Sumário

Introdução

A ciência da computação se fundamenta em modelos matemáticos descritivos de todos os processos computacionais. Os modelos utilizados atualmente foram construídos na primeira metade do século XX. Aparentemente, uma necessidade de caracterizar rigorosamente o que significaria *ser computável* deve ter sido sentida em um momento histórico específico, pois diversos pesquisadores trabalharam simultaneamente nesse problema.

Um fato muito curioso na história da ciência aconteceu quando esses pesquisadores apresentaram seus resultados, construídos de forma independente embora relacionada, e eles se mostraram matematicamente equivalentes. Em outras palavras, parecia já existir de fato um conceito característico de computabilidade, que podia ser formulado de diversas maneiras.

As demonstrações de equivalência das formulações iniciais do conceito de computabilidade estimularam outros pesquisadores a produzirem novas formulações, e hoje contamos com uma coleção razoavelmente grande dessas formulações equivalentes.

A equivalência matemática, entretanto, acabou por ofuscar as *diferenças* entre essas formulações. Cada formulação nasceu de um contexto de pesquisas distinto, sendo, portanto, influenciada por um arcabouço conceitual e cultural também distinto. Nesse sentido, podemos afirmar que as formulações *não são* equivalentes, elas têm papéis diferenciados na evolução da ciência da computação, e cada uma evidencia com maior clareza aspectos específicos da noção de computabilidade.

Atualmente, a prática da construção de programas e sistemas computacionais é uma disciplina multifacetada. Contamos com diferentes técnicas e

métodos para a construção de programas e sistemas, que se fundamentam nos diferentes *paradigmas de programação* (funcional, orientado a objetos, etc.). Neste livro desenvolvemos uma tese original:

> *Diferentes modelos de computação se mostram mais adequados para explicar diferentes paradigmas de programação. Portanto, o estudo dos diferentes paradigmas de programação deve fazer uso de diversos modelos de computação.*

Para defender essa tese, apresentamos três modelos de computação publicados no mesmo ano, 1936, mas formulados de forma independente e depois demonstrados como sendo matematicamente equivalentes. Hoje, eles são os mais usados para apresentar os fundamentos teóricos da computação. Em nossa apresentação desses modelos, evidenciamos suas *diferenças* em vez de sua equivalência formal. Apresentamos então, para cada modelo de computação, o(s) paradigma(s) de programação mais bem explicado(s) segundo aquele modelo.

Os três modelos são:

- A *teoria de funções recursivas* de Stephen Cole Kleene, que caracteriza a classe de funções matemáticas computáveis, formuladas como funções de números naturais. A tradução de uma função qualquer em uma função de números naturais equivalente em formato adequado para verificar se ela é ou não uma função recursiva não é tarefa fácil. Matematicamente, entretanto, essa tradução é sempre possível. Assim, ao menos na teoria, as funções recursivas catalogam tudo o que pode ser resolvido com a computação.

- O λ-*cálculo* de Alonzo Church, que associa computabilidade a uma noção de demonstração explícita da validade de uma equação: uma função f é computável se, para quaisquer x e y tais que $f(x) = y$, podemos demonstrar que $f(x) = y$. Intuitivamente, é como se o λ-cálculo desenhasse um mapa com todas as funções computáveis. Cada função é uma rota nesse mapa, levando os valores iniciais x aos valores finais y. Assim como ocorre com os entroncamentos e as ligações nos caminhos em um mapa rodoviário, as funções computáveis podem ser combinadas e desmembradas.

Conforme veremos, existe uma classe de paradigmas de programação, denominados *declarativos*, que são mais bem explicados com base na teoria de funções recursivas e no λ-cálculo.

- A *máquina abstrata* de Alan Turing – atualmente conhecida como máquina de Turing –, que identifica como função computável qualquer função que possa ser processada por ela. Intuitivamente, a máquina de Turing torna a computação uma atividade semelhante à que ocorre em uma fábrica. Os valores iniciais de uma função computável são a matéria-prima e os valores finais são o produto acabado.

  Também veremos que os paradigmas de programação denominados *imperativos* são mais bem explicados com base na máquina de Turing.

Alonzo Church, que nasceu nos Estados Unidos em 1903, concluiu seu doutorado na Universidade de Princeton em 1927. Logo em seguida, ele passou um ano em Harvard e um ano na Holanda, onde trabalhou com Luitzen Egbertus Jan Brouwer. Muito provavelmente, a influência da filosofia intuicionista de Brouwer levou Church a desenvolver o trabalho que culminou no λ-cálculo.

Em 1931, o matemático austríaco Kurt Gödel (1906-1978) havia publicado seu famoso *Teorema da Incompletude*, demonstrando a impossibilidade de se provar a veracidade ou a falsidade de certas proposições matemáticas denominadas *indecidíveis*. Um corolário importante desse teorema é a impossibilidade de garantir a consistência da aritmética sem fazer uso de propriedades que precisam ser válidas para conjuntos infinitos. Esses resultados foram apresentados em um curso em Princeton, em 1934, e impulsionaram Church a trabalhar em uma formulação geral das proposições matemáticas *decidíveis*. O resultado foi o λ-cálculo.

O norte-americano Stephen Cole Kleene, nascido em 1909, foi aluno de doutorado de Church e também participou do curso ministrado por Gödel em 1934. Naquele mesmo ano, Kleene havia concluído seu doutorado. A influência da filosofia intuicionista de Brouwer e o estudo dos trabalhos de Gödel levaram Kleene a construir sua própria formulação das proposições decidíveis, na forma de uma teoria de funções recursivas.

Alan Turing nasceu na Inglaterra em 1912 e em 1935 participou de um curso em Cambridge devotado a discutir, entre outros temas, os resultados de Gödel. Em 1936, Turing publicou a sua versão de funções computáveis, fundamentada em uma máquina abstrata. Diferentemente de seus colegas

norte-americanos, entretanto, a motivação para seus estudos parece ter vindo não dos fundamentos intuicionistas da matemática mas sim da proposição de Bertrand Russell (1872-1970) e Alfred North Whitehead (1861-1947), refutada por Gödel, que sugeria ser possível deduzir todos os resultados da matemática a partir de um conjunto de princípios básicos. A proposição de Russell e Whitehead seguia a linha sugerida por David Hilbert (1862-1943), denominada formalismo, que divergia frontalmente do intuicionismo. Os resultados de Turing, no entanto, corroboraram os de Church e Kleene.

Turing teve grande dificuldade para publicar seu trabalho, pois alguns meses antes Church havia publicado um resultado semelhante, fundamentado no λ-cálculo. O formato dos dois resultados era tão distinto que se acreditou, em princípio, que uma das formulações deveria ser rejeitada. Como o trabalho de Church já havia sido publicado, e o pesquisador norte-americano já contava com grande prestígio nos meios acadêmicos, coube a Turing o "ônus da prova" da validade de suas formulações.

O trabalho de Turing só foi publicado quando ele acrescentou um apêndice demonstrando a equivalência de seus resultados com os de Church. Isso chamou a atenção de Church, que então convidou Turing para trabalhar em Princeton como seu estudante de doutorado. Posteriormente, também foi demonstrado que as funções computáveis conforme formuladas por Turing também coincidiam com as funções parciais recursivas da teoria de Kleene.

A Universidade de Princeton e o estudioso Alonzo Church, portanto, constituem o eixo ao redor do qual surgiram os resultados que estudaremos neste livro. Aparentemente, os fatores de maior influência para estimular o desenvolvimento dessas formulações foram os resultados de incompletude da matemática obtidos por Gödel e o intuicionismo de Brouwer.

Nosso objetivo é equilibrar a importância dos diferentes modelos de computação para a fundamentação das atividades de programação. A adoção do modelo mais apropriado em função do paradigma de programação selecionado, que por sua vez decorre do tipo de problema tratado, deve simplificar a formulação de resolução de problemas, permitindo a construção mais eficiente de programas elegantes, com a garantia de estarem corretos.

Estrutura do livro

Este livro tem caráter introdutório. Em outras palavras, tomamos cuidado para que fosse auto-suficiente e não exigisse conhecimentos prévios do leitor.

Por outro lado, o tratamento dado a cada tema apresentado é geral, visando a proporcionar um entendimento amplo sobre cada assunto. No final de cada capítulo incluímos uma série de sugestões bibliográficas para quem se interessar por um entendimento mais profundo sobre qualquer um dos temas tratados.

No Capítulo 1 apresentamos a teoria de funções recursivas de Kleene. De todas as teorias de computabilidade, essa é a mais concisa e matematicamente elegante. Ela é útil para identificar propriedades matemáticas das funções computáveis, mas é a mais difícil de operacionalizar na forma de programas.

No Capítulo 2 apresentamos o λ-cálculo de Church, que permite formular com clareza uma teoria de programação declarativa, apresentada no Capítulo 3.

No Capítulo 4 apresentamos as funções computáveis conforme caracterizadas pela máquina de Turing, que permite formular com clareza uma teoria de programação imperativa, também apresentada no mesmo capítulo.

Finalmente, na Conclusão, discutimos possíveis desdobramentos e extensões do material apresentado.

Público-alvo

Apesar do tratamento introdutório dado aos temas desenvolvidos neste livro, os temas em si são um tanto elaborados e exigem do leitor interesse e gosto pela matemática e pelo tratamento formal e conceitual dos fundamentos da ciência da computação.

Este livro é material básico para disciplinas dos currículos de informática de diversas escolas no Brasil. Nossa expectativa é que ele seja útil e prazeroso para estudantes de graduação em ciência da computação e áreas afins, interessados nos fundamentos matemáticos da ciência da computação. De maneira "simétrica", o livro também é útil para estudantes de graduação em matemática e áreas correlatas, que buscam entender melhor os conceitos fundamentais para a construção da computação como ciência. Considerando esse uso de nosso livro, acrescentamos ao longo do texto diversos exercícios que podem ser utilizados em sala de aula.

O material apresentado aqui de forma sintética pode ser usado para a revisão de conceitos por estudantes de mestrado e doutorado em informática, especialmente aqueles provenientes de áreas diversas ou que concluíram a

graduação há tempos e precisam de um material de apoio direto e objetivo que auxilie a complementar e relembrar certos conceitos.

Finalmente, consideramos que um fator diferencial para os profissionais na indústria mais bem capacitados é o domínio dos aspectos abstratos, teóricos e conceituais da ciência da computação. Esse fator diferencial está tornando-se cada vez mais claro e evidente à medida que os sistemas computacionais progressivamente se tornam maiores e mais complexos. Esperamos que esta obra contribua para a reciclagem e atualização de profissionais da indústria a fim de atender essas exigências cada vez mais rigorosas do mercado de trabalho.

Notas e referências

A teoria de funções recursivas foi originalmente publicada em Benacerraf, Putnam (1983). O λ-cálculo foi originalmente publicado em Barendregt (1990), e a máquina de Turing, em 1936.

Church foi um professor notável, orientando os trabalhos de outros 29 alunos além de Kleene e Turing. Ele faleceu em 1995, aos 92 anos. Kleene faleceu aos 85 anos, em 1994, tendo formado diretamente 13 alunos. Turing teve uma vida dramática com um final trágico, tendo falecido em 1954 com apenas 42 anos. A biografia de Turing pode ser encontrada em Hodges (1992).

A equivalência entre as funções computáveis usando máquinas de Turing e as funções redutíveis pelo λ-cálculo, conforme já mencionado, encontra-se no artigo original de Turing. A equivalência entre as funções computáveis usando máquinas de Turing e as funções parciais recursivas pode ser encontrada em diversas referências, entre as quais sugerimos Mendelson (1987) e Epstein, Carnielli (2000).

A distinção entre a matemática formalista e a matemática intuicionista, que de certa forma inspirou o desenvolvimento dos trabalhos que estudaremos neste livro, está primorosamente caracterizada na coletânea de textos (Benacerraf, Putnam, 1983).

1 Teoria de Funções Recursivas

No presente capítulo consideraremos uma classe de funções de números naturais. Essa classe de funções, caracterizada por S. C. Kleene, representa o que é possível resolver usando o processamento de um computador. Em outras palavras, ela representa o *conjunto de funções computáveis*, ou seja, um problema pode ser resolvido usando o computador se, e somente se, ele puder ser "traduzido" na forma de uma função de números naturais pertencente à classe que estudaremos a seguir.

Esse resultado, embora importante, é eminentemente teórico. Conforme veremos com alguns exemplos, a "tradução" de um problema na forma de funções de números naturais para verificar se ele pertence ou não ao conjunto das funções computáveis está longe de ser uma questão trivial.

1.1 Números naturais

Por números naturais, neste capítulo, consideramos as entidades matemáticas caracterizadas pelos *postulados de Peano*. Giuseppe Peano (1858-1932) foi um importante matemático italiano que em 1889 propôs as seguintes propriedades como definição dos números naturais:

- 0 é um número natural.
- Se x for um número natural, então existe um número natural definido a partir de x, denotado como $s(x)$ e denominado o *sucessor de* x.

- Para qualquer número natural x, temos que $0 \neq s(x)$.
- Se $s(x) = s(y)$, então $x = y$.
- Se uma propriedade $\mathcal{Q}$ for tal que:
  - 0 tem a propriedade $\mathcal{Q}$;
  - se x tem a propriedade $\mathcal{Q}$, então $s(x)$ também tem a propriedade $\mathcal{Q}$ para qualquer número natural x;

  então todos os números naturais têm a propriedade $\mathcal{Q}$.

Denotaremos os números naturais com a notação usual, que facilita nossa leitura. Assim, escreveremos 0, 1, 2, 3, ... em vez de 0, $s(0)$, $s(s(0))$, $s(s(s(0)))$ ... .

Uma função n-ária de números naturais é uma função matemática de n argumentos, em que tanto os argumentos como os resultados da função são números naturais. Uma função n-ária de números naturais pode ser caracterizada por uma tabela com $n + 1$ colunas, em que as n primeiras colunas representam os argumentos da função e a última coluna representa os resultados da função para cada n-tupla de argumentos.

Exemplo 1.1.1

Por exemplo, a função $f(x) = 2 \times x$ pode ser representada em uma tabela com 2 colunas:

x	$f(x)$
0	0
1	2
2	4
3	6
...	

Exemplo 1.1.2

Como um segundo exemplo, a função $g(x, y) = x + y$ pode ser representada em uma tabela com 3 colunas:

x	y	$g(x, y)$
0	0	0
0	1	1
0	2	2
	...	
1	0	1
1	1	2
1	2	3
	...	

Exemplo 1.1.3

Como um terceiro exemplo, podemos também representar a função $h(x) =$ um número arbitrariamente selecionado para cada valor de x, sem respeitar qualquer regra de regularidade ou dependência dos valores específicos da variável x:

x	$h(x)$
0	35
1	3
2	5429
3	6
	...

Podemos construir, para os dois primeiros exemplos, algoritmos que calculam os valores das funções para quaisquer valores dos argumentos. Para o terceiro exemplo, entretanto, isso não pode ser efetuado, e os valores da função que não forem explicitamente fornecidos permanecerão desconhecidos. Nesse sentido, as duas primeiras funções são *computáveis* e a terceira função não é.

Kleene propôs um conjunto de funções iniciais e um conjunto de construtores de funções que podem ser encadeados. Uma função computável, segundo a formulação de Kleene, é uma função inicial ou uma função obtida a partir das funções iniciais por meio do encadeamento dos construtores.

1.2 Funções recursivas primitivas

As *funções iniciais* são as seguintes:

- A função anuladora $Ze(x) = 0$ para qualquer número natural x.
- A função sucessor $Suc(x) = s(x)$ para qualquer número natural x.
- As funções de projeção $Pr_i^n(x_1, \ldots, x_n) = x_i$ para qualquer número natural x, qualquer $n > 0$ e qualquer i, tal que $1 \leqslant i \leqslant n$.

As funções anuladora e sucessor são únicas, mas existem infinitas funções de projeção.

O conjunto das *funções recursivas primitivas* é o menor conjunto, composto por:

- Funções iniciais.
- Função resultante de um número finito de aplicações dos seguintes construtores sobre funções recursivas primitivas:
  - substituição: se $g(x_1, \ldots, x_m), h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n)$, $m, n > 0$ são funções de números naturais, então construímos por substituição a função $f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n))$;
  - recursão: se $g(x_1, \ldots, x_n)$ e $h(x_1, \ldots, x_n, y_1, y_2), n \geqslant 0$ são funções de números naturais, então construímos recursivamente a função
    - $f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n)$;
    - $f(x_1, \ldots, x_n, s(y)) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y))$. Observe que n pode valer zero. Nesse caso, obtemos as seguintes expressões, que permitem visualizar com maior clareza a noção intuitiva de recursão implícita na formulação geral:
    - $f(0) = k$, uma constante arbitrária;
    - $f(s(y)) = h(y, f(y))$.

Vejamos alguns exemplos de funções recursivas primitivas.

Exemplo 1.2.1

Função constante. Seja k um número natural qualquer. A função $f(x) = k$ é recursiva primitiva.

De fato, considerando a definição de recursão anterior, se $h(x, y) = Pr_1^2(x, y) = x$, temos para qualquer valor de k a seguinte função recursiva primitiva:

- $f(0) = k$.
- $f(s(y)) = h(y, f(y)) = Pr_1^2(y, f(y)) = y$. É fácil demonstrar que $f(x) = k$ para qualquer valor de x.

Exemplo 1.2.2

Função soma. A função $g(x, y) = x + y$ é recursiva primitiva.

Considerando a definição de recursão com $n = 1$, se

- $g(x, 0) = Pr_1^1(x) = x$.
- $g(x, s(y)) = h(x, y, g(x, y))$. Por substituição, se definirmos que $h(x, y, z) = Suc(Pr_3^3(x, y, z))$, temos que $g(x, s(y)) = s(g(x, y))$.

Exemplo 1.2.3

Seja k um número natural qualquer. A função $f(x) = k \times x$ é recursiva primitiva.

Por substituição, obtemos que $f(x) = Pr_2^2(u(k, x))$ para alguma função $u(k, x)$. Se $u(k, x)$ for recursiva primitiva, então $f(x)$ também será.

Por recursão, seja

- $u(k, 0) = Ze(k) = 0$.
- $u(k, s(y)) = h(k, y, u(k, y))$. Por substituição, se definirmos que $h(x, y, z) = g(Pr_1^3(x, y, z), Pr_3^3(x, y, z))$, em que $g(x, Y)$ é a função soma do exemplo anterior, temos que $u(k, s(y)) = g(k, u(k, y))$.

Com esses exemplos, pretendemos ilustrar como trabalhar com as funções recursivas primitivas. Ilustramos também a complexidade inerente a verificar se uma função é recursiva primitiva. É bastante trabalhoso efetuar tais verificações, até mesmo para funções aparentemente simples. A dificuldade cresce ainda mais quando o resultado esperado é negativo. Por exemplo, como demonstrar que a função do exemplo 1.1.3 não é recursiva primitiva?

Essa demonstração exige analisar todas as (infinitas) formas de relacionar os valores de função com seus parâmetros, o que é naturalmente muito trabalhoso.

Exercício

1.1 As funções a seguir são recursivas primitivas. Demonstre esse fato:

a. $f(x, y) = x \times y$.

b. $f(x, y) = x^y$.

c. $f(x) = x!$.

d. $div(x, y) =$ o quociente da divisão de y por x.

e. $mod(x, y) =$ o resto da divisão de y por x.

f. $f(x) = sqr(x) =$ o maior número natural y tal que $y \leqslant \sqrt{x}$.

g. $f(x) = pr(x) =$ a quantidade de números primos menores que ou iguais a x.

h. $f(x) = fib(x) =$ o n-ésimo número de Fibonacci: $f(0) = 1, f(1) = 1, f(k + 2) = f(k) + f(k + 1), k \geqslant 0$.

1.3 Funções recursivas parciais

As funções recursivas primitivas caracterizam uma classe de funções bastante ampla e expressiva. Intuitivamente, elas capturam a noção de computabilidade que desejamos apresentar. Existem, entretanto, algumas funções "estranhas" que, embora sejam computáveis (ou seja, sabemos construir um programa para calcular o seu valor para quaisquer números naturais), não são recursivas primitivas.

Demonstrar que uma função computável *não* é recursiva primitiva é bastante complexo e, em geral, requer técnicas matemáticas além do escopo

deste trabalho. Algumas dessas funções "estranhas", no entanto, são surpreendentemente simples.

A função computável não-recursiva primitiva mais simples que se conhece é a função de Ackermann:

$$f(x,y) = \begin{cases} y+1 & \text{se } x = 0 \\ f(x-1,1) & \text{se } y = 0 \\ f(x-1,f(x,y-1)) & \text{caso contrário} \end{cases}$$

Para incluir essas funções "estranhas" na classe de funções recursivas primitivas, foi proposto um terceiro construtor, definido a seguir:

- Seja $g(x_1,\ldots,x_n,y)$ uma função tal que para quaisquer valores de $x_1,\ldots,x_n$ existe pelo menos um valor de y de modo que $g(x_1,\ldots,x_n,y) = 0$. Seja $\mu_y[g(x_1,\ldots,x_n,y) = 0]$ o menor valor de y para que $g(x_1,\ldots,x_n,y) = 0$. Então, apenas para as funções $g(x_1,\ldots,x_n,y)$ com essa propriedade podemos construir a função $f(x_1,\ldots,x_n) = \mu_y[g(x_1,\ldots,x_n,y) = 0]$.

As funções recursivas primitivas, estendidas com esse novo construtor, são denominadas *funções recursivas totais*.

Consideremos, entretanto, a função recursiva primitiva $g(x,y) = x+y$. Se $x = 0$, então $\mu_y[g(x,y) = 0] = 0$. Mas se $x > 0$, então $\mu_y[g(x,y) = 0] = 0$ não tem valor definido. A função $\mu_y[g(x,y) = 0] = 0$, portanto, é uma *função parcial*.

Na maioria das linguagens de programação conhecidas, contamos com verificadores condicionais (como o par `if ... else` em C). A programação de funções parciais é perfeitamente factível, e essa função deveria também ser considerada computável.

Para obter esse efeito, o construtor mencionado é generalizado, eliminando a restrição de $g(x_1,\ldots,x_n,y)$ precisar ter pelo menos um valor de y tal que $g(x_1,\ldots,x_n,y) = 0$ para *quaisquer* valores de $x_1,\ldots,x_n$. A formulação generalizada desse construtor fica assim:

- Seja $g(x_1,\ldots,x_n,y)$ uma função de números inteiros. Seja $\mu_y[g(x_1,\ldots,x_n,y) = 0]$ o menor valor de y – se existir – tal que $g(x_1,\ldots,x_n,y) = 0$. Então podemos construir a função $f(x_1,\ldots,x_n) = \mu_y[g(x_1,\ldots,x_n,y) = 0]$.

Pode ocorrer de $f(x_1, \ldots, x_n) = \mu_y[g(x_1, \ldots, x_n, y) = 0]$ ser uma função parcial, ou seja, definida apenas para algumas n-tuplas de números naturais $x_1, \ldots, x_n$. Por esse motivo, as funções caracterizadas pela adição desse construtor aos construtores das funções recursivas primitivas são denominadas *funções recursivas parciais*.

As funções recursivas parciais correspondem exatamente ao conjunto de funções computáveis, definido de maneiras distintas nos próximos capítulos.

Notas e referências

Boa parte do exposto nesse capítulo foi adaptado de Mendelson (1987). Essa referência apresenta muitas das demonstrações omitidas aqui.

Uma demonstração de que a função de Ackermann é recursiva total mas não é recursiva primitiva pode ser encontrada no livro *Computability: computable functions, logic and the foundations of mathematics* (Epstein, Carnielli 2000).

Informações adicionais a respeito de funções recursivas podem ser encontradas no excelente – embora complexo – livro de Martin Davis, *Computability and unsolvability* Davis (1958).