

Capítulo 3

Funções Recursivas

Neste capítulo será estudada uma classe especial de funções, as *funções recursivas* que têm como conjuntos de partida \mathbb{N}^n para algum $n > 0$ e como conjunto de chegada \mathbb{N} . A motivação para esta classe de funções é de capturar a intuição sobre funções computáveis. Estas funções serão apresentadas de maneira abstrata e informal, como é usual em matemática.

A partir das funções recursivas serão apresentados certos conceitos matemáticos básicos para o estudo de computabilidade, relacionados aos domínios das funções. Serão abordados os *sistemas de representação*, maneiras de fazer corresponder termos abstratos a *representações simbólicas*. Também será demonstrada a independência entre a propriedade “computável” de uma função e o sistema de representação adotado para o conjunto de partida e chegada, tanto com respeito ao conjunto de símbolos quanto à dimensão.

Sob esta ótica, observe que no início do século XX, a crença na possibilidade de resolução de qualquer problema matemático era amplamente aceita, principalmente devido ao matemático David Hilbert. Sua ambição de *mecanizar* o tratamento de problemas matemáticos contribuiu fortemente para a reformulação da matemática, na direção de sistemas axiomáticos, nos moldes da geometria Euclidiana.

Entretanto, em 1931 Kurt Gödel publicou o resultado da *incompletude da aritmética* estabelecendo a não existência de procedimentos efetivos para julgar a

veracidade ou falsidade de asserções (potenciais teoremas) a respeito da teoria dos números naturais. Na demonstração deste resultado, tão fundamental para a matemática, Gödel utilizou funções cujo cálculo é feito através de procedimentos finitos, com passos bem determinados. Gödel e Jacques Herbrand definiram um conjunto de funções, hoje conhecido como *conjunto das funções recursivas*, que capturam a noção de cálculo ou computação finita.

Em 1936, Alonzo Church demonstrou a equivalência entre o conjunto de funções definido por Gödel e Herbrand, e uma outra caracterização de procedimento efetivo devida a Church e Stephen Kleene, o λ -calculus. Esta equivalência levou Church a conjecturar que a definição de procedimento efetivo é completamente caracterizada pelas funções recursivas. Esta conjectura é conhecida como *Tese de Church* e até hoje é amplamente aceita.

Os formalismos usados para especificar algoritmos podem ser classificados em *Operacional*, *Axiomático* e *Denotacional* (ou *Funcional*). O formalismo Operacional define uma máquina abstrata baseada em estados, em instruções primitivas e na especificação de como cada instrução modifica cada estado. No formalismo Axiomático, associam-se regras às componentes da linguagem. As regras permitem afirmar o que será verdadeiro após a ocorrência de cláusulas, considerando o que era verdadeiro antes da ocorrência. Por fim, o formalismo Denotacional trata-se de uma função construída a partir de funções elementares de forma composicional no sentido em que o algoritmo denotado pela função pode ser determinado em termos de suas funções componentes.

A classe das funções recursivas pode ser representada por uma gama bastante variada de formalismos. Neste capítulo será utilizada uma linguagem básica cuja origem pode ser encontrada em Eilenberg [39] ou em Brainerd [38]. O material apresentado é uma teoria mais *concreta*, ou seja, mais *formal* e construtiva, na qual as funções serão definidas *dentro* de um plano simbólico e de tal modo que os referentes conceituais fiquem bem determinados.

Quando se fala do sucessor de um número natural, consideramos a função $\underline{suc} : \mathbb{N} \rightarrow \mathbb{N}$ tal que $\underline{suc}(x) = x + 1$ como se o conjunto \mathbb{N} fosse totalmente conhecido e pressupondo um conhecimento anterior do significado da operação de somar 1.

É claro que sabemos efetuar tal operação na representação arábica decimal, mas esta representação não foi nem sequer mencionada na definição de *suc*, e ilustra a necessidade premente de um cuidado mais formal. Nos próximos capítulos serão definidas as funções recursivas via sistemas de representação diversificados.

Serão introduzidas uma série de funções que realizam transformações no espaço \mathcal{W} e que servem de base para a construção de linguagens, máquinas e programas. As funções são apresentadas na seguinte ordem: 1) funções iniciais, que servem de base para todas as demais; 2) funcionais, ou seja, funções que geram funções a partir de funções (composição, combinação e expoentização); 3) A partir de 1) e 2) serão construídas funções aritméticas, funções que manipulam *tuplas*, funções para processamento de cadeias e alguns funcionais de particular interesse computacional. Finalmente é apresentado o funcional repetição, que completa a linguagem básica (LB). O leitor deve prosseguir como se estivesse aprendendo uma nova linguagem de programação funcional.

A apresentação das funções é feita paralelamente com o uso ícones, que são chamadas de componentes atômicos e moleculares. Esta linguagem gráfica assemelha-se aos gráficos conhecidos como circuitos estudados em eletricidade.

3.1 Funções Recursivas Primitivas

Uma das maneiras usuais de apresentar funções matemáticas é através de uma *definição recursiva*: um conjunto de valores da função é explicitado e os demais valores são obtidos destes iniciais. Por exemplo, a conhecida sequência de Fibonacci $1, 1, 2, 3, 5, 8, 13, \dots$ é definida como se segue:

$$\begin{aligned} Fib(0) &= 1 \\ Fib(1) &= 1 \\ Fib(x+2) &= Fib(x+1) + Fib(x) \end{aligned}$$

A definição recursiva é simples e possui operações matemáticas pouco custosas, ao contrário de outras formas de representação. Note a diferença significativa

da definição recursiva para a algébrica a seguir:

$$Fib(x) = \frac{\sqrt{5}}{5} \left(\frac{1 + \sqrt{5}}{2} \right)^{x+1} - \frac{\sqrt{5}}{5} \left(\frac{1 - \sqrt{5}}{2} \right)^{x+1}$$

Esta última definição apresenta operações matemáticas extremamente demoradas, tais como divisão, raiz quadrada e exponenciação. Contudo, existe um ponto de inflexão a partir do qual a representação algébrica conduz a um resultado mais rapidamente do que a recursiva. Isto ocorre quando o valor de x é alto, pois neste caso, a recursão precisaria passar por todos os $x - 1$ resultados até computar o valor de $Fib(x)$.

Será utilizado o mesmo método de *definição recursiva* para definir a classe de funções (primitivas) recursivas. Começa-se pela definição das funções iniciais, cuja simplicidade é óbvia, que são de indiscutível *calculabilidade*. As funções iniciais apresentadas nesta seção são formalismos que representam objetos finitos e realizáveis¹.

Definição 3.1 As seguintes funções são chamadas iniciais:

Zero $\underline{zero} : \mathbb{N} \rightarrow \mathbb{N}$, tal que para todo $x \in \mathbb{N}$

$$\underline{zero}() = 0$$

Projeção para cada $n > 0$ e cada $1 \leq i \leq n$

$\underline{pr} : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$, tal que para todo $\underline{x}_n = \langle x_1, x_2, \dots, x_n \rangle \in \mathbb{N}^n$

$$\underline{pr}(n, i, \underline{x}_n) = x_i$$

A função de projeção também é chamada de função de seleção. Além disto, a

$\underline{pr}(1, 1, x) = x$ é muitas vezes é chamada de função identidade $id(x) = x$.

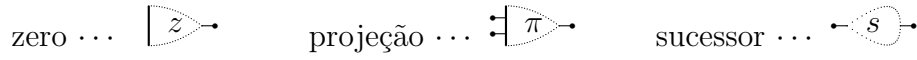
¹São nomes de *processadores* elementares, de um ponto de vista apenas conceitual é que são funções, e portanto abstrações. A idéia é que tais processadores elementares sejam suficientemente simples de tal modo que seja fácil imaginar sua possível existência no plano real, e devem ser suficientes para se constituírem de base para a construção de processadores mais complexos que representam a classe completa das funções recursivas. Assim em todas as definições se está tratando da função (entidade abstrata) que os processadores reais computam e que são representadas por uma linguagem. São usados os mesmos símbolos para representar tais processadores e as correspondentes funções.



Sucessor $\underline{suc} : \mathbb{N} \rightarrow \mathbb{N}$, tal que para todo $x \in \mathbb{N}$

$$\underline{suc}(x) = y, \quad \nu(y) = x + 1$$

Observe que a função adição ainda não foi definida na hierarquia de funções recursivas, por isto não deve ser utilizada na definição da função $\underline{suc}(x)$. O leitor deve entender que o resultado desta função é y , cuja avaliação significa o elemento posterior a x em uma enumeração qualquer, que aqui foi simplifi-
cadamente representado por $x + 1$.


As funções iniciais são representadas pelas seguintes componentes atômicas:

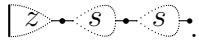


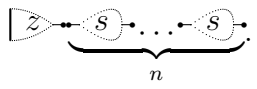
A idéia é que $\{ \text{zero}, \text{sucessor}, \text{projeção} \}$ é o conjunto de componentes básicas atômicas para a construção de máquinas mais complexas. Estas componentes são fornecidas como caixas pretas obedecendo às especificações dadas pelas definições abstratas. Pode-se também imaginar  como um agente que cria um registro con-
tendo a palavra nula 0 cujo valor é 0. As demais componentes apenas transformam
ou destroem registros criados anteriormente. Assim, segundo esta interpretação,
apenas  tem utilidade quando considerada isoladamente.

Os números naturais, por exemplo, podem ser definidos como uma sucessão
de componentes básicas atômicas justapostas uma após a outra:

0 : função primitiva recursiva $\underline{zero}()$, isto é, .

1 : função primitiva recursiva $\underline{suc}(\underline{zero}())$, isto é, .


2 : função primitiva recursiva $\underline{suc}(\underline{suc}(\underline{zero}()))$, isto é, .

n : função primitiva recursiva $\underbrace{\underline{suc}(\dots \underline{suc}(\underline{zero}()) \dots)}_n$, isto é, .

De modo análogo, os valores booleanos também podem ser definidos como
uma sucessão de componentes básicas atômicas justapostas uma após a outra:

Falso : função primitiva recursiva $\underline{zero}()$, isto é, .

Verdadeiro : função primitiva recursiva $\underline{suc}(\underline{zero}())$, isto é, .

A função identidade $id(x) = x$, apesar de ser derivada de $\underline{pr}(1, 1, x) = x$ e não ser uma função inicial, também terá uma componente básica atômica de modo a simplificar a representação. Esta componente será .

A seguir são definidas maneiras de *construir* funções a partir de funções anteriormente construídas, no caso básico, as iniciais.

Definição 3.2 [Recursão Primitiva] Sejam $f : \mathbb{N}^n \rightarrow \mathbb{N}$ e $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$, diz-se que $h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ é definida por recursão primitiva se os valores de h são obtidos por

$$\begin{aligned} h(0, x_1, \dots, x_n) &= f(x_1, \dots, x_n) \\ h(y + 1, x_1, \dots, x_n) &= g(y, h(y, x_1, \dots, x_n), x_1, \dots, x_n) \end{aligned}$$

Exemplo 3.1 A adição de números naturais pode ser definida por:

$$\begin{aligned} 0 + x &= x \\ (y + 1) + x &= (y + x) + 1 \end{aligned}$$

Assim, toma-se $f = \underline{pr}(1, 1, ??) : \mathbb{N}^3 \rightarrow \mathbb{N}$ e $g = \underline{suc} \circ \underline{pr}(3, 2, ??, ??, ??) : \mathbb{N}^5 \rightarrow \mathbb{N}$, obtidas por composição a partir das funções iniciais, pode-se escrever:

$$\begin{aligned} h(0, x) &= \underline{soma}(0, x) = f(x) = \underline{pr}(1, 1, x) \\ h(y + 1, x) &= \underline{soma}(y + 1, x) = g(y, h(y, x), x) = \underline{suc}(\underline{pr}(3, 2, y, \underline{soma}(y, x), x)) \\ &= \underline{suc}(\underline{soma}(y, x)) \end{aligned}$$

É fácil ver que para todo x e todo y , $\underline{soma}(x, y) = x + y$, portanto \underline{soma} é a adição de números naturais. E, além disso \underline{soma} foi obtida por composição e recursão primitiva a partir das funções iniciais. Neste caso, partindo da segunda parcela, faz-se sucessor tantas vezes quantas forem indicadas pela primeira parcela. Note também que para calcular a soma de dois números, por exemplo 3 e 5 seria computado:

$$\begin{aligned} h(3, 5) &= 3 + 5 \\ \underline{soma}(0, 5) &= \underline{pr}(1, 1, 5) = 5 \\ \underline{soma}(1, 5) &= \underline{suc}(\underline{pr}(3, 2, 0, \underline{soma}(0, 5), 5)) = \underline{suc}(\underline{soma}(0, 5)) = \underline{suc}(5) = 6 \\ \underline{soma}(2, 5) &= \underline{suc}(\underline{pr}(3, 2, 1, \underline{soma}(1, 5), 5)) = \underline{suc}(\underline{soma}(1, 5)) = \underline{suc}(6) = 7 \\ \underline{soma}(3, 5) &= \underline{suc}(\underline{pr}(3, 2, 2, \underline{soma}(2, 5), 5)) = \underline{suc}(\underline{soma}(2, 5)) = \underline{suc}(7) = 8 \end{aligned}$$

Definição 3.3 [Função Recursiva Primitiva] Uma função $f : \mathbb{N}^n \rightarrow \mathbb{N}$ é *recursiva primitiva* se f for uma função inicial ou for obtida pela utilização de composição e recursão primitiva a partir das funções iniciais.

Para mostrar que uma função $f : \mathbb{N}^n \rightarrow \mathbb{N}$ é recursiva primitiva, basta exibir a sua definição informal como feito com a adição. Assim para a multiplicação tem-se:

$$\begin{array}{lcl} 0.x & = & 0 \\ (y+1).x & = & y.x + x \end{array} \quad \left\| \quad \begin{array}{lcl} h(0, x) & = & f(x) \\ h((y+1), x) & = & g(y, h(y, x), x) \end{array} \right.$$

sem precisar explicitar a forma

$$\begin{aligned} \underline{produto}(0, x) &= \underline{zero}() \\ \underline{produto}(y+1, x) &= \underline{soma}(\underline{pr}(3, 2, y, \underline{produto}(y, x), x), \underline{pr}(3, 3, y, \underline{produto}(y, x), x)) \\ &= \underline{soma}(\underline{produto}(y, x), x) \end{aligned}$$

que obedece a definição formal de recursão primitiva.

Exemplo 3.2 Definição informal das seguintes funções:

Predecessor \underline{pred} , função que retorna o antecessor de um número de uma lista \underline{x} de n objetos.

$$\begin{aligned} \underline{pred}(0) &= 0 \\ \underline{pred}(x) &= \underline{pr}(\underline{suc}(n), x, \underline{zero}(), \underline{x}) \end{aligned}$$

Note que para calcular o predecessor de 3, seria computado:

$$\underline{pred}(3) = \underline{pr}(\underline{suc}(n), 3, 0, 1, 2, 3, 4, \dots, n)$$

Subtração própria \underline{monus} , função cujo resultado é $(x - y)$ se $x \geq y$ e 0 de $x < y$

$$\begin{aligned} \underline{monus}(x, 0) &= \underline{pr}(1, 1, x) \\ \underline{monus}(x, y+1) &= \underline{pred}(\underline{monus}(x, y)) \end{aligned}$$

Note que para calcular $3 - 2$, deve ser computado:

$$\begin{aligned} \underline{monus}(3, 0) &= 3 \\ \underline{monus}(3, 1) &= \underline{pred}(\underline{monus}(3, 0)) = \underline{pred}(3) = 2 \\ \underline{monus}(3, 2) &= \underline{pred}(\underline{monus}(3, 1)) = \underline{pred}(2) = 1 \end{aligned}$$

Lema 3.1 As funções a seguir são recursivas primitivas:

(a) $\underline{exp}(x, y) = x^y, y \geq 0$

(b) $\underline{!}(x) = x!$

(c) $\underline{somatório}(x) = \sum_{n=0}^x n$

(d) $\underline{modulo}(x, y) = |x - y| = \begin{cases} x - y & \text{se } x \geq y \\ y - x & \text{se } x < y \end{cases}$

(e) $\underline{diferente}(x, y) = \begin{cases} 1 & \text{se } x \neq y \\ 0 & \text{se } x = y \end{cases}$

(f) $\underline{igual}(x, y) = \begin{cases} 1 & \text{se } x = y \\ 0 & \text{se } x \neq y \end{cases}$

(g) $\underline{menorigual}(x, y) = \begin{cases} 1 & \text{se verdadeiro} \\ 0 & \text{se falso} \end{cases}$

(h) $\underline{menor}(x, y) = \begin{cases} 1 & \text{se verdadeiro} \\ 0 & \text{se falso} \end{cases}$

(i) $\underline{maior}(x, y) = \begin{cases} 1 & \text{se verdadeiro} \\ 0 & \text{se falso} \end{cases}$

(j) $\underline{maiorigual}(x, y) = \begin{cases} 1 & \text{se verdadeiro} \\ 0 & \text{se falso} \end{cases}$

(k) $\underline{max}(x, y) = \text{maior número entre } x \text{ e } y$

(l) $\underline{min}(x, y) = \text{menor número entre } x \text{ e } y$

(m) $\underline{quo}(x, y) = x \text{ div } y$

(n) $\underline{not}(x) = \begin{cases} 1 & \text{se } x = 0 \\ 0 & \text{se qualquer outro valor de } x \end{cases}$

(o) $\underline{and}(x, y)$ onde $x, y = 0$ ou 1

(p) $\underline{or}(x, y)$ onde $x, y = 0$ ou 1

$$(q) \quad \underline{if}(x, y, z) \begin{cases} y & \text{se } x \neq 0 \\ z & \text{se } x = 0 \end{cases}$$

(r) $\underline{mod}(x, y)$, resto da divisão de x por y

(s) $\underline{for}(i, n, c) = \text{for}(k = i; k \leq n; k++)c;$

(t) $\underline{dowhile}(c, i, n) = \text{do } c \text{ while}(i \leq n);$

Lema 3.2 Seja $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$. As funções:

$$(a) \cdots \sum_{y=0}^{y=m} f(\underline{x}_n, y) = f(\underline{x}_n, 0) + f(\underline{x}_n, 1) + \cdots + f(\underline{x}_n, m)$$

$$(b) \cdots \prod_{y=0}^{y=m} f(\underline{x}_n, y) = f(\underline{x}_n, 0) \cdot f(\underline{x}_n, 1) \cdot \cdots \cdot f(\underline{x}_n, m)$$

$$(c) \cdots \bigvee_{y=0}^{y=m} f(\underline{x}_n, y) = f(\underline{x}_n, 0) \vee f(\underline{x}_n, 1) \vee \cdots \vee f(\underline{x}_n, m)$$

$$(d) \cdots \bigwedge_{y=0}^{y=m} f(\underline{x}_n, y) = f(\underline{x}_n, 0) \wedge f(\underline{x}_n, 1) \wedge \cdots \wedge f(\underline{x}_n, m)$$

onde $x \vee y = \vee(x, y)$ e $x \wedge y = \wedge(x, y)$, são recursivas primitivas

Demonstração:

(a) Seja a função g definida por

$$\begin{aligned} g(0) &= f(0) \\ g(x+1) &= g(x) + f(x+1) \end{aligned}$$

É claro que

$$g(n) = \sum_{x=0}^{x=n} f(x)$$

(b) semelhante à (a) bastando tomar g como sendo:

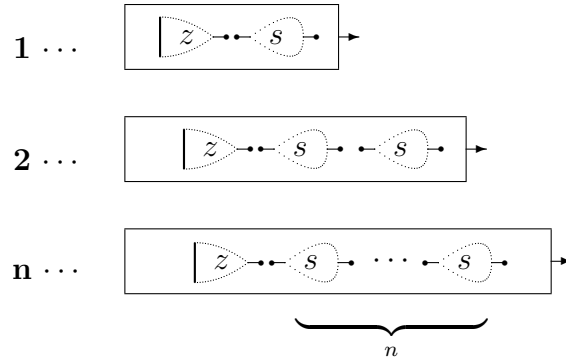
$$\begin{aligned} g(0) &= f(0) \\ g(x+1) &= g(x) \cdot f(x+1) \end{aligned}$$

(c),(d) Sugere-se que o leitor demonstre.

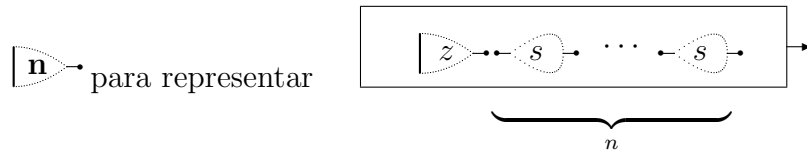
Se for desejado criar espaços de representação e transformá-los deve-se buscar envelopamentos que permitam juntar tais componentes atômicas em objetos estruturados. Se for considerado juntar componentes e objetos já preparados em *série*, pode-se obter componentes moleculares com estrutura linear.

Exemplo 3.3 A função constante $f(x) = 2$ pode ser obtida por composição como $f(x) = \underline{suc}(\underline{suc}(\underline{zero}()))$

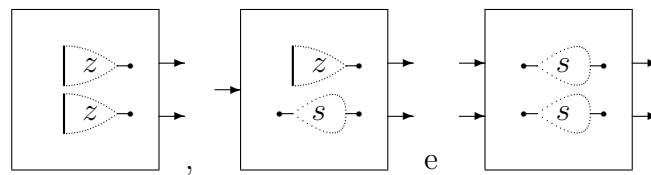
Assim pode-se ter estruturas como a seguir:



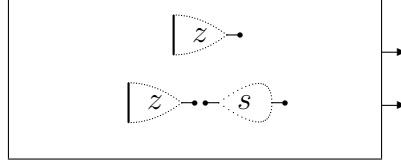
Estas componentes moleculares foram obtidas via uma *disposição linear* em que a saída de uma componente atômica é usada como entrada para a próxima, da esquerda para a direita. Note que se pode assim criar registros contendo todos os elementos do espaço \mathcal{W} . Estas componentes moleculares representam os números naturais e pode-se utilizar figuras mais simples,



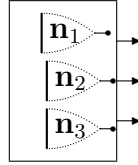
Uma outra maneira de se obter componentes moleculares é por estruturação vertical ou em *paralelo*, por exemplo:



que representam combinações em paralelo de 2 componentes atômicas. Pode-se ter combinações em paralelo de componentes moleculares e atômicas:



Pode-se ter componentes moleculares para representar qualquer tupla $\langle x_1, x_2, \dots, x_r \rangle$ de \mathcal{W}_r . Por exemplo, para $r = 3$ e tomando $\nu(x_i) = n_i$, tem-se que a componente molecular a seguir cria três registros contendo n_1 , n_2 e n_3 .



Para obter funções entre espaços quaisquer utiliza-se esquemas de construção de funções a partir de funções. Em geral, se houver uma classe \mathcal{F} de funções definidas em um conjunto A_1 e tomando valores em um conjunto A_2 , diz-se que \mathcal{F} é uma subclasse da classe $A_2^{A_1}$, que é a classe de todas as funções de A_1 em A_2 . Um funcional é qualquer função que tome como parâmetros outras funções, por exemplo F de $(A_2^{A_1})^n$ em $A_2^{A_1}$. No caso particular que está sendo estudado, tem-se funções em $\mathcal{W}^{\mathcal{W}}$. Assim nestes esquemas de construção de funções os funcionais são de $(\mathcal{W}^{\mathcal{W}})^n$ em $\mathcal{W}^{\mathcal{W}}$.

Definição 3.4 [Composição] Sejam as funções $f : \mathbb{N}^m \rightarrow \mathbb{N}$, $g_1, \dots, g_m : \mathbb{N}^n \rightarrow \mathbb{N}$. A composição h de f com g_1, \dots, g_m é a função $h : \mathbb{N}^n \rightarrow \mathbb{N}$, definida por:

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

A composição de uma função f com uma outra g é usualmente denotada por $f \circ g$, e ilustrada na Figura 3.1.

Definição 3.5 Sejam as funções $f : \mathcal{W}_r \rightarrow \mathcal{W}_s$ e $g : \mathcal{W}_s \rightarrow \mathcal{W}_t$. A composição de f e g é a função $g \circ f : \mathcal{W}_r \rightarrow \mathcal{W}_t$ definida por:

$$g \circ f(\underline{x}_r) = g(f(\underline{x}_r))$$

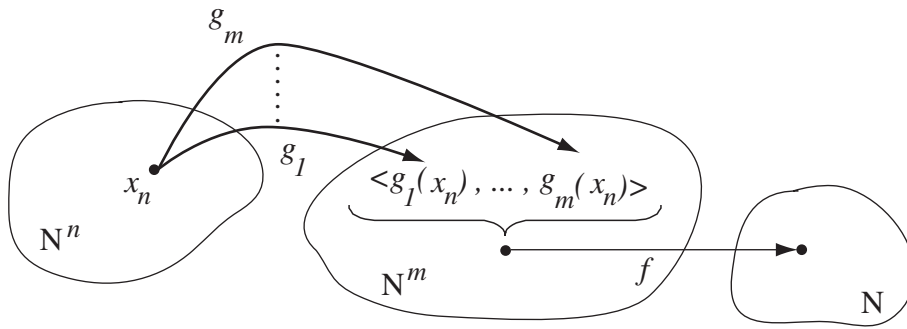
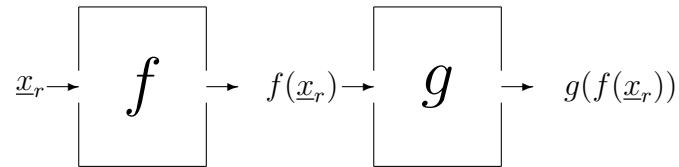


Figura 3.1: Composição de funções.

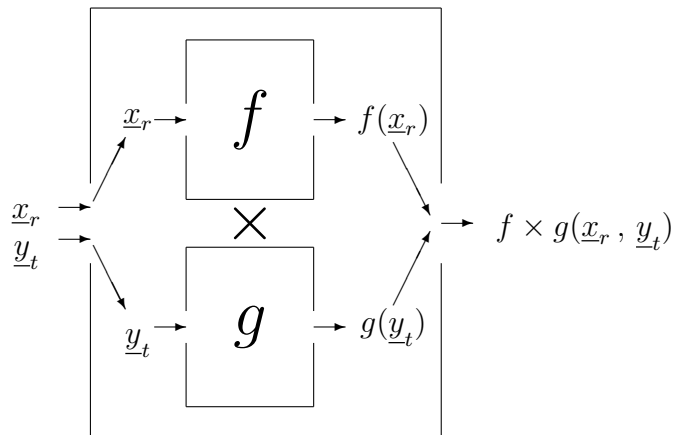


Exemplo 3.4 A partir das funções iniciais pode-se obter, pelo uso da composição:

$$\begin{aligned}
 s \circ z &= 1 \\
 s \circ s \circ z &= 2 \\
 \underbrace{s \circ s \circ \dots \circ s}_n \circ z &= n
 \end{aligned}$$

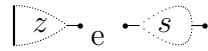
Definição 3.6 Sejam as funções $f : \mathcal{W}_r \rightarrow \mathcal{W}_s$ e $g : \mathcal{W}_t \rightarrow \mathcal{W}_u$. A combinação de f com g é a função $f \times g : \mathcal{W}_{r+t} \rightarrow \mathcal{W}_{s+u}$, definida por:

$$f \times g(\underline{x}_r, \underline{y}_t) = \langle f(\underline{x}_r), g(\underline{y}_t) \rangle$$



Todas as tuplas de \mathcal{W}_2 podem agora ser representadas, do mesmo modo que são representados os elementos de \mathcal{W}_1 . Assim $\langle 0, 0 \rangle$ é representado por $z \times z$. As tuplas de \mathcal{W}_3 também podem ser representadas, por exemplo $\langle 0, 3, 1 \rangle$ é representado por $z \times s \circ s \circ s \circ z \times s \circ z$. Em geral, os pontos do espaço \mathcal{W}_k são representados por $n_1 \times n_2 \times \cdots \times n_k$ e os n_i por $s^{n_i} \circ z$. Note que os pontos do espaço \mathcal{W} agora podem ser representados como r -tuplas de palavras no alfabeto Σ , ou como componentes moleculares, ou ainda como expressões formadas por composição e combinação das funções iniciais.

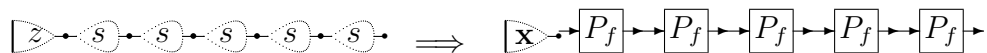
Todas as tuplas do espaço \mathcal{W} podem ser realizadas (dependendo de interpretação) no plano real por objetos compostos pelas componentes atômicos



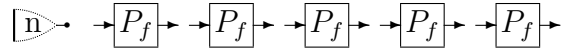
através dos esquemas de composição (componentes arranjados em série) e combinação (componentes arranjados em paralelo). Por exemplo, considerando o alfabeto $\Sigma = \{\square, \diamond\}$, com $\nu_2(\square) = 1$ e $\nu_2(\diamond) = 2$, pode-se construir $\mathcal{W}_1 = \{\square, \diamond, \square\square, \square\diamond, \dots\}$. Assim, tem-se que $\nu_2(\square\diamond) = 4$ e sua componente molecular é $\boxed{z} \rightarrow \bullet \bullet \bullet \bullet \rightarrow$. De um modo geral uma palavra de \mathcal{W}_1 de valor n corresponde ao componente molecular $\boxed{z} \rightarrow \underbrace{\bullet \bullet \bullet \bullet \rightarrow}_n$.

Note que não existe circularidade nesta representação, pois o n é utilizado como recurso metalinguístico para explicar a representação. Assim tem-se representações finitárias para objetos reais, seja através da linguagem de representação numérica, seja como componente molecular.

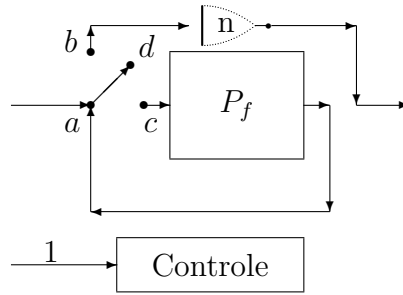
Se for considerado uma componente P_f realizando $f : \mathcal{W}_1 \rightarrow \mathcal{W}_1$ e deseja-se reaplicar tal função, por exemplo 5 vezes a uma certa palavra x de \mathcal{W}_1 , pode-se realizar a componente correspondente substituindo $\boxed{z} \rightarrow$ por $\boxed{x} \rightarrow$ e cada $\bullet \bullet \bullet \bullet \rightarrow$ pela componente P_f .



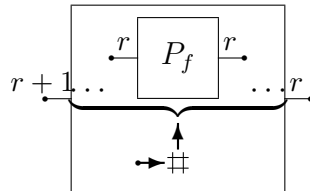
Esta substituição corresponderia a reaplicar a função f a x . Para se obter a função que seria aplicada a uma entrada qualquer x basta substituir $\boxed{z} \rightarrow$ por $\boxed{n} \rightarrow$, obtendo-se assim a componente molecular



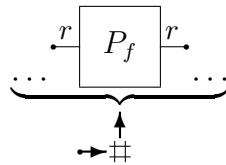
A figura a seguir representa uma maneira em que a operação de substituição pode ser realizada utilizando-se apenas um componente molecular correspondendo a P_f e o componente atômico \boxed{n} . A chave em a está ligada inicialmente na posição d . O controle recebe como entrada a representação do número de vezes que P_f deve ser reaplicado. Se a entrada for \boxed{z} então a chave em a é ligada na a posição b ; se a entrada for diferente de 0 então o controle liga a chave para a posição c e em seguida retorna a chave para a posição d aguardando a próxima entrada atômica. Isto é como ligar a chave cada vez que se passa uma conta do ábaco para a direita.



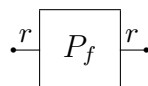
Pode-se representar esta figura de modo mais simples por



onde indica-se o número de entradas para P_f . O 1 somado a r significa mais uma entrada para o controle, e



é para ser entendido como a repetição do componente P_f , o número de vezes indicado por #.



Se $P_f^\# : \mathcal{W}_{r+1} \rightarrow \mathcal{W}_1$ for a função computada por este componente, então tem-se:

$$\begin{aligned} P_f^\#(\underline{x}_r, 0) &= \underline{x}_r \\ P_f^\#(\underline{x}_r, y + 1) &= P_f(P_f^\#(\underline{x}_r, y)) \end{aligned}$$

evidenciando que esta maneira de realizar este tipo de repetição de uma função coincide com o esquema de recursão primitiva. Mais tarde este tópico será rediscutido. Por enquanto define-se o esquema de obtenção de novas funções que é motivado por esta discussão.

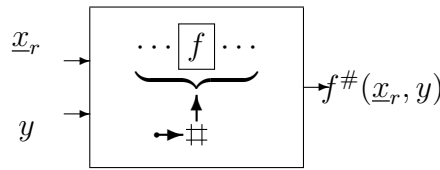
Definição 3.7 [Expoentização] Seja a função $f : \mathcal{W}_r \rightarrow \mathcal{W}_r$. A expoentização de f é a função $f^\# : \mathcal{W}_{r+1} \rightarrow \mathcal{W}_r$, definida por:

$$\begin{aligned} f^\#(\underline{x}_r, 0) &= \underline{x}_r \\ f^\#(\underline{x}_r, y + 1) &= f(f^\#(\underline{x}_r, y)) \end{aligned}$$

Diz-se, também, que $f^\#$ é definida por recursão primitiva a partir de f . Observe que:

$$\begin{aligned} f^\#(x, 0) &= x \\ f^\#(x, 1) &= f(x) \\ f^\#(x, 2) &= f(f(x)) \\ &\vdots \end{aligned}$$

A componente molecular para a expoentização de uma função f é representada pela figura:



A rigor seria necessário também dizer que a função *expoentização* é a composição da função f consigo mesma, tantas vezes quanto o valor de y , aplicada a x_r . Logo, se f é computável, $f^\#$ também é computável.

Informalmente, as funções recursivas primitivas são aquelas obtidas das funções iniciais e aplicação dos esquemas de composição (\circ), combinação (\times) e expoentização ($\#$), o que gera uma coleção muito grande de funções.

Definição 3.8 [Funções recursivas primitivas]

1. Diz-se que uma função $f : \mathcal{W} \rightarrow \mathcal{W}$ é recursiva primitiva se existe uma seqüência f_1, \dots, f_n de funções em $\mathcal{W}^{\mathcal{W}}$ tal que $f = f_n$ e para todo i , $1 \leq i < n$ f_i é uma função inicial ou f_i satisfaz um dos itens abaixo:

$$(I) \quad f_i = f_j \circ f_k \quad 1 \leq j, k < i$$

$$(II) \quad f_i = f_j \times f_k \quad 1 \leq j, k < i$$

$$(III) \quad f_i = (f_j)^\# \quad 1 \leq j < i$$

2. Seja F um funcional de $(\mathcal{W}^{\mathcal{W}})^n$ em $\mathcal{W}^{\mathcal{W}}$. Diz-se que F é recursivo primitivo se e somente se para toda $f \in \text{dom}(F)$, se f recursiva primitiva então $F(f)$ é recursiva primitiva.

Seja $f = \iota^\# \circ (s \circ z \times s \circ z)$, então a seguinte seqüência de funções evidencia que f é uma função recursiva primitiva:

$$\begin{array}{ll} f_1 & \cdots \quad z \quad \cdots (\text{inicial}) \\ f_2 & \cdots \quad s \quad \cdots (\text{inicial}) \\ f_3 & \cdots \quad s \circ z \quad \cdots (f_1 \circ f_2) \\ f_4 & \cdots \quad s \circ z \times s \circ z \quad \cdots (f_3 \times f_3) \\ f_5 & \cdots \quad \iota \quad \cdots (\text{inicial}) \\ f_6 & \cdots \quad \iota^\# \quad \cdots (f_5^\#) \\ f_7 & \cdots \quad (\iota)^\# \circ (s \circ z \times s \circ z) \quad \cdots (f_6 \circ f_4) \end{array}$$

Chega-se a um ponto em que se tem esboçada uma linguagem de programação, definida a seguir.

3.2 A Linguagem Básica

Nesta seção será introduzida uma linguagem de programação chamada de linguagem básica- ∇ ou $LB-\nabla$. Esta linguagem é naturalmente obtida ao considerarem-se as expressões formadas a partir das funções iniciais π, z, s e os esquemas de composição \circ , combinação \times , expoentização $\#$. O esquema de repetição ∇ é definido na seção 3.8 e estende a linguagem posteriormente.

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*, Coleção Schaum. São 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.