

com corda única, procuramos investigar que consequência traz para a estrutura do grafo, e que problemas desafiadores tornam-se polinomiais. O teorema é do tipo: todo grafo na classe C pode ser obtido a partir de grafos básicos de C aplicando uma série de operações que "identificam" grafos de C. Outra propriedade investigada é a de família de grafos χ-limitada, introduzida por Gyárfás [25], como uma generalização natural da propriedade que define os grafos perfeitos. Uma família de grafos $\mathcal{G}$ é χ-limitada com *função χ-limitada f* se, para todo subgrafo induzido G' de $G \in \mathcal{G}$, temos $\chi(G') \leq f(\omega(G'))$, onde $\chi(G')$ é o número cromático de G' e $\omega(G')$ denota o tamanho da clique máxima em G'. Os grafos perfeitos satisfazem: para todo subgrafo induzido G' de G na classe dos grafos perfeitos, temos $\chi(G') = \omega(G')$, a função χ-limitada f é a função identidade. Por outro lado, o teorema dos grafos perfeitos estabelece que um grafo G é perfeito se e somente se G e o complemento de G não admitem um ciclo ímpar sem cordas e com pelo menos 5 vértices. A pesquisa nesta área é dedicada principalmente a entender para que escolhas de subgrafos induzidos proibidos, a família resultante de grafos é χ-limitada; veja [38] para um *survey* sobre o assunto. Pelo Teorema de Vizing para coloração de arestas, a classe dos grafos de linha de grafos simples é uma classe de grafos χ-limitada com função χ-limitada $f(x) = x + 1$ (este limite superior especial é chamado o *limite de Vizing*) e, além disso, os grafos de linha são caracterizados por nove subgrafos induzidos proibidos [47]. A classe C dos grafos sem ciclos com corda única é também χ-limitada com o limite de Vizing [46]. Em [46] os seguintes resultados são obtidos para os grafos na classe C: um algoritmo $O(nm)$ para a coloração dos vértices, um algoritmo $O(n+m)$ para clique máxima, um algoritmo $O(nm)$ para o reconhecimento da classe, e a prova de NP-completude para conjunto independente máximo.

Consideramos a complexidade do problema de coloração de arestas para a classe C [31]. O problema de coloração de arestas, também conhecido como índice cromático, é o problema de determinar o menor número $\chi'(G)$ de cores necessárias para colorir as arestas do grafo G. Também investigamos subclasses obtidas da classe C quando proibimos o ciclo sem cordas com 4 vértices e quando proibimos o ciclo sem cordas com 6 vértices. As Tabelas 1.2 e 1.3 resumem os resultados obtidos, mostrando que classes com muita estrutura ainda definem problemas difíceis e desafiadores. Nas tabelas, denotamos por C a classe dos grafos sem ciclo com corda única e por Δ o grau máximo no grafo.

A classe inicialmente investigada em [31] foi a classe C dos grafos sem ciclo com corda única. Para os objetivos do trabalho, um grafo G é *básico* se G é completo, um ciclo sem cordas com pelo menos 5 vértices, um grafo fortemente 2-bipartido, ou um subgrafo induzido do grafo de Petersen ou do grafo de Heawood; e G não possui um vértice de articulação, um corte próprio com 2 vértices ou uma junção própria. Após provar que coloração de arestas é NP-completo para grafos em C, consideramos os grafos na subclasse $C' \subset C$ que não têm C_4. Ao proibir um ciclo sem cordas com 4 vértices

classe de grafos	$\Delta = 3$	$\Delta \geq 4$	regular
grafos em C	NP-completo	NP-completo	NP-completo
grafos em C sem C_4	NP-completo	polinomial	polinomial
grafos em C sem C_6	NP-completo	NP-completo	NP-completo
grafos em C sem C_4, sem C_6	polinomial	polinomial	polinomial

Tabela 1.2. Dicotomia de complexidade NP-completo versus polinomial para a coloração de arestas dos grafos sem ciclo com corda única.

C_4, evitamos a decomposição por junção própria, uma decomposição difícil para coloração de arestas [4, 40, 41], o que significa que cada grafo da classe C' que não é básico pode ser decomposto através de um vértice de articulação ou através de um corte próprio com 2 vértices. Para a classe C', estabelecemos então uma dicotomia: coloração de arestas é NP-completo para grafos de C' com grau máximo 3 e é polinomial para grafos de C' com grau máximo *não* 3. Adicionalmente, determinamos uma condição necessária para um grafo $G \in C'$ com grau máximo 3 ser Classe 2, isto é, ter índice cromático $\chi'(G) = \Delta(G) + 1 = 4$. A condição necessária é ter um grafo P^* — subgrafo do grafo de Petersen — como um bloco básico na árvore de decomposição. Consequentemente, se proibimos ambos C_4 e C_6, o índice cromático dos grafos sem ciclo com corda única pode ser determinado em tempo polinomial. Os resultados alcançados em [31] podem ser relacionados com várias áreas da pesquisa em coloração de arestas, como observamos a seguir.

A primeira observação diz respeito ao resultado de dicotomia encontrado para a classe class C'. A dicotomia é muito interessante porque esta é a primeira classe de grafos para a qual coloração de arestas é NP-completo para grafos com um dado grau máximo fixo Δ mas é polinomial para grafos com grau máximo $\Delta' > \Delta$. Além disso, a classe C' é a primeira classe de grafos interessante onde coloração de arestas é NP-completo em geral, mas é polinomial quando restrito a grafos regulares. É interessante observar que a classe de grafos C' é uma classe de grafos com poucos grafos regulares — somente o grafo de Petersen, o grafo de Heawood , os grafos completos e os ciclos sem cordas.

A segunda observação diz respeito à conjectura de Chetwynd e Hilton. Uma ferramenta importante para identificar classes de grafos que são Classe 2 é o conceito de *sobrecarregado* [17]. Um grafo é dito *sobrecarregado* se satisfaz $|E| > \Delta(G)\lfloor |V|/2 \rfloor$, uma condição suficiente para que $\chi'(G) = \Delta(G) + 1$. Por exemplo, o ciclo sem cordas com 5 vértices é um grafo sobrecarregado. Uma condição suficiente mais geral é: G é *subgrafo sobrecarregado* se G possui o mesmo grau máximo que o grafo e G é sobrecarregado. Grafos que são subgrafo sobrecarregados são Classe 2 [17] e podemos verificar em tempo polinomial se um grafo é subgrafo sobrecarregado [33]. Para algumas classes

classe de grafos	$k \leq 2$	$k \geq 3$
grafos k-partidos	polinomial	NP-completo

Tabela 1.3. Dicotomia de complexidade NP-completo versus polinomial para a coloração de arestas de grafos multipartidos.

de grafos, ser subgrafo sobrecarregado é equivalente a ser Classe 2. Exemplos de tais classes são: grafos com vértice universal [35], grafos multipartidos completos [26], e grafos split com grau máximo ímpar [9]. A conjectura de Chetwynd e Hilton [10] diz que um grafo $G = (V, E)$ com $\Delta(G) > |V|/3$ é Classe 2 se e somente se é subgrafo sobrecarregado. De fato, para a maioria das classes para as quais o problema de coloração de arestas é resolvido em tempo polinomial, a equivalência "Class 2 = Subgrafo Sobrecarregado" é válida. É notável que a maioria destas classes é composta de grafos cujo grau máximo é alto — sempre superior a um terço do número de vértices. Portanto, para estas classes, a equivalência "Class 2 = Subgrafo Sobrecarregado" — e o consequente algoritmo polinomial para o problema de coloração de arestas — seriam consequência direta da Conjectura de Chetwynd e Hilton para grafos subgrafo sobrecarregados, caso a conjectura seja válida. Neste sentido, a classe C' investigada em [31, 46] apresenta um grande interesse: para grafos em C' não há limite na relação entre "número de vértices versus grau máximo"; mesmo assim, se o grau máximo não é 3, a equivalência "Class 2 = Subgrafo Sobrecarregado" é válida. Portanto, a classe dos grafos em C' com grau máximo não 3 é uma classe de grafos que não encaixa nas hipóteses da Conjectura de Chetwynd e Hilton para grafos subgrafo sobrecarregados, mas para a qual coloração de arestas é ainda solúvel em tempo polinomial através da equivalência "Class 2 = Subgrafo Sobrecarregado".

A terceira observação é relacionada com o estudo dos *snarks* [43]. Um *snark* é um grafo cúbico sem pontes com índice cromático igual a 4. Para evitar casos triviais, snarks são em geral restritos a grafos com cintura pelo menos 5 e não contêm 3 arestas cuja remoção resulta em um grafo desconexo, com cada componente não trivial. O estudo dos snarks é relacionado fortemente ao famoso e histórico Teorema das Quatro Cores, para a coloração de mapas. Através de um resultado de [31], o único snark não trivial que *não* possui ciclo com corda única é o grafo de Petersen.

A quarta observação diz respeito ao problema de determinar o índice cromático de um grafo k-partido, isto é, um grafo cujo conjunto de vértices pode ser particionado em k conjuntos independentes. O problema de determinar o índice cromático de um grafo k-partido, já estava classificado, é polinomial para $k = 2$ [28, 29], e para grafos multipartido completos [26]. A prova de NP-completude do índice cromático dos grafos na classe C implica que coloração de arestas é NP-completo para grafos k-partidos que são r-regulares, para cada $k \geq 3$, $r \geq 3$ [31].

1.1.3. Notas Bibliográficas

Indicamos algumas publicações da autora para o leitor interessado. Para um público amplo, a autora possui um artigo de divulgação [18] na revisa Ciência Hoje: "Resolver ou Verificar? Uma pergunta que vale um milhão de dólares". Para o público especialista na área de Teoria dos Grafos e Complexidade Computacional, a autora reuniu as suas contribuições principais e apontou problemas em aberto relacionados no artigo *survey* [19]: "The P vs. NP-complete dichotomy of some challenging problems in graph theory". A autora tem dois textos introdutórios anteriores na Jornada de Atualização em Informática (JAI), um texto sobre Emparelhamento em Grafos [20] e um texto sobre Coloração em Grafos [21].

No JAI 1999, em co-autoria com Jayme Szwarcfiter, escreveu sobre "Emparelhamento em Grafos — Algoritmos e Complexidade". Encontrar um emparelhamento máximo em um grafo é um problema clássico no estudo de algoritmos, com muitas aplicações: designação de tarefas, determinação de rotas de veículos, problema do carteiro chinês, determinação de percursos mínimos, e muitos outros. O artigo histórico de Edmonds que descreveu um algoritmo $O(n^4)$ para o problema geral de emparelhamento, na verdade, introduziu a noção de algoritmo de tempo polinomial. Implementações mais eficientes do algoritmo de Edmonds foram apresentadas por vários pesquisadores como Lawler (algoritmo $O(n^3)$), Hopcroft e Karp (algoritmo $O(m\sqrt{n})$ para o caso bipartido), Micali e Vazirani (algoritmo $O(m\sqrt{n})$ para o caso geral, o mais eficiente até o momento). O texto apresenta uma introdução ao problema de emparelhamento em grafos, e um estudo de seus algoritmos e complexidade. Apresentamos algoritmos eficientes para os quatro problemas relacionados com encontrar um emparelhamento de cardinalidade máxima ou de peso máximo, em grafos bipartidos ou em grafos quaisquer. Estes quatro problemas são todos casos particulares do problema de emparelhamento de peso máximo em grafos quaisquer, mas é interessante considerá-los em ordem crescente de dificuldade por razões de ordem didática [20].

No JAI 1997, em co-autoria com João Meidanis e Célia Mello, escreveu sobre "Coloração em Grafos". O texto é uma introdução à coloração em grafos. Consideramos dois tipos de problemas de coloração em grafos: coloração de vértices e coloração de arestas. Aplicamos a estes problemas técnicas clássicas de coloração tais como: algoritmos gulosos, decomposição e alteração estrutural. Consideramos classes de grafos para as quais tais técnicas resolvem estes problemas de coloração eficientemente. O estudo de coloração, um tópico básico em Teoria dos Grafos, surgiu a partir do conhecido "problema das quatro cores". Em 1852, Francis Guthrie questionou se todo grafo planar poderia ser colorido com quatro cores. Embora estivesse claro que quatro cores eram necessárias, a questão se referia ao número mínimo de cores, i.e., quantas cores são suficientes para colorir qualquer grafo planar. Esta questão foi respondida afirmativamente por Appel e Haken, usando computador, após mais de 100 anos. Robertson,

Sanders, Seymour e Thomas mais recentemente confirmaram esta resposta fornecendo uma prova mais simples, mais aceita pela comunidade matemática, embora ainda use o computador. O problema das quatro cores corresponde à coloração dos vértices de um grafo. Mais precisamente, corresponde à coloração mínima dos vértices de um grafo. Outros problemas de coloração existem, tais como coloração das arestas, coloração total (vértices e arestas), coloração a partir de conjuntos de cores atribuídos préviamente aos vértices ou às arestas de um grafo. O escopo do texto, no entanto, é mais restrito: coloração mínima de vértices e coloração mínima de arestas em classes de grafos [21].

Referências

[1] L. Alcon, L. Faria, C. M. H. de Figueiredo, and M. Gutierrez, The complexity of clique graph recognition, *Theoret. Comput. Sci.* **410** (2009) 2072–2083.

[2] L. Alcon, L. Faria, C. M. H. de Figueiredo, and M. Gutierrez, Split clique graph complexity, *Theoret. Comput. Sci.* **506** (2013) 29–42.

[3] L. Alcón, and M. Gutierrez, Cliques and extended triangles. A necessary condition for planar clique graphs, *Discrete Appl. Math.* **141** (2004) 3–17.

[4] M. M. Barbosa, C. P. de Mello, and J. Meidanis, Local conditions for edge-colouring of cographs, *Congr. Numer.* **133** (1998) 45–55.

[5] C. Berge, and V. Chvátal, *Topics on Perfect Graphs*, North-Holland Mathematics Studies, 88. *Annals of Discrete Mathematics*, 21. North-Holland Publishing Co., Amsterdam, 1984.

[6] H. L. Bodlaender, C. M. H. de Figueiredo, M. Gutierrez, T. Kloks, and R. Niedermeier, Simple max-cut for split-indifference graphs and graphs with few P_4's, Proc. of Third International Workshop on Experimental and Efficient Algorithms (WEA 2004). *Lecture Notes in Comput. Sci.* 3059 (2004), 87–99.

[7] A. Brandstädt, V. B. Le, and J. P. Spinrad, *Graph Classes: A survey*. SIAM Monographs on Discrete Mathematics and Applications, 1999.

[8] K. Cameron, *Polyhedral and Algorithmic Ramifications of Antichains*, Ph.D. Thesis, Department of Combinatorics and Optimization, University of Waterloo, Waterloo, 1982.

[9] B. L. Chen, H.-L. Fu, and M. T. Ko, Total chromatic number and chromatic index of split graphs, *J. Combin. Math. Combin. Comput.* **17** (1995) 137–146.

[10] A. G. Chetwynd, and A. J. W. Hilton, Star multigraphs with three vertices of maximum degree, *Math. Proc. Cambridge Philos. Soc.* **100** (1986) 303–317.

[11] M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour, and K. Vušković, Recognizing Berge graphs, *Combinatorica* **25** (2005) 143–186.

[12] M. Chudnovsky, N. Robertson, P. Seymour, and R. Thomas, The strong perfect graph theorem, *Ann. of Math.* **164** (2006) 51–229.

[13] V. Chvátal, Star-cutsets and perfect graphs, *J. Combin. Theory Ser. B* **39** (1985) 189–199.

[14] T. Feder, P. Hell, S. Klein, and R. Motwani, List partitions, *SIAM J. Discrete Math.* **16** (2003) 449–478.

[15] C. M. H. de Figueiredo, S. Klein, Y. Kohayakawa, and B.A. Reed, Finding skew partitions efficiently, *J. Algorithms* **37** (2000) 505–521.

[16] C. M. H. de Figueiredo, J. Meidanis, and C. P. de Mello, On edge-colouring indifference graphs, *Theoret. Comput. Sci.* **181** (1997) 91–106.

[17] C. M. H. de Figueiredo, J. Meidanis, and C. P. de Mello, Local conditions for edge-coloring, *J. Combin. Math. Combin. Comput.* **32** (2000) 79–91.

[18] C. M. H. de Figueiredo, Resolver ou Verificar? Uma pergunta que vale um milhão de dólares. *Ciência Hoje*, Rio de Janeiro pp. 42–46, novembro 2011.

[19] C. M. H. de Figueiredo, The P vs. NP-complete dichotomy of some challenging problems in graph theory, *Discrete Appl. Math.* **160** (2012) 2681-2693.

[20] C. M. H. de Figueiredo, and J. L. Szwarcfiter, Emparelhamentos em Grafos: Algoritmos e Complexidade, *XIX Congresso da Sociedade Brasileira de Computação. (Org.). XVIII Jornada de Atualização em Informática.* 1999, pp. 127–161.

[21] C. M. H. de Figueiredo, J. Meidanis, and C. P. Mello, Coloração em Grafos, *XVII Congresso da Sociedade Brasileira de Computação. (Org.). XVI Jornada de Atualização em Informática* 1997, pp. 01–45.

[22] L. Fortnow, The status of the P versus NP problem, *Communications of the ACM* **52** (2009) 78–86.

[23] M. R. Garey, and D. S. Johnson, *Computers and Intractability. A Guide to the Theory of NP-completeness*. WH Freeman, New York, 1979.

[24] M. Grötschel, L. Lovász, and A. Schrijver, Polynomial Algorithms for Perfect Graphs, In *Topics on Perfect Graphs*, C. Berge and V. Chvátal, eds., North-Holland Mathematics Studies 88, Annals of Discrete Mathematics 21, North-Holland Publishing Co., Amsterdam, 1984, pp. 325–356.

[25] A. Gyárfás, Problems from the world surrounding perfect graphs, *Zastos. Mat.* **19** (1987) 413–441.

[26] D. G. Hoffman, and C. A. Rodger, The chromatic index of complete multipartite graphs, *J. Graph Theory* **16** (1992) 159–163.

[27] I. Holyer, The NP-completeness of edge-coloring, *SIAM J. Comput.* **10** (1981) 718–720.

[28] D. S. Johnson, *NP-Completeness Columns,* available at `http://www2.research.att.com/ dsj/columns/`

[29] D. S. Johnson, Graph restrictions and their effect, *J. Algorithms* **6** (1985) 434–451.

[30] R. C. S. Machado, and C. M. H. de Figueiredo, Complexity separating classes for edge-colouring and total-colouring, *J. Braz. Comput. Soc.* **17** (2011) 281–285.

[31] R. C. S. Machado, C. M. H. de Figueiredo, and K. Vušković, Chromatic index of graphs with no cycle with a unique chord, *Theoret. Comput. Sci.* **411** (2010) 1221–1234.

[32] T. A. McKee, and F. R. McMorris, *Topics in Intersection Graph Theory*, SIAM Monographs on Discrete Mathematics and Applications, 1999.

[33] T. Niessen, How to find overfull subgraphs in graphs with large maximum degree, *Electron. J. Combin.* **8** (2001) #R7.

[34] C. Ortiz Z., N. Maculan, and J. L. Szwarcfiter, Characterizing and edge-colouring split-indifference graphs, *Discrete Appl. Math.* **82** (1998) 209–217.

[35] M. Plantholt, The chromatic index of graphs with a spanning star, *J. Graph Theory* **5** (1981) 45–53.

[36] E. Prisner, *Graph Dynamics*, Pitman Research Notes in Mathematics 338, Longman, 1995.

[37] J. L. Ramirez Alfonsin, and B. A. Reed, *Perfect graphs*, Wiley-Interscience Series in Discrete Mathematics and Optimization, 2001.

[38] B. Randerath, and I. Schiermeyer. Vertex colouring and forbidden subgraphs—a survey, *Graphs Combin.* **20** (2004) 1–40.

[39] F. Roberts, and J. Spencer, A characterization of clique graphs, *J. Combin. Theory Ser. B* **10** (1971) 102–108.

[40] C. Simone, and C. P. de Mello, Edge-colouring of join graphs, *Theoret. Comput. Sci.* **355** (2006) 364–370.

[41] C. Simone, and A. Galluccio, Edge-colouring of regular graphs of large degree, *Theoret. Comput. Sci.* **389** (2007) 91–99.

[42] J. P. Spinrad, *Efficient graph representations*, Fields Institute Monographs, 19. AMS, 2003.

[43] E. Steffen, Classifications and characterizations of snarks, *Discrete Math.* **188** (1998) 183–203.

[44] J. L. Szwarcfiter, Recognizing clique-Helly graphs, *Ars Combin.* **25** (1997) 29–32.

[45] J. L. Szwarcfiter, A Survey on Clique Graphs, In *Recent Advances in Algorithms and Combinatorics*, C. Linhares-Sales and B. Reed, eds., CMS Books Math./Ouvrages Math. SMC, 11, Springer, New York, 2003, pp. 109–136.

[46] N. Trotignon, and K. Vušković, A structure theorem for graphs with no cycle with a unique chord and its consequences, *J. Graph Theory* **63** (2010) 31–67.

[47] D. B. West, *Introduction to Graph Theory*. Second edition. Prentice Hall, 2001.

1.2. Lógica Para Computação: Uma Introdução Curta - Luís C. Lamb

Abstract

The understanding of logical systems is fundamental to Computer Science research. Even the notion of computability has originated in logic-based research. Several logics have been used and developed in Computing since the dawn of this science, in particular in the second half of the XX Century. The use of logical methods has been fundamental to several fields in Computing research. These include, e.g. the specification and verification of computational systems, database systems design, descriptive complexity, integrated circuits design, semantic web, artificial intelligence and distributed systems. This chapter presents a brief introduction to computational logic. We shall also present historical notions on the evolution of logic in Computer Science, illustrating its use by means of applications in relevant fields.

Resumo

O entendimento de sistemas lógicos é fundamental em Ciência da Computação. A própria noção de computabilidade é originária de estudos em lógica. Diversas lógicas têm sido utilizadas e desenvolvidas em Computação desde o estabelecimento desta como ciência organizada, particularmente a partir da segunda metade do Século XX. O uso de métodos lógicos tem sido fundamentais em múltiplas subáreas da computação. Estas áreas incluem: especificação e verificação de sistemas computacionais, projeto de sistemas de bancos de dados, projeto lógico de circuitos integrados, complexidade descritiva, web semântica, inteligência artificial e sistemas distribuídos. Este capítulo apresenta uma breve introdução à lógica computacional. Apresentaremos, também, noções históricas sobre a evolução da lógica em Ciência da Computação, ilustrando com algumas aplicações em áreas relevantes.

1.2.1. Introdução: Brevíssimas Considerações Históricas

It takes an extraordinary intelligence to contemplate the obvious.
Alfred North Whitehead

Lógica é uma área organizada do domínio do conhecimento humano. É uma área de estudo caracterizada por conceitos e definições precisas. O estudo das técnicas e métodos de lógica(s) permite a melhor definição, entendimento e raciocínio sobre diversos domínios do conhecimento.[1]

Na Grécia antiga ("clássica"), os *Estóicos*[2] estudaram e definiram noções iniciais de lógica proposicional e, preliminarmente, de inferência. O nome de Aristóteles, fundamental na história da ciência define os silogismos (padrões de raciocínio em que premissas tem uma conclusão lógica) e o raciocínio com quantificadores (séc. IV A.C). Exemplo simples de silogismo: *Todos os lógicos são filósofos; Luís é lógico; Portanto, Luís é filósofo*; silogismos foram utilizados até a idade média, servindo, inclusive para a formalização de textos escolásticos. No período medieval, alguns lógicos se destacam, como Pierre Abélard (França), no Século XIII, e William of Ockham (Inglaterra) no Século XIV. Ockham é referenciado até hoje pelo conceito da "navalha de Ockham", mas também introduziu princípios similares ao que conhecemos hoje como Leis de *De Morgan* em linguagem natural, além de lógicas tri-valoradas. Abélard publica o *Logica ingredientibus*, em 1121 (lógica para principiantes) e destaca-se como grande propagador do sistema de Aristóteles. Ambos fazem uso do sistema silogístico para verificação de argumentos. No mesmo período, Ramon Llull (*latim:* Lulio), no Séc. XIII, desenvolve estudos para verificar automaticamente a verdade de silogismos, desenvolvendo um sistema de raciocínio como computação. É importante ressaltar que o trabalho de Llull, de certa forma, influenciou o trabalho de Leibniz.

Durante um longo período, o conhecimento acadêmico foi pouco desenvolvido ou relatado e reportado, algumas vezes justificado pelo período da idade média no mundo ocidental. Na idade média, o ensino da lógica era parte do *trivium*: conjunto

[1] O objetivo deste texto não é oferecer uma história completa da lógica. As considerações históricas são apresentadas, apenas, para estimular o leitor a buscar informações adicionais sobre a evolução da lógica e seu impacto na ciência da computação. O texto também *não oferece* um curso completo de lógica; apenas introduz, de forma não exaustiva, conceitos e noções básicas e - em reduzida síntese - sua relação com a ciência da computação. O leitor é convidado a consultar as referências bibliográficas citadas no texto.

[2] O pensamento estóico tem como pioneiro Zeno (ou Zenão) de Cítio (344-262 AC). O pensamento estóico foi predominante na filosofia ocidental até o crescimento da doutrina cristã. Para os estóicos, a lógica inclui a análise de argumentos, a retórica, gramática, teoria de conceitos, percepções, proposições, pensamentos, o que - de certa forma - poderia ser visto como epistemologia ou filosofia da linguagem. No entanto, não há espaço suficiente neste capítulo para maiores detalhes sobre o estoicismo. Uma referência na moderna "Stanford Encyclopedia of Philosophy" [1] é uma boa bibliografia inicial sobre o assunto.

de disciplinas básicas das artes liberais: gramática, lógica e retórica. O trivium era base para o *quadrivium*, conjunto de disciplinas formado pela aritmética, geometria, música e astronomia. Neste período, há desenvolvimentos relevantes no oriente médio e vizinhança, particularmente na pérsia, sub-continente asiático (Índia) e Oriente.[3] Outras contribuições significativas são de Gottfried Willhem Leibniz (1646-1716). Leibniz, grande nome da matemática, idealizou uma linguagem universal a *lingua characteristica universalis* para expressar todo conhecimento humano. Desenvolveu também o *Calculus ratiocinator*: modelo universal teórico de cálculo lógico - executado por máquinas - para derivar relações lógicas. Esta contribuição é relevante devido ao objetivo computacional. Tanto que Leibniz é referido como o "Patron saint of Computer Science" por Moshe Vardi, na *Communications of the ACM*, no Editorial de Dezembro de 2011.

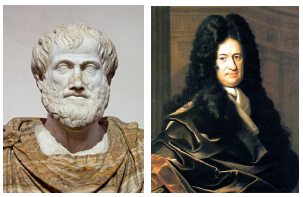


Figura 1.1. Aristoteles, G. W. Leibniz

No século XIX há grandes desenvolvimentos em lógica matemática. George Boole (1815-1864) nome de forte impacto em matemática (posteriormente em ciência da computação, economia e engenharia) desenvolve a lógica proposicional, sob rigorosa formalização. Historiadores da ciência afirmam que Boole, de fato, introduz a lógica simbólica em *An Investigation of the Laws of Thought (1854)*. No mesmo século, em torno de 1879, Gottlob Frege (1848-1925), no *Begriffsschrift*, formaliza a Lógica de Primeira Ordem, introduzindo definições precisas da quantificação existencial e universal em lógica matemática. Augustus De Morgan (1806-1871) formaliza a lógica das relações e define *indução matemática* de forma rigorosa - técnica hoje muito utilizada em ciência da computação - além de formalizar as leis que levam seu nome. Além desses, Charles Sanders Peirce (1839-1914) define também o raciocínio abdutivo, propõe uma definição rigorosa de indução matemática e princípios de lógica relacional, estendendo trabalho de De Morgan. Ademais, mostra que a utilização do operador *nor* permite representar todas as funções booleanas. No final do século XIX, Bertrand Russell mostra que o sistema de Frege é inconsistente. Isto leva Russell ao desenvolvimento da teoria de tipos em lógica matemática (que terá certa influência e inspiração sobre a teoria de ti-

[3]Para melhor entendimento dos estudos em lógica desenvolvidos neste período histórico recomendamos a consulta do *Handbook of Philosophical Logic* e *Handbook of the History of Logic*. Estas importantes obras modernas de organização do conhecimento sobre lógica pura e aplicada publicaram capítulos específicos sobre o desenvolvimento da lógica nestas regiões e nestes períodos históricos [20, 22].

pos em ciência da computação, particularmente na área de linguagens de programação).

Figura 1.2. Grandes Lógicos do Século XIX: Boole, De Morgan, Frege, Peirce

Entre 1903 e 1910, Alfred North Whitehead (1861-1947) e Bertrand Russell (1872-1970) escrevem o monumental *Principia Mathematica* [52], em 3 volumes, com mais de 2 mil páginas. O objetivo era formalizar na linguagem da lógica, utilizando axiomas e regras de inferência, todo o conhecimento matemático. A partir deste período, a lógica matemática e a teoria de conjuntos estabelecem-se como fundamentos da matemática moderna.[4]

Luitzen Egbertus Jan Brouwer (1881-1966), desenvolve a lógica intuicionista [39, 49]. A lógica intuicionista (ou construtiva) rejeita a lei do meio excluído em seu sistema, a eliminação da dupla negação $\neg\neg\alpha \nvdash \alpha$, assim como provas por contradição. Assim, as provas nesta lógica são construtivas, o que remete à relação com conceitos de computação. O isomorfismo de Curry-Howard, simplificadamente, relaciona provas construtivas (intuicionistas) e programas. A lógica intuicionista tem sido pesquisada

[4]O *Principia* ou PM, como também é conhecido, foi considerado um dos 100 livros de não-ficção mais importantes do Século XX pelo New York Times(http://www.nytimes.com/library/books/042999best-nonfiction-list.html). Whitehead e Russell foram filósofos de grande influência no Século XX. Russell foi considerado por muitos especialistas o "filósofo do século" [42]. Whitehead foi considerado genial pela famosa intelectual da primeira metade do século, Gertrude Stein, que nas primeiras frases da "Autobiografia de Alice B. Toklas" (na verdade sua própria autobiografia, uma obra vanguardista na cultura ocidental) afirmou *"... that only three times in my life have I met a genius ... the three geniuses of whom I wish to speak are Gertrude Stein, Pablo Picasso and Alfred Whitehead."*

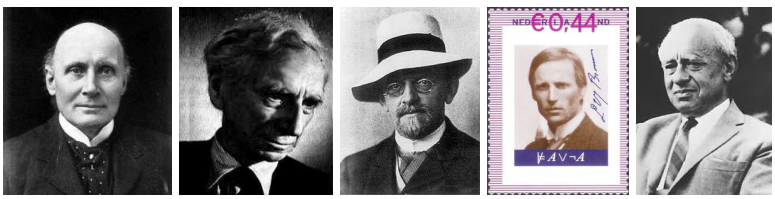


Figura 1.3. Alfred Whitehead, Bertrand Russell, David Hilbert, L.E.J. Brouwer, Alfred Tarski

como um sistema lógico que pode ser fundamental em diversas áreas da ciência da computação, incluindo na teoria de tipos de linguagens de programação.

Na década de 1920, David Hilbert (1862-1943), um dos matemáticos mais influentes do início do século XX, propôs a busca por uma nova fundamentação para a matemática. Sinteticamente, Hilbert queria mostrar (1) a completude da matemática: todas as verdades podem ser provadas. (2) a consistência da matemática: uma afirmação matemática e sua contradição não podem ser ambas provadas através de um sistema lógico. (3) a decidibilidade: existe uma forma mecânica de mostrar se uma afirmação matemática arbitrária qualquer é verdadeira ou falsa. Através de um procedimento formal, Hilbert queria mostrar que era possível construir todas as verdades matemáticas. Deste programa deriva-se o *Entscheidungsproblem* (ou "problema da decisão") - demonstrar a (in)existência de um algoritmo para determinar se um enunciado da lógica de primeira ordem pode ser provado. Em 1922, Ludwig Wittgenstein publica o *Tractatus Logico-Philosophicus* [53]. O *Tractatus* é uma obra influente na filosofia analítica e estabeleceu o seu autor como um dos filósofos mais importantes do século XX. Nesta obra, Wittgenstein define, entre diversas conceituações notáveis, a noção de tabelas-verdade, na forma em que passam a ser utilizadas até hoje.

Na década de 1930 Kurt Gödel, Alan Turing e Alonzo Church estudam as limitações da lógica de primeira ordem.[5] Entre as contribuições fundamentais de Gödel, citamos o seu primeiro teorema da incompletude: um sistema axiomático suficientemente poderoso para descrever a aritmética dos naturais, não é completo e o segundo teorema de Gödel: em um sistema axiomático suficientemente poderoso para descrever a aritmética dos naturais, a consistência destes axiomas não pode ser provada pelo próprio sistema. Tais resultados responderam negativamente ao programa de Hilbert, mostrando não ser possível definir um sistema axiomático correto e completo para toda matemática. É interessante observar que estudos em decidibilidade contribuem decisivamente para o desenvolvimento da noção de computibilidade. Neste mesmo período histórico e até os anos 1950, a definição precisa de consequência lógica, a noção de verdade em uma estrutura e trabalhos fundamentais na teoria de modelos são desenvolvidos por Alfred Tarski.

Além disso, Gödel desenvolve um sistema completo e correto para a lógica de primeira ordem. Church encontra uma solução negativa ao *Entscheidungsproblem* [5]: é impossível decidir algoritmicamente se afirmações da aritmética são verdadeiras ou falsas; Church define a noção de *calculabilidade* e também introduz o cálculo-λ para estudo das funções referidas por ele como calculáveis - mas referidas por Turing como *computáveis*. Posteriormente, o cálculo-λ também passa a ter forte influência na ciência da computação, principalmente na área de linguagens de programação, a partir da dé-

[5] A contribuição de Kurt Gödel foi inclusive popularizada pelos grandes meios de comunicação da imprensa. Em 1999, Gödel foi escolhido o matemático do século pela revista *Time Magazine*.

cada de 1960. De grande influência na comunidade de lógica, Church orientou as teses de nomes relevantes em lógica e ciência da computação como: A.M. Turing, S. Kleene, J.B. Rosser, D.S. Scott, M.O. Rabin, H. Rogers Jr., R. Smullyan e Leon Henkin.

Atualmente, em poucos textos a referência ao termo *calculabilidade* é encontrada. No famoso artigo de Turing [46], ele faz referência pioneira ao termo "computable" enquanto esclarece que o conceito de "calculable" (calculável) de Church ao estudar funções calculáveis é equivalente à sua definição de computável [46]:

> *Although the class of computable numbers is so great, and in many ways similar to the class of real numbers, it is nevertheless enumerable. In §8 I examine certain arguments which would seem to prove the contrary. By the correct application of one of these arguments, conclusions are reached which are superficially similar to those of Gödel [1]. These results have valuable applications. In particular, it is shown (§11) that the Hilbertian Entscheidungsproblem can have no solution. In a recent paper Alonzo Church [2] has introduced an idea of "effective calculability", which is equivalent to my "computability", but is very differently defined. Church also reaches similar conclusions about the Entscheidungsproblem [3]. The proof of equivalence between "computability" and "effective calculability" is outlined in an appendix to the present paper.*
>
> [1] Uber formal unentscheidbare Satze der Principia Mathematica und verwandter Systeme, I. *Monatshefte fur Mathematik und Physik*, 38(1931):173-198.
>
> [2] An unsolvable problem of elementary number theory, *Amer. J. Math*, 58(1936): 345-363.
>
> [3] A note on the Entscheidungsproblem. *J. of Symbolic Logic*, 1(1936): 40-41.

Ademais, em [46] as noções de algoritmo, procedimento efetivo e computabilidade são explicitadas. A partir das contribuições de Turing, Church, Gödel e muitos outros, a Ciência da Computação passa a ser desenvolvida sob rigor científico e, fundamentalmente, lógico-matemático. As contribuições de Church e Turing levam à formulação da Tese de Church-Turing: uma função sobre números inteiros é computável se, e somente se, ela é computável em uma Máquina de Turing. Outra ideia de Turing - o programa armazenado em fita tem grande impacto, inclusive tecnológico. Este princípio muito simples influenciou von Neumann no desenvolvimento do conceito de programa armazenado em memória.

A partir do final da segunda guerra mundial, o desenvolvimento da Ciência da Computação acontece de forma muito acelerada. O próprio desenvolvimento do transistor, no final da década de 1940, a construção dos primeiros computadores e o desenvolvimento do modelo de arquitetura de computadores conhecido como modelo de von Neumann, levam a impressionantes impactos sociais e econômicos. Ainda nesta

época, Turing novamente causa impacto, ao levantar questões relacionados ao raciocínio e aprendizado de máquinas, i.e., pavimentando o caminho para o surgimento da Inteligência Artificial. Evidentemente, para formalizar raciocínio torna-se necessário o desenvolvimento de novos sistemas lógicos, "implementáveis" em computadores. A partir da década de 1960 novas lógicas são desenvolvidas em ciência da computação, tendo em vista a evolução da ciência e a necessidade de especificar, verificar e raciocinar sobre programas, sistemas e construções computacionais. Em suma, o impacto da lógica em computação foi tão significativo que muitos pesquisadores se referem à lógica como o "cálculo da computação" [25, 36].

Figura 1.4. Alan Turing, FRS (1912-54), Kurt Gödel (1906-78), Alonzo Church (1903-95)

1.2.2. Lógica em Computação: Uma Breve Introdução

A seguir, apresentamos uma introdução muito básica à lógica proposicional. Há excelentes livros que recomendamos para o leitor, com extenso material sobre o assunto, inclusive sobre aspectos relacionados à Ciência da Computação. Um dos melhores textos em língua portuguesa é o livro de Corrêa da Silva, Finger e Melo [7]. Neste livro, são abordados em detalhe temas como SAT (satisfatibilidade), prova automática de teoremas e verificação de programas. Este capítulo tem como objetivo, apenas, oferecer uma breve introdução à lógica em computação. Na literatura há outras obras de grande abrangência e qualidade sobre lógica em computação, e.g. [3, 19, 29].

1.2.3. Lógica Proposicional

Lembre-se que para falar sobre algum domínio, é necessário estabelecer (isto é, definir) uma *linguagem*. Em sistemas lógicos, as noções de sintaxe, semântica, e teoria de prova (sistemas de prova) são fundamentais. A teoria de prova se refere à forma sintática de obter ou identificar as afirmações válidas da linguagem.

Também dizemos que a sintaxe se refere às regras utilizadas para construir as fórmulas de um sistema lógico. Por sua vez, a semântica se refere ao significado destas fórmulas bem-formadas (formadas de acordo com regras definidas de sintaxe), mapeando as sentenças a uma interpretação. Em ciência da computação, estes conceitos também são essenciais, tanto do ponto de vista fundamental, quanto do ponto de vista de aplicações, como em linguagens de programação. Assim, o conhecimento básico de

lógica clássica torna-se útil em um grande número de aplicações. Tipicamente, a lógica clássica inclui e tem como objeto de estudo a lógica proposicional e a lógica de predicados de primeira ordem. Iniciaremos, desta forma, nossa abordagem através de breves introduções à lógica proposicional e à lógica de predicados.

Em lógica proposicional as afirmações atômicas são *proposições*: fatos que podem ser verdadeiros/falsos em um determinado mundo possível/situação. Para representar variáveis proposicionais, por exemplo, podemos usar $p, q, r...$; para representar os conetivos (ditos "Booleanos") tipicamente utilizamos $\neg, \vee, \wedge, \rightarrow$ (ou $\supset$)[6]. Estes símbolos de conetivos representam negação (conetivo unário), disjunção ("ou" lógico), conjunção ("e" lógico) e implicação material ("se então", informalmente). Exemplos de formulae podem ser construídos a partir destes símbolos da linguagem proposicional: $\neg(p \wedge q) \rightarrow (\neg p \vee \neg q)$; $(p \rightarrow q) \rightarrow (\neg p \vee q)$; $(p \rightarrow q) \rightarrow p$.

Note que a lógica proposicional tem expressividade limitada. Por exemplo, a afirmação *todos os homens são mortais*, é expresso como um átomo proposicional, abstraindo-se relações e quantificações. Os conetivos "Booleanos" têm uma interpretação similar à utilizada em linguagem natural. Assim, $p \wedge q$ intuitivamente significa que p e q são interpretados como verdadeiros em uma determinada situação. Lógicas são utilizadas para aplicações, por exemplo, em representação do conhecimento (humano). Portanto, encontrar e estabelecer correspondência entre lógica(s) e linguagem natural é essencial nestas aplicações. Esta "tradução" nem sempre é simples. Lembre que a lógica clássica nos permite representar somente afirmações. A idéia fundamental na tradução é tentar extrair proposições atômicas de uma afirmação e combiná-las através de conetivos. Por exemplo, "Maria gosta de João e de Pedro" pode ser "traduzida" como "Maria gosta de João $\wedge$ Maria gosta de Pedro" e formalizada como $(p \wedge q)$. Por sua vez, a afirmação (proposição) "Spinoza não gosta de Tomás" pode ser representada como $\neg$(Spinoza gosta de Tomás) e formalizada simplesmente como $\neg p$. Note que esta formalização proposicional abstrai relações entre objetos/pessoas das afirmações em linguagem natural. Finalmente, um exemplo envolvendo implicação material (ou um "condicional"): "Se o Internacional vence, então todos estamos felizes" pode ser traduzida como "Inter vencedor $\rightarrow$ felizes" e formalizada $p \supset q$ ou $p \rightarrow q$. No entanto, como a linguagem natural é ambígua, encontrar uma tradução aceita universalmente nem sempre é possível: por exemplo, "Pedro será rico e famoso" não pode ser apenas

[6] A sintaxe para o condicional $p \rightarrow q$ é hoje a mais comumente utilizada. Na notação do Principia Mathematica [52] a implicação material seria representada como $p \supset q$. No entanto, deve-se observar que existem diversos condicionais em lógica(s). O condicional da lógica clássica é conhecido como "implicação material", sendo que $p \rightarrow q$ é logicamente equivalente a $\neg p \vee q$. No entanto, esta equivalência não é válida em outros sistemas lógicos não-clássicos, por exemplo em lógica intuicionista, na qual esta equivalência não é um teorema. Há diversas outras formalizações de condicionais: contrafatuais, condicionais estritos, causais e outros. No entanto, o escopo deste capítulo é restrito, e não exploraremos estes "condicionais não-clássicos" [4].

traduzido como "Pedro será rico $\wedge$ Pedro será famoso", pois Pedro poderá ficar famoso somente após perder sua fortuna em um cassino, o que possivelmente seria melhor formalizado com uma *lógica temporal*. Algumas fórmulas são relativamente simples: Por exemplo, "*está chovendo* $\rightarrow$ *Candide não sai de casa*" pode ser traduzido como "se está chovendo então Candide não sai de casa". Entretanto, *não* devemos ler a fórmula $p \rightarrow q$ como "p causa q", pois a relação de causa e efeito tem interpretação distinta da implicação material da lógica clássica.

É claro que nem sempre existe uma correspondência "trivial" entre aquilo que é expresso através de uma linguagem natural e através de uma linguagem simbólica de uma lógica. Em computação, a fim de expressarmos as afirmações e o conhecimento sobre um determinado domínio, idealmente imaginamos que não existam ambiguidades nas interpretações de fórmulas. Na lógica clássica, trabalhamos com valores-verdade de fórmulas em situações nas quais determina-se se os componentes (átomos) de fórmulas são verdadeiras ou não. Se pensarmos num programa de computador, avaliaríamos um teste condicional (um "if") em função dos valores das variáveis do programa. Para átomos $p, q, r, ...$ uma situação especificará os valores-verdade dos mesmos, i.e. se eles são verdadeiros ou falsos nesta situação. Note que os valores-verdade dos átomos podem ser diferentes em diferentes situações. O valor-verdade de uma fórmula proposicional em uma situação é definida como a seguir. É conveniente incluir $\top$ e $\bot$ como as fórmulas que são sempre verdadeiras e sempre falsas, respectivamente. Assim, $\top$ e $\bot$ denotam, também, tautologias e contradições, respectivamente. $\top$ é verdadeiro e $\bot$ é falso; $\neg p$ verdadeiro se p é falso, e falso se p é verdadeiro. $p \wedge q$ é verdadeiro se p, q são ambos verdadeiros; caso contrário, $p \wedge q$ é falso. $p \vee q$ é verdadeiro se um ou mais dentre p, q são verdadeiros, falso em caso contrário. $p \rightarrow q$ é verdadeiro se p é falso ou q é verdadeiro (ou ambos). Caso contrário, a implicação é falsa.[7] As noções de *verdade, validade e equivalência* são essenciais em lógica. Uma fórmula proposicional é logicamente válida se ela é verdadeira em qualquer situação. Escreve-se $\models p$ se p é válida, para uma fórmula arbitrária p. Fórmulas proposicionais válidas são também conhecidas como *tautologias*.

Por sua vez, uma fórmula p é uma *contradição* se ela for falsa em todas interpretações. Uma fórmula proposicional é *satisfatível* (ou ainda *consistente*) se ela é verdadeira em pelo menos uma situação. Duas fórmulas p, q são logicamente equivalentes se elas são verdadeiras em exatamente nas mesmas situações. Podemos escrever $p \equiv q$ neste caso. Outra noção relevante é a de consistência de um conjunto de fórmulas. Um conjunto de fórmulas bem formadas $\{A_1, A_2, ..., A_n\}$ é consistente se existe uma

[7] Podemos expressar o significado definido acima através de *tabelas-verdade*. Ludwig Wittgenstein, no *Tractatus Logico-Philosophicus* [53] define a noção de tabelas-verdade, na forma em que passaram a ser utilizadas até hoje. Cada linha de uma tabela-verdade corresponderá a uma *interpretação* da fórmula para a qual construímos a tabela.

interpretação de sua conjunção que é verdadeira; i.e. se valores podem ser atribuídos a todas as sentenças atômicas no conjunto de fórmulas $\{A_1, A_2, ..., A_n\}$ que fazem com que as fórmulas sejam verdadeiras. Por exemplo, o conjunto $\{p \rightarrow q, \neg p, \neg q\}$ é consistente, enquanto que o conjunto $\{r \rightarrow s, r, \neg s\}$ não é consistente.

Dizemos que uma fórmula q é *consequência lógica* de um conjunto de fórmulas $p_1, p_2, ...p_n$, representado por $p_1, p_2, ...p_n \models q$ se em toda situação (interpretação) na qual $p_1, p_2, ...p_n$ são todas fórmulas verdadeiras, q também é verdadeira. Note que esta é uma noção de consequência *semântica*. Podemos, também definir consequência lógica como uma noção *sintática*; neste caso, $p_1, p_2, ...p_n \vdash q$ significa que existe um *prova* de q a partir das fórmulas $p_1, p_2, ...p_n$, utilizando um sistema de prova (e.g. sistema de dedução natural, tableaux, etc). Uma *prova* é uma sequência (finita) de fórmulas bem-formadas que são deduzidas a partir de outras fórmulas através do uso de regras de um sistema de provas ou de axiomas. Resultados anteriormente já provados (teoremas) podem ser utilizados como *lemas* em provas. Note que um sistema de provas utiliza regras puramente sintáticas. Podemos dizer que q pode ser *provado* a partir das premissas $p_1, p_2, ...p_n$. Assim, as noções de consequência lógica no nível sintático ($p_1, p_2, ...p_n \vdash q$) e semântico ($p_1, p_2, ...p_n \models q$) são claramente distintas. O conjunto de fórmulas (premissas) $p_1, p_2, ...p_n$ é também chamado de teoria. A construção sintática $p_1, p_2, ...p_n \vdash q$ é denominada, também, de *sequente*. As fórmulas à esquerda de $\vdash$ são chamadas de antecedente, enquanto que a fórmula à direita de $\vdash$ é chamada de consequente. Por sua vez, *teoremas* são fórmulas que seguem logicamente de outras fórmulas válidas; são afirmações deduzidas a partir de outras afirmações (e que são, obviamente, válidas) em um cálculo formal. Formalmente, no entanto, um teorema é a última linha de uma prova. A notação $\vdash \alpha$ denota que α é um teorema. Existem diversos métodos para verificação de validade de fórmulas, incluindo: (1) tabelas-verdade, onde simplesmente verificamos mecanicamente todas as situações relevantes; têm crescimento exponencial no número de subfórmulas atômicas e são aplicáveis somente à lógica proposicional; (2) argumentação direta (utilizado por matemáticos/lógicos), que é relativamente rápida, desde que o usuário seja familiarizado com o método; (3) Sistemas de prova: sistemas de Hilbert (axiomáticos), tableaux, dedução natural e métodos automáticos (provadores automáticos de teoremas - "automated theorem provers").

Propriedades importantes de sistemas lógicos (ditas metapropriedades) incluem a *correção* e a *completude*. Se todas as fórmulas válidas de um sistema lógico são provadas por um sistema de provas para este sistema, ele é dito *completo* em relação ao modelo semântico; enquanto que a *correção* garante que as provas obtidas por este sistema de prova efetivamente são válidas em relação ao modelo semântico. Por exemplo, para a lógica proposicional, existem tanto sistemas de prova axiomáticos, quanto sistemas de dedução natural que são *corretos e completos*.

1.2.3.1. Exemplo de Sistema de Provas: Sistema Axiomático (*à la* Frege-Hilbert)

Existem diversos sistemas de prova para lógica proposicional e de primeira ordem. Entre as diversas formas de apresentação de sistemas lógicos, sistemas axiomáticos estão entre as mais utilizadas. Esta apresentação também é conhecida como apresentação *à la* Frege-Hilbert, ou, no "estilo de Hilbert" [37]. O sistema axiomático é historicamente muito utilizado por filósofos e matemáticos. Tem como limitação (en. "drawback") o fato de levar à construção de provas complexas, mesmo para experientes. Um sistema axiomático consiste de fórmulas que são definidas como axiomas e pela especificação de suas regras de inferência. Para exemplificarmos a sua utilização, apresentamos a seguinte axiomatização da lógica proposicional. Se α, β e γ são quaisquer fórmulas bem-formadas de L, então os seguintes são axiomas de L [37]; esta axiomatização utiliza apenas os conetivos $\rightarrow, \neg$.
(A1). $(\alpha \rightarrow (\beta \rightarrow \alpha))$
(A2). $((\alpha \rightarrow (\beta \rightarrow \gamma)) \rightarrow ((\alpha \rightarrow \beta) \rightarrow (\alpha \rightarrow \gamma)))$
(A3). $((\neg\beta \rightarrow \neg\alpha) \rightarrow ((\neg\beta \rightarrow \alpha) \rightarrow \beta))$.
Uma regra de inferência de L: *modus ponens*: $\alpha, \alpha \rightarrow \beta \vdash \beta$
Exemplo simples: prova em L de $A \rightarrow A$.

1. $(A \rightarrow ((A \rightarrow A) \rightarrow A)) \rightarrow ((A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A))$ (Instância Ax. A2)
2. $A \rightarrow ((A \rightarrow A) \rightarrow A)$ Esquema Ax. A1
3. $(A \rightarrow (A \rightarrow A)) \rightarrow (A \rightarrow A)$ 1, 2, MP
4. $A \rightarrow (A \rightarrow A)$ Esquema Ax. A1
5. $A \rightarrow A$ A partir de 3, 4 por MP

Há mais de uma forma de axiomatizar a lógica proposicional. O sistema axiomático proposto por Hilbert para a lógica proposicional inclui a regra de inferência modus ponens e os seguintes axiomas:

1. $A \rightarrow (B \rightarrow A)$
2. $(A \rightarrow (B \rightarrow C)) \rightarrow ((A \rightarrow B) \rightarrow (A \rightarrow C))$
3. $(A \rightarrow B) \rightarrow (\neg B \rightarrow \neg A)$
4. $\neg\neg A \rightarrow A$
5. $A \rightarrow \neg\neg A$

Embora a apresentação axiomática seja sintética, a construção de provas neste sistema é notadamente mais difícil do que em outros sistemas de prova.

1.2.3.2. Exemplo de Sistema de Provas: Sistema de Dedução Natural

O método de dedução natural, foi proposto por Gentzen [23], e refinado por Fitch [16] em um formato onde provas e subprovas são representadas hierarquicamente. O método consiste em uma representação formal da argumentação direta, em que utilizamos regras do sistema para manipular premissas e obter conclusões. A formalização do raciocínio é feita através de regras ditas de *introdução* e de *eliminação*. Regras de *introdução* permitem introduzir em uma linha de prova uma fórmula que utilizam um determinado conetivo que está sendo "introduzido". Regras de *eliminação* permitem a inferência de uma nova fórmula em que um conetivo foi "eliminado" (estas noções são detalhadas a seguir). Nossa representação utilizará o formato de "boxes" (caixas) que lembram a hierarquia da estrutura original no formato de árvores como Gentzen utilizava. Os livros [3, 4, 19, 29] utilizam os formatos de caixas, que - de certa forma, ressalte-se - lembram também a estrutura de um programa de computador. O método de dedução natural é muito utilizado como formalização do raciocínio utilizado em provas por especialistas; i.e. existe uma correspondência próxima com a forma de como teoremas são provados (na "prática" lógico-matemática) e como são construídas as relações entre premissas e conclusões em dedução natural. Em vez de axiomas, há regras de inferência, que descrevem o que pode ser derivado a partir de premissas e hipóteses.

Em síntese, o sistema de dedução natural tem uma série de vantagens em relação a outros sistemas, como em relação ao sistema axiomático, pois: (i) concentra-se na noção de dedução, que é a noção fundamental em lógica; (ii) evita que tenhamos de definir um conjunto de fórmulas como axiomas; (iii) freqüentemente as regras de dedução natural são mais fáceis de serem manipuladas. (iv) de certa forma, ele espelha a forma como os seres humanos realizam inferências.

As Regras de Dedução do Sistema de Dedução Natural

A implicação material e uso do raciocínio hipotético: Nas ciências exatas o raciocínio hipotético é fundamental. Existem inúmeras publicações em filosofia da ciência sobre o tema, e neste curso apresentamos algumas formalizações elementares sobre o mesmo. Lembre-se que lógica é fundamentalmente baseada sobre o raciocínio condicional ("se... então...") e este padrão de raciocínio constitui a espinha dorsal das derivações (algumas não triviais) obtidas através de raciocínios hipotéticos. O método de dedução natural ilustra muito bem a formalização metódica deste tipo de raciocínio, fundamental na ciência. Nas regras que manipulam a implicação material "$\rightarrow$", a negação "$\neg$", a disjunção "$\vee$" e nas provas por contradição, a utilização de hipóteses é essencial. Hipóteses são fórmulas utilizadas de uma forma distinta das demais fórmulas. Usa-se suposições (através de raciocínios em situações *hipotéticas*) para construir uma situação

na qual esta fórmula é válida. Ou seja, *assumimos* que em uma determinada situação uma hipótese (fórmula) é válida a fim de obter alguma conclusão a partir desta hipótese (fórmula). Por que usamos hipóteses? De forma simplificada, para derivar informações adicionais (em subprovas) sobre a prova maior que estamos tentando construir.

Em dedução natural, uma das ***regras para o condicional*** $\rightarrow$ faz uso de hipóteses. Assim, a *regra de introdução da implicação material* ($\rightarrow I$) é explicada a seguir: Para introduzirmos (i.e. mostrarmos) uma fórmula $\alpha \rightarrow \beta$ em uma linha de prova, assume-se α e então constrói-se a prova de β. Pode-se utilizar α e as demais fórmulas anteriormente derivadas nesta subprova. No entanto, ao obter β, a hipótese α **não** pode ser utilizada novamente, pois esta era uma hipótese adicional para provar β. Esta prova é isolada em uma caixa para indicar que a hipótese é válida apenas "localmente". Note que hipóteses são anotadas entre colchetes "[α]". *Eliminação da implicação material*: Para "eliminar" a implicação material em $\alpha \rightarrow \beta$ é necessário também provar α, e assim podemos derivar β. Dizemos "eliminação" por estarmos deduzindo uma fórmula β (que é o consequente de uma implicação $\alpha \rightarrow \beta$). Esta regra também é conhecida como *modus ponens* e é usualmente abreviada em livros de lógica como MP. A seguir, apresentamos as regras de introdução e eliminação da implicação material (representadas por $\rightarrow I$ e $\rightarrow E$, respectivamente).

1. [α] hipótese	1. $\alpha \rightarrow \beta$
2. $\vdots$	2. α
3. β	3. β $\quad \rightarrow E(1,2)$
4. $\alpha \rightarrow \beta$ $\quad \rightarrow I(1,3)$	

Regras para a conjunção $\wedge$: *Regra de Introdução*: Para introduzir (ou para escrever) uma fórmula do tipo $\alpha \wedge \beta$ em uma prova, devemos provar α **e** provar β. *Regra de Eliminação*: Se provamos $\alpha \wedge \beta$ então podemos escrever α e/ou podemos também escrever β. As regras podem ser representadas como a seguir.

1. α	1. $\alpha \wedge \beta$
2. $\vdots$	2. α $\quad \wedge E(1)$
3. β	3. β $\quad \wedge E(1)$
4. $\alpha \wedge \beta$ $\quad \wedge I(1,3)$	

Regras para a disjunção $\vee$: *Regra de Introdução*: Para provar $\alpha \vee \beta$, basta provar α ou basta provar β. (A rigor, há duas regras de introdução do $\vee$: uma a partir de α e outra para eliminação de β). Note que α,β são fórmulas quaisquer da linguagem proposicional. *Regra de Eliminação*: Para provar algo a partir de $\alpha \vee \beta$, deve-se prová-lo a partir da hipótese que α é válida e a partir da hipótese que β é válida. Esta construção é uma argumentação por casos.

1. α
2. $\alpha \vee \beta$ $\vee I(1)$

1. $\alpha \vee \beta$

2.	$[\alpha]$	hipótese	5.	$[\beta]$	hipótese
3.	$\vdots$	$\langle$prova$\rangle$	6.	$\vdots$	$\langle$prova$\rangle$
4.	γ		7.	γ	

8. γ $\vee E(1, 2-4, 5-7)$

Regras para ($\neg$): *Introdução*: Para mostrar $\neg\alpha$, assumimos α e provamos $\perp$. A hipótese não pode ser utilizada fora da subprova. Note que nesta regra, usa-se também o raciocínio hipotético, e interpreta-se $\neg\alpha$ como $\alpha \rightarrow \perp$. *Eliminação*: A partir de α e de $\neg\alpha$, provamos $\perp$.

1.	$[\alpha]$	hipótese
2.	$\vdots$	
3.	$\perp$	

4. $\neg\alpha$ $\neg I(1,3)$

1. α
2. $\vdots$
3. $\neg\alpha$
4. $\perp$ $\neg E(1,3)$

Entre as regras derivadas do sistema de dedução natural, podemos citar a regra de eliminação da dupla negação $\neg\neg E$: A partir de $\neg\neg\alpha$ provamos α. Esta regra é utilizada para simplificar passos de prova, mas pode ser derivada de outras regras do sistema de dedução natural. A seguir mostramos que, em lógica clássica, *qualquer fórmula pode ser obtida a partir de uma prova de* $\perp$ *(também referido como falsum)*. Isto é, mostramos que $\perp \vdash A$ para qualquer fórmula A.

1. $\perp$ Premissa

2.	$[\neg A]$ hipót.
3.	$\perp$ $\checkmark$ (1)

4. $\neg\neg A$ $\neg I(2,3)$
5. A $\neg\neg E(4)$

Ou seja, ao assumirmos uma situação na qual $\perp$ é verdadeiro (linha 1), se $\neg A$ é verdadeiro (linha 2), então $\perp$ é verdadeiro, o que não é possível. Portanto, temos $\neg\neg A$ (linha 4) e consequentemente, A é verdadeira nesta situação. Isto é: *"qualquer coisa" (fórmula)* segue de uma contradição. Na prova acima, anotamos uma linha com o símbolo "$\checkmark$". Esta anotação simplesmente justifica a utilização de uma fórmula já provada anteriormente (esta notação é utilizada por alguns autores, incluindo [3]).

Note que é impossível que $\perp$ seja verdadeira. $\perp$ só é provado através de hipóteses contraditórias, e isto acontece no meio de uma prova maior. Em uma subprova, podemos então deduzir aquilo que necessitamos/desejamos. Claramente, mostrar contradições, i.e. mostrar $\perp$ em uma sub-prova, pode ser útil em construção de provas.

Prova por Contradição[8] Para provar α, assume-se $\neg\alpha$ e prova-se $\bot$. Novamente, esta é uma regra que utiliza raciocínio condicional (hipotético).

1. $[\neg\alpha]$ hipótese
2. $\vdots$
3. $\bot$

4. α PC(1,3)

Regras para $\bot$: *Introdução*: Para provar $\bot$ temos de provar α e provar $\neg\alpha$, para fórmulas quaisquer. Esta regra é análoga à regra $\neg E$. *Eliminação*: A partir de uma prova de $\bot$ podemos derivar qualquer fórmula.

1. $\neg\alpha$
2. $\vdots$
3. α
4. $\bot$ $\bot I(1,3)$

1. $\bot$
2. α $\bot E(1)$

Existem também regras para $\top$, de introdução e eliminação. $\top$ pode ser introduzido em qualquer ponto de uma prova, pois é uma fórmula sempre válida, enquanto que se temos $\top$ nada podemos afirmar a partir desta fórmula.

Utilização de Lemas: Lemas são fórmulas provadas (válidas), e assim denominadas quando utilizadas na construção de uma prova maior. Por exemplo, um lema muito utilizado em dedução natural para lógica clássica é a lei do meio excluído, i.e. $\alpha \vee \neg\alpha$. Este lema - em geral - deve ser utilizado em conjunto com uma eliminação de $\vee$, onde $\neg\alpha$ e α serão hipóteses para um raciocínio por casos.

Regras para $\leftrightarrow$: Lembre-se que $\alpha \leftrightarrow \beta$ é equivalente a $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$, e também é equivalente a $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$. *Introdução*: Para provar $\alpha \leftrightarrow \beta$, devemos provar α e β ou devemos provar $\neg\alpha$ e $\neg\beta$. Devido à equivalência acima, também podemos derivar $\alpha \leftrightarrow \beta$ ao provarmos $(\alpha \rightarrow \beta)$ e $(\beta \rightarrow \alpha)$. *Eliminação*: A partir de $\alpha \leftrightarrow \beta$, podemos provar $(\alpha \rightarrow \beta)$ e $(\beta \rightarrow \alpha)$ e $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$. Assim como as demais regras para os conetivos, as regras para $\leftrightarrow$ também estão representadas na Figura 1.6.

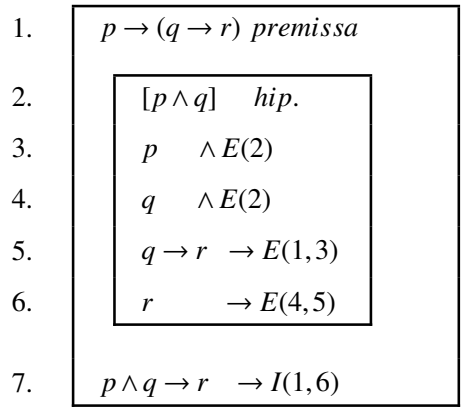
1. α
2. $\vdots$
3. β
4. $\alpha \leftrightarrow \beta$ $\leftrightarrow I(1,3)$

1. $\alpha \leftrightarrow \beta$
2. $(\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$ $\leftrightarrow E(1)$
3. $(\alpha \rightarrow \beta) \wedge (\beta \rightarrow \alpha)$ $\leftrightarrow E(1)$

A seguir, apresentamos a prova de um sequente simples, representada na Figura 1.5. Na prova do sequente $p \rightarrow (q \rightarrow r) \vdash p \wedge q \rightarrow r$, escrevemos a premissa $p \rightarrow (q \rightarrow r)$

[8] Também conhecida como redução ao absurdo (Reductio ad absurdum - RAA).

Figura 1.5. Exemplo: Prova do seqüente $p \to (q \to r) \vdash p \wedge q \to r$ através do sistema de dedução natural, ilustrada nesta figura.

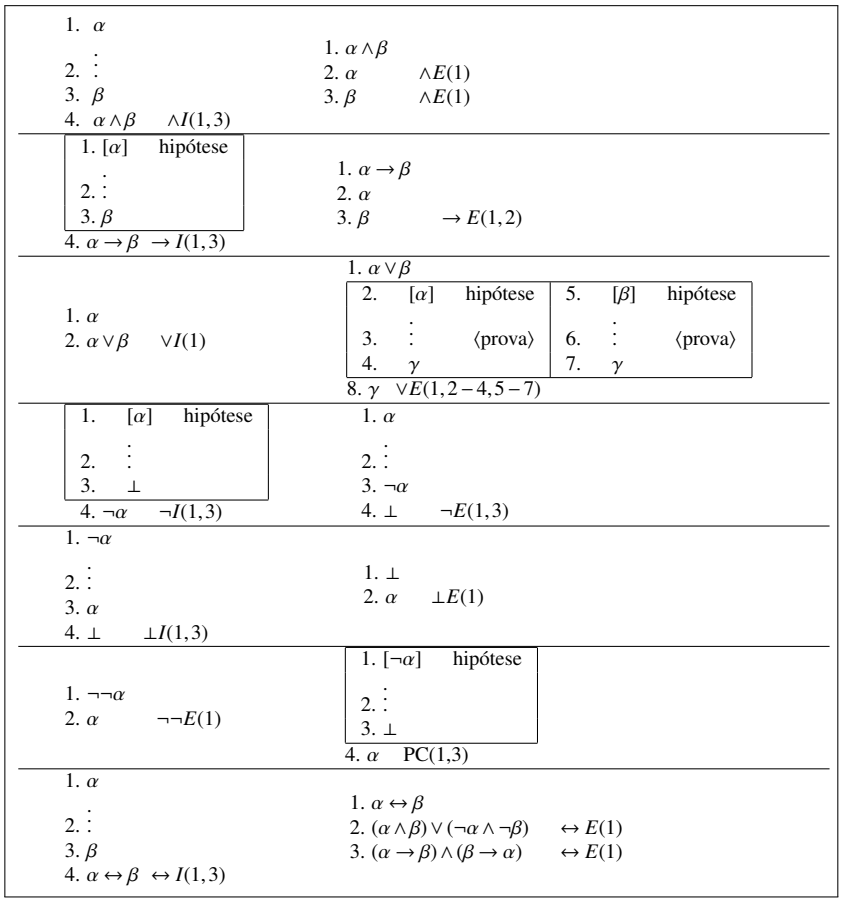


na linha 1 da Figura 1.5. O nosso objetivo é provar $p \wedge q \to r$ (que está na linha 7). A conclusão a ser provada é uma implicação material, $\alpha \to \beta$. Para provar uma implicação material, assumimos como hipótese o antecedente (α) da implicação: portanto, na linha 2 assumimos $[p \wedge q]$ com o objetivo de mostrarmos r. Note que r ocorre como consequente de uma implicação, na linha 1. Assim, será necessário usar a regra de eliminação do condicional $\to E$ (modus ponens) para mostrarmos r. Isto é feito sucessivamente: primeiramente mostramos p, na linha 3 e posteriormente mostramos q na linha 4. Ao mostrarmos estes antecedentes das implicações da linha 1, finalmente podemos mostrar q, a partir de duas aplicações de modus ponens (linhas 5 e 6). Note que a prova da Figura 1.5 não é desenvolvida apenas de "cima para baixo". Uma analogia muitíssimo simples sobre como construir uma prova nos remete à construção de caminhos entre dois pontos em um mapa. Ou seja, a construção de uma prova muitas vezes é análoga à construção de um caminho: temos de saber o ponto de saída (premissas, hipóteses) e o ponto de chegada (a conclusão, i.e. a fórmula que temos de provar). A partir destes pontos construímos o "caminho" (a prova) entre estes "pontos", usando este raciocínio, recursivamente, até que o caminho (prova) esteja concluído.

As regras aqui apresentadas permitem que todas as fórmulas válidas da lógica clássica proposicional sejam provadas usando o sistema de dedução natural. Esta propriedade é conhecida como *completude* de um sistema de provas. Um sistema lógico é completo se, e somente se, todas as fórmulas bem formadas válidas são teoremas do sistema. Idealmente, um sistema de provas teria que permitir que todas as fórmulas válidas

sejam deriváveis através das regras do sistema. Por outro lado, é desejável também que as fórmulas provadas através do sistema de prova sejam efetivamente fórmulas válidas do sistema lógico em questão. Esta propriedade é conhecida como *correção* de um sistema de provas. O sistema de dedução natural apresentado aqui é correto e completo em relação à semântica apresentada. Isto garante que o sistema de dedução natural pode ser utilizado para verificar a validade de fórmulas proposicionais clássicas. A seção acima resume regras básicas do sistema de dedução natural para a lógica clássica proposicional. Estas regras seriam diferentes para outros sistemas lógicos. Na lógica intuicionista, por exemplo, não se admitem provas por contradição, nem eliminação da dupla negação $\neg\neg E$ e não é aceita a lei (ou o princípio) do meio excluído $\alpha \vee \neg\alpha$, considerado válido na lógica clássica.

Figura 1.6. Resumo das Regras para os Conetivos Clássicos. As linhas da tabela servem como referência. Obviamente poderíamos adotar uma numeração genérica para as linhas de prova; mas como exemplo didático, a numeração a seguir é clara.



1.2.4. O Problema da Satisfatibilidade (SAT)

O problema da satisfatibilidade - SAT - consiste em determinar se existe uma interpretação que satisfaça uma fórmula da lógica proposicional. Dada uma fórmula α existe uma atribuição v tal que $v(\alpha) = 1$ (ou *true*)? Embora o enunciado seja "simples" e extensivamente estudado, apresenta um dos grandes desafios da Ciência da Computação. Até hoje, não há algoritmo em tempo polinomial para o problema. Steven A. Cook [9] identificou a relevância de SAT em um clássico artigo, publicado no início da década de 1970 [6]. O artigo é intitulado *The Complexity of Theory Proving Procedures*. O próprio título já indica a forte relevância que as questões de lógica tem e sua relação com a área de complexidade computational. Neste artigo, Cook formalizou as noções de redução em tempo polinomial e mostrou que SAT é um problema NP-completo. Neste artigo, também foi formulada famosa questão P versus NP, problema que ainda está em aberto.

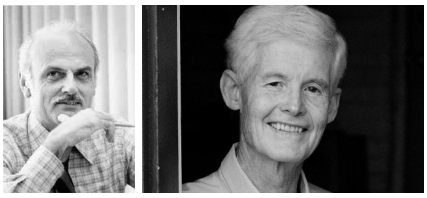


Figura 1.7. Edgar (Ted) F. Codd, Stephen A. Cook

1.2.5. Lógica de Primeira Ordem

A lógica proposicional, em termos de poder de expressão, é mais "fraca" que a lógica de predicados de primeira ordem. Por exemplo, a afirmação (proposição) "um herói é alguém admirado por todos" é representada simplesmente como uma proposição em lógica proposicional, digamos p; note que, por exemplo, o predicado "admirado" e o quantificador "todos" são abstraídos na representação. A lógica de predicados aumenta o poder de expressão da lógica proposicional ao adicionar a possibilidade de expressarmos relações entre objetos, inclusive fazendo uso de quantificadores: $\forall$ - que representa a quantificação universal e $\exists$ - que representa a quantificação existencial, além de outras extensões à linguagem. Símbolos de relação n-ários, símbolos de função, constantes e quantificadores sobre variáveis são introduzidos. Na sintaxe, define-se a noção de *assinatura*, que consiste que um conjunto de constantes, símbolos de função e símbolos de relação. A noção de *termo* é fundamental para nomear objetos (utilizamos constantes,

[9] Cook recebeu o ACM A.M. Turing Award em 1982 pelo seu trabalho pioneiro em complexidade computacional. Este prêmio, o mais importante da Ciência da Computação, tem esta denominação em reconhecimento às contribuições de Alan M. Turing.

variáveis, símbolos de função n-ários como termos). Exemplos de fórmulas atômicas na lógica de primeira ordem, construídas com símbolos de relação unária *homem* e de relação binária *amigo*, constantes *(Pedro, Celina, Luis)*, variáveis (x, y) e símbolo de função unária f: $homem(Pedro)$, $amigo(Celina, Luis)$, $\forall x \exists y(y = f(x))$. Note que constantes e símbolos de função se referem a indivíduos, a objetos, enquanto que os símbolos de relação especificam, representam relações entre estes objetos. Esta especificação de relações entre objetos não é representada de forma precisa na lógica proposicional. Ademais, a semântica da lógica de predicados é mais complexa. Neste curso, apresentamos apenas uma noção muito simplificada desta semântica, por questões de espaço.

Para que possamos interpretar esta linguagem mais expressiva, a semântica deve considerar quantificadores, variáveis e novas fórmulas atômicas, que envolvem relações. A noção de *estrutura* ou *modelo* torna-se fundamental. Uma estrutura identifica uma coleção de objetos em seu domínio, bem como o significado dos símbolos da linguagem. A interpretação de uma constante corresponde a uma constante no domínio da estrutura (ou do modelo); a interpretação de um símbolo de relação em uma estrutura corresponde a uma relação sobre os objetos do domínio desta estrutura. Uma *sentença* é uma fórmula sem ocorrência de variáveis livres, e.g. $\forall x \exists y(y = f(x))$; as variáveis deste exemplo estão sob o escopo dos quantificadores $\forall$ e $\exists$, são ditas amarradas (ou limitadas, ou ligadas) ou ainda *bound* - em inglês. Por sua vez a fórmula $\forall x(P(x, y) \rightarrow R(x, y))$ não é uma sentença, pois a variável y é livre: y não ocorre no escopo de um quantificador $\forall$ ou $\exists$. Variáveis livres são interpretadas através de uma atribuição de valor para um objeto do domínio da estrutura. De forma muito sucinta, fórmulas quantificadas existencialmente são interpretadas em uma estrutura de forma que alguma atribuição torne a fórmula verdadeira, enquanto que para fórmulas quantificadas universalmente todas as atribuições tornam a fórmula verdadeira.[10]

Portanto, a lógica de primeira ordem permite especificar propriedades de estruturas como (grafos, ordens parciais, grupos, bancos de dados) usando predicados (relações). As sentenças atômicas da lógica de predicados têm argumentos *(termos)* que denotam objetos e símbolos de predicados de n argumentos. A utilização de variáveis quantificadas pelo quantificador existencial $(\exists x)$ e universal $(\forall x)$ permite grande expressividade sobre conjuntos de objetos. Por exemplo, podemos expressar propriedades sobre grados, $G = (V, E), E \subseteq V^2$, como *Cada nodo do grafo tem pelo menos dois vizinhos distintos*: $\forall x \exists y \exists z(\neg(y = z) \wedge E(x, y) \wedge E(x, z))$. Outra observação simples, mas poderosa é que uma *estrutura relacional é essencialmente um banco de dados relacional*. Este "insight" de Ted Codd (no final da década de 1960) posteriormente levou ao desenvolvimento de uma indústria bilionária em bancos de dados relacionais. Codd posteriormente foi agraciado com o ACM Turing Award.

[10]Por questões de espaço a apresentação formal da semântica da lógica de primeira ordem não é apresentada neste texto. No entanto, pode ser compreendida através da leitura dos excelentes livros [7, 29].

1.2.5.1. Dedução Natural para Lógica de Predicados

As provas no cálculo de dedução natural para lógica de predicados são similares às provas para lógica proposicional. Introduz-se regras para os quantificadores e para o símbolo de igualdade. As regras para os conetivos são preservadas: as regras para lógica proposicional continuam válidas para a lógica de predicados. Novamente, nossa apresentação sumariza os sistemas apresentados em [4, 19, 29] que apresentam um sistema de dedução natural para lógica de predicados com estrutura hierárquica no formato de caixas. Primeiramente, temos as regras para igualdade (=). A reflexividade da igualdade permite escrever em qualquer linha de prova que $t = t$. Isto é, qualquer *termo* t é igual a si mesmo, o que na verdade é um axioma. Dada uma fórmula arbitrária α, se provamos $\alpha(t)$ e provamos que $t = u$, sendo t, u termos, então pela substituição de termos iguais = *sub* podemos mostrar $\alpha(u)$. As regras estão representadas na Figura 1.8. Exemplo: Podemos mostrar, usando as regras acima que (exemplo de [29]):

				1.	$\alpha(t)$	provado
1.	$\vdots$			2.	$\vdots$	
2.	$t = t$	=refl		3.	$t = u$	provado
				4.	$\alpha(u)$	=sub (1,3)

Figura 1.8. Regras de Dedução Natural para Igualdade (=)

$$x + 1 = 1 + x, (x + 1 > 1) \rightarrow (x + 1 > 0) \vdash (1 + x) > 1 \rightarrow (1 + x) > 0$$

1.	$(x + 1) = (1 + x)$	premissa
2.	$(x + 1 > 1) \rightarrow (x + 1 > 0)$	premissa
3.	$(1 + x > 1) \rightarrow (1 + x > 0)$	=sub(1,2)

Regras para o Quantificador Universal ($\forall$)

Para qualquer fórmula $\forall x \alpha(x)$, podemos derivar $\alpha(t)$ para qualquer termo fechado t. Isto se deve ao fato de afirmações universais serem verdade para quaisquer objetos de um domínio, isto é para quaisquer termos fechados. Assim, a regra de eliminação do quantificador universal denotada por $\forall E$ pode ser representada como na Figura 1.9. A regra de introdução do quantificador universal $\forall I$ requer a utilização de raciocínio hipotético. É necessário provar, para um termo arbitrário t, ainda não utilizado em nenhuma outra linha de prova, que $\alpha(t)$ é válido. Note que nesta regra escrevemos um termo em uma linha de prova, o que não ocorre nas demais regras do sistema de dedução natural. Note que t é arbitrário, não sendo utilizado na prova posteriormente. A anotação feita à direita $t - \forall I$, ao lado do termo t na linha 1, denota este termo arbitrário t, para o qual temos de provar a fórmula $\alpha(t)$, conforme a Figura 1.9.

Regras para o Quantificador Existencial $\exists$

Para mostrarmos uma sentença $\exists x\alpha$ através do sistema de dedução natural, temos que mostrar que $\alpha(t)$ para um termo fechado t (que não inclui variáveis). Assim, a regra de introdução do quantificador existencial $\exists I$ é representada na Figura 1.9. A eliminação do quantificador existencial exige a utilização de uma análise de casos, através de raciocínio hipotético. Lembre-se que a quantificação existencial $\exists x\alpha(x)$ significa que α é verdadeira para pelo menos uma atribuição de x; isto é $\exists x\alpha(x)$ pode ser interpretado como uma disjunção. Se provamos a sentença $\exists x\alpha(x)$, a partir da hipótese de que $\alpha(t)$ é verdadeiro construímos uma prova de β, então podemos afirmar que β foi derivado a partir de $\exists x\alpha(x)$. No formato hierárquico que estamos utilizando, a regra é representada como a seguir. Como exemplo, provamos o seguinte sequente: $\exists x\neg\alpha(x) \vdash \neg\forall x\alpha(x)$.

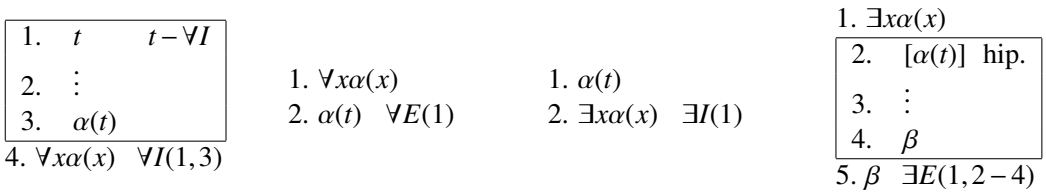
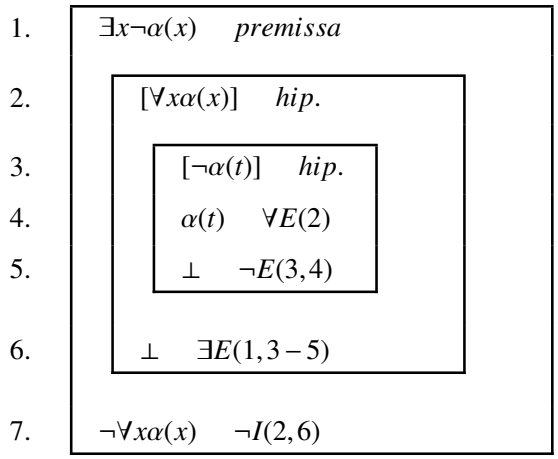


Figura 1.9. Regras de Dedução Natural para os Quantificadores $\forall$, $\exists$.



Explicação: Para provarmos $\neg\forall x\alpha(x)$, temos $\exists x\neg\alpha(x)$ como premissa. Note que a conclusão é uma fórmula negada. Portanto, podemos usar a regra de $\neg I$ para prová-la; fazemos isto assumindo $\forall x\alpha(x)$ como hipótese para derivar uma contradição $\perp$. Observe que, ao assumirmos $\forall x\alpha(x)$ e tendo a premissa da linha 1 (i.e. $\exists x\neg\alpha(x)$), a hipótese

$\neg\alpha(t)$ da linha 3 pode levar a uma contradição com a fórmula $\alpha(t)$ da linha 4. Assim, obtemos a conclusão.

1.2.5.2. Outros Métodos de Prova

Infelizmente, devido a restrições de espaço, a apresentação de outros métodos avançados, incluindo métodos de prova automática de teoremas é mais adequada para um curso de automação do raciocínio. Por exemplo, o princípio (ou método) da *Resolução* é uma regra de inferência do cálculo de predicados e utilizado extensivamente em inteligência artificial, devido à sua eficiência [31, 32]. O princípio da resolução foi estudado inicialmente por Davis e Putnam na década de 1960. Robinson estendeu o princípio, tornando-o mais eficiente, notadamente por fazer uso de um algoritmo de unificação (detalhes em [32]). A resolução é o mecanismo de execução/automação da linguagem Prolog. Neste método, todas as fórmulas envolvidas devem estar na forma clausal. Para quaisquer duas cláusulas C_1 e C_2, se existe um literal L_1 em C_1 que é complementar a um literal L_2 em C_2, então remova L_1 e L_2 de C_1 e C_2, respectivamente, e construa a disjunção das cláusulas restantes. Ou, se α,β são cláusulas e p_i, q_j são fórmulas atômicas, então a regra é:

$$\frac{\alpha \vee p_1 \vee ... \vee p_m \qquad \beta \vee \neg q_1 \vee ... \vee \neg q_m}{(\alpha \vee \beta)\theta}$$

onde θ é o *Unificador Mais Geral (UMG)* de todas as fórmulas p_i e q_j. O UMG é obtido por *unificação*; a unificação busca determinar se dois termos coincidem. No caso da lógica proposicional, é possível construir um provador de teoremas utilizando o princípio da resolução que é correto e completo. Uma característica essential da resolução é o fato deste método funcionar exclusivamente sobre sentenças em *forma clausal* - disjunções de literais (átomos ou negações de átomos). A forma clausal é utilizada por métodos automáticos de prova desde os primórdios desta área. A transformação de uma sentença em forma clausal preserva a consistência (ou inconsistência) da sentença original, e desta forma o uso de cláusulas facilita a busca por refutações. Os métodos não-clausais também fazem uso das idéias sugeridas para melhorar o desempenho de provadores de teoremas clausais.

Outro método de inferência relevante é o conhecido como tableaux analítico, desenvolvido inicialmente por Beth e aprimorado por Smullyan [7, 45]. É um método refutacional. Ou seja, para provarmos o sequente $\beta_1,...,\beta_n \vdash \alpha_1,...\alpha_m$, afirmamos os antecedentes $\beta_1,...,\beta_n$ em busca de uma contradição; se a contradição é mostrada pelo tableau, o sequente está provado.

Os assuntos relacionados à prova automática de teoremas são de alta complexidade, demandando conhecimento de diversas lógicas, algoritmos e complexidade computacional. A automação do raciocínio tem uma longa história. Ainda na década de 1950, Martin Davis construiu um programa para automatizar a aritmética de Pressburger (a teoria de primeira ordem dos naturais com adição). Allan Newell, Herbert Simon and Cliff Shaw desenvolveram o "Logic Theorist": um programa que imitava o raciocínio humano, que também foi fundamental nos primórdios da inteligência artificial. Este sistema foi capaz de provar 38 dos 52 primeiros teoremas do *Principia Mathematica* [52]. Na década de 1960, os algoritmos de Davis-Putnam (1960) e o algoritmo DPLL (Davis-Putnam-Logemann-Loveland, 1962) foram desenvolvidos. DPLL é aplicado na resolução de problemas de satisfatibilidade (SAT). Até hoje variações e extensões do algoritmo DPLL são utilizadas pelos modernos "SAT solvers". Para problemas de SAT em lógica de primeira ordem podemos utilizar SMT (Satisfiability Modulo Theories), que têm obtido sucesso notável na área de verificação, particularmente em aplicações em engenharia de software [13]. A área de projeto de circuitos integrados e aplicações críticas também fazem uso intensivo de métodos automáticos de prova.

1.2.6. Lógicas-Não-Clássicas

Em lógica clássica assumimos uma única situação, um único mundo possível em que fórmulas são avaliadas, isto é, em que assumem valores-verdade. Em diversos domínios do conhecimento, este modelo não é satisfatório. Em Ciência da Computação, nosso conhecimento do mundo, pode se referir a outros "mundos/estados possíveis" [25]. Raciocinar/refletir sobre conhecimento, computações e ações é fundamental em computação: como processos se coordenam para executar uma ação? O que robôs precisam saber para executar suas tarefas? O que programas ("agentes") precisam saber sobre os outros programas para se comunicarem? As lógicas que investigam estas questões são lógicas não-clássicas: modais, temporais, epistêmicas, entre outras.

Além disso, sistemas computacionais são dinâmicos e interativos. O tempo, por exemplo, é um aspecto fundamental, assim como estados, transições, trocas de mensagens. Para aplicações em Ciência da Computação, Inteligência Artificial entre outras, estas lógicas requerem uma formalização/axiomatização distinta da realizada através de lógica clássica. A lógica clássica é monotônica, i.e. um resultado não é refutado sob novas informações. Em inteligência artificial, no entanto, muitas vezes é necessário utilizar raciocínios não-monotônicos. Tendo em vista as características e a complexidade das aplicações em IA e computação, surgem então diversas propostas de lógicas "não-clássicas".[11]

[11] Além disso, as lógicas não-clássicas encontram um grande respaldo da comunidade de Ciência da Computação, inclusive através de um grande número de Periódicos (Journals), Handbooks, Livros e Conferências. É também notável que um percentual significativo de pesquisadores que atuam em áreas relacionadas à lógica computacional tenham sido vencedores do ACM Turing Award. A lista de premia-

Entre as lógicas originárias da Ciência da Computação podemos citar as lógicas desenvolvidas para raciocinar sobre programas, a partir dos trabalhos de Hoare, Dijkstra e Floyd [29]. Hoare e Milner também desenvolveram cálculos computacionais para expressar propriedades de concorrência e interação, que demandam novas lógicas computacionais para expressar propriedades de sistemas concorrentes [27, 38]. Dijkstra foi um pioneiro da área de verificação formal e da *derivação de programas*, método no qual um programa e sua prova são desenvolvidos concomitantemente. Uma lógica modal que

Figura 1.10. E.W. Dijkstra, C.A.R. Hoare, Robin Milner: Turing Awards (1972, 1980, 1991)

se tornou extremamente útil em ciência da computação é a lógica temporal. O raciocínio sobre tempo é fundamental em computação. Este requisito essencial do ponto de vista computacional, tem conduzido ao desenvolvimento de um considerável número de sistemas lógicos temporais. Estes desenvolvimentos iniciaram-se com o estudo de lógicas modais temporais, com forte impacto desde o trabalho de Pnueli *On the temporal logic of programs*, a partir de 1977 [40]. As lógicas temporais tem mostrado efetiva aplicabilidade na verificação de programas, verificação de hardware, teoria de bancos de dados, computação distribuída. É relevante observar que a lógica LTL (Linear Temporal Logic) proposicional tem o poder expressivo da lógica de primeira ordem sobre os naturais [21].

1.2.6.1. Lógicas Não-Clássicas: Lógicas Modais

> *Not ignorance, but ignorance of ignorance, is the death of knowledge.*
> Alfred North Whitehead

As lógicas modais formalizam modos de verdade. Em filosofia, são relacionadas à epistemologia, sendo relevantes no estudo do conhecimento. São as lógicas das verdades contingênciais, do "necessário" e "possível". Sua origem moderna remonta a origem a Lewis [35]. Há, também, extensões para lógica modal de primeira ordem. Na sintaxe das lógicas modais, adiciona-se inicialmente operadores unários à lógica clássica proposicional. Os operadores básicos $\Box$, $\Diamond$ adicionados à lógica proposicional/predicados são

dos inclui M. Rabin & D. Scott, C.A.R. Hoare, R. Milner, J. McCarthy, E. Codd, S. Cook, A. Pnueli, E. Clarke, A. Emerson e J. Sifakis, que realizaram trabalhos em áreas diretamente vinculadas à lógica.

$\Box\phi$: que lemos ϕ "é necessariamente verdadeiro" , e $\Diamond\phi$ que é lido "ϕ é possivelmente verdadeiro" (e equivalente a $\neg\Box\neg\phi$). Em lógica modal uma proposição é necessária em um mundo ω se ela for verdadeira em todos os mundos que são possíveis em relação a este mundo ω, enquanto que uma proposição é possível em um mundo se ela for verdadeira em pelo menos um mundo possível alternativo, em relação a ω. Esta relação é formalizada através de uma relação binária de acessibilidade entre mundos possíveis. As lógicas modais foram consideradas apropriadas para estudar provas (lógica de provas), tempo, conhecimento, crenças, obrigações e outras modalidades [4, 15]. Em inteligência artificial e computação, lógicas modais estão entre os formalismos mais adequados para análise e representação de conhecimento e raciocínio sobre sistemas distribuídos e concorrentes [15, 18].

Em aplicações em epistemologia, e em Ciência da Computação, na modelagem de conhecimento em sistemas distribuídos e em inteligência artificial, $\Box\phi$ é lido como "de acordo com o meu conhecimento", "de acordo com as leis da física", "após o término do programa". Notadamente a partir da década de 1950, os filósofos da lógica (e epistemologistas) Hintikka e Kripke propuseram a *semântica de mundos possíveis* [26, 34]. A partir da década de 1970, as lógicas modais passaram a ser muito utilizadas em Ciência da Computação na modelagem e o raciocínio sobre tempo, espaço, crenças, ações e conhecimento, individualmente ou através de combinações destas. Uma obra de referência que tornou as lógicas modais relativamente populares em computação (dizemos relativamente, pois o estudo de lógica computacional reúne uma comunidade científica pequena) foi o agora clássico "Reasoning about Knowledge" de Fagin, Halpern, Moses, Vardi [15]. Ademais, as propriedades de decidibilidade da lógica modal motivam seu uso em computação. Vardi mostra que elas são robustamente decidíveis, sendo adequadas, portanto, para aplicações em computação. [51].

Figura 1.11. Saul Kripke, Amir Pnueli, Moshe Vardi

A lógica modal encontra diversas aplicações. Diversas leituras (ou interpretações) das modalidades são possíveis, por exemplo:

- Modalidades Aléticas:
  $\Box\phi$: ϕ é necessariamente verdadeiro.
  $\Diamond\phi$: ϕ é possivelmente verdadeiro (equivalente a $\neg\Box\neg\phi$).

- Modalidades epistêmicas
  $K_a\phi$: Agente a sabe ϕ.
  $B_a\phi$: Agente a acredita que ϕ é verdade.

- Modalidades temporais lineares:
  $\bigcirc\phi$ (ou $X\phi$): ϕ será verdade no próximo ponto de tempo (estado do mundo).
  $\Box\phi$ (ou $G\phi$): ϕ será verdadeiro sempre.
  $\Diamond\phi$ (ou $F\phi$): ϕ será possivelmente (contingencialmente) verdadeiro no futuro (equivalente a $\neg\Box\neg\phi$: ϕ não será sempre falso).
  $\phi\mathbb{U}\psi$: ϕ é verdadeiro até que ψ seja verdadeiro.

A axiomatização básica da lógica modal acrescenta novos axiomas à lógica proposicional. O Sistema Axiomático da lógica modal conhecido como sistema K, consiste dos axiomas abaixo e duas regras de inferência:

A1 Todas as tautologias proposicionais.

A2 (**K**) $\Box(\alpha \rightarrow \beta) \rightarrow (\Box\alpha \rightarrow \Box\beta)$

R1 De α, $\alpha \rightarrow \beta$ infira β (MP)

R2 De α infira $\Box\alpha$ (Generalização)

Kripke demonstrou, na década de 1960, que o sistema K é um sistema axiomático correto e completo em relação ao modelo semântico. O axioma A2 é hoje chamado de **K** em homenagem a Kripke. Os sistemas lógicos modais ditos normais contém o axioma **K** e a regra de generalização acima. Quanto a exemplos de fórmulas, podemos citar: $\Box\Box A$: é necessário que A seja necessário (ou "é conhecido que A é conhecido"); $\Box\Diamond A$: é necessário que A é possível; $\Box A \rightarrow A$: se A é conhecido, então A é verdadeiro (ou se A é necessariamente verdadeiro, então A é verdadeiro). É importante observar que a cada propriedade da relação de acessibilidade corresponde um axioma da lógica modal. Por exemplo, a seguinte lista apresenta alguns axiomas da lógica modal e a correspondência com as correspondentes propriedades da relação binária R.

T:	$\Box\alpha \rightarrow \alpha$	$\forall x R(x,x)$	(Reflexiva)
4:	$\Box\alpha \rightarrow \Box\Box\alpha$	$\forall x,y,z(R(x,y) \wedge R(y,z)) \rightarrow R(x,z)$	(Transitiva)
B:	$\alpha \rightarrow \Box\Diamond\alpha$	$\forall x,y(R(x,y) \rightarrow R(y,x))$	(Simétrica)
D:	$\Box\alpha \rightarrow \Diamond\alpha$	$\forall x\exists y R(x,y)$	(Serial)
5:	$\Diamond\alpha \rightarrow \Box\Diamond\alpha$	$\forall x,y,z(R(x,y) \wedge R(x,z)) \rightarrow R(y,z)$	(Euclidiana)

Assim, obtemos diferentes sistemas de lógicas modais considerando subconjuntos destes axiomas. Por exemplo:

K: A1, A2, R1, R2.
KD45: K, D, 4, 5 + A1, R1, R2.
KT4: tipicamente chamado de **S4**
KT45: tipicamente chamado de **S5**.
O axioma T corresponde à propriedade em R da reflexividade; 4 corresponde à propriedade da transitividade; B corresponde à propriedade da simetria; D corresponde à propriedade serial e 5 corresponde à propriedade euclidiana.

1.2.6.2. Semântica de Mundos Possíveis

Embora existam diversos modelos semânticos para as lógicas modais, esta síntese se concentra na semântica relacional de Kripke/Hintikka. O modelo dos mundos possíveis de Kripke é utilizado de forma ampla, inclusive como modelo de conhecimento, para lógicas modais epistêmicas. Kripke [33, 34] definiu uma teoria de modelos baseada em "mundos possíveis" e relações de acessibilidade explícita entre mundos. Estes modelos são aplicáveis em uma classe ampla de lógicas. A noção de que mundos são "possíveis" depende do mundo atual, ou de referência. Por exemplo, em lógica modal temporal as noções de passado e futuro dependem do mundo atual. Se considerarmos um conjunto W de mundos possíveis e uma relação binária R em W, podemos ler $R(\omega_1, \omega_2)$ como ω_2 é um mundo possível tendo ω_1 como referência, ou a partir de ω_1. Podemos considerar diversas propriedades das relações de acessibilidade. Um *frame* de Kripke é definido por uma estrutura (W,R) onde W é um conjunto de não vazio e R é uma relação em W.

A noção de relação de acessibilidade também é relevante pois sistemas diferentes de lógicas modais se caracterizam pelas propriedades das relações de acessibilidade. Por exemplo, se definirmos que a relação de acessibilidade é reflexiva, simétrica e transitiva definimos um sistema lógico conhecido como lógica modal $S5$ (ou $KT45$) [15]. De forma simplificada, podemos definir então uma Estrutura de Kripke: $M = (W,R,v)$, onde: W é um conjunto de mundos possíveis; $R \subseteq W^2$ é a relação de acessibilidade entre mundos; e $v(\omega)$ associa a cada mundo $\omega \in W$ um conjunto de variáveis proposicionais. A satisfação de fórmulas $(M,\omega) \models \alpha$ (que pode ser lida α é verdadeira em (M,ω) ou α é satisfeita em (M,ω), pode então ser definida:

- $(M,\omega) \models p$ se, e somente se, $p \in v(\omega)$.
- $(M,\omega \models \neg\alpha \iff (M,\omega) \not\models \alpha$
- $(M,\omega \models \alpha \wedge \beta \iff (M,\omega) \models \alpha$ and $(M,\omega) \models \beta$
- $(M,\omega \models \alpha \vee \beta \iff (M,\omega) \models \alpha$ or $(M,\omega) \models \beta$
- $(M,\omega \models \alpha \rightarrow \beta \iff (M,\omega) \models \beta$ or $(M,\omega) \not\models \alpha$

- $(M,\omega) \models \Box\varphi \iff$ para todo $\omega_i \in W$, se $R(\omega,\omega_i)$ então $(M,\omega_i) \models \varphi$
- $(M,\omega) \models \Diamond\varphi \iff$ existe ω_i tal que $R(\omega,\omega_i)$ e $(M,\omega_i) \models \varphi$

Observe que a definição de satisfação para os conetivos clássicos reflete a definição da lógica proposicional clássica. Em cada mundo possível, a lógica proposicional clássica segue a sua interpretação usual. A lógica modal é uma extensão da lógica clássica pois um modelo da lógica modal é uma estrutura de modelos da lógica clássica interdependentes de acordo com a satisfatibilidade das fórmulas modais. A noção de *validade* também é mais geral. A validade pode ser definida em relação a um dado modelo ou em relação a uma dada classe de modelos. Diferentes funções de avaliação para um mesmo frame (W,R) podem gerar um classe C de modelos associados a (W,R). Dizemos então que, para um modelo $M = (W,R,v)$ e α uma fórmula bem-formada, α é *válida* no modelo M, denotado por $M \models \alpha$, se para todo ω em W, então $M,\omega \models \alpha$ (i.e. α é verdadeiro em todo mundo possível de W). A fórmula α será válida em uma classe C de modelos se para todo modelo M' de C, temos $M' \models \alpha$.
Também é possível considerar múltiplos agentes em uma lógica modal. Neste caso, cada agente tem a sua própria visão de mundo, expresso por uma relação de acessibilidade individual. Cada agente teria uma modalidade $\Box_i$, indexada pelo agente, além de uma relação R_i. Em uma lógica epistêmica, podemos utilizar uma modalidade K para representar conhecimento. Assim $K_i\alpha$ significa que o agente i sabe α; $K_2K_1\neg\alpha$ pode ser lido: o agente 2 sabe que o agente 1 sabe $\neg\alpha$. Por exemplo, o modelo a seguir representando três relações, expressaria um modelo de Kripke para a lógica $KT45^n$, com três agentes (exemplo de [29]). A Figura 1.12 representa o Modelo M a seguir (não mostramos reflexividade, transitividade na figura). $M = \langle W = \{\omega_1,\omega_2,\omega_3,\omega_4,\omega_5,\omega_6\}; R_1 = \{(\omega_1,\omega_2),(\omega_1,\omega_3),(\omega_3,\omega_2),(\omega_4,\omega_5)\}\ R_2 = \{(\omega_1,\omega_3),(\omega_4,\omega_5)\}, R_3 = \{(\omega_3,\omega_2),(\omega_5,\omega_6)\}; v = \{(\omega_1,q),(\omega_2,p),(\omega_2,q),(\omega_3,p),(\omega_4,q),(\omega_6,p)\}\rangle$

1.2.6.3. Sistemas de Provas Para Lógicas Modais

Vários tipos de sistemas de prova para lógicas modais foram propostos, e.g., [4, 17]. Em alguns destes sistemas, as fórmulas são rotuladas pelos mundos (estados) em que elas são verdadeiras, o que pode facilitar no entendimento e desenvolvimento das provas. Nas regras apresentadas a seguir, no estilo de dedução natural, a notação $\varphi : \omega$ significa que φ é verdadeira no mundo possível ω. A relação de acessibilidade R também é explicitada neste sistema de dedução rotulado, o que também auxilia na derivação de em que mundos (estados) as fórmulas podem ser inferidas a partir da informação sobre as relações R entre mundos. As diferenças entre as diversas lógicas, por exemplo lógicas modais, são ditadas pelas propriedades diferentes das relações de acessibilidade. Isso é possível pela união, nas *unidades declarativas* que são constituídas de fórmulas com

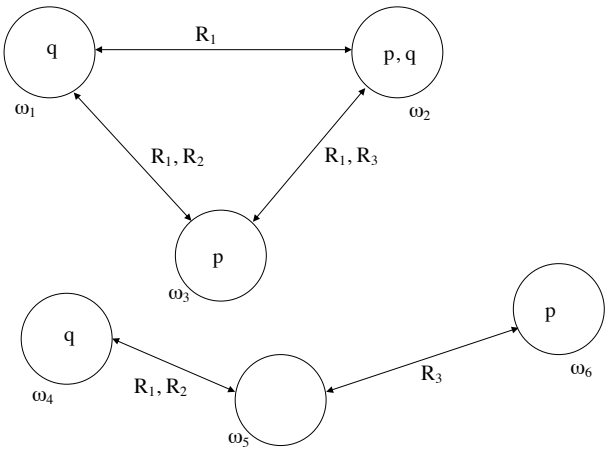


Figura 1.12. Exemplo de Modelo para $KT45^n$ (de [29])

rótulos. Também definimos uma álgebra de rotulação que representa as propriedades específicas de cada sistema lógico. Há também regras de inferência estruturais (não formalizadas aqui por simplicidade) que permitem raciocínio sobre diagramas (relações) em configurações.

Na regra $\Box I$ a hipótese $[\mathcal{R}(\omega, g_\varphi(\omega))]$ é interpretada como: dado um mundo possível arbitrário $g_\varphi(\omega)$, se podemos inferir $\varphi : g_\varphi(\omega)$ então é possível provar que $\Box\varphi : \omega$. A regra $\Diamond E$ pode ser vista, informalmente, como uma *skolemização* do quantificador existencial sobre mundos possíveis, o que é semanticamente implicado pela fórmula $\Diamond\varphi$ na premissa. Assim, o termo $f_\varphi(\omega)$ define um mundo possível particular unicamente associado à fórmula φ inferido e acessível a partir do mundo possível ω (i.e. $\mathcal{R}(\omega, f_\varphi(\omega))$). A regra de $\Diamond I$ representa que se tivermos a relação $\mathcal{R}(\omega_1, \omega_2)$ e se φ é verdade em ω_2 então derivamos $\Diamond\varphi$ no mundo ω_1. Finalmente, a regra $\Box E$ significa que, se $\Box\varphi$ vale no mundo ω_1 e ω_1 está relacionado com (é acessível de) ω_2 então podemos inferir que φ é verdadeiro em ω_2. Assim, as fórmulas são rotuladas pelos mundos em que elas são verdadeiras para facilitar o processo de raciocínio e inferência. As referências explícitas às relações de acessibilidade também auxiliam na derivação de fórmulas nos mundos relacionados por R [4].

Tabela 1.4. Regras para operadores modais

$$\begin{array}{|ll|}\hline
\begin{array}{c}[R(\omega, g_\varphi(\omega))] \\ \vdots \\ \varphi : g_\varphi(\omega) \\ \hline \Box\varphi : \omega\end{array} \Box I & \dfrac{\Box\varphi : \omega_1, R(\omega_1, \omega_2)}{\varphi : \omega_2} \Box E \\[2ex]
\dfrac{\Diamond\varphi : \omega}{\varphi : f_\varphi(\omega), R(\omega, f_\varphi(\omega))} \Diamond E & \dfrac{\varphi : \omega_2, R(\omega_1, \omega_2)}{\Diamond\varphi : \omega_1} \Diamond I \\[2ex]\hline
\end{array}$$

Exemplo: Vamos provar $\Box p : w_0 \vdash \neg\Diamond\neg p : w_0$ usando regras de dedução rotuladas.

1. $C_1\langle \Box p : w_0 \vdash \neg\Diamond\neg p : w_0\rangle$ *premissa*
2. $C_2\langle \Box p : w_0, [\Diamond\neg p : w_0]\rangle$ *hip.*
3. $C_3\langle \Box p : w_0, R(w_0, f_{\neg p}(w_0)), \neg p : f_{\neg p}(w_0)\rangle$ $\Diamond E(C_2)$
4. $C_4\langle p : f_{\neg p}(w_0), \neg p : f_{\neg p}(w_0)\rangle$ $\Box E(C_2)$
5. $C_5\langle \bot : f_{\neg p}(w_0)\rangle$ $\wedge I(C_3)$
6. $C_6\langle \neg\Diamond\neg p : w_0\rangle$ $\neg I(C_2, C_5)$

Explicação: Nesta prova, temos como premissa $\Box p : w_0$, isto é no mundo w_0, $\Box p$ é verdadeiro. Para provar a conclusão $\neg\Diamond\neg p : w_0$, assumimos, como hipótese que $\Diamond\neg p : w_0$ para obtermos uma contradição $\bot$ usando a regra de $\neg I$ em w_0. Note que, em w_0, temos agora $\Box p : w_0$, $\neg\Diamond\neg p : w_0$ e $\Diamond\neg p : w_0$ (hipótese). Para obtermos uma contradição, temos de usar as regras de eliminação $\Box E$, quanto $\Diamond E$. Para usar $\Diamond E$, temos mostrar que p é verdade em um novo mundo possível unicamente associado com p, isto é em $f_{\neg p}(w_0)$. Com a contradição entre $p : f_{\neg p}(w_0)$ e $\neg p : f_{\neg p}(w_0)$, concluímos a prova. Note que neste sistema rotulado de provas, fazemos uso das regras para os conetivos clássicos, conforme a Tabela 1.5.

1.2.6.4. Lógica em Inteligência Artificial: Integrando Raciocínio e Aprendizado

Em 2003, Valiant propôs um grande desafio para a Ciência da Computação [48]. O desafio consiste no desenvolvimento de modelos efetivos de computação cognitiva inte-

Tabela 1.5. Regras para conetivos clássicos

$$\frac{\alpha \vee \beta : \omega \qquad \begin{array}{c}[\alpha : \omega] \\ \vdots \\ \gamma : \omega\end{array} \qquad \begin{array}{c}[\beta : \omega] \\ \vdots \\ \gamma : \omega\end{array}}{\gamma : \omega} \vee E \qquad \frac{\alpha : \omega}{\alpha \vee \beta : \omega} \vee I$$

$$\frac{\alpha \wedge \beta : \omega}{\alpha : \omega, \beta : \omega} \wedge E \qquad \frac{\alpha : \omega, \beta : \omega}{\alpha \wedge \beta : \omega} \wedge I$$

$$\frac{\alpha \supset \beta : \omega, \alpha : \omega}{\beta : \omega} \supset E \qquad \frac{\begin{array}{c}[\alpha : \omega] \\ \vdots \\ \beta : \omega\end{array}}{\alpha \supset \beta : \omega} \supset I$$

$$\frac{\neg\neg\alpha : \omega}{\alpha : \omega} \neg E \qquad \frac{\begin{array}{c}[\alpha : \omega] \\ \vdots \\ \bot : \omega'\end{array}}{\neg\alpha : \omega} \neg I$$

grando raciocínio e aprendizado.

> *"The aim here is to identify a way of looking at and manipulating commonsense knowledge that is consistent with and can support what we consider to be the two most fundamental aspects of intelligent cognitive behaviour: the ability to* learn from experience*, and the ability* to reason from what has been learned*. We are therefore seeking a semantics of knowledge that can computationally support the basic phenomena of intelligent behaviour."*[48]

Recentemente, esta área tem sido objeto de pesquisas de um crescente número de pesquisadores, tendo em vista resultados promissores em inteligência artificial [8]. Para responder efetivamente a este desafio, além de conhecimentos sólidos de ciência da computação, é necessário entendimento da ciência cognitiva, área de estudo interdisciplinar da mente e cérebro. Esta área envolve psicologia, filosofia, IA, neurociência, linguística, antropologia. Em cognição, estudamos os processos do raciocínio humano, enquanto que em computação estes processos podem ou não ter inspiração na natureza. Várias áreas nos auxiliam a construir artefatos/algoritmos/sistemas computacio-

nais mais expressivos, que podem contribuir neste desafio. Outro desafio, relacionado ao proposto por Valiant é o clássico questionamento levantado por Turing:

> *I propose to consider the question, "Can machines think?" This should begin with definitions of the meaning of the terms "machine" and "think". The definitions might be framed so as to reflect so far as possible the normal use of the words,... but this attitude is dangerous, If the meaning of the words "machine" and "think" are to be found by examining how they are commonly used it is difficult to escape the conclusion that the meaning and the answer to the question, "Can machines think?" is to be sought in a statistical survey such as a Gallup poll. But this is absurd....* [47].

Para realizar computação cognitiva, diversas habilidades devem ser consideradas. Uma alternativa inicial seria expressar algumas destas habilidades em sistemas artificiais. Estes modelos cognitivos mais completos requerem, portanto, a integração entre aprendizado, formação de regras (lógicas) e raciocínio (computação) [30]. Para construir modelos computacionais cognitivos que exigem a integração de raciocínio expressivo (através de lógicas) e aprendizado robusto é necessário definir uma semântica adequada para computação cognitiva, i.e. que represente habilidades cognitivas.

Como aprendizado de máquina é usualmente estudado através de abordagens estatísticas, experimentais, enquanto que o raciocínio em IA é estudado através de abordagens baseadas em lógicas, uma alternativa seria desenvolver modelos integrados, híbridos, por exemplo, modelos neuro-simbólicos. A computação neuro-simbólica explora os benefícios de cada paradigma: aprendizado (neural) e raciocínio simbólico (lógico) [12]. O exemplo a seguir mostra como modelos neuro-simbólicos podem ser realizados. Nesta abordagem, o conhecimento é representado por uma linguagem lógica simbólica, enquanto que a computação é realizada por algoritmos conexionistas, através de redes neurais. Para ilustrar a abordagem, utilizarmos a Lógica Modal Conexionista (Connectionist Modal Logic - CML) [11]. O *insight* em CML é considerar as redes neurais como redes de mundos possíveis, como na Figura 1.13.

No exemplo, considere que as 3 redes estão relacionadas, podendo se comunicar. Considere as redes neurais N_1, N_2, N_3 como mundos possíveis. Observe que em N_1 é possível, por exemplo, usar o raciocínio sobre a fórmula $\Box q$ para deduzirmos q em N_2. Por sua vez, em N_2 temos s; assim, podemos mostrar $\Diamond s$ em N_1. Outros raciocínios são possíveis. Regras de inferência podem ser aprendidas pela rede neural, integrando, assim, raciocínio lógico e aprendizado de máquina. Observe que estas redes neuro-simbólicas tem propriedades interessantes, inclusive em relação à computação de pontos fixos de programas lógicos. Por exemplo, para qualquer programa P, existe uma rede neural N tal que N computa o operador de ponto fixo de programas modais, de

programas lógicos que representam raciocínio intuicionista e de linguagens restritas de programas temporais [9, 10, 11].

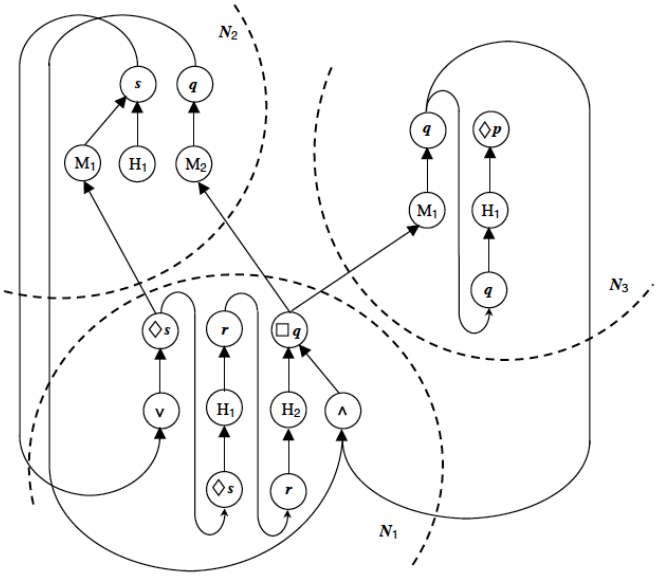


Figura 1.13. Conjunto de redes que representa o programa $P = \{r \rightarrow \Box q : \omega_1; \Diamond s \rightarrow r : \omega_1; s : \omega_2; q \rightarrow \Diamond p : \omega_3, R(\omega_1, \omega_2), R(\omega_2, \omega_3)\}$

Outras áreas em que a integração de lógica e aprendizado teve aplicações relevantes incluem raciocínio e aprendizado temporal, inclusive no desenvolvimento de simuladores de treinamento [14], devido à capacidade de aprender novas regras observando-se experts e novos usuários em treinamento. Nos problemas tipicamente utilizados como casos de estudo em sistemas distribuídos (*muddy children puzzle, wise men puzzle, dining philosophers* [15]), os resultados de aplicação desta metodologia também foram promissores, pois o sistema é capaz de aprender a resolver estes estudos de caso, encontrando a solução dos mesmos (veja detalhes em [12]). Finalmente, em engenharia de software, a abordagem permite adaptação e evolução de modelos e especificações temporais, servindo para verificar sistemas quando descrições de modelos são incompletas [2].

Também é relevante mencionar que a integração entre **lógica e probabilidade** tem sido tema de pesquisas por um longo período. Historicamente, De Morgan e Boole já haviam realizado estudos em que associavam raciocínio probabilístico e lógica clássica. Esta área de pesquisa tem como um de seus objetivos integrar a representação simbólica da lógica - que representa verdades absolutas, com a representação de incertezas - através de probabilidade [24, 28]. Embora este pareça ser um objetivo

contraditório, a combinação de raciocínio qualitativo (simbólico) e quantitativo (probabilístico) pode levar à construção de sistemas, particularmente em inteligência artificial, mais expressivos [41].

Figura 1.14. Michael O. Rabin, Leslie G. Valiant

1.2.7. Conclusões e Perspectivas

O estudo de sistemas lógicos têm sido fundamental desde as origens da Ciência da Computação, a ponto da lógica ser citada como o cálculo da computação [25]. Recentemente, temos presenciado resultados muito promissores de pesquisas em Ciência da Computação e Inteligência Artificial [43, 50], que demonstram que os sistemas computacionais atuais já são capazes de integrar habilidades cognitivas complexas, como o aprendizado a partir de experiências e o raciocínio sobre o conhecimento adquirido do ambiente. A construção de sistemas computacionais com complexidade crescente, exige, portanto, conhecimento sofisticado de técnicas, modelos e fundamentos da Ciência da Computação.

Os grandes desafios científicos sempre exigiram e, cada vez mais, irão demandar habilidade e capacidade de formalizar e construir raciocínios não triviais em sistemas de computação. Este capítulo ofereceu uma introdução mínima e incompleta à lógica aplicada. Esperamos que o leitor, a partir da bibliografia, possa explorar novos caminhos de forma independente. Para concluir de forma promissora, citamos Michael Rabin, vencedor do ACM Turing Award em 1976:

> *Our field is still in its embryonic stage. It's great that we haven't been around for 2000 years. We are at a stage where very, very important results occur in front of our eyes.* M.O. Rabin em [44]

Referências

[1] Dirk Baltzly. Stoicism. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2014 edition, 2014.

[2] Rafael V. Borges, Artur d'Avila Garcez, and Luís C. Lamb. Learning and representing temporal knowledge in recurrent networks. *IEEE Transactions on Neural Networks*, 22(12):2409–2421, 2011.

[3] K. Broda, S. Eisenbach, H. Khoshnevisan, and S. Vickers. *Reasoned Programming*. Prentice Hall, 1994.

[4] K. Broda, D.M. Gabbay, L.C. Lamb, and A. Russo. *Compiled Labelled Deductive Systems: A Uniform Presentation of Non-classical Logics*. Studies in Logic and Computation. Research Studies Press/Institute of Physics Publishing, Baldock, UK, Philadelphia, PA, 2004.

[5] Alonzo Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 58(2):345–363, April 1936.

[6] Stephen A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158, 1971.

[7] Flavio Corrêa da Silva, Marcelo Finger, and Ana C. V. de Melo. *Lógica para Computação*. Thomson, São Paulo, 2006.

[8] Artur d'Avila Garcez, Marco Gori, Pascal Hitzler, and Luís C. Lamb. Neural-Symbolic Learning and Reasoning (Dagstuhl Seminar 14381). *Dagstuhl Reports*, 4(9):50–84, 2015.

[9] A.S. d'Avila Garcez and L.C. Lamb. A connectionist computational model for epistemic and temporal reasoning. *Neural Computation*, 18(7):1711–1738, 2006.

[10] A.S. d'Avila Garcez, L.C. Lamb, and D.M. Gabbay. Connectionist computations of intuitionistic reasoning. *Theoretical Computer Science*, 358(1):34–55, 2006.

[11] A.S. d'Avila Garcez, L.C. Lamb, and D.M. Gabbay. Connectionist modal logic: Representing modalities in neural networks. *Theoretical Computer Science*, 371(1-2):34–53, 2007.

[12] A.S. d'Avila Garcez, L.C Lamb, and D.M. Gabbay. *Neural-Symbolic Cognitive Reasoning*. Cognitive Technologies. Springer, 2009.

[13] Leonardo M. de Moura and Nikolaj Bjørner. Satisfiability modulo theories: introduction and applications. Commun. ACM, 54(9):69-77, 2011.

[14] Leo de Penning, Artur S. d'Avila Garcez, Luís C. Lamb, and John-Jules Ch. Meyer. A neural-symbolic cognitive agent for online learning and reasoning. In Toby Walsh, editor, *IJCAI-11*, pages 1653–1658. IJCAI/AAAI, 2011.

[15] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. MIT Press, 1995.

[16] Frederic B. Fitch. *Symbolic Logic*. The Ronald Press Company, New York, 1952.

[17] M. Fitting. *Proof methods for modal and intuitionistic logics*. D. Reidel Publishing Company, Dordrecht, 1983.

[18] D. Gabbay, A. Kurucz, F.Wolter, and M. Zakharyaschev. *Many-dimensional modal logics: theory and applications*, volume 148 of *Studies in Logic and the Foundations of Mathematics*. Elsevier Science, 2003.

[19] D. M. Gabbay. *Elementary Logics: a Procedural Perspective*. Prentice Hall, London, 1998.

[20] Dov M. Gabbay and F. Guenthner, editors. *Handbook of Philosophical Logic*, volume I-XVIII. Springer, 2008-2015.

[21] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. On the temporal basis of fairness. In *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980*, pages 163–173, 1980.

[22] Dov M. Gabbay and John Woods, editors. *Handbook of The History of Logic*, volume I-XI. Elsevier, 2008-2015.

[23] G. Gentzen. Unstersuchungen über das logische Schliessen. *Mathematische Zeitschrift*, 39:176–210, 1934.

[24] J.Y. Halpern. *Reasoning about Uncertainty*, MIT Press, 2003.

[25] J.Y. Halpern, R. Harper, N. Immerman, P.G. Kolaitis, M.Y. Vardi, and V. Vianu. On the unusual effectiveness of logic in computer science. *Bulletin of Symbolic Logic*, 7(2):213–236, 2001.

[26] J. Hintikka. *Knowledge and Belief.* Cornell University Press, 1962.

[27] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[28] C. Howson. Probability and logic. *J. Applied Logic* 1(3-4), 151-165, 2003.

[29] M. Huth and M. Ryan. *Logic in Computer Science: modelling and reasoning about systems*. Cambridge University Press, 2000.

[30] R. Khardon and D. Roth. Learning to reason. *Journal of the ACM*, 44(5):697–725, 1997.

[31] Robert A. Kowalski. *Computational Logic and Human Thinking: How to be Artificially Intelligent*. Cambridge University Press, 2011.

[32] Robert A. Kowalski. *Logic for Problem Solving, Revisited*. Herstellung und Verlag: Books on Demand, 2014.

[33] S. Kripke. A completeness theorem in modal logic. *Journal of Symbolic Logic*, 24:1–4, 1959.

[34] S. Kripke. Semantic analysis of modal logics I, normal propositional calculi. *Zeitschrift für mathematische Logik und Glundlagen der Mathematik*, 9:67–96, 1963.

[35] C. Lewis. *A Survey of Symbolic Logic*. University of California Press, Berkeley, 1918.

[36] Z. Manna and R. Waldinger. *The logical basis for computer programming. Volume 1: deductive reasoning*. Addison-Wesley, Boston, 1985.

[37] E. Mendelson. *Introduction to mathematical logic*. Van Nostrand Reinhold, New York, 1964.

[38] Robin Milner. *Communication and concurrency*. PHI Series in computer science. Prentice Hall, 1989.

[39] Joan Moschovakis. Intuitionistic logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Spring 2015 edition, 2015.

[40] A. Pnueli. The temporal logic of programs. In *Proceedings of 18th IEEE Annual Symposium on Foundations of Computer Science*, pages 46–57, 1977.

[41] S.J. Russell. Unifying Logic and Probability: A New Dawn for AI? In *Information Processing and Management of Uncertainty in Knowledge-Based Systems - 15th International Conference, IPMU 2014*, pages 10–14, 2014.

[42] Ralph Schoenman, editor. *Bertrand Russell: Philosopher of the Century*. Allen and Unwin, London, 1967.

[43] Bernhard Schoelkopf. Artificial intelligence: Learning to see and act. *Nature*, 518:486–487, 2015.

[44] Denis Shasha and Cathy Lazere. *Out of Their Minds: The Lives and Discoveries of 15 Great Computer Scientists*. Copernicus, 1995.

[45] R. M. Smullyan. *First-Order Logic*. Dover Publications, New York, revised edition, 1995.

[46] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, 42(2), 1936.

[47] Alan M. Turing. Computing machinery and intelligence. *Mind*, 59:433–460, 1950.

[48] L. G. Valiant. Three problems in computer science. *Journal of ACM*, 50(1):96–99, 2003.

[49] Dirk Van Dalen. Intuitionistic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume 166 of *Synthese Library*, pages 225–339. Springer Netherlands, 1986.

[50] Moshe Y. Vardi. Is information technology destroying the middle class? *Commun. ACM*, 58(2):5, 2015.

[51] M.Y. Vardi. Why is modal logic so robustly decidable? In N. Immerman and P. Kolaitis, editors, *Descriptive Complexity and Finite Models*, volume 31 of *Discrete Mathematics and Theoretical Computer Science*, pages 149–184. DIMACS, 1997.

[52] Alfred North Whitehead and Bertrand Russell. *Principia Mathematica*, volume I,II, III. Cambridge University Press, Cambridge, 1910, 1912, 1913.

[53] Ludwig Wittgenstein. *Tractatus Logico-Philosophicus*. Kegan Paul, Trench, Tubner & Co, London, 1922.