

elemento que está sendo copiado. Tal como uma variável temporária, ao término do processamento ela deve ser descartada pelo gerenciador de memória do compilador. Assim, se antes um símbolo 1 foi sobrescrito por um X , agora este mesmo X deve ser removido, restaurando o símbolo original. Desta forma, ao término da computação, a Máquina de Turing irá apresentar somente os símbolos pertencentes ao alfabeto Σ .

7.2 Máquinas de Turing Combinadas

A combinação de Máquinas de Turing permite que sejam criadas máquinas mais complexas a partir de máquinas mais simples. Muitas vezes é difícil conceber uma máquina que execute uma tarefa qualquer simplesmente por conta da inabilidade de quem a constrói em lidar com tantos estados e transições. Nestes casos, a estratégia de dividir para conquistar torna-se uma opção valiosa. Divide-se o problema inicial em partes menores mais simples e se realiza a construção de máquinas para cada uma destas partes individualmente, algo que a Engenharia de Software chama de análise. Logo após, realiza-se a síntese, isto é, a combinação destas partes para que juntas provejam a solução do problema inicial.

Sob esta ótica, torna-se desejável entender como combinar estas máquinas. Considere dois grupos de Máquinas de Turing com alfabeto $\Sigma = \{a, b, \dots, \sqcup\}$. O primeiro grupo é composto por máquinas que escrevem na fita um determinado símbolo, sobre outro já existente, e depois pára no estado h . Por exemplo, a função de transição da máquina M_a que escreve o símbolo a nestas condições (e similarmente M_b, \dots, M_{\sqcup}) é definida como:

q	σ	$\delta(q, \sigma)$
q_0	a	(h, a)
q_0	b	(h, a)
q_0	\vdots	(h, a)
q_0	\sqcup	(h, a)

O segundo grupo é composto por máquinas que movem a cabeça leitora uma unidade à esquerda (M_L) ou uma unidade à direita (M_R) e em seguida pára no estado h . Por

exemplo, a função de transição da máquina M_R que desloca a cabeça leitora para a esquerda é definida como:

q	σ	$\delta(q, \sigma)$
q_0	a	(h, \rightarrow)
q_0	b	(h, \rightarrow)
q_0	\vdots	(h, \rightarrow)
q_0	\sqcup	(h, \rightarrow)

Agora, imagine duas Máquinas de Turing M_1 e M_2 que executam tarefas quaisquer. A máquina resultante da sequenciação de M_1 seguida de M_2 é uma máquina que primeiro se comporta como M_1 e depois como M_2 . Esta composição, denotada por $> M_1 \rightarrow M_2$ é realizada da seguinte forma:

1. Altere o nome de todas as etiquetas dos estados de M_2 de tal forma que nenhuma etiqueta coincida com as etiquetas dos estados de M_1 .
2. Mude todos os estados de parada de M_1 para o estado inicial de M_2 , já com a nova etiqueta.
3. Anexe todas as transições de M_2 ao final da tabela de função de transição de M_1 .

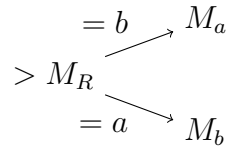
Por exemplo, $> M_a \rightarrow M_R$ é definida como:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_1, a)
q_0	b	(q_1, a)
q_0	\vdots	(q_1, a)
q_0	\sqcup	(q_1, a)
q_1	a	(h, \rightarrow)
q_1	b	(h, \rightarrow)
q_1	\vdots	(h, \rightarrow)
q_1	\sqcup	(h, \rightarrow)

Outra combinação ilustrativa é $> M_a \rightarrow M_R \rightarrow M_{\sqcup} \rightarrow M_L \rightarrow M_b$ (deixamos para o leitor a construção da função de transição).

Observe ainda que a conexão entre duas Máquinas de Turing não precisa ser incondicional, como até o momento. Essa conexão pode depender do símbolo sob a cabeça de leitura no ponto onde a primeira máquina pára. Esta dependência é representada por um teste na conexão entre as duas máquinas, isto é, $M_1 \xrightarrow{\text{teste}} M_2$. Por exemplo, $> M_R \xrightarrow{=b} M_{\sqcup}$ primeiro move o cabeçote da máquina para a direita. Em seguida, se existe um símbolo b sob a cabeça leitora, então ela sobrescreve um \sqcup e pára. Se nesta posição existisse qualquer outro símbolo diferente de b , a máquina pararia imediatamente. Não existe uma regra rígida sobre como indicar a semântica deste teste, por isso também são válidas estas outras conexões: $M_R \xrightarrow{\in\{a,b\}} M_{\sqcup}$, $M_R \xrightarrow{\neq a} M_{\sqcup}$, $M_R \xrightarrow{\notin\{a,b\}} M_{\sqcup}$.

A partir desse momento, as conexões podem ser realizadas segundo condições de contorno, por isso uma demanda natural que se segue é a possibilidade de realizar múltiplas conexões (múltiplas setas), desde que os testes sejam mutuamente exclusivos. Como por exemplo:



Inicialmente, esta máquina move a cabeça leitora para a direita. Depois, se existe um a sob a cabeça leitora, então ela escreve um b e pára. Se existe um b sob a cabeça leitora, então um a é escrito e a máquina pára. Se qualquer outro símbolo surgir, a máquina pára imediatamente após a cabeça leitora ter se deslocado para a direita.

A tabela desta função de transição é escrita da seguinte forma:

1. Renomear todos os estados de M_a e M_b para evitar duplicatas, não só entre si mas também com M_R .
2. Alterar os estados de parada da máquina M_R . Todos os estados de parada associados ao símbolo a são renomeados para coincidirem com o estado inicial de M_b . Já os estados de parada associados ao símbolo b são alterados para o

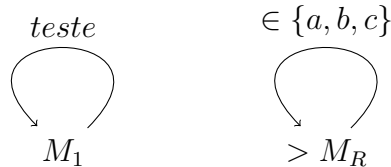
estado de partida de M_a .

3. Anexe todas as tabelas em uma única tabela.

Assim, a tabela fica definida como:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_2, \rightarrow)
q_0	b	(q_1, \rightarrow)
q_0	\vdots	(h, \rightarrow)
q_0	\sqcup	(h, \rightarrow)
q_1	a	(h, a)
q_1	b	(h, a)
q_1	\vdots	(h, a)
q_1	\sqcup	(h, a)
q_2	a	(h, b)
q_2	b	(h, b)
q_2	\vdots	(h, b)
q_2	\sqcup	(h, b)

Por fim, uma vez de posse dos desvios condicionais, resta a hipótese de que a conexão da primeira máquina seja feita com a entrada dela mesma, isto é, que seja feita uma retroalimentação, gerando um loop. Assim, a máquina M_1 é executada, até o ponto quando normalmente ela pararia mas a avaliação de um teste, caso se mostre verdadeiro, faz com que ela retorne para o estado inicial. No exemplo a seguir, a máquina move-se repetidamente para a direita enquanto existe um a , b ou c sob a cabeça leitora. Quando o símbolo sob esta cabeça leitora é diferente de a , b ou c , a máquina pára.



Nos casos com loop, é importante a inserção de um teste condicional para evitar que a máquina entre em loop eterno. A tabela da função de transição para esta máquina é construída da seguinte forma:

1. Altere os estados de parada da máquina M_1 . Qualquer estado de parada h associado à leitura de um símbolo envolvido no teste deve ser alterado para o estado inicial.

No caso da máquina M_R tem-se:

q	σ	$\delta(q, \sigma)$
q_0	a	(q_0, \rightarrow)
q_0	b	(q_0, \rightarrow)
q_0	\vdots	(h, \rightarrow)
q_0	\sqcup	(h, \rightarrow)

7.3 Variações da Máquina de Turing

É comum se encontrar diversas aplicações que requerem cálculos da ordem de bilhões de operações por segundo, tais como o processamento de sinais e imagens, cálculos para renderização, simulações numéricas, criptografia, entre outras. Uma solução natural para estas demandas, oferecida pelos fabricantes de hardware, é o desenvolvimento de processadores específicos dedicados para um determinado tipo de atividade, tais como os Processadores de Sinais Digitais (*Digital Signal Processor* - DSPs) e *Field-Programmable Gate Arrays* (FPGAs). Entretanto, esta abordagem possui um custo e um esforço de desenvolvimento elevado [89, 90], além do que o produto final carece de flexibilidade, pois não se adapta com facilidade à evolução dos algoritmos e dos aplicativos. Os projetistas de computadores sempre buscaram construir máquinas mais rápidas. Entretanto, existem vários limites físicos que impedem o aumento indiscriminado desta velocidade de hardware, a fim de atingir computadores mais velozes, tais como temperatura de operação, miniaturização do hardware, entre outros. Uma abordagem alternativa, que tem encontrado muitos adeptos, é a construção de máquinas paralelas que usam a simultaneidade de execução de tarefas, como meio para acelerar um processamento. Segundo Hwang [91], o processamento paralelo pode ser definido como uma forma eficiente do processamento de dados, com ênfase na exploração de eventos concorrentes do processo computacional. Neste sentido, uma das soluções é o emprego de *Stream Processors*

programáveis [92], que exploram a baixa localidade de referência, e a pequena necessidade de concorrência de alguns algoritmos. Estes dispositivos passaram a exigir linguagens de programação que permitissem seu uso com mais facilidade. Estas linguagens, por sua vez, evoluíram concomitantemente com o progresso tecnológico das *Graphical Processing Units* (GPUs), criando uma disputa acirrada pelo mercado de computação de alto desempenho, através das *General Purpose Computing on Graphics Processing Units* (GPGPUs), uma evolução dos *Stream Processors* programáveis. A proposta das GPGPUs é utilizar as GPUs não somente para aplicações gráficas, mas também para realizar computação em um sentido mais amplo, tarefa esta que normalmente está associada as *Central Processing Units* (CPUs). Trata-se de um rompimento com a estrutura clássica de processadores programáveis, conhecida como arquitetura de von Neumann [93]. Esse paradigma é uma quebra, que no final da década de 1970, já havia sido apontada por Backus como necessária [94]. As arquiteturas modernas de GPUs apresentam características importantes como [95]: (1) uma largura de banda elevada; (2) uma capacidade de processamento significativa; (3) são programáveis; (4) e possuem uma relação custo/benefício vantajosa. O paradigma da construção de um algoritmo paralelo, resolutor de um determinado problema, depende significativamente da maneira pela qual a carga de processamento é dividida entre os processadores disponíveis. Este particionamento do problema está intimamente ligado, não só a sua natureza, mas também à arquitetura paralela disponível na máquina. Esse cenário se contrapõe a uma primeira impressão de que tudo pode ser paralelizado, e de que o paralelismo pode aumentar indefinidamente a velocidade de processamento, desde que haja recursos computacionais disponíveis. A Lei de Amdahl [96] e a Lei de Gustafson-Barsis [96] fazem a relação de como os trechos não paralelizáveis de um programa comprometem o aumento de velocidade do paralelismo, a Conjectura de Minsky [97] estabelece como ocorre a queda de desempenho de processadores paralelos devido ao conflito de acesso aos dados e a Lei de Grosch [98] postula sobre o custo financeiro dos MFLOPS (*Mega Floating-point Operations Per Second*). Estes princípios, por exemplo, são limites clássicos das condições de contorno associadas ao processamento paralelo, como será apresentado nos capítulos a seguir. Os programas e os dispositivos de processamento podem ser classificados de acordo com a quantidade de fluxo de instruções,

combinada com a quantidade de dados usada por tais instruções. Essa rotulação é conhecida como taxonomia de Flynn, e classifica os dispositivos e programas segundo quatro grandes grupos, a saber [93]:

- a) SISD (*Single Instruction, Single Data*): uma Única Instrução e um Único Dado. A grande parte dos computadores da atualidade segue este modelo de arquitetura, que está diretamente associada à arquitetura de von Neumann;
- b) SIMD (*Single Instruction, Multiple Data*): uma Única Instrução e Múltiplos Dados. São os computadores vetoriais, que aplicam as mesmas operações matemáticas em um grande volume de dados;
- c) MISD (*Multiple Instruction, Single Data*): Múltiplas Instruções e Dados Únicos. Não é uma representação muito utilizada, sendo conhecida como Vetores Sistólicos², apresentando com certa frequência, uma predisposição a gerar ociosidade dos processadores;
- d) MIMD (*Multiple Instruction, Multiple Data*): Múltiplas Instruções e Múltiplos Dados. Através de diversas unidades de processamento, manipula-se conjuntos de dados distintos, como por exemplo as GPGPUs.

Apesar do determinismo desta taxonomia, a prática mostra que existe uma dualidade combinatória entre as classificações. Os *Stream Processors* programáveis podem ser considerados como uma combinação de SIMD/MIMD, de acordo com a abordagem adotada na solução do problema a ser paralelizado. Deste modo, as GPGPUs podem ser consideradas, ora como pertencentes à taxonomia de SIMD, ora como MIMD, conforme seu emprego.

No início da seção anterior, foi mencionado que uma Máquina de Turing é um mecanismo que se assemelha em muito aos computadores modernos, e ainda que possui o mesmo poder computacional de qualquer outro computador de propósito geral. Isto não quer dizer que a máquina de Turing seja mais rápida, ou mesmo igualmente rápida, quando comparada com os computadores atuais. Afinal, uma

²Caracterizam-se pelo emprego de processadores simples (células), responsáveis por operações elementares, que são aplicadas a um fluxo de dados, “bombeado” através do sistema.

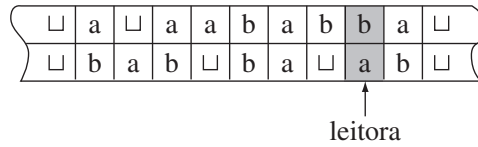


Figura 7.3: Máquina de fita *multitrack*: exemplo com duas trilhas.

máquina de fita seqüencial, que lê e escreve um único dado em uma fita, tem que ser mais lenta que as atuais tecnologias. O que é proposto ao afirmar sobre o mesmo poder computacional é que a Máquina de Turing pode computar exatamente as mesmas coisas que o computador mais rápido da atualidade, seja qual for a tecnologia que ele emprega, isto é, vários núcleos de processamento, computação quântica, entre outros. A Máquina de Turing pode efetuar uma computação muito mais lenta, mas certamente chegará aos mesmos resultados de qualquer outro computador, e por este motivo é considerada como a máquina universal de computação. Esta é a beleza da criação de Turing, que concebeu um modelo trivial para algo que nunca perdeu sua atualidade. Sob esta ótica, esta seção tem por objetivo mostrar que diversas variações de máquinas possuem uma Máquina de Turing equivalente, que através de uma única cabeça leitora realiza a leitura e escrita de um único símbolo.

Uma primeira variação é a máquina de fitas *multitrack*, uma representação análoga à categoria SIMD. O leitor poder também imaginar um processador que faz a leitura e escrita simultânea em um barramento com 32, 64, ou até mesmo mais bits. Uma ilustração esquemática deste tipo de situação é apresentada na Figura 7.3, cuja principal característica é a quantidade de símbolos que são lidos e escritos ao mesmo tempo. No exemplo da figura em questão a máquina de fita *multitrack* possui, por simplicidade, apenas duas trilhas e conseqüentemente lê e escreve dois símbolos simultaneamente. Esta máquina M poderia estar sujeita a seguinte função de transição δ :

M com fita de duas trilhas

q	σ	$\delta(q, \sigma)$
q_0	$\begin{smallmatrix} a \\ a \end{smallmatrix}$	$(q_0, \begin{smallmatrix} a \\ b \end{smallmatrix})$
q_0	$\begin{smallmatrix} a \\ b \end{smallmatrix}$	$(q_1, \begin{smallmatrix} a \\ b \end{smallmatrix})$
q_0	$\begin{smallmatrix} b \\ a \end{smallmatrix}$	$(q_1, \begin{smallmatrix} b \\ b \end{smallmatrix})$
q_0	$\begin{smallmatrix} b \\ b \end{smallmatrix}$	$(q_1, \begin{smallmatrix} a \\ a \end{smallmatrix})$
q_1	$\begin{smallmatrix} a \\ a \end{smallmatrix}$	$(q_1, \begin{smallmatrix} b \\ a \end{smallmatrix})$
q_1	$\begin{smallmatrix} a \\ b \end{smallmatrix}$	$(h, \begin{smallmatrix} a \\ b \end{smallmatrix})$
q_1	$\begin{smallmatrix} b \\ a \end{smallmatrix}$	$(q_0, \begin{smallmatrix} b \\ a \end{smallmatrix})$
q_1	$\begin{smallmatrix} b \\ b \end{smallmatrix}$	$(q_1, \begin{smallmatrix} a \\ a \end{smallmatrix})$

Observe que a representação do tipo $\sigma = \begin{smallmatrix} b \\ a \end{smallmatrix}$ indica que é lido o símbolo b da primeira trilha, e o símbolo a da segunda trilha (analogamente no caso da escrita). Note que esta função de transição δ utiliza um alfabeto $\Sigma = \{a, b\}$. Uma vez que a composição da fita com a cabeça leitora permite a leitura e escrita de símbolos simultaneamente, e ainda, que existem apenas dois símbolos no alfabeto, temos então que todas as possibilidades de combinação de símbolos nesta fita de duas trilhas é dada por $\begin{smallmatrix} a \\ a \end{smallmatrix}, \begin{smallmatrix} a \\ b \end{smallmatrix}, \begin{smallmatrix} b \\ a \end{smallmatrix}, \begin{smallmatrix} b \\ b \end{smallmatrix}$. Este resultado é importante porque indica que o conjunto de duplas de símbolos que podem ser lidos e escritos pela máquina M é fixo e limitado. Assim, pode-se fazer uso de um artifício matemático onde cada dupla fixa de símbolos é substituída por uma nova representação, e assim se obtém:

$$\begin{smallmatrix} a \\ a \end{smallmatrix} = 0 \qquad \begin{smallmatrix} a \\ b \end{smallmatrix} = 1 \qquad \begin{smallmatrix} b \\ a \end{smallmatrix} = 2 \qquad \begin{smallmatrix} b \\ b \end{smallmatrix} = 3$$

Este artifício permite realizar uma substituição direta dos símbolos existentes na função de transição δ da máquina M com fita de duas trilhas, resultando na função de transição δ' que representará a simulação da mesma máquina M em uma fita simples. Esta função δ' pode ser facilmente aplicada em uma Máquina de Turing com fita simples que ao invés de ter um alfabeto de dois símbolos, possui um alfabeto de quatro símbolos dado por $\{0, 1, 2, 3\}$. O leitor terá facilidade para entender que se considerarmos uma fita *multitrack* de k trilhas e um alfabeto de n símbolos, sempre poderemos convertê-la em uma Máquina de Turing de fita simples com n^k símbolos.

Simulação com fita simples

q	σ	$\delta'(q, \sigma)$
q_0	0	$(q_0, 1)$
q_0	1	$(q_1, 1)$
q_0	2	$(q_1, 3)$
q_0	3	$(q_1, 0)$
q_1	0	$(q_1, 2)$
q_1	1	$(h, 1)$
q_1	2	$(q_0, 2)$
q_1	3	$(q_1, 0)$

Deste modo, um computador atual contendo um barramento de 64 bits, e ainda sabendo que os níveis de quantização são apenas dois (0 e 1), temos que existe uma Máquina de Turing de fita simples análoga ao computador, a qual está associado um alfabeto de 2^{64} símbolos. Sob a ótica da Engenharia, este número gigante de símbolos é desconfortável de ser manipulado, o que tornaria a construção de tal máquina muito trabalhosa. Entretanto, desconsiderando esta questão, tanto a máquina como o computador são capazes de manipular a mesma quantidade de dados.

A segunda variação de Máquina de Turing é a máquina com várias fitas, uma representação análoga à categoria MIMD. Esta máquina possui uma única unidade de controle, porém apresenta mais de um par de fitas com cabeças leitoras. A Figura 7.4 apresenta uma máquina de duas fitas e duas cabeças. Nesta máquina a leitura e a escrita na fita é realizada de modo síncrono, isto é, símbolos são lidos e escritos simultaneamente em suas respectivas fitas. Por sua vez, o movimento das fitas é independente, pois estas não precisam obrigatoriamente avançar para o mesmo lado.

Na parte superior direita da Figura 7.4 são apresentadas de forma esquemática as duas fitas, bem como o posicionamento da cabeça leitora, denotado por \uparrow . Neste caso, a proposta é imitar o comportamento da cabeça leitora através de uma fita auxiliar. Nesta fita auxiliar este posicionamento da cabeça leitora é indicado pela inscrição do símbolo 1 na posição correspondente. Assim, no exemplo proposto

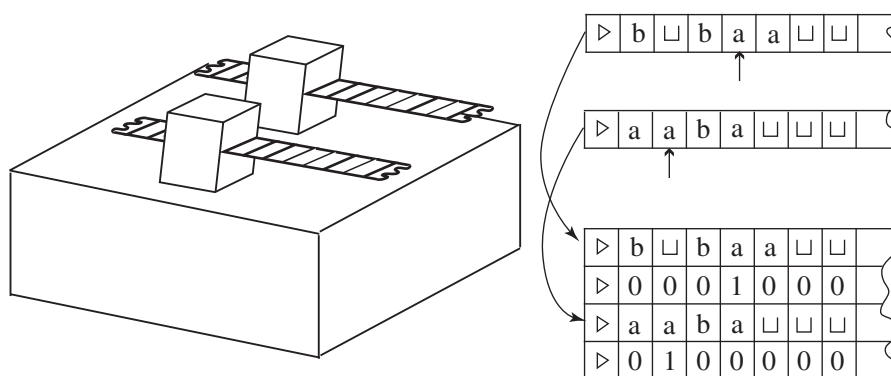


Figura 7.4: Máquina de várias fitas.

transforma-se a máquina em questão em uma máquina de fita *multitrack* de quatro trilhas. Em seguida, converte-se esta fita de quatro trilhas para uma fita simples com um alfabeto mais abrangente.

A máquina de fitas multidimensionais também é uma variação da máquina de fita simples. Nesta situação, a fita multidimensional é transformada em *multitrack*, que em seguida é transformada em fita simples. Já uma máquina multiprocessada na qual existam várias cabeças leitoras para uma única fita, faz-se uma abordagem similar à máquina de várias fitas, marcando-se em uma fita auxiliar o posicionamento das cabeças leitoras com símbolos diferentes para cada uma delas. Casos mais complexos são reduzidos sucessivamente a casos mais simples, até que se obtenha o mapeamento adequado para a máquina de fita simples.

7.4 Máquina Universal de Turing

As Máquinas de Turing desenvolvidas até o momento, são dispositivos construídos para tratar especificamente um determinado problema, isto é, elas são configuradas para uma determinada tarefa, ou de forma simplista, são computadores com um programa armazenado estaticamente. Quando uma nova tarefa precisa ser computada, uma construção nova de máquina precisa ser realizada. Assim, a Máquina de Turing é construída conforme o problema que se deseja resolver, enquanto que os computadores atuais são máquinas de propósito geral podendo ser programadas.

Referências Bibliográficas

- [1] HOPCROFT, J. E., *Theory os Machines and Compuatations*, chapter An n log n algorithm for minimizing states in a finite automaton, Academic Press, pp. 189–196, 1971.
- [2] MONTEIRO, A. A., PAULO, J. D. S., *Aritmética Racional*. Lisboa, Livraria Avelar Machado, 1945.
- [3] IFRAH, G., *Os Números: a história de uma grande invenção*. São Paulo, Globo S.A., 2005.
- [4] DEDEKIND, R., *Was sind und was sollen die Zahlen*, v. 3, *Gesammelte Mathematische Werke*. New York, Chelsea Publishing Company, 1969. pp. 335-391.
- [5] GÖDEL, K., *Collected Works*, v. 2, *Gesammelte Mathematische Werke*. Oxford, Oxford University Press, 1990.
- [6] ISRAEL, D., “Reflections on Gödel’s and Gandy’s Reflection on Turing’s Thesis”, *Minds and Machines*, v. 12, n. 2, pp. 181–201, 2002.
- [7] SOBRINHO, J. Z., “Aspectos da Tese de Church-Turing”, *Revista Matemática Universitária - USP*, v. 1, n. 6, pp. 1–23, 1987.
- [8] MCDERMOTT, D., “Artificial Intelligence Meets Natural Stupidity”, *SI-GART Newsletter*, v. 57, pp. 4–9, April 1976.
- [9] SETTI, M. D. O. G., *O Processo de Discretiza çã o do Raciocínio Matemático na Tradu çã o para o Raciocínio Computacional*, Report, Universidade Federal do Paraná, 2009.

- [10] GUERREIRO, G., “A Vanguarda Matemática e os Limites da Razão”, *Scientific American Brasil*, v. 5, n. 12, pp. 39–56, 2007. Coleção Gênios da Ciência.
- [11] SMULLYAN, R., *Gödel’s Incompleteness Theorems*. Oxford University Press, 1992.
- [12] GÖDEL, K., *The Undecidable*, chapter On Formally Undecidable Propositions of Principia Mathematica and Related Systems, New York, Raven Press, pp. 5–38, 1965.
- [13] WHITEHEAD, A. N., RUSSELL, B., *Principia Mathematica*. Londres, Cambridge University Press, 1913.
- [14] NAGEL, E., NEWMAN, J. R., *Gödel’s Proof*. USA, Routledge, 1989.
- [15] GÖDEL, K., “Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme - On Formally Undecidable Propositions of Principia Mathematica and Related Systems”, *Kurt Gödel: Collected Works*, v. 1, n. 1, pp. 144–195, 1986. Tradução para o inglês por Martin Hirzel. www.research.ibm.com/people/h/hirzel/papers/canon00-goedel.pdf [capturado em 13 de agosto de 2011].
- [16] MELO, A. C. V. D., SILVA, F. S. C. D., *Modelos Clássicos de Computação*, Coleção Schaum. São Paulo, Thomson, 2006.
- [17] KUBRUSLY, R. D. S., *Uma viagem informal ao Teorema de Gödel ou (O preço da matemática é o eterno matemático)*, Report, Universidade Federal do Rio de Janeiro, 2007. IM/UFRJ.
- [18] SMULLYAN, R., *Recursion Theory for Metamathematics*. Oxford University Press, 1993.
- [19] SMULLYAN, R., *What’s the Name of This Book*. Penguin Books, 1978.
- [20] SMULLYAN, R., *Forever Undecided: A Puzzle Guide to Gödel*. Oxford University Press, 1987.
- [21] GOLDSTEIN, R., *Incompleteness: The Proof and Paradox of Kurt Gödel*. W. W. Norton Company, Inc., 2005.

- [22] HOFSTADTER, D. R., *Gödel, Escher e Bach: an Eternal Golden Braid*. Nova Iorque, Basic Books, 1979.
- [23] Rodríguez-Consuegra, F. A. (ed.), *Kurt Gödel - Unpublished Philosophical Essays*. Berlin, Birkhäuser Verlag, 1995.
- [24] Feferman, S., *et al.* (eds.), *Kurt Gödel - Collected Works*, v. I, II, III. New York, Oxford University Press, 1986.
- [25] WANG, H., *Reflections on Kurt Gödel*. Cambridge, Massachusetts, The MIT Press, 1988.
- [26] SUPPES, P., *Axiomatic Set Theory*. New York, Dover, 1972.
- [27] LIPSCHUTZ, S., *Teoria Elementar dos Conjuntos*, Cole çã o Schaum. Sã o Paulo, McGraw-Hill do Brasil, 1990.
- [28] SUPPES, P., *Axiomatic Theory Set*. USA, Van Nostrand Company Inc., 1960.
- [29] MELLO, F. L. D., CARVALHO, R. L. D., “Knowledge Geometry”, *Journal of Information and Knowledge Management*, v. 14, pp. 1550028, 2015.
- [30] STOLL, R. R., *Set Theory and Logic*. USA, Dover Publications Inc., 1961.
- [31] MIRAGLIA, F., *Teoria dos Conjuntos: um mínimo*. Brasil, Edusp - Editora da Universidade de São Paulo, 1992.
- [32] HALMOS, P., *Teoria Ingênua dos Conjuntos*. Brasil, Edusp - Editora da Universidade de São Paulo, 1970.
- [33] GRATZER, G., *Universal Algebra*. USA, Van Nostrand Company Inc., 1968.
- [34] COHN, P. M., *Universal Algebra*. USA, Harper and Row, 1965.
- [35] GALLIER, J. H., *Logic for Computer Science*. USA, John Wiley and Sons Inc., 1987.
- [36] PREPARATTA, F. P., YEH, R. T., *Introduction to Discrete Structures for Computer Science and Engineering*. USA, Addison-Wesley Publishing Company, 1973.

- [37] SHOENFIELD, J. R., *Degrees of Unsolvability*. North-Holland Publishing Company, 1971.
- [38] BRAINERD, W. S., LANDWEBER, L. H., *Theory of Computation*. USA, John Wiley and Sons, 1974.
- [39] EILENBERG, S., ELGOT, C., *Recursiveness*. New York: Academic Press, 1970.
- [40] CARVALHO, R. L. D., OLIVEIRA, C. M. G. M. D., *Modelos de Computação e Sistemas Formais*, 11^a Escola de Computação. Rio de Janeiro, Universidade Federal do Rio de Janeiro, 1998.
- [41] BRAINERD, W. S., LANDWEBER, L. H., *Theory of Computation*. John Wiley & Sons, 1974.
- [42] KLEENE, S. C., *Introduction to Metamathematics*. D. Van Nostrand Company, Inc., 1952.
- [43] Rogers Jr., H., *Theory of Recursive Functions and Effective Computability*. USA, McGraw-Hill Book Company, 1967.
- [44] DAVIS, M., *Computability and Unsolvability*. New York, Dover, 1983.
- [45] BOOLOS, G. S., JEFFREY, R. C., *Computability and Logic*. Cambridge University Press, 1974.
- [46] MARTIN DAVIS, R. S., WEYUKER, E. J., *Computability Complexity and Languages*. New York, Academic Press, 1994.
- [47] MALLOZZI, J. S., LILLO, N. J. D., *Computability with Pascal*. New Jersey, Prentice-Hall, Inc., 1984.
- [48] TURING, A. M., *The Undecidable*, chapter On Computable Numbers, with an Application to the Entscheidungsproblem, New York, Raven Press, pp. 115–151, 1965.
- [49] ELGOT, C. C., ROBINSON, A., “Random-access stored-program machines, an approach to programming languages”, *Journal of the ACM*, v. 11, pp. 365–399, 1964.

- [50] PREPARATTA, F. P., YEH, R. T., *Introduction to Discrete Structures for Computer Science and Engineering*. Addison-Wesley Publishing Company, 1973.
- [51] HENNIE, F., *Introduction to Computability*. Addison-Wesley Publishing Company, 1977.
- [52] NELSON, R. J., *Introduction to Automata*. USA, Jonh Wiley & Sons, Inc., 1968.
- [53] CARVALHO, R. L. D., *Máquinas, Programas e Algoritmos*, 2^a Escola de Computação. Campinas, Universidade Estadual de Campinas, 1981.
- [54] MENDELSON, E., *Introduction to Mathematical Logic*, Cole Mathematics Series. 3 ed. The Wadsworth and Brooks, 1987.
- [55] MANIN, Y. I., *A Course in Mathematical Logic*, Graduate Texts in Mathematics 53. 1 ed. Springer-Verlag, 1977.
- [56] HOMER, S., SELMAN, A. L., *Computability and Complexity Theory*, Texts in Computer Science. 2 ed. Springer, 2011.
- [57] CUTLAND, N. J., *Computability: An introduction to recursive function theory*. Cambridge University Press, 1980.
- [58] SIPSER, M., *Introduction to The Theory of Computation*, Course Technology Series. 2 ed. Thomson, 2006.
- [59] WALTER CARNIELLI, R. L. E., *Computability: computable functions, logic and the foundations of mathematics*. Belmont, Wadsworth and Brooks, 1989.
- [60] TARSKI, A., *Logic, semantics, metamathematics*, chapter Fundamental concepts of the methodology of the deductive sciences, London, Oxford at the Clarendon Press, pp. 60–109, 1969.
- [61] ADAM YOUNG, M. Y., *Malicious Cryptography: Exposing Cryptovirology*. John Wiley and Sons Inc., 2004.

- [62] BONFANTE, G., KACZMAREK, M., MARION, J.-Y., *A Classification of Viruses through Recursion Theorems*, volume 4497 of Lecture Notes in Computer Science. 2 ed. CiE 2007, 2007.
- [63] MACHTEY, M., YOUNG, P., *An Introduction to the General Theory of Algorithms*. New York, North Holland, 1978.
- [64] ROBINSON, J. A., “A machine oriented logic based on the resolution principle”, *J. Assoc. Comput.*, v. 12, pp. 23–41, 1965.
- [65] ROBINSON, J. A., “Automatic deduction with hyper-resolution”, *Internat. J. Comput. Math.*, v. 1, pp. 227–234, 1965.
- [66] GILMORE, P. C., “A proof method for quantification theory: Its justification and realization”, *IBM Journal of Research and Development*, v. 4, n. 1, pp. 28–35, 1960.
- [67] DAVIS, M., PUTNAM, H., “A computing procedure for quantification theory”, *Journal of the ACM*, v. 7, n. 3, pp. 201–215, 1960.
- [68] CHANG, C.-L., LEE, R. C.-T., *Symbolic Logic and Mechanical Theorem Proving*. Academic Press Inc, 1973.
- [69] KLEENE, S. C., *Mathematical Logic*. Wiley, 1967.
- [70] HILBERT, D., ACKERMANN, W., *Prinmciples of Mathematical Logic*. Chelsea, 1950.
- [71] MCCAWLEY, J. D., *Everything That Linguists Have Always Wanted To Know About Logic*. 2 ed. The University of Chicago Press, 1993.
- [72] SUPPES, P., *Introduction to Logic*. D. van Nostrand, 1966.
- [73] RUSSELL, B., *A Filosofia do Atomismo Lógico*, *Lógica e Conhecimento*. 1 ed. Abril Cultural, 1974. (Os Pensadores, 42).
- [74] RUSSELL, B., *Significado e Verdade*. 1 ed. Zahar, 1978.
- [75] POPPER, K. R., *A Lógica da Pesquisa Científica*. 2 ed. Cultrix, 1974.

- [76] LAKATOS, I., , MUSGRAVE, A., *A Crítica e o Desenvolvimento do Conhecimento*. 1 ed. EDUSP, Cultrix, 1979. Tradução: M. O. Caiado.
- [77] WANG, H., *From Mathematics to Philosophy*. 1 ed. Cultrix, 1974.
- [78] GREEN, C. C., *The Application of Theorem Proving to Question-Answering Systems*. Ph.D. dissertation, Stanford, June 1969. AI Project MEMO AI-96.
- [79] LOVELAND, D. W., *Automated Theorem Proving: a Logical Basis*. 1 ed. North Holland, 1978.
- [80] HUGHES, G. E., LONDEY, D. G., *The Elements of Formal Logic*. USA, Methuen and Co Ltd, 1965.
- [81] BOOK, R. V., OTTO, F., *String-Rewriting Systems*. USA, Springer-Verlag, 1993.
- [82] HOPCROFT, J. E., ULLMAN, J. D., *Introduction to Automata Theory, Language and Computation*. USA, Addison-Wesley Publishing Company, 1979.
- [83] HOPCROFT, J. E., ULLMAN, J. D., *Formal Languages and their Relation to Automata*. USA, Addison-Wesley Publishing Company, 1969.
- [84] CURRY, H. B., *Foundations of Mathematical Logic*. New York, Academic Press, 1977.
- [85] SMULLYAN, R., *Theory of Formal Systems*. USA, Princeton, 1961.
- [86] BERSTEL, J., BOASSON, L., CARTON, O., *et al.*, *Handbook of Automata: from Mathematics to Applications*, chapter Minimization of automata, European Mathematical Society, pp. 189–196, 2010.
- [87] BEAL, M. P., CROCHEMORE, M., “Minimizing incomplete automata”, *Workshop on Finite State Methods and Natural Language Processing*, , september 2008. Ispra.
- [88] VALMARI, A., LEHTINEN, P., “Efficient minimization of DFAs with partial transition”, *Proc. 25th Symp. Theoretical Aspects of Comp. Sci.*, v. 08001, pp. 645–656, 2008. S. Albers and P. Weil, editors.

- [89] PAPADONIKOLAKIS, M., BOUGANIS, C.-S., CONSTANTINIDES, G.,
“Performance comparison of GPU and FPGA architectures for the SVM training problem”, *IEEE International Conference on FieldProgrammable Technology*, pp. 388–391, 2009.
- [90] MU, S., WANG, C., LIU, M., *et al.*, “Evaluating the potential of graphics processors for high performance embedded computing”, *Proc. IEEE Design, Automation and Test in Europe Conference and Exhibition*, pp. 709–714, 2011.
- [91] KAI HWANG, F. A. B., *Computer Architecture and Parallel Processing*. McGraw-Hill, 1984.
- [92] AGARWAL, P., KRISHNAN, S., MUSTAFA, N., *et al.*, *Algorithms - ESA 2003, Lecture Notes in Computer Science*, chapter Streaming Geometric Optimization Using Graphics Hardware, Springer Berlin-Heidelberg, pp. 115–151, 2003.
- [93] TANENBAUM, A. S., *Organização Estruturada de Computadores*. 3 ed. Prentice Hall do Brasil, 1997.
- [94] BACKUS, J., “Can Programming be Liberated from von Neumann Style? A functional style and its algebra of program”, *ACM Turing Award Lecture, Communications of the ACM*, v. 21, n. 8, pp. 613–641, 1978.
- [95] OWENS, J. D., LUEBKE, D., GOVINDARAJU, N., *et al.*, “A Survey of General-Purpose Computing on Graphics Hardware”, *Eurographics 2005, State of the Art Reports*, pp. 21–51, 2005.
- [96] GUSTAFSON, J. L., “Reevaluating Amdahl’s law”, *Communications of the ACM*, v. 5, n. 31, pp. 532, 1988.
- [97] HANDLER, W., *Parallel Processing Systems, an advanced course*, chapter Innovative computer architecture - how to increase parallelism but not complexity, Cambridge University Press, pp. 1–41, 1982.
- [98] LOBUR, J., NULL, L., *The Essentials of Computer Organization And Architecture*. Jones and Bartlett Pub, 2006.

- [99] LEWIS, H. R., PAPADIMITRIOU, C. H., *Elements of the Theory of Computation*. 2 ed. New York, Prentice-Hall, 1998.
- [100] DUNNE, P., *Computability Theory: Concepts and Applications*. Ellis Horwood, 1991.
- [101] AARONSON, S., “NP-complete Problems and Physical Reality”, *ACM SIGACT News*, , march 2005. Complexity Theory Column 46.