A Computação e as funções recursivas parciais

Prof. Sanderson L. Gonzaga de Oliveira - sanderson@dcc.ufla.br

1. Introdução

A Computação, assim como o conhecimento humano, é resultado do esforço e do trabalho dedicado de indivíduos através dos séculos. Entretanto, pode-se afirmar que a Computação passou a ser considerada uma Ciência e desprender-se da Matemática com as pesquisas de Herbrand, Gödel, Church, Turing, Post, Rosser e Kleene, para citar alguns dos principais matemáticos que começaram a definir precisamente a Computação. Muitas das pesquisas desses matemáticos foram decorrentes de problemas postos por Hilbert. Interessantemente, a Computação iniciou com os estudos de seus limites. O livro de Fonseca Filho (2007) [1] traz uma excelente e didática História da Computação com suas bases matemáticas.

Estudos básicos sobre os limites da Computação adentram na teoria das funções recursivas. Especificamente, a classe das funções recursivas parciais foi mostrada como sendo a classe de funções que podem ser computadas. Para o estudo dessa teoria, o conhecimento de conteúdos do Ensino Médio é essencial. Dentre eles, podem ser citados as sequências, os tipos de relações, as funções e as suas composições.

Considere um exemplo simples de função parcial, que é uma relação não definida em todos os valores do domínio. Sejam os conjuntos D={1,2,3} e I={a,b}. A relação R={(1,a),(2,b)} não é função[1] em D×I, pois nem todos os elementos do conjunto de partida são associados a elementos do conjunto de chegada (veja a Figura 1). R pode ser chamada de função parcial, ou seja, o domínio de definição D contém propriamente a pré-imagem {1,2}.

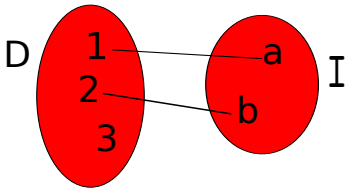

Figura 1: Relação R={(1,a),(2,b)}.

Em outro contexto, pode-se calcular 2^3 em termos de $2{\cdot}2^2 = 2{\cdot}2{\cdot}2^1 = 2{\cdot}2{\cdot}2{\cdot}1$ e afirmar que 1, 2, 4, 8, $\cdots$ é uma sequência em que cada termo é o dobro do anterior. Este texto apresenta vínculos de conceitos essenciais, como funções parciais e sequências, com a Computação e seus limites.

2. Funções recursivas

Uma função recursiva é uma definição formal da classe de funções cujos valores podem ser calculados por recursão. Obtém-se uma família de *funções teórico-numéricas* de um conjunto de funções geradoras e operadores básicos. Uma função teórico-numérica recebe zero ou mais argumentos, sendo cada argumento um número natural e tem como valor também um número natural.

Funções recursivas primitivas são obtidas por meio de operações extremamente básicas. Considere as *funções recursivas primitivas geradoras*, em que $S = \{x_1, \cdots, x_n\}$ é um conjunto de n números naturais:

1. $(z : \mathbb{N}^n \to \mathbb{N}) \mid z(S) = 0$, significa que a *função zero* z retorna zero para qualquer subconjunto de números naturais;
2. $(\forall x \in \mathbb{N})(s : \mathbb{N} \to \mathbb{N}) \mid s(x) = x + 1$, significa que a *função sucessora* s incrementa 1 a um número natural x;
3. $(p_i^n : \mathbb{N}^n \to \mathbb{N}) \mid p_i^n(S) = x_i$, em que $1 \leq i \leq n$, significa que o valor da classe das *funções de projeção* p_i^n é retornado diretamente do i-ésimo argumento - adiante, o superscrito será omitido por clareza;

Uma dessas é a função inicial da recursão. Note que a projeção é uma família de funções, pois depende do número de argumentos e de qual argumento deseja-se projetar [2].

Outras funções recursivas primitivas, ligeiramente mais complexas que as anteriores, são obtidas por meio das duas operações básicas:

[1]Grosso modo, uma relação é uma função total, ou simplesmente função, se é funcional (um elemento do conjunto de partida é associado a exatamente um elemento do conjunto de chegada da relação) e total (definida para todos os valores do domínio).

1. seja $T = \{x_1, \cdots, x_m\}$, a função

$$f(T) = g(h_1(T), \cdots, h_n(T)) \tag{1}$$

de m variáveis, é uma *composição* (ou substituição) das funções: $h_1, \cdots, h_n$ de m argumentos cada e g de n argumentos;

2. a *recursão primitiva* define o valor da função ao utilizar o valor do argumento exatamente anterior, isto é, a partir das funções teórico-numéricas g e h de n e n+2 variáveis, respectivamente, a função recursiva primitiva f de n+1 variáveis é definida por

$$\begin{aligned} \boldsymbol{f(0, S)} &= \boldsymbol{g(S)}, \\ f(1, S) &= h(0, S, f(S, 0)), \\ f(2, S) &= h(1, S, f(S, 1)), \\ \cdots &= \cdots \\ \boldsymbol{f(y+1, S)} &= \boldsymbol{h(y, S, f(S, y))}, \end{aligned} \tag{2}$$

em que $y \in \mathbb{N}$ é chamada de variável recursiva.

Gödel (1934) [3] definiu a classe de funções recursivas primitivas, segundo terminologia introduzida por Kleene (1936) [4], como a totalidade das funções que podem ser geradas pela aplicação de zero ou mais substituição(ões), de acordo com (1), e recursão, de acordo com (2), a partir das funções recursivas primitivas geradoras. Gödel (1934) [3] explicou que uma função f é recursiva se há uma sequência finita de funções $f_1, \cdots, f_n$ que *termina* com f, tal que cada função da sequência é uma das funções recursivas primitivas geradoras ou é composta pelas funções precedentes ou é recursiva em relação às funções precedentes. Gödel (1934) [3] percebeu que as funções recursivas têm a importante propriedade de que, para cada dado conjunto de valores dos argumentos, o valor da função pode ser computado por um *procedimento finito* (ou *algoritmo*).

2.1. Exemplos de funções recursivas primitivas

Veja a seguir alguns exemplos de funções recursivas primitivas.

1. Para cada número natural k, obtém-se a função constante $c_k(x) = k$ ao compor k funções s e uma função z: $s(s(z(x))) = c_2(x) = 2$. Note que $z(x) = 0$, para qualquer número natural x. A função $s(z(x)) = c_1(x) = 1$, pois $s(0) = 1$, isto é, a função s incrementa 1 ao argumento 0. Consequentemente, a função $s(s(z(x))) = c_2(x) = 2$, pois $s(1) = 2$, isto é, a função s incrementa 1 ao argumento 1.

2. Se $S = \{0, 1, 2, \cdots\}$, então as funções constantes[2] também podem ser obtidas através das funções de projeção $c_k(x) = p_k(S) = k$. Note que a função $p_k(S)$ de projeção recebe um conjunto de números naturais e, simplesmente, retorna o k-ésimo argumento.

3. Considere *soma* a função que computa a soma de dois números naturais, definida por recursão primitiva de[3]

   - $soma(x, 0) = g(x, 0) = p_1(x, 0) = x$, indicando que a soma de um número natural com zero é o próprio número;
   - $soma(x, y+1) = h(x, y, soma(x, y)) = s(p_3(x, y, soma(x, y))) = s(soma(x, y)) = soma(x, y) + 1$, indicando que a soma de dois números naturais x e y é dada pela soma de x e *y-1* incrementada de 1.

   Por exemplo,

$$\begin{aligned} soma(2, 3) &= soma(2, 2) + 1 \\ &= (soma(2, 1) + 1) + 1 \\ &= ((\overbrace{soma(2, 0)}^{p_1(x,0)=2} + 1) + 1) + 1 \\ &= ((2 + 1) + 1) + 1 \\ &= 5 \end{aligned}.$$

[2] Este é o exemplo 13.1.1 de Sudkamp (2006) [5].
[3] Exemplo 4.3a de Cutland (1980) [6] e 13.1.2 de Sudkamp (2006) [5].

4. Considere *mult* a função que computa a multiplicação de dois números naturais definida por recursão primitiva de[4]

   - $mult(x, 0) = g(x, 0) = p_2(x, 0) = 0$, indicando que o produto de um número natural com zero é zero;
   - $mult(x, y + 1) = h(x, y, mult(x, y)) = soma(p_1(x, y, mult(x, y)), p_3(x, y, mult(x, y))) = soma(x, mult(x, y))$, indicando que o produto de dois números naturais x e y é dado adicionando-se x ao produto de x e *y-1*.

   Por exemplo,

$$\begin{aligned} mult(2,3) &= soma(2, mult(2,2)) \\ &= soma(2, soma(2, mult(2,1))) \\ &= soma(2, soma(2, soma(2, \overbrace{mult(2,0)}^{p_2(x,0)=0}))) \\ &= soma(2, soma(2, \overbrace{soma(2,0)}^{p_1(x,0)=2})) \\ &= soma(2, soma(2,2)) \\ &= 6 \end{aligned} .$$

5. Considere o conjunto formado pela sequência *seq* definida por recursão recursiva de

   - $seq(1) = g(1) = c_1(1) = 1$;
   - $seq(y + 1) = h(y, seq(y)) = mult(c_2(y, seq(y)), p_2(y, seq(y))) = mult(2, seq(y))$, indicando que o elemento y é dado pelo dobro do elemento *y-1*.

   Por exemplo,

$$\begin{aligned} seq(4) &= mult(2, seq(3)) \\ &= mult(2, mult(2, seq(2))) \\ &= mult(2, mult(2, mult(2, \overbrace{seq(1)}^{1}))) \\ &= mult(2, mult(2, mult(2,1))) \\ &= 8 \end{aligned} ,$$

   que é o exemplo dado no início deste texto.

2.2. Funções recursivas parciais

As funções recursivas parciais são exatamente aquelas com domínio em $\mathbb{N}$ que podem ser definidas por algoritmos. Um função recursiva parcial (ou μ-recursiva) tem domínio em $\mathbb{N}^n$ e pode ser obtida inicialmente de funções recursivas primitivas geradoras e, então, por um número finito de aplicações de composição, recursão primitiva e *minimização* de *funções minimizáveis*. A minimização é o processo de definir uma nova função f ao procurar por valores de uma dada função g usando o operador μ de minimização de g. Novamente, seja $S = \{x_1, \cdots, x_n\}$ um conjunto de n números naturais. A função g de $n+1$ variáveis é minimizável se $(\exists y)|(g(S, y) = 1)$. Kleene (1936) [4] mostrou o processo de minimização de uma função g como a operação $\mu y[g(S, y) = 1]$, ou seja, que satisfaz g, dada por [7]

$$f(S) = \begin{cases} \text{se } g(S,i) = 0 \text{ para } 0 \leq i < y, \text{ em que } i \in \mathbb{N}, \\ \text{ o menor } y \text{ tal que } g(S,y) = 1 \text{ se } y \text{ existe;} \\ 0, \text{ caso contrário,} \end{cases} \tag{3}$$

em que 1 e 0 representam os valores lógicos verdadeiro e falso, respectivamente.

O operador μ falha quando aplicado a alguns argumentos e tal função é, portanto, parcial. Uma maneira de se entender o operador μ é tentar computar, sucessivamente, todos os valores

[4]Exemplo 13.1.3 de Sudkamp (2006) [5].

$g(S,0), g(S,1), g(S,2), \cdots$ até que, para algum $yg(S,y)$=1, y é retornado. Veja este trecho de pseudocódigo [7]:

$y \leftarrow 0$; // comentário: y representa uma posição de memória que recebe o conteúdo 0
enquanto $(g(S,y) \neq verdadeiro)$ faça $y \leftarrow y + 1$;
retorne y;

Apesar de a minimização de g ser sempre bem definida, não há um método efetivo para computá-la. O trecho anterior não compõe um algoritmo porque, eventualmente, pode não terminar [7]. Em mais detalhes, esse trecho de um procedimento falha em retornar um valor, sendo assim, torna-o não *efetivo* e, com isso, g não é minimizável em dois casos: i) se esse y não existe e, por causa disso, um procedimento para essa tarefa não pararia e, portanto, não seria um algoritmo ou ii) se alguma das próprias computações $g(x,0), g(x,1), g(x,2), \cdots$ falha em retornar um valor.

Com isso, Kleene (1936) [4] gerou a classe de funções recursivas parciais. Essa classe é formada pelas funções que podem ser obtidas a partir de funções iniciais por meio de composição de funções naturais simples, constante zero e sucessora, da classe de funções de projeção e por recursão primitiva e por funções μ-recursivas.

3. Considerações finais

A composição de funções naturais simples (constante zero, sucessora e a classe de funções de projeção), da recursão e da minimização, constitui uma forma compacta e natural para definir muitas funções e suficientemente robusta para descrever toda função intuitivamente computável. Veja Diverio e Menezes (1999) [2], para detalhes. Note que a recursão e a minimização compõem uma maneira especial de compor funções.

Este texto apresentou apenas uma breve introdução às funções recursivas. Post (1944) [8] é uma excelente referência para introdução à teoria das funções recursivas e sua relação com o teorema da incompletude de Gödel. Note que a *teoria da computabilidade*, às vezes, é chamada de teoria da recursão. Veja também Cutland (1980) [6], a seção 4.7 de Lewis e Papadimitriou (2004) [7], Odifreddi (2005) [9] e o capítulo 13 de Sudkamp (2006) [5], para detalhes sobre funções recursivas.

Referências

[1] C. F. Filho, História da computação: O caminho do pensamento e da tecnologia, webpage, acessado em dezembro de 2010 (2007).
URL `http://www.pucrs.br/edipucrs/online/historiadacomputacao.pdf`

[2] T. A. Diverio, P. B. Menezes, Teoria da Computação: Máquinas Universais e Computabilidade, Sagra Luzzato, 1999.

[3] K. Gödel, On Undecidable Propositions of Formal Mathematical Systems, Lecture notes taken by Kleene and Rosser at the Institute for Advanced Study. Reimpresso em M. Davis (ed.) 1965. The Undecidable. Raven, 1934.

[4] S. C. Kleene, General recursive functions on natural numbers, Mathematische Annalen 112 (1936) 727–742.

[5] T. A. Sudkamp, Languages and Machines: An Introduction to the Theory of Computer Science, 3rd ed., Addison-Wesley Publishing Company, Reading, 2006.

[6] N. J. Cutland, Computability: an introduction to recursive function theory, Cambridge University Press, 1980.

[7] H. R. Lewis, C. H. Papadimitriou, Elementos de Teoria da Computação, 2a. ed., Bookman, Porto Alegre, 2004.

[8] E. L. Post, Recursively enumerable sets of positive integers and their decision problems, Bulletin of the American Mathematical Society 50 (1944) 284–316.

[9] P. Odifreddi, Recursive functions, webpage, acessado em dezembro de 2010 (Março 2005).
URL `http://plato.stanford.edu/entries/recursive-functions/`