Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

Sobre a existência ou inexistência de uma função universal para a classe das funções recursivas primitivas

Victor Pereira GOMES[1]

∞

RESUMO

Embora a classe das funções recursivas primitivas não constitua uma versão formal para a classe de todas as funções algorítmicas, estuda-se esta classe especial de funções numéricas pelo fato de que muitas das funções conhecidas como algorítmicas são recursivas primitivas. Tomando em consideração esta classe de funções, o presente artigo destina-se a solucionar o seguinte problema: existe uma função universal para a classe das funções recursivas primitivas? Se sim, ela é recursiva primitiva? Para apresentar solução para este problema, a argumentação será realizada com base nos manuais de Cooper (2004), Davis (1982), Dias & Weber (2010), Mendelson (2009), Rogers (1987), Soare (1987, 2016), entre outros. Como resultado, é demonstrado, em termos matematicamente precisos, que embora tal função universal exista, ela não é recursiva primitiva.

PALAVRAS-CHAVE: funções algorítmicas. funções recursivas primitivas. função universal.

INTRODUÇÃO

A Teoria da Computabilidade, Teoria da Recursão, ou Teoria das funções recursivas é uma subárea da Lógica Matemática e também da Teoria da Computação. Ela surgiu[2] durante a década de 30 e tem como um dos seus objetivos classificar certos problemas como algoritmicamente *solúveis* ou *insolúveis*. Tais problemas são denominados *problemas de decisão*. Intuitivamente, um problema de decisão é caracterizado como uma questão que busca pela existência ou não-existência de um procedimento mecânico, efetivo, para solucioná-lo, e geralmente é expresso em forma de conjunto, da seguinte forma: dado um conjunto A e um elemento x qualquer, há um *algoritmo* para sempre decidir se $x \in A$ ou $x \notin A$? Tal problema é chamado de problema de decisão para A. Caso a resposta seja positiva, diz-se que A é algoritmicamente solúvel ou *decidível*; caso contrário, diz-se que A é algoritmicamente insolúvel ou *indecidível*.

[1] Doutorando em Filosofia pela Universidade Federal do Rio Grande do Norte. E-mail: victorpereiragomes@yahoo.de

[2] O leitor interessado nos aspectos históricos da Teoria da Computabilidade pode encontra-los em SOARE, Robert. *Turing Computability*: Theory and Applications. Berlin, Heidelberg: Springer-Verlag, 2016.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

Para que fosse possível classificar os problemas de decisão nestas duas categorias, lógicos e matemáticos tiveram, num primeiro momento, a tarefa de formalizar a noção intuitiva de algoritmo, a saber, um conjunto finito (não-vazio) de instruções precisas, não ambíguas, para computar uma função numérica, função cujo domínio e codomínio são subconjuntos de ω[3].

Durante os anos 30, várias versões formais, que constituem modelos de computabilidade, foram apresentadas para a noção intuitiva de algoritmo e função algorítmica, como a teoria das funções recursivas, a teoria das funções λ-computáveis, a teoria das máquinas de Turing (Turing-computabilidade), entre outras. Diante deste fato, é importante destacar que cada uma destas versões formais foi obtida de forma independente através dos trabalhos de lógicos e matemáticos do calibre de Gödel & Herbrand, Kleene, Church, Turing, entre outros, e que embora cada versão formal seja conceitualmente distinta uma das outras, elas acabaram por produzir uma mesma classe[4] de funções, a saber, funções algorítmicas, isto é, funções cujos valores podem ser calculados algoritmicamente, o que é em parte argumento para a tese de Church-Turing, que afirma que toda função algorítmica é Turing-computável, e vice-versa. Assim, acredita-se que os trabalhos destes lógicos e matemáticos foram decisivos para capturar formalmente, de uma vez por todas, a noção intuitiva de algoritmo e função algorítmica.

Um modelo de computabilidade que não contempla todas as funções algorítmicas é a teoria das funções recursivas primitivas, a qual foi inicialmente desenvolvida por Gödel & Herbrand, sendo depois estudada à fundo e aprimorada por Kleene. Embora ela não contemple todas as funções algorítmicas, esta classe especial de funções numéricas ainda é estudada e tem sua importância para a Lógica Matemática, uma vez que foi através dela que Gödel demonstrou seus dois famosos teoremas de incompletude. Tomando em consideração esta classe de funções, este artigo[5] tem como objetivo apresentar solução para o seguinte problema: existe uma função universal para a classe das funções recursivas primitivas? Se sim, ela é recursiva primitiva?

Para que seja possível apresentar solução para este problema, faz-se necessário dividir o texto em três partes. Na primeira parte será exibida uma versão formal para a noção intuitiva de algoritmo e função algorítmica, a saber, a teoria das máquinas de Turing. Na segunda parte, será exibida algumas noções básicas acerca da classe das funções recursivas primitivas e também será demonstrado que ela constitui uma subclasse das funções Turing-computáveis. Isto feito, a terceira e última parte será reservada para apresentar o resultado

[3] Conjunto dos números naturais.
[4] A palavra classe é empregada aqui como sinônimo de conjunto.
[5] Este artigo é um recorte da minha dissertação de mestrado: GOMES, Victor P. *Funções recursivas primitivas*: caracterização e alguns resultados para esta classe de funções. 2016. 55 f. Dissertação (Mestrado em Filosofia) - Universidade Federal da Paraíba, João Pessoa, 2016.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

obtido para o problema supracitado. Embora esta questão já tenha sido por demais trabalhada, a abordagem que se seguirá será feita de forma diferente da encontrada nos manuais pesquisados na literatura especializada.

Além do objetivo traçado acima, o presente artigo também objetiva contribuir para uma maior divulgação, ou até mesmo ampliação das discussões acerca da Teoria da Computabilidade, cujas aplicações em áreas como Filosofia da Mente, Inteligência Artificial, entre outras, são de grande interesse até mesmo para não-matemáticos.

1 TURING-COMPUTABILIDADE: MÁQUINAS OU PROGRAMAS DE TURING

Nesta seção será apresentada[6] a teoria das máquinas de Turing, a qual, juntamente com a teoria das funções recursivas, se tornou um dos mais estudados modelos de computabilidade, e é tomada como modelo *standard* em vários manuais especializados.

Uma máquina de Turing (*a-machine*, máquina automática) é um mecanismo teórico composto por um cabeçote de leitura/escrita, e por uma fita infinita em sentido horizontal, tanto para a direita quanto para a esquerda, dividida em quadrados de tamanhos iguais, que são examinados individualmente pelo cabeçote, como mostra a figura abaixo.

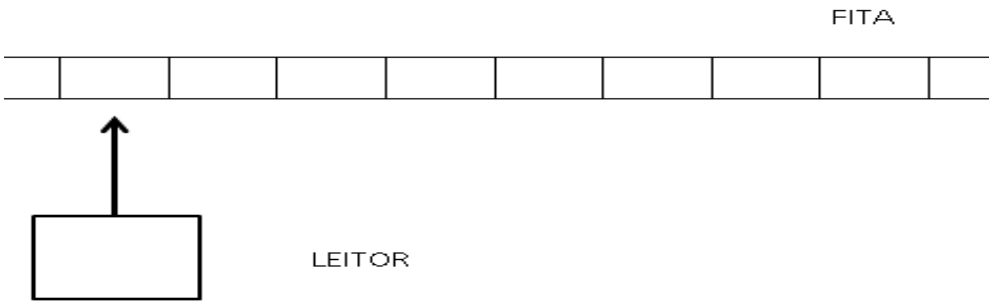

Figura 1 - Máquina de Turing[7]

Quando o cabeçote examina um quadrado, somente um dos elementos do conjunto alfabeto $A = \{|, B\}$ chamados de *símbolos da fita* pode estar escrito, e a

[6] A apresentação aqui realizada acerca da teoria das máquinas de Turing segue de perto (com algumas modificações e definições adicionais) a apresentação encontrada em DIAS, Matias Francisco; WEBER, Leonardo. *Teoria da recursão*. São Paulo: Editora UNESP, 2010.

[7] Figura retirada do site <https://numeroimaginario.wordpress.com/2015/12/27/teoria-da-recursao-maquinas-de-turing-e-computabilidade-de-funcoes/>

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

máquina sempre se encontra em um dos elementos do conjunto de estados internos $Q = \{q_0, q_1, q_2, \dots\}$ chamados de *símbolos de estados internos*. Como o cabeçote de leitura/escrita examina apenas um quadrado por vez, a ação da máquina é determinada pelo símbolo que está sendo examinado e pelo estado interno em que a máquina se encontra. Isto posto, a máquina só pode realizar uma das seguintes ações:

1. Escrever o símbolo | no quadrado que está sendo examinado, se nenhum | já estiver escrito em tal quadrado;
2. Escrever o símbolo B (*blank*), se tal quadrado já não estiver em branco;
3. Mover-se para o quadrado imediatamente à direita do quadrado previamente examinado (deslocando a fita um quadrado à esquerda);
4. Mover-se para o quadrado imediatamente à esquerda do quadrado previamente examinado (deslocando a fita um quadrado à direita);
5. Mudar de um estado interno para outro.

Utiliza-se o símbolo R (de *right*) para indicar que a máquina passa a examinar o quadrado imediatamente à direita do anteriormente examinado, e o símbolo L (de *left*) para indicar que a máquina passa a examinar o quadrado imediatamente à esquerda do anteriormente examinado. Tais símbolos são elementos do conjunto de movimento $M = \{R, L\}$ e são chamados de *símbolos de movimento*.

Para que a máquina possa executar tais ações é necessário que ela seja guiada por um algoritmo, chamado de *programa de Turing*, que será apresentado a seguir.

Definição 1.1: A linguagem utilizada para criar programas de Turing é composta pelos elementos dos conjuntos $A \cup Q \cup M$, que serão chamados de *símbolos de programas de Turing*.

Definição 1.2: Uma *expressão* da linguagem de programas é uma sequência finita de símbolos de programas de Turing.

Definição 1.3: Uma *quádrupla* é uma expressão de um dos seguintes tipos:

1. $q_i s_j s_k q_l$
2. $q_i s_j R q_l$
3. $q_i s_j L q_l$

onde $i \geq 0$; $l \geq 0$; $j = 0{,}1$; $k = 0{,}1$; $s_0 = B$ e $s_1 = |$.

Observação 1.1: Com relação à primeira quádrupla, quando a máquina se encontra no estado interno q_i examinando o quadrado cujo símbolo escrito é s_j, ela apaga s_j, escreve s_k em seu lugar e assume o estado interno q_l. Já de acordo com a segunda quádrupla, quando a máquina se encontra no estado interno q_i

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

examinando o quadrado cujo símbolo escrito é s_j, ela move-se para o quadrado imediatamente à direita do anteriormente examinado e assume o estado interno q_l. Por fim, no que concerne à terceira quádrupla, quando a máquina se encontra no estado interno q_i examinando o quadrado cujo símbolo escrito é s_j, ela move-se para o quadrado imediatamente à esquerda do anteriormente examinado e assume o estado interno q_l.

Definição 1.4: Duas quádruplas são ditas *inconsistentes* se ambas têm os dois primeiros símbolos iguais, e pelo menos um dos dois símbolos restantes diferentes. Caso contrário, são chamadas consistentes.

Definição 1.5: Um *programa de Turing* é um conjunto finito (não-vazio) de quádruplas consistentes. Tal restrição, adotada para evitar comandos contraditórios, é chamada de *restrição de consistência*.

Observação 1.2: Se um programa de Turing T possuísse as quádruplas $q_0|Rq_1$ e $q_0|Lq_3$, ambas seriam inconsistentes e, assim, a máquina não executaria ação alguma, por não saber como reagir diante de tais comandos.

Definição 1.6: Uma *descrição instantânea* é uma expressão da linguagem de programas de Turing que possui um único q_i, não possui símbolos de movimento, e q_i não pode ser o último símbolo à direita.

Definição 1.7: Sendo t_1 e t_2 sequências finitas de símbolos da fita, dado um programa de Turing T e dadas descrições instantâneas α e β, diz-se que $\alpha \rightarrow \beta$ (T) ["α acarreta β via T"] se uma das seguintes condições vale:

1. $q_i s_j s_k q_l \in T$, $\alpha = t_1 q_i s_j t_2$, $\beta = t_1 q_l s_k t_2$;
2. $q_i s_j R q_l \in T$, $\alpha = t_1 q_i s_j s_k t_2$, $\beta = t_1 s_j q_l s_k t_2$, ou $\alpha = t_1 q_i s_j$, $\beta = t_1 s_j q_l B$;
3. $q_i s_j L q_l \in T$, $\alpha = t_1 s_k q_i s_j t_2$, $\beta = t_1 q_l s_k s_j t_2$, ou $\alpha = q_i s_j t_2$, $\beta = q_l B s_j t_2$.

Nota 1.1: Note-se que no caso 2, o símbolo B é inserido após q_l na segunda descrição instantânea β, pois, de acordo com a definição 1.6, q_l não pode ser o último símbolo à direita em uma descrição instantânea. Já no caso 3, o símbolo B é inserido após q_l na segunda descrição instantânea β, pois o quadrado que seria examinado pela máquina no estado interno q_l está vazio.

Definição 1.8: Dado um programa de Turing T, diz-se que α é uma *descrição instantânea inicial* (*input*) se $\alpha = q_0|t_1$.

Definição 1.9: Dado um programa de Turing T, diz-se que α é uma *descrição instantânea terminal* (*output*) se $\alpha = t_1 q_i s_j t_2$ e T não possui quádruplas começando com $q_i s_j$. Se isso ocorre, a máquina para.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

No que se segue, algumas considerações notacionais serão apresentadas, a fim de que uma máquina de Turing possa realizar computações numéricas. São elas:

1. No que concerne ao *input*, para representar um número natural n, escreve-se $|$ em $n+1$ quadrados adjacentes;
2. No que concerne ao *output*, um número natural n é representado por n $|$'s não necessariamente adjacentes;
3. Representa-se $n+1$ $|$'s por $|^{n+1}$.
4. Para representar uma n-upla $k_1, \dots, k_n$, escreve-se $|^{k_1+1}B \dots B|^{k_n+1}$.

Definição 1.10: Uma computação de acordo com um programa de Turing T é uma sequência finita $\alpha_1, \dots, \alpha_m$ de descrições instantâneas, tal que α_1 é inicial, α_m é terminal e $\alpha_i \rightarrow \alpha_{i+1}(T)$, para todo i, tal que $1 \leq i < m$. Nesse caso, diz-se que $Res_T(\alpha_1) = \alpha_m$, isto é, α_m é a resultante de α_1 com respeito a T.

Exemplo 1.1: Seja $S = \{q_0|Bq_1,\ q_1BRq_2,\ q_2|Rq_2,\ q_2BRq_0\}$ e $\alpha_1, \dots, \alpha_9$ a seguinte sequência finita de descrições instantâneas:

α_1. $q_0||B||$ (*input*)

α_2. $q_1B|B||$

α_3. $Bq_2|B||$

α_4. $B|q_2B||$

α_5. $B|Bq_0||$

α_6. $B|Bq_1B|$

α_7. $B|BBq_2|$

α_8. $B|BB|q_2B$

α_9. $B|BB|Bq_0B$ (*output*)

A sequência de descrições instantâneas apresentada acima é uma computação obtida através do programa de Turing S[8], aplicado ao *input* α_1. Assim, diz-se que $Res_S(\alpha_1) = \alpha_9$, isto é, $Res_S(q_0||B||) = B|BB|Bq_0B$.

Observação 1.3: De acordo com a definição 1.8, toda computação será iniciada com um *input* da forma $q_0|t_1$.

Definição 1.11: Seja T um programa de Turing. Para cada $n \geq 1$, uma única função n-ária $\psi_T^n(x_1, \dots, x_n)$ está associada a T da seguinte forma: para cada n-upla $(k_1, \dots, k_n)$, faz-se $\alpha_1 = q_0|^{k_1+1}B \dots B|^{k_n+1}$ e distingue-se dois casos:

[8] Note-se que S é o programa de Turing para a função soma $(x+y)$.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

1. Há uma computação $\alpha_1, \dots, \alpha_m$ de acordo com T e, neste caso, diz-se que $\psi_T^n(k_1, \dots, k_n) = (\alpha_m) = \big(Res_T(\alpha_1)\big)$, onde (α_m) representa o número de traços verticais no *output* α_m;
2. Não há uma computação $\alpha_1, \dots, \alpha_m$ de acordo com T e, neste caso, $\psi_T^n(k_1, \dots, k_n)$ fica indefinida.

Definição 1.12: Uma função numérica n-ária $f(x_1, \dots, x_n)$ é chamada *parcialmente Turing-computável*, se existe um programa de Turing T tal que $f(x_1, \dots, x_n) = \psi_T^n(x_1, \dots, x_n)$. Não obstante, se $f(x_1, \dots, x_n)$ é total, isto é, se ela está definida para todas as n-uplas dos números naturais, diz-se que ela é Turing-computável.

Observação 1.4: Intuitivamente, pode-se afirmar que uma função é (parcialmente) Turing-computável, se existe uma máquina de Turing para computar seus valores.

De um ponto de vista formal, falar-se-ia apenas em máquinas de Turing, uma vez que máquinas e programas denotam a mesma coisa. Assim, a distinção intuitiva entre máquinas e programas aqui empregada, tem como objetivo tornar a exposição mais clara.

2 FUNÇÕES RECURSIVAS PRIMITIVAS

A classe das funções recursivas primitivas (inicialmente nomeada por Gödel como *rekursiv*[9]) é uma classe especial de funções numéricas de grande importância para a Lógica Matemática e a Ciência da Computação. Embora ela não capture[10] todas as funções algorítmicas, essa classe de funções é estudada por dois motivos: (a) foi através delas que Gödel provou seus dois teoremas da incompletude; e (b) Kleene, no livro *Introduction to Metamathematics*, demonstrou que muitas das funções conhecidas como algorítmicas (soma, multiplicação, exponenciação, entre outras) são, de fato, recursivas primitivas. No que se segue, serão exibidas algumas noções[11] básicas acerca desta classe de funções.

Definição 2.1: As seguintes funções são chamadas *funções iniciais*:

---

[9] Do alemão recursivo.

[10] Este fato deve-se à existência de funções que embora sejam algorítmicas, não são recursivas primitivas. Pode-se citar como exemplo a função exponencial generalizada de Ackermann. Uma definição informal dessa função pode ser encontrada em ROGERS, Hartley. *Theory of Recursive Functions and Effective Computability*. Cambridge, Massachusetts, London: MIT Press, 1987. p. 8.

[11] Uma exposição mais detalhada sobre a classe das funções recursivas primitivas pode ser encontrada em MENDELSON, E. *Introduction to Mathematical Logic*. 5. ed. New York: Chapman and Hall/CRC, 2009, pp. 171-188.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

1. A função nulo, $N(x) = 0$ para todo x;
2. A função sucessor, $S(x) = x + 1$ para todo x;
3. A função constante, $C_n(x) = n$, com $n \geq 0$, para todo n;
4. A função projeção, $U_i^n(x_1, \dots, x_n) = x_i$, com $1 \leq i \leq n$, para todo $x_1, \dots, x_n$.

Definição 2.2: As seguintes operações são chamadas *operações básicas* entre funções:

1. A função f (de n variáveis, $n \geq 1$) é obtida por *composição*, da função h (de m variáveis, $m \geq 1$) e das funções $g_1, \dots, g_m$ (todas de n variáveis) se, e somente se:

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

2. A função f (de $n + 1$ variáveis, $n \geq 0$) é obtida por *recursão primitiva*, da função g (de n variáveis) e da função h (de $n + 2$ variáveis) se, e somente se:

$$f(x_1, \dots, x_n, 0) = g(x_1, \dots, x_n)$$
$$f\big(x_1, \dots, x_n, S(y)\big) = h\big(x_1, \dots, x_n, y, f(x_1, \dots, x_n, y)\big)$$

Observação 2.1: Note-se que quando $n = 0$, a função f será obtida por recursão primitiva somente pela função h da seguinte forma:

$$f(0) = k$$
$$f\big(S(y)\big) = h\big(y, f(y)\big)$$

Nota 2.1: k é uma constante individual.

Definição 2.3: A classe das funções recursivas primitivas é a menor classe de funções numéricas totais que contém as funções iniciais e é *fechada* com respeito às operações de composição e recursão primitiva.

Definição 2.4: Uma função f é recursiva primitiva se, e somente se, existe uma sequência finita de funções $(f_1, \dots, f_n)$ tal que $f_n = f$ e, para $1 \leq i \leq n$, ou f_i é uma função inicial, ou é obtida de funções anteriores da sequência por aplicação de pelo menos uma das operações básicas.

Dada uma sequência finita de funções para f, é possível especificar como cada função da sequência foi obtida e, portanto, reconhecer se tal sequência é recursiva primitiva. Dessa forma, ao especificar como cada função da sequência é obtida, obtém-se uma derivação recursiva primitiva para f. Como consequência define-se o seguinte:

Definição 2.5: Uma função f é recursiva primitiva se, e somente se, existe uma derivação recursiva primitiva para ela.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

Para exemplificar a definição acima, uma derivação recursiva primitiva para a função soma $(x + y)$ será exibida abaixo.

Definição intuitiva:

$x + 0 = x$
$x + (y + 1) = (x + y) + 1$

Derivação:

1. $S(x) = x + 1$ função inicial
2. $U_3^3(x, y, z) = z$ função inicial
3. $g_0(x, y, z) = S\big(U_3^3(x, y, z)\big)$ 1,2 / composição
4. $U_1^1(x) = x$ função inicial
5. $g_1(x, 0) = U_1^1(x)$
   $g_1\big(x, S(y)\big) = g_0\big(x, y, g_1(x, y)\big)$ 4,3 / recursão primitiva

Uma derivação recursiva primitiva para uma função f fornece um algoritmo para computá-la. Assim, a computação é realizada a partir das funções mais inferiores para as mais superiores, no sentido das funções mais abaixo para as funções mais acima; e parte-se das funções mais interiores para as mais exteriores, no sentido das funções à direita para as funções à esquerda.

No que se segue, será exibido como uma computação para a função soma $g_1(2,2)$ é obtida, a partir de sua derivação recursiva primitiva, apresentada acima.

$$
\begin{aligned}
g_1(2,2) &= g_0\big(2,1, g_1(2,1)\big) \\
&= g_0\Big(2,1, g_0\big(2,0, g_1(2,0)\big)\Big) \\
&= g_0\Big(2,1, g_0\big(2,0, U_1^1(2)\big)\Big) \\
&= g_0\big(2,1, g_0(2,0,2)\big) \\
&= g_0\Big(2,1, S\big(U_3^3(2,0,2)\big)\Big) \\
&= g_0\big(2,0, S(2)\big) \\
&= g_0(2,0,3) \\
&= S\big(U_3^3(2,0,3)\big) \\
&= S(3) \\
&= 4
\end{aligned}
$$

Teorema 2.1: As seguintes funções são recursivas primitivas: a) multiplicação; b) exponenciação; c) predecessor; d) subtração própria; e) diferença absoluta; f) sinal; g) contrassinal; h) fatorial; i) mínimo de um par ordenado; j) mínimo de uma sequência finita; k) máximo de um par ordenado; l) máximo de uma sequência finita; m) resto da divisão de y por x; e n) quociente da divisão de y por x.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

Prova:

Devido à extensão do teorema, será exibida aqui apenas a prova para o item a)[12].

a) Multiplicação:

Definição intuitiva:

$x.0 = 0$
$x.(y+1) = (x.y) + x$

Derivação:

1. $N(x) = 0$ função inicial
2. $U_1^3(x,y,z) = x$ função inicial
3. $U_3^3(x,y,z) = z$ função inicial
4. $g_1(x,0) = U_1^1(x)$
   $g_1\big(x,S(y)\big) = g_0\big(x,y,g_1(x,y)\big)$ função recursiva primitiva [soma]
5. $g_2(x,y,z) = g_1\big(U_1^3(x,y,z), U_3^3(x,y,z)\big)$ 4,2,3 / composição
6. $g_3(x,0) = N(x)$
   $g_3\big(x,S(y)\big) = g_2\big(x,y,g_3(x,y)\big)$ 1,5 / recursão primitiva ■

Definição 2.6 (*função característica de um conjunto*): Seja A um conjunto arbitrário, x um elemento qualquer, e $\chi_A(x)$ a *função característica* de A. Dessa forma, tem-se que:

$$\chi_A(x) = \begin{cases} 1, se\ x \in A \\ 0, se\ x \notin A \end{cases}$$

Como a Teoria da Computabilidade Clássica trabalha apenas com funções numéricas, define-se que uma relação numérica n-ária $R(x_1, \dots, x_n)$, com $n \geq 1$, é um subconjunto de ω^n.

Definição 2.7: Uma relação numérica $R(x_1, \dots, x_n)$ é recursiva primitiva se, e somente se, sua função característica $\chi_R(x_1, \dots, x_n)$ é recursiva primitiva.

A partir das noções e definições apresentadas até então, é fácil perceber que a classe das funções recursivas primitivas constitui uma subclasse das funções Turing-computáveis. De fato, como para toda função recursiva primitiva existe uma derivação recursiva primitiva, isto é, um algoritmo para computar seus valores, pode-se afirmar que toda função recursiva primitiva é algorítmica. Logo, pela tese de Church-Turing, as funções recursivas primitivas são parcialmente Turing-computáveis. Contudo, como toda função recursiva primitiva é total, ela é Turing-computável.

[12] A prova para o restante das funções pode ser encontrada em DIAS, Matias Francisco; WEBER, Leonardo. *Teoria da recursão*. São Paulo: Ed. UNESP, 2010, pp. 31-35.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

Observação 2.2: Como mencionado no início desta seção, a classe das funções recursivas primitivas não captura todas as funções algorítmicas. Isto posto, é importante destacar que embora toda função recursiva primitiva seja algorítmica e, portanto, via tese de Church-Turing, seja Turing-computável, o contrário não acontece, isto é, nem toda função algorítmica (Turing-computável) é recursiva primitiva.

3 FUNÇÃO UNIVERSAL PARA A CLASSE DAS FUNÇÕES RECURSIVAS PRIMITIVAS

Como se sabe, não existe uma função universal para a classe das funções Turing-computáveis, uma vez que seus elementos não podem ser algoritmicamente enumerados. Entretanto, é possível definir uma função universal para uma subclasse desta classe supracitada, a saber, a classe das funções recursivas primitivas. Para que isto seja possível, faz-se necessário provar que embora não exista um algoritmo reconhecedor para a classe de todos os algoritmos, isto é, não existe um algoritmo para sempre decidir, dado um conjunto finito (não-vazio) de instruções precisas, não-ambíguas, se este conjunto pertence ou não à classe de todos os algoritmos, existe um algoritmo reconhecedor para uma subclasse desta classe supracitada, a saber, a classe das derivações recursivas primitivas.

Teorema 3.1: Existe um algoritmo reconhecedor para a classe das derivações recursivas primitivas.

Prova intuitiva:

Seja $DRP = \{x: x \text{ é } o\ número\ de\ Gödel\ de\ uma\ derivação\ recursiva\ primitiva\}$

Sua função característica é a seguinte:

$$\chi_{DRP}(x) = \begin{cases} 1, se\ x \in DRP \\ 0, se\ x \notin DRP \end{cases}$$

Esta função característica é algorítmica. De fato, dado um número natural x, é possível reconhecer, a partir da decodificação[13] de x, se x é número de Gödel de uma derivação recursiva primitiva.■

Nota 3.1: A numeração de Gödel, também conhecida como aritmetização ou codificação, foi inicialmente utilizada para traduzir os enunciados metalinguísticos da aritmética elementar de primeira ordem de Peano na linguagem-objeto da aritmética, cujo domínio é constituído de números naturais. De forma análoga à aritmética de Peano, a numeração de Gödel também pode ser

[13] Cf. DIAS, Matias Francisco; WEBER, Leonardo. *Teoria da recursão*. São Paulo: Ed. UNESP, 2010. p. 47.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

aplicada a qualquer linguagem formal, atribuindo números naturais aos seus componentes básicos, de forma a codificar toda a teoria. O leitor poderá encontrar uma exposição detalhada acerca da numeração de Gödel para a classe das funções e derivações recursivas primitivas em DIAS, Matias Francisco; WEBER, Leonardo. *Teoria da recursão*. São Paulo: Ed. UNESP, 2010. pp. 45-47.

Com base no algoritmo reconhecedor para a classe das derivações recursivas primitivas, é possível listar efetivamente todas as derivações recursivas primitivas, definindo a seguinte função:

$$f(0) = \overset{\mu}{y}(\chi_{DRP}(y) = 1)$$
$$f(n+1) = \overset{\mu}{y}\big(\chi_{DRP}(y) = 1 \wedge y > f(n)\big)$$

Observação 3.1: O símbolo μ é aqui empregado significando "o menor". No caso da função acima, lê-se "o menor y".

Com relação à função acima, $f(0)$ fornece o menor número natural que é número de Gödel de uma derivação recursiva primitiva e, portanto, a primeira derivação recursiva primitiva da lista. $f(1)$ fornece a segunda derivação recursiva primitiva da lista. E assim sucessivamente. Embora esta lista seja infinita, cada derivação recursiva primitiva pode ser encontrada em um ponto finito da mesma.

No que se segue será demonstrado que é possível definir uma função universal para a classe das funções recursivas primitivas. Por uma função universal para uma classe de funções computáveis C, entende-se uma função que, estando associada à máquina de Turing universal[14] (capaz de simular qualquer máquina de Turing), enumera e computa todas as funções de C. Além disso, também será demonstrado que embora essa função nos permita listar e computar todas as funções recursivas primitivas, ela não é recursiva primitiva.

Definição 3.1: Uma função universal U para uma classe C de funções computáveis n-árias, com $(n \geq 1)$, é uma função $1+n$-ária que enumera as funções de C e seus argumentos. Assim, tem-se que:

$$U(x, y_1, \dots, y_n) = \begin{cases} \varphi_x(y_1, \dots, y_n), & se\ \varphi_x(y_1, \dots, y_n) \downarrow \\ \uparrow, & se\ \varphi_x(y_1, \dots, y_n) \uparrow \end{cases}$$

Observação 3.2: φ_x é a função cuja derivação tem o número natural x como número de Gödel.

Observação 3.3: $\downarrow$ e $\uparrow$ simbolizam respectivamente que a função está definida e indefinida.

[14] Uma definição formal de máquina de Turing universal pode ser encontrada em DAVIS, Martin. *Computability and Unsolvability*. Dover: New York, 1982. pp. 64-65.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

Definição 3.2: U é definida para a classe das funções recursivas primitivas n-árias, da seguinte forma:

$$U(f(x), y_1, \dots, y_n) = \begin{cases} \varphi_{f(x)}(y_1, \dots, y_n), & se\ \varphi_{f(x)}(y_1, \dots, y_n) \downarrow \\ \uparrow, & se\ \varphi_{f(x)}(y_1, \dots, y_n) \uparrow \end{cases}$$

De posse dessa função universal, é possível listar e computar efetivamente todas as funções recursivas primitivas n-árias, como segue:

1. Após listar efetivamente todas as derivações recursivas primitivas (com o auxílio do algoritmo exibido no teorema 3.1), pode-se listar efetivamente todas as funções recursivas primitivas n-árias, da seguinte forma:

$$\varphi_{f(0)}(y_1, \dots, y_{n-1}, 0), \varphi_{f(0)}(y_1, \dots, y_{n-1}, 1), \varphi_{f(0)}(y_1, \dots, y_{n-1}, 2) \cdots$$
$$\varphi_{f(1)}(y_1, \dots, y_{n-1}, 0), \varphi_{f(1)}(y_1, \dots, y_{n-1}, 1), \varphi_{f(1)}(y_1, \dots, y_{n-1}, 2) \cdots$$
$$\varphi_{f(2)}(y_1, \dots, y_{n-1}, 0), \varphi_{f(2)}(y_1, \dots, y_{n-1}, 1), \varphi_{f(2)}(y_1, \dots, y_{n-1}, 2) \cdots$$
$$\vdots \qquad\qquad \vdots \qquad\qquad \vdots \qquad\qquad \ddots$$

Observação 3.4: Percebe-se que $\varphi_{f(x)}(y_1, \dots, y_{n-1}, z)$ é a função recursiva primitiva n-ária cuja derivação tem $f(x)$ como número de Gödel e $(y_1, \dots, y_{n-1}, z)$ como argumentos. Os argumentos $(y_1, \dots, y_{n-1})$ são fixados aqui como parâmetros.

2. U computa todas as funções recursivas primitivas n-árias da seguinte forma:

Como exibido acima, $U(f(x), y_1, \dots, y_n) = \varphi_{f(x)}(y_1, \dots, y_n)$. Dessa forma, se U tem como argumentos $(f(x), y_1, \dots, y_n)$, o que se tem de fazer é encontrar a $f(x)$-ésima derivação recursiva primitiva e aplicá-la aos argumentos de φ, a saber, $(y_1, \dots, y_n)$. Quando, e somente quando, $\varphi_{f(x)}(y_1, \dots, y_n)$ tem um valor, atribua esse valor a $U(f(x), y_1, \dots, y_n)$. Entretanto, como toda função recursiva primitiva é total, sempre haverá um valor para $U(f(x), y_1, \dots, y_n)$.

Observação 3.5: É possível reduzir todas as funções recursivas primitivas n-árias, com $(n \geq 2)$, a funções recursivas primitivas de uma variável, através da função J de Cantor[15], que possibilita atribuir números de Gödel à n-uplas.

Como visto na seção anterior, nem toda função algorítmica é recursiva primitiva (a função de Ackermann é um exemplo deste fato). Assim, nas linhas que se seguem será demonstrado um teorema que fornece um outro exemplo de função algorítmica que não é recursiva primitiva, a saber, a função universal para a classe das funções recursivas primitivas, definida acima.

[15] Uma exposição detalhada acerca desta função, pode ser encontrada em DIAS, Matias Francisco e WEBER, Leonardo. *Teoria da recursão*. São Paulo: Ed. UNESP, 2010, p.134.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

Teorema 3.2: A função universal definida para a classe das funções recursivas primitivas não é recursiva primitiva.

Prova:

Com base nos resultados anteriores, e levando em consideração a possibilidade de reduzir todas as funções recursivas primitivas n-árias a funções recursivas primitivas de uma variável, é possível, após a listagem efetiva de todas as derivações recursivas primitivas, listar todas as funções recursivas primitivas de uma variável da seguinte forma:

Defina $U(f(x), y) = \varphi_{f(x)}(y)$, a fim de obter a seguinte listagem:

$$\varphi_{f(0)}(0), \varphi_{f(0)}(1), \varphi_{f(0)}(2) \cdots$$
$$\varphi_{f(1)}(0), \varphi_{f(1)}(1), \varphi_{f(1)}(2) \cdots$$
$$\varphi_{f(2)}(0), \varphi_{f(2)}(1), \varphi_{f(2)}(2) \cdots$$
$$\vdots \qquad \vdots \qquad \vdots \qquad \ddots$$

De posse desta lista com todas as funções recursivas primitivas de uma variável, pode-se agora demonstrar por diagonalização – método que pode ser aplicado a uma variedade de classes formalmente caracterizadas de funções, e que, em cada caso, a partir do traçado da diagonal a uma classe previamente dada, produz uma função que não pertence à classe – que a função universal para a classe das funções recursivas primitivas não é recursiva primitiva.

Hipótese: Se $U(f(x), y) = \varphi_{f(x)}(y)$ é recursiva primitiva, então $U(f(x), x) = \varphi_{f(x)}(x) = h(x)$ é recursiva primitiva por composição.

Defina a seguinte função $g(x) = h(x) + 1 = \varphi_{f(x)}(x) + 1$. É fácil perceber que g é algorítmica. De fato, basta encontrar $\varphi_{f(x)}(x)$ na lista de todas as funções recursivas primitivas, e somar 1 ao seu valor. Agora, suponha que $g(x)$ é uma função recursiva primitiva de uma variável. Se isto ocorre, g é uma das funções da lista, isto é, $g = \varphi_{f(k)}$ para algum k. Portanto, $g(k) = \varphi_{f(k)}(k)$. Mas por definição, $g(k) = \varphi_{f(k)}(k) + 1$. Portanto, tem-se que, $\varphi_{f(k)}(k) = \varphi_{f(k)}(k) + 1$, o que leva a uma contradição. Dessa forma, conclui-se que, embora g seja algorítmica, ela não está na lista e, portanto, ela não é recursiva primitiva.■

Note-se que se $U(f(x), y)$ fosse recursiva primitiva, então $U(f(x), x)$ seria recursiva primitiva e, portanto, $U(f(x), x) + 1$ seria recursiva primitiva. Entretanto, tem-se que $U(f(x), x) + 1 = \varphi_{f(x)}(x) + 1 = g(x)$, que não é recursiva primitiva.

CONCLUSÃO

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

A partir da análise realizada acerca da classe das funções recursivas primitivas, conclui-se o seguinte: 1) que ela constitui uma subclasse das funções Turing-computáveis; 2) que existe um algoritmo reconhecedor para a classe das derivações recursivas primitivas; 3) que existe uma função universal que enumera e computa todas as funções recursivas primitivas n-árias ($n \geq 1$); 4) que esta função universal não é recursiva primitiva. Embora estas questões já tenham sido por demais discutidas, a abordagem e os resultados aqui apresentados diferem da forma como são expostos nos manuais pesquisados na literatura especializada. Esta análise acerca das funções recursivas primitivas possibilita também uma melhor compreensão da classe das funções algorítmicas, no sentido de que a primeira possui um algoritmo reconhecedor e uma função universal, características que a segunda não possui. De toda forma, a análise aqui realizada favorece a uma compreensão mais acurada da classe das funções algorítmicas.

On the existence or nonexistence of a universal function for the class of primitive recursive functions

ABSTRACT

Although the class of primitive recursive functions does not constitute a formal version for the class of all algorithmic functions, this special class of numerical functions is studied by the fact that many of the functions known as algorithmic are primitive recursive. Taking this class of functions into consideration, this article is intended to solve the following problem: is there a universal function for the class of primitive recursive functions? If so, is this function primitive recursive? To present a solution to this problem, the discussion will be based on the books of Cooper (2004), Davis (1982), Dias & Weber (2010), Rogers (1987), Soare (1987, 2016), among others. As a result, it is shown, in mathematically precise terms, that although such a universal function exists, it is not primitive recursive.

KEYWORDS**:** Algorithmic functions. Primitive recursive functions. Universal function.

REFERÊNCIAS

COOPER, S. Barry. **Computability Theory**. New York: Chapman and Hall/CRC, 2004.

DAVIS, Martin. **Computability and Unsolvability**. Dover: New York, 1982.

DIAS, Matias Francisco; WEBER, Leonardo**. Teoria da recursão**. São Paulo: Editora UNESP, 2010.

Revista Instante, Campina Grande-PB, Brasil, v.1, n.2, p.76-91, 2018.

EPSTEIN, Richard L.; CARNIELLI, Walter A.. **Computability: Computable Functions, Logic, and the Foundations of Mathematics**. Wadsworth & Brooks/Cole Advanced Books & Software: Pacific Grove, California, 1989.

GOMES, Victor P.. **Funções recursivas primitivas: caracterização e alguns resultados para esta classe de funções**. 2016. 55 f. Dissertação (Mestrado em Filosofia) - Universidade Federal da Paraíba, João Pessoa, 2016.

KLEENE, Stephen C.. **Introduction to Metamathematics**. Ishi Press: New York and Tokyo, 2009.

MENDELSON, Elliott. **Introduction to Mathematical Logic**. 5. ed. New York: Chapman and Hall/CRC, 2009.

MONK, J. Donald. **Mathematical Logic**. New York, Heidelberg, Berlin: Springer-Verlag, 1976.

ODIFREDDI, Piergiorgio. **Classical Recursion Theory: The Theory of Functions and Sets of Natural Numbers**, I. North-Holland: Amsterdam, New York, 1989.

ROGERS, Hartley. **Theory of Recursive Functions and Effective Computability**. Cambridge, Massachusetts, London: MIT Press, 1987.

SOARE, Robert**. Recursively Enumerable Sets and Degrees: A Study of Computable Functions and Computably Generated Sets**. Berlin, Heidelberg, New York: Springer-Verlag, 1987.

SOARE, Robert. **Turing Computability: Theory and Applications**. Berlin, Heidelberg: Springer-Verlag, 2016.