Capítulo 7

Teorema de Gödel

7.1 Funções recursivas primitivas

Introduziremos uma classe de funções numéricas que são evidentemente efetivamente computáveis. O procedimento pode parecer um tanto ad hoc, mas ele nos dá uma classe surpreendentemente rica de algoritmos. Usamos o método indutivo, ou seja, fixamos um número de funções iniciais que são tão efetivas quanto se pode desejar; depois disso especificamos maneiras de construir novos algoritmos a partir de outros algoritmos.

Os algoritmos iniciais são de fato extremamente simples: a função sucessor, as funções constantes e as funções de projeção. É óbvio que a composição (ou substituição) de algoritmos dá origem a algoritmos. O uso de recursão foi como um dispositivo para se obter novas funções já conhecido de Dedekind; que recursão produz algoritmos a partir de algoritmos dados é também facilmente visto. Em lógica o estudo de funções recursivas primitivas foi iniciado por Skolem, Herbrand, Gödel e outros.

Continuaremos agora com uma definição precisa, que será dada na forma de uma definição indutiva. Primeiro apresentamos uma lista de funções iniciais de uma natureza inequivocamente algorítmica, e então especificamos como obter novos algoritmos de outros algoritmos. Todas as funções têm sua própria aridade, o que significa dizer que elas mapeiam $\mathbb{N}^k$ em $\mathbb{N}$ para um k adequado. Em geral não especificaremos as aridades das funções envolvidas, e assumiremos que elas são escolhidas corretamente. As chamadas *funções iniciais* são

- a *função constante* C_m^k com $C_m^k(n_0, \ldots, n_{k-1}) = m$,
- a *função sucessor* S com $S(n) = n + 1$,
- as *funções de projeção* P_i^k com $P_i^k(n_0, \ldots, n_{k-1}) = n_i$ $(i < k)$

Novos algoritmos são obtidos a partir de outros algoritmos por *substituição* ou *composição* e *recursão primitiva*.

- Uma classe $\mathcal{F}$ de funções é *fechada sob substituição* se $g, h_0, \ldots, h_{p-1} \in \mathcal{F} \Rightarrow f \in \mathcal{F}$, onde $f(\vec{n}) = g(h_0(\vec{n}), \ldots, h_{p-1}(\vec{n}))$
- $\mathcal{F}$ é *fechada sob recursão primitiva* se, $g, h \in \mathcal{F} \Rightarrow f \in \mathcal{F}$, onde

$$\begin{cases} f(0, \vec{n}) = g(\vec{n}) \\ f(m+1, \vec{n}) = h(f(m, \vec{n}), \vec{n}, m). \end{cases}$$

Definição 7.1.1 *A classe das funções recursivas primitivas é a menor classe de funções contendo a função constante, a função sucessor, e as funções de projeção, que é fechada sob substituição e recursão primitiva.*

Observação. Substituição foi definida de uma maneira particular: as funções que são substituídas têm todas a mesma cadeia de entradas. De modo a fazer substituições arbitrárias é preciso fazer um pouco de trabalho adicional. Considere por exemplo a função $f(x, y)$ na qual desejamos substituir $g(z)$ no lugar de x e $f(z, x)$ no lugar de y: $f(g(z), f(z, x))$. Isso é realizado como se segue: ponha $h_0(x, z) = g(z) = g(P_1^2(x, z))$ e $h_1(x, z) = f(z, x) = f(P_1^2(x, z), P_0^2(x, z))$. Então o $f(g(z), f(z, x))$ desejado é obtido como $f(h_0(x, z), h_1(x, z))$. Assume-se que o leitor é capaz de lidar com casos de substituição que virão a aparecer mais adiante.

Vamos começar construindo um estoque de funções recursivas primitivas. A técnica não é de forma nenhuma difícil, e maioria dos leitores a terão usado em diveras ocasiões. O fato surpreendente é que tantas funções possam ser obtidas por esses procedimentos simples. Aqui vai um primeiro exemplo:

$x + y$, definida por

$$\begin{cases} x + 0 = x \\ x + (y + 1) = (x + y) + 1. \end{cases}$$

Reformularemos essa definição de modo que o leitor possa ver que ela de fato se encaixa no formato prescrito:

$$\begin{cases} +(0, x) = P_0^1(x) \\ +(y + 1, x) = S(P_0^3(+(y, x), P_0^2(x, y), P_1^2(x, y))). \end{cases}$$

Via de regra, continuaremos a usar a notação tradicional, de modo que escreveremos simplesmente $x+y$ no lugar de $+(x, y)$. Também usaremos tacitamente abreviações da matemática, e.g. omitiremos o ponto da multiplicação.

Há dois truques convenientes para adicionar ou remover variáveis. O primeiro é a introdução de variáveis fantasma.

Lema 7.1.2 (variáveis fantasma) *Se f é recursiva primitiva, então g também o é, com $g(x_0, \ldots, x_{n-1}, z_0, \ldots, z_{m-1}) = f(x_0, \ldots, x_{n-1})$.*

Demonstração. Ponha

$g(x_0, \ldots, x_{n-1}, z_0, \ldots, z_{m-1}) = f(P_0^{n+m}(\vec{x}, \vec{z}), \ldots, P_n^{n+m}(\vec{x}, \vec{z}))$. □

Lema 7.1.3 (identificação de variáveis) *Se f é recursiva primitiva, então $f(x_0, \ldots, x_{n-1})[x_i/x_j]$ também o é, onde $i, j \leq n$.*

Demonstração. Precisamos considerar apenas o case em que $i \neq j$. $f(x_0, \ldots, x_{n-1})[x_i/x_j] = f(P_0^n(x_0, \ldots, x_{n-1}), \ldots, P_i^n(x_0, \ldots, x_{n-1}), \ldots, P_i^n(x_0, \ldots, x_{n-1}), \ldots, P_{n-1}^n(x_0, \ldots, x_{n-1}))$, onde o segundo P_i^n está na j-ésima entrada. □

Uma notação mais mundana é $f(x_0, \ldots, x_i, \ldots, x_i, \ldots, x_{n-1})$.

Lema 7.1.4 (permutação de variáveis) *Se f é recursiva primitiva, então g também o é com $g(x_0, \ldots, x_{n-1}) = f(x_0, \ldots, x_{n-1})[x_i, x_j/x_j, x_i]$, onde $i, j \leq n$.*

Demonstração. Use substituição e funções de projeção. □

De agora em diante usaremos as notações informais tradicionais, e.g. $g(x) = f(x, x, x)$, ou $g(x, y) = f(y, x)$. Por conveniência usamos e usaremos, sempre que nenhuma confusão possa surgir, a notação vetorial para cadeias de entradas.

O leitor pode facilmente verificar que os exemplos abaixo podem ser moldados no formato requerido das funções primitivas recursivas.

1. $x + y$

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

2. $x \cdot y$

$$\begin{cases} x \cdot 0 = 0 \\ x \cdot (y + 1) = x \cdot y + x \quad \text{(usamos (1))} \end{cases}$$

3. x^y

$$\begin{cases} x^0 = 1 \\ x^{y+1} = x^y \cdot x \end{cases}$$

4. *função predecessor*

$$p(x) = \begin{cases} x - 1 & \text{se } x > 0 \\ 0 & \text{se } x = 0 \end{cases}$$

Aplique recursão:

$$\begin{cases} p(0) = 0 \\ p(x + 1) = x \end{cases}$$

5. *subtração de corte (monus)*

$$x \dot{-} y = \begin{cases} x - y & \text{se } x \geq y \\ 0 & \text{caso contrário} \end{cases}$$

6. *função fatorial*

$$n! = 1 \cdot 2 \cdot 3 \cdots (n - 1) \cdot n.$$

7. *função signum*

$$sg(x) = \begin{cases} 0 & \text{se } x = 0 \\ 1 & \text{caso contrário} \end{cases}$$

Aplique recursão:

$$\begin{cases} sg(0) = 0 \\ sg(x + 1) = 1 \end{cases}$$

8. $\overline{sg}(x) = 1 \dot{-} sg(x)$.

$$\overline{sg}(x) = \begin{cases} 1 & \text{se } x = 0 \\ 0 & \text{caso contrário} \end{cases}$$

9. $|x - y|$.
   Observe que $|x - y| = (x \dot{-} y) + (y \dot{-} x)$

10. $f(\vec{x}, y) = \sum_{i=0}^{y} g(\vec{x}, i)$, onde g é recursiva primitiva.

$$\begin{cases} \sum_{i=0}^{0} g(\vec{x}, i) = g(\vec{x}, 0) \\ \sum_{i=0}^{y+1} g(\vec{x}, i) = \sum_{i=0}^{y} g(\vec{x}, i) + g(\vec{x}, y+1) \end{cases}$$

11. $\prod_{i=0}^{y} g(\vec{x}, i)$, idem.

12. Se f é recursiva primitiva e π é uma permutação do conjunto $\{0, \ldots, n-1\}$ então g com $g(\vec{x}) = f(x_{\pi 0}, \ldots, x_{\pi(n-1)})$ também é recursiva primitiva.

13. Se $f(\vec{x}, y)$ é recursiva primitiva, então $f(\vec{x}, k)$ também o é.

A definição de funções recursivas primitivas diretamente é um desafio que vale a pena, e o leitor encontrará casos interessantes entre os exercícios. Para um acesso rápido e eficiente a um grande estoque de funções recursivas primitivas existem, no entanto, técnicas que cortam alguns caminhos. Vamos apresentá-las aqui.

Em primeiro lugar podemos relacionar conjuntos e funções por meio de *funções características*. No cenário das funções da teoria dos números, definimos funções características da seguinte forma: para $A \subseteq \mathbb{N}^k$ a função característica $K_A : \mathbb{N}^k \to \{0, 1\}$ de A é dada por $\vec{n} \in A \Leftrightarrow K_A(\vec{n}) = 1$ (e portanto $\vec{n} \notin A \Leftrightarrow K_A(\vec{n}) = 0$). Advertência: em lógica a função característica é às vezes definida com 0 e 1 intercambiados. Para a teoria isso não faz diferença alguma. Note que um subconjunto de $\mathbb{N}^k$ é também chamado de uma *relação k-ária*. Quando lidamos com relações tacitamente assumimos que temos o número correto de argumentos, e.g. quando escrevemos $A \cap B$ supomos que A, B são subconjuntos do mesmo $\mathbb{N}^k$.

Definição 7.1.5 *Uma relação R é recursiva primitiva se sua função característica o é.*

Note que isso corresponde à idéia de usar K_R como um teste de pertinência.

Os seguintes conjuntos (relações) são recursivos primitivos:

1. $\emptyset$: $K_\emptyset(n) = 0$ para todo n.

2. O conjunto P dos números pares:

$$\begin{cases} K_P(0) = 1 \\ K_P(x+1) = \overline{sg}(K_P(x)) \end{cases}$$

3. A relação de igualdade: $K_=(x, y) = \overline{sg}(|x - y|)$.

4. A relação de ordem: $K_<(x, y) = \overline{sg}((x+1) \dot{-} y)$.

Lema 7.1.6 *As relações recursivas primitivas são fechadas sob $\cup, \cap, {}^c$ e quantificação cotada.*

Demonstração. Seja $C = A \cap B$, então $x \in C \Leftrightarrow x \in A \wedge x \in B$, portanto $K_C(x) = 1 \Leftrightarrow K_A(x) = 1 \wedge K_B(x) = 1$. Por conseguinte, fazemos $K_C(x) = K_A(x) \cdot K_B(x)$. Logo, a interseção de conjuntos recursivos primitivos é recursiva

primitiva. Para a união, tome $K_{A\cup B}(x) = sg(K_A(x)+K_B(x))$, e para o complemento $K_{A^c}(x) = \overline{sg}(K_A(x))$.

Dizemos que R *é obtida por quantificação cotada a partir de* S se $R(\vec{n}, m) := Qx \leq mS(\vec{n}, x)$, onde Q é um dos quantificadores $\forall$, $\exists$.

Considere a quantificação existencial cotada: $R(\vec{x}, n) := \exists y \leq nS(\vec{x}, y)$, então $K_R(\vec{x}, n) = sg(\sum_{y\leq n} K_S(\vec{x}, y))$, portanto se S for recursivo primitivo, então R também o é.

O caso do $\forall$ é semelhante; deixo ao leitor. □

Lema 7.1.7 *As relações recursivas primitivas são fechadas sob substituições recursivas primitivas, i.e. se* $f_0, \ldots, f_{n-1}$ *e* R *forem recursivas primitivas, então* $S(\vec{n}) := R(f_0(\vec{x}), \ldots, f_{n-1}(\vec{x}))$ *também o é.*

Demonstração. $K_S(\vec{x}) = K_R(f_0(\vec{x}), \ldots, f_{n-1}(\vec{x}))$. □

Lema 7.1.8 (definição por casos) *Sejam* $R_1, \ldots, R_p$ *predicados recursivos primitivos mutuamente exclusivos, tais que* $\forall\vec{x}(R_1(\vec{x}) \vee R_2(\vec{x}) \vee \cdots \vee R_p(\vec{x}))$ *e suponha que* $g_1, \ldots, g_p$ *sejam funções recursivas primitivas, então* f *com*

$$\begin{cases} g_1(\vec{x}) \text{ se } R_1(\vec{x}) \\ g_2(\vec{x}) \text{ se } R_2(\vec{x}) \\ \qquad\vdots \\ g_p(\vec{x}) \text{ se } R_p(\vec{x}) \end{cases}$$

é recursiva primitiva.

Demonstração. Se $K_{R_i}(\vec{x}) = 1$, então todas as outras funções características produzem 0, portanto fazemos $f(\vec{x}) = g_1(\vec{x}) \cdot K_{R_1}(\vec{x}) + \ldots + g_p(\vec{x}) \cdot K_{R_p}(\vec{x})$. □

Os números naturais são bem-ordenados, o que significa dizer que cada subconjunto não-vazio tem um elemento mínimo. Se pudermos testar o subconjunto com relação a pertinência, então podemos sempre encontrar esse elemento mínimo efetivamente. Isso é tornado preciso para conjuntos recursivos primitivos.

Um pouco de notação: $(\mu y)R(\vec{x}, y)$ representa o menor número y tal que $R(\vec{x}, y)$ caso exista pelo menos um. $(\mu y < m)R(\vec{x}, y)$ representa o menor número $y < m$ tal que $R(\vec{x}, y)$ se um tal número existe; senão, simplesmente tomamo-lo como sendo m.

Lema 7.1.9 (minimização limitada) R *é recursiva primitiva* $\Rightarrow (\mu y < m)R(\vec{x}, y)$ *é recursiva primitiva.*

Demonstração. Considere a seguinte tabela:

R	$R(\vec{x}, 0)$,	$R(\vec{x}, 1)$,	$\ldots$,	$R(\vec{x}, i)$,	$R(\vec{x}, i+1)$,	$\ldots$,	$R(\vec{x}, m)$
K_R	0	0	$\ldots$	1	0	$\ldots$	1
g	0	0	$\ldots$	1	1	$\ldots$	1
h	1	1	$\ldots$	0	0	$\ldots$	0
f	1	2	$\ldots$	i	i	$\ldots$	i

Na primeira linha escrevemos os valores de $K_R(\vec{x}, i)$ para $0 \leq i \leq m$; na segunda linha tornamos a seqüência monotônica, e.g. tome $g(\vec{x}, i) = sg\sum_{j=0}^{i} K_R(\vec{x}, j)$. A seguir trocamos 0 por 1 e vice-versa: $h(\vec{x}, i) = \overline{sg}(g(\vec{x}, i))$ e finalmente somamos o

$h : f(\vec{x}, i) = \sum_{j=0}^{i} h(\vec{x}, j)$. Se $R(\vec{x}, j)$ se dá pela primeira vez em i, então $f(\vec{x}, m-1) = i$, e se $R(\vec{x}, j)$ não se verifica para nenhum $j < m$, então $f(\vec{x}, m-1) = m$. Logo, $(\mu y < m)R(\vec{x}, y) = f(\vec{x}, m-1)$, e portanto *minimização limitada* produz uma função recursiva primitiva. □

Fazemos $(\mu y \leq m)R(\vec{x}, y) := (\mu y < m+1)R(\vec{x}, y)$.

Agora é hora de aplicar nosso arsenal de técnicas para obter uma grande variedade de relações e funções recursivas primitivas.

Teorema 7.1.10 *São recursivas primitivas as seguintes:*

1. *O conjunto de primos:*

   $Primo(x) \Leftrightarrow x$ *é um primo* $\Leftrightarrow x \neq 1 \wedge \forall yz \leq x(x = yz \rightarrow y = 1 \vee z = 1)$.

2. *A relação de divisibilidade:*

   $x|y \Leftrightarrow \exists z \leq y(x \cdot z = y)$

3. *O expoente do primo p na fatoração de x:*

   $(\mu y \leq x)(p^y|x \wedge \neg p^{y+1}|x)$

4. *A função 'n-ésimo primo':*

$$\begin{cases} p(0) = 2 \\ p(n+1) = (\mu x \leq p(n)^n)(x \text{ é primo } \wedge x > p(n)). \end{cases}$$

Note que começamos a contar os números primos a partir de zero, e usamos a notação $p_n = p(n)$. Portanto $p_0 = 2$, $p_1 = 3$, $p_2 = 5$, ... O primeiro primo é p_0, e o i-ésimo primo é p_{i-1}.

Demonstração. Verifica-se facilmente que os predicados definidores são recursivos primitivos aplicando-se os teoremas acima. □

Codificação de seqüências finitas

Uma das características interessantes do sistema de números naturais é que ele permite uma codificação razoavelmente simples de pares de números, triplas de números, ..., e n-uplas em geral. Existe um bom número dessas codificações por aí, cada uma tendo seus próprios pontos fortes. As duas mais conhecidas são aquelas de Cantor e de Gödel. A codificação de Cantor é dada no exercício 6, e a codificação de Gödel será usada aqui. Ela é baseada no fato bem-conhecido de que números têm uma única (a menos da ordem) fatoração em primos.

A idéia é associar a uma seqüência $(n_0, \ldots, n_{k-1})$ o número $2^{n_0+1} \cdot 3^{n_1+1} \cdot \ldots \cdot p_i^{n_i+1} \cdot \ldots \cdot p_{k-1}^{n_{k-1}+1}$. O $+1$ extra nos expoentes é para levar em conta que a codificação tem que mostrar os zeros que ocorrem numa seqüência. Da fatoração prima de uma seqüência codificada podemos efetivamente extrair a seqüência original. A forma como introduzimos esses códigos faz com que a codificação infelizmente não seja uma bijeção; por exemplo, 10 não é uma seqüência codificada, enquanto que 6 é. Isso não é uma limitação terrível; há remédios, que não consideraremos aqui.

Lembremos que, no arcabouço da teoria dos conjuntos uma seqüência de comprimento n é um mapeamento de $\{0, \ldots, n-1\}$ para $\mathbb{N}$, portanto definimos a *seqüência vazia* como a única seqüência de comprimento 0, i.e. o único mapa de $\emptyset$ para $\mathbb{N}$, que é a função (i.e. conjunto) vazia. Fazemos com que o código da seqüência vazia seja 1.

Definição 7.1.11 1. $Seq(n) := \forall p, q \leq n(Primo(p) \wedge Primo(q) \wedge q < p \wedge p|n \rightarrow q|n) \wedge n \neq 0$. ***(*número de seqüência*)***

Em palavras: n é um número de seqüência se ele é o produto de potências primas positivas consecutivas.

2. $comp(n) := (\mu x \leq n+1)(\neg p_x|n)$ ***(*comprimento*)***

3. $(n)_i = (\mu x < n)(p_i^x|n \wedge \neg(p_i^{x+1}|n)) \dot{-} 1$ ***(*decodificação *ou* projeção*)***

Em palavras: o expoente do i-ésimo primo na fatoração de n, menos 1. $(n)_i$ extrai o i-ésimo elemento da seqüência.

4. $n * m = n \cdot \prod_{i=0}^{comp(m)-1} p_{comp(n)+i}^{(m)_i+1}$ ***(*concatenação*)***

Em palavras: se m, n forem os códigos de duas seqüências $\vec{m}, \vec{n}$, então o código da concatenação de $\vec{m}$ e $\vec{n}$ é obtido pelo produto de n e as potências primas que se obtém 'subindo' todos os primos na fatoração de m pelo fator correspondente ao comprimento de n.

Observação: 1 é trivialmente um número de seqüência. A função comprimento só produz a saída correta para números de seqüência, e.g. $comp(10) = 1$. Além do mais, o comprimento de 1 é de fato 0, e o comprimento de uma 1-upla é 1.

Notação. Usaremos abreviações para as funções de decodificação iteradas: $(n)_{i,j} = ((n)_i)_j$, etc.
Números de seqüência são, de agora em diante, escritos como $\langle n_0, \ldots, n_{k-1} \rangle$. Daí, por exemplo, $\langle 5, 0 \rangle = 2^6 \cdot 3^1$. Escrevemos $\langle\,\rangle$ para representar o código da seqüência vazia. A codificação binária, $\langle x, y \rangle$, é usualmente chamada de uma *função de pareamento*.

Até agora usamos uma forma imediata de recursão, cada saída seguinte depende dos parâmetros e da saída anterior. Mas já a *seqüência de Fibonacci* nos mostra que existem mais formas de recursão que ocorrem na prática:

$$\begin{cases} F(0) = 1 \\ F(1) = 1 \\ F(n+2) = F(n) + F(n+1) \end{cases}$$

A generalização óbvia é uma função, onde cada saída depende dos parâmetros e de todas as saídas anteriores. Isso é chamado de *recursão por curso de valores*.

Definição 7.1.12 *Para uma dada função $f(y, \vec{x})$ sua função 'curso de valores' $\bar{f}(y, \vec{x})$ é dada por*

$$\begin{cases} \bar{f}(0, \vec{x}) = 1 \\ \bar{f}(y+1, \vec{x}) = \bar{f}(y, \vec{x}) \cdot p_y^{f(y,\vec{x})+1} \end{cases}$$

Exemplo: Se $f(0) = 1$, $f(1) = 0$, $f(2) = 7$, então $\bar{f}(0) = 1$, $\bar{f}(1) = 2^{1+1}$, $\bar{f}(2) = 2^{1+1} \cdot 3^1$, $\bar{f}(3) = 2^2 \cdot 3 \cdot 5^8 = \langle 1, 0, 7 \rangle$.

Lema 7.1.13 *Se f é recursiva primitiva, então $\bar{f}$ também o é.*

Demonstração. Óbvia. □

Como $\bar{f}(n+1)$ 'codifica', por assim dizer, toda a informação sobre f até o n-ésimo valor, podemos usar $\bar{f}$ para formular a recursão por curso-de-valor.

Teorema 7.1.14 *Se g é recursiva primitiva e $f(y,\vec{x}) = g(\bar{f}(y,\vec{x}),y,\vec{x})$, então f é recursiva primitiva.*

Demonstração. Primeiro definimos $\bar{f}$.

$$\begin{cases} \bar{f}(0,\vec{x}) = 1 \\ \bar{f}(y+1,\vec{x}) = \bar{f}(y,\vec{x}) * \langle g(\bar{f}(y,\vec{x}),y,\vec{x})\rangle. \end{cases}$$

$\bar{f}$ é obviamente recursiva primitiva. Como $f(y,\vec{x}) = (\bar{f}(y+1,\vec{x}))_y$ vemos que f é recursiva primitiva. □

Nesse ponto coletamos fatos suficientes para uso futuro sobre as funções recursivas primitivas. Poderíamos perguntar se existem mais algoritmos que apenas as funções recursivas primitivas. A resposta vem a ser sim. Considere a seguinte construção: cada função recursiva primitiva f é determinada por sua definição, que consiste de uma cadeia de funções $f_0, f_1, \ldots, f_{n-1} = f$ tal que cada função é uma função inicial, ou é obtida de funções anteriores por meio de substituição ou recursão primitiva.

É uma questão de rotina codificar toda a definição num número natural tal que toda a informação pode ser efetivamente extraída do código (veja [Hinman], p.34). A construção mostra que podemos definir uma função F tal que $F(x,y) = f_x(y)$, onde f_x é a função recursiva primitiva com código x. Agora considere $D(x) = F(x,x)+1$. Suponha que D seja recursiva primitiva, portanto $D = f_n$ para um certo n, mas então $D(n) = F(n,n) + 1 = f_n(n) + 1 \neq f_n(n)$. Contradição. Está claro, entretanto, da definição de D que ela é efetiva, portanto indicamos como obter uma função efetiva que não é recursiva primitiva. Ao resultado acima pode também ser dada a seguinte formulação: não existe função recursiva primitiva binária $F(x,y)$ tal que cada função recursiva primitiva unária é $F(n,y)$ para algum n. Em outras palavras, as funções recursivas primitivas não podem ser recursivo-primitivamente enumeradas.

O argumento é, na verdade, completamente geral; suponha que tenhamos uma classe de funções efetivas que pode se enumerar a si própria da maneira considerada acima; então podemos sempre "diagonalizar para fora da classe" pela função D. Chamamos isso '*diagonalização*'. O moral dessa observação é que temos pouca esperança de obter *todas* as funções efetivas de uma maneira efetiva. A técnica da diagonalização vai lá atrás para Cantor, que a introduziu para mostrar que os reais não são enumeráveis. Em geral ele usou diagonalização para mostrar que a cardinalidade de um conjunto é menor que a cardinalidade de seu conjunto das partes.

Exercícios

1. Se h_1 e h_2 forem recursivas primitivas, então o mesmo acontece com f e g, onde

$$\begin{cases} f(0) &= a_1 \\ g(0) &= a_2 \\ f(x+1) &= h_1(f(x),g(x),x) \\ g(x+1) &= h_2(f(x),g(x),x) \end{cases}$$

2. Mostre que a série de Fibonacci é recursiva primitiva, onde

$$\begin{cases} f(0) = f(1) = 1 \\ f(x+2) = f(x) + f(x+1) \end{cases}$$

3. Suponha que $[a]$ denote a parte inteira do número real a (i.e. o maior inteiro $\leq a$). Mostre que $\left[\frac{x}{y+1}\right]$, para números naturais x e y, é recursiva primitiva.

4. Mostre que $\max(x, y)$ e $\min(x, y)$ são recursivas primitivas.

5. Mostre que mdc (máximo divisor comum) e mmc (mínimo múltiplo comum) são recursivas primitivas.

6. A função de pareamento de Cantor é dada por $P(x, y) = \frac{1}{2}((x+y)^2 + 3x + 2y)$. Mostre que P é recursiva primitiva, e que P é uma bijeção de $\mathbb{N}^2$ sobre $\mathbb{N}$. (Sugestão: Considere no plano uma caminhada sobre todos os pontos de um reticulado da seguinte forma: $(0,0) \to (0,1) \to (1,0) \to (0,2) \to (1,1) \to (2,0) \to (0,3) \to (1,2) \to \ldots$). Defina as 'inversas' E e D tais que $P(E(z), D(z)) = z$ e mostre que elas são recursivas primitivas.

7. Mostre que $p_n \leq 2^{2^n}$. Para mais informações sobre limitantes sobre p_n veja *Smoryński 1980*.

7.2 Funções Recursivas Parciais

Dado o fato de que as funções recursivas primitivas não esgotam os algoritmos numéricos, estendemos de uma maneira natural a classe das funções efetivas. Como vimos que uma geração efetiva de todos os algoritmos invariavelmente nos traz para um conflito com a diagonalização, alargaremos nosso escopo permitindo funções parciais. Dessa maneira a situação conflitante $D(n) = D(n) + 1$ para um certo n nos diz que D não é definida para n.

No contexto presente funções têm domínios naturais, i.e. conjuntos da forma $\mathbb{N}^n$ ($= \{(m_0, \ldots, m_{n-1}) \mid m_i \in \mathbb{N}\}$, assim-chamados produtos cartesianos), uma função parcial tem um domínio que é um subconjunto de $\mathbb{N}^n$. Se o domínio é todo o conjunto $\mathbb{N}^n$, então chamamos a função de *total*.

Exemplo: $f(x) = x^2$ é total, $g(x) = \mu y(y^2 = x)$ é parcial e não total, ($g(x)$ é a raiz quadrada de x se ela existe).

Os algoritmos que vamos introduzir são chamados *funções recursivas parciais*; talvez funções parciais recursivas teria sido uma melhor denominação. Entretanto, o nome veio a ser sendo geralmente aceito. A técnica específica para definir funções recursivas parciais que empregamos aqui vai lá atrás até Kleene. Tal qual antes, usamos uma definição indutiva; exceto a cláusula R7 adiante, poderíamos ter usado uma formulação quase idêntica àquela da definição das funções recursivas primitivas. Como desejamos uma função universal 'interna', ou seja, uma função que efetivamente enumera as funções, temos que empregar uma técnica mais refinada que permite referência explícita a vários algoritmos. O truque não é de forma alguma esotérico, simplesmente atribuímos um código numérico a cada algoritmo, que chamamos seu *índice*. Fixamos tais índices antecipadamente de modo que podemos falar do 'algoritmo com índice e

produz como saída y quando recebe como entrada $(x_0, \ldots, x_{n-1})$', simbolicamente representado como $\{e\}(x_0, \ldots, x_{n-1}) \simeq y$.

A heurística desse 'índice aplicado à entrada' é que um índice é visto com uma descrição de uma máquina abstrata que opera sobre entradas de uma aridade fixa. Portanto $\{e\}(\vec{n}) \simeq m$ deve ser lido como 'a máquina com índice e opera sobre $\vec{n}$ e produz como saída m'. Pode muito bem ser o caso que a máquina não produz uma saída, e nesse caso dizemos que $\{e\}(\vec{n})$ diverge. Se existe uma saída, dizemos que $\{e\}(\vec{n})$ converge. Que a máquina abstrata é um algoritmo vai se tornar aparente da especificação na definição abaixo.

Note que não sabemos antecipadamente que o resultado é uma função, i.e. que para cada entrada existe no máximo uma saída. Independentemente do quão plausível isso seja, tem que haver uma demonstração. Kleene introduziu o símbolo $\simeq$ para 'igualdade' em contextos onde termos podem resultar em valores indefinidos. Isso acontece de ser útil no estudo de algoritmos que não necessariamente precisam produzir uma saída. As máquinas abstratas acima podem, por exemplo, entrar numa computação que executa para sempre. Por exemplo, poderia haver uma instrução da forma 'a saída em n é o sucessor da saída em $n + 1$'. É fácil ver que para nenhum n uma saída pode ser obtida. Nesse contexto o uso do predicado de existência seria útil, e $\simeq$ seria o $\equiv$ da teoria de objetos parciais (cf. *Troelstra–van Dalen*, 2.2). A convenção regulando $\simeq$ é: se $t \simeq s$ então t e s têm valores simultaneamente definidos e são idênticos, ou eles têm valores simultaneamente indefinidos.

Definição 7.2.1 *A relação $\{e\}(\vec{x}) \simeq y$ é indutivamente definida por*

R1 $\{\langle 0, n, q\rangle\}(m_0, \ldots, m_{n-1}) \simeq q$

R2 $\{\langle 1, n, i\rangle\}(m_0, \ldots, m_{n-1}) \simeq m_i$, *para* $0 \leq i < n$

R3 $\{\langle 2, n, i\rangle\}(m_0, \ldots, m_{n-1}) \simeq m_i + 1$, *para* $0 \leq i < n$

R4 $\{\langle 3, n + 4\rangle\}(p, q, r, s, m_0, \ldots, m_{n-1}) \simeq p$, *se* $r = s$
$\{\langle 3, n + 4\rangle\}(p, q, r, s, m_0, \ldots, m_{n-1}) \simeq q$, *se* $r \neq s$

R5 $\{\langle 4, n, b, c_0, \ldots, c_{k-1}\rangle\}(m_0, \ldots, m_{n-1}) \simeq p$, *se existem* $q_0, \ldots, q_{k-1}$ *tais que* $\{c_i\}(m_0, \ldots, m_{n-1}) \simeq q_i$ $(0 \leq i < k)$ *e* $\{b\}(q_0, \ldots, q_{k-1}) \simeq p$

R6 $\{\langle 5, n + 2\rangle\}(p, q, m_0, \ldots, m_{n-1}) \simeq S^1_n(p, q)$

R7 $\{\langle 6, n + 1\rangle\}(b, m_0, \ldots, m_{n-1}) \simeq p$ *se* $\{b\}(m_0, \ldots, m_{n-1}) \simeq p$.

A função S^1_n em R6 será especificada no teorema S^m_n adiante.

Mantendo em mente a leitura acima de $\{e\}(\vec{x})$, podemos parafrasear o esquema da seguinte forma:

R1 a máquina com índice $\langle 0, n, q\rangle$ produz para a entrada $(m_0, \ldots, m_{n-1})$ a saída q (a *função constante*),

R2 a máquina com índice $\langle 1, n, i\rangle$ produz para a entrada $\vec{m}$ a saída m_i (a *função de projeção* P^n_i),

R3 a máquina com índice $\langle 2, n, i\rangle$ produz para a entrada $\vec{m}$ a saída $m_i + 1$ (a *função sucessor* sobre o i-ésimo argumento),

R4 a máquina com índice $\langle 3, n+4\rangle$ testa a igualdade do terceiro e do quarto argumento da entrada e produz o primeiro argumento no caso da igualdade, e o segundo argumento caso contrário (a *função discriminadora*),

R5 a máquina com índice $\langle 4, n, b, c_0, \ldots, c_{k-1}\rangle$ primeiro simula as máquinas com índices $c_0, \ldots, c_{k-1}$ com entrada $\vec{m}$, e então usa a seqüência de saída $(q_0, \ldots, q_{k-1})$ como entrada e simula as máquinas com índice b (*substituição*),

R7 a máquina com índice $\langle 6, n+1\rangle$ simula para uma dada entrada $b, m_0, \ldots, m_{n-1}$, a máquina com índice b e entrada $m_0, \ldots, m_{n-1}$ (*reflexão*).

Uma outra maneira de ver R7 é que ela provê uma *máquina universal* para todas as máquinas com entradas de n-argumentos, o que quer dizer que ela aceita como uma entrada os índices de máquinas, e aí as simula. Isso é o tipo de máquina requerida para o processo de diagonalização. Se se pensa em máquinas abstratas idealizadas, então R7 é bastante razoável. Esperar-se-ia que se os índices podem ser 'decifrados', uma máquina universal pode ser construída. Isso foi de fato realizado por Alan Turing, que construiu (abstratamente) uma assim-chamada máquina de Turing universal.

O leitor escrupuloso poderia chamar R7 de um caso de trapaça, pois ela se livra de todo o trabalho árduo que se tem que fazer de modo a obter uma máquina universal, por exemplo, no caso das máquinas de Turing.

Como $\{e\}(\vec{x}) \simeq y$ é indutivamente definido, tudo que provamos sobre conjuntos indutivamente definidos se aplica aqui. Por exemplo, se $\{e\}(\vec{x}) \simeq y$ for o caso, então sabemos que existe uma seqüência de formação (ver página 9) para ele. Essa seqüência especifica como $\{e\}$ é construída a partir de funções recursivas parciais mais simples. Note que poderíamos também ter visto a definição acima como uma definição indutiva do conjunto de índices (de funções recursivas parciais).

Lema 7.2.2 *A relação $\{e\}(\vec{x}) \simeq y$ é funcional.*

Demonstração. Temos que mostrar que $\{e\}$ se comporta como uma função, ou seja, $\{e\}(\vec{x}) \simeq y$, $\{e\}(\vec{x}) \simeq z \Rightarrow y = z$. Isso é feito por indução sobre a definição de $\{e\}$. Deixamos a prova ao leitor. □

A definição de $\{e\}(\vec{n}) \simeq m$ tem um conteúdo computacional, pois nos diz o que fazer. Quando apresentados com $\{e\}(\vec{n})$, primeiro olhamos para e; se a primeira 'entrada' de e for 0, 1 ou 2, computamos a saída via a função inicial correspondente. Se a primeira 'entrada' for 3, determinamos a saída 'por casos'. Se a primeira entrada for 4, primeiro realizamos as subcomputações indicadas por $\{c_i\}(\vec{m})$, e então usamos as saídas para conduzir as subcomputações para $\{b\}(\vec{n})$. E assim por diante.

Se R7 for usada em tal composição, não temos mais garantia de que a computação vai parar; de fato, podemos cair num laço, como mostra o exemplo a seguir.

De R7 segue, como veremos adiante, que existe um índice e tal que $\{e\}(x) \simeq \{x\}(x)$. Para computar $\{e\}$ para o argumento e passamos, conforme R7, para o lado direito, i.e. temos que computar $\{e\}(e)$, e como e foi introduzido por R7, temos que repetir as transições para o lado direito, etc. Evidentemente que nosso procedimento não nos leva a lugar algum!

Convenções. A relação $\{e\}(\vec{x}) \simeq y$ define uma função sobre um domínio, que é um subconjunto do 'domínio natural', i.e. um conjunto da forma $\mathbb{N}^n$. Tais funções são chamadas *funções recursivas parciais*; elas são tradicionalmente denotadas por

símbolos do alfabeto grego, φ, ψ, σ, etc. Se tal função é total sobre seu domínio natural, ela é chamada de *recursiva*, e denotada por um símbolo romano, f, g, h, etc. O uso do símbolo de igualdade '$=$' é próprio no contexto de funções totais. Na prática, no entanto, quando não houver confusão, usá-la-emos freqüentemente ao invés de '$\simeq$'. O leitor deve tomar cuidado para não confundir fórmulas e funções recursivas parciais; ficará sempre claro do contexto o que o símbolo representa. Conjuntos e relações serão denotados por letras romanas maiúsculas. Quando nenhuma confusão pode ocorrer, às vezes omitiremos parênteses, como em $\{e\}x$ para $\{e\}(x)$. Alguns autores usam uma notação tipo 'bolinha preta' para funções recursivas parciais: $e \bullet \vec{x}$. Insistiremos nos 'parênteses de Kleene': $\{e\}(\vec{x})$.

A terminologia abaixo é tradicionalmente usada em teoria da recursão:

Definição 7.2.3 *1. Se para uma função parcial φ $\exists y(\varphi(\vec{x}) = y)$, então dizemos que φ converge em $\vec{x}$, caso contrário φ diverge em $\vec{x}$.*

2. Se uma função parcial converge para todas as entradas (próprias), ela é chamada de total*.*

3. Uma função recursiva parcial total (sic!) será chamada de função recursiva*.*

4. Um conjunto (relação) é chamado de recursivo *se sua função característica (que, por definição, é total) é recursiva.*

Observe que é uma importante característica de computações conforme definidas na Definição 7.2.1, que $\{e\}(\psi_0(\vec{n}), \psi_1(\vec{n}), \ldots, \psi_{k-1}(\vec{n}))$ diverge se um de seus argumentos $\psi_i(\vec{n})$ diverge. Daí, por exemplo, a função recursiva parcial $\{e\}(x) - \{e\}(x)$ pode não convergir para todo e e x, pois primeiro temos que saber que $\{e\}(x)$ converge!

Essa característica é às vezes inconveniente e levemente paradoxal, e.g. em aplicações diretas do esquema discriminador R4, $\{\langle 3,4\rangle\}(\varphi(x), \psi(x), 0, 0)$ tem valor indefinido quando a função (aparentemente irrelevante) $\psi(x)$ tem valor indefinido.

Com um pouco de trabalho extra, podemos obter um índice para uma função recursiva parcial que faz *definição por casos* sobre funções recursivas parciais:

$$\{e\}(\vec{x}) = \begin{cases} \{e_1\}(\vec{x}) \text{ se } g_1(\vec{x}) = g_2(\vec{x}) \\ \{e_2\}(\vec{x}) \text{ se } g_1(\vec{x}) \neq g_2(\vec{x}) \end{cases} \qquad \text{para } g_1, g_2 \text{ recursivas.}$$

Defina

$$\varphi(\vec{x}) = \begin{cases} e_1 \text{ se } g_1(\vec{x}) = g_2(\vec{x}) \\ e_2 \text{ se } g_1(\vec{x}) \neq g_2(\vec{x}) \end{cases}$$

por $\varphi(\vec{x}) = \{\langle 3,4\rangle\}(e_1, e_2, g_1(\vec{x}), g_2(\vec{x}))$. Portanto $\{e\}(\vec{x}) = \{\psi(\vec{x})\} =$ [por R7] $\{\langle 6, n+1\rangle\}(\psi(\vec{x}), \vec{x})$. Agora use R5 (substituição) para obter o índice requerido.

Como as funções recursivas primitivas formam tal classe natural de algoritmos, nosso primeiro objetivo será mostrar que elas estão contidas na classe das funções recursivas.

O teorema que vem a seguir tem uma bela motivação de máquina. Considere uma máquina com índice e operando sobre dois argumentos x e y. Mantendo x fixo, temos uma máquina operando sobre y. Portanto obtemos uma seqüência de máquinas, uma para cada x. O índice de cada máquina dessas depende, de uma maneira decente, de x? A resposta plausível parecer ser 'sim'. O teorema a seguir confirma isso.

Teorema 7.2.4 (O Teorema S_n^m) *Para todo m, n com $0 < m < n$ existe uma função recursiva primitiva S_n^m tal que*

$$\{S_n^m(e, x_0, \ldots, x_{m-1})\}(x_m, \ldots, x_{n-1}) = \{e\}(\vec{x}).$$

Demonstração. A primeira função, S_n^1, ocorre em R6, e havíamos adiado sua definição precisa, que aqui está:

$$S_n^1(e, y) = \langle 4, (e)_1 \dot{-} 1, e, \langle 0, (e)_1 \dot{-} 1, y\rangle, \langle 1, (e)_1 \dot{-} 1, 0\rangle, \ldots, \langle 1, (e)_1 \dot{-} 1, n \dot{-} 2\rangle\rangle.$$

Note que as aridades estão corretas, $\{e\}$ tem um argumento a mais que a função constante e as funções de projeção envolvidas.

Agora, $\{S_n^1(e, y)\}(\vec{x}) = z \Leftrightarrow$ existem $q_0, \ldots, q_{n-1}$ tais que

$$\begin{aligned}
&\{\langle 0, (e)_1 \dot{-} 1, y\rangle\}(\vec{x}) = q_0\\
&\{\langle 1, (e)_1 \dot{-} 1, 0\rangle\}(\vec{x}) = q_1\\
&\ldots\\
&\{\langle 1, (e)_1 \dot{-} 1, n \dot{-} 2\rangle\}(\vec{x}) = q_{n-1}\\
&\{e\}(q_0, \ldots, q_{n-1}) = z.
\end{aligned}$$

Pelas cláusulas R1 e R2 obtemos $q_0 = y$ e $q_{i+1} = x_i$ $(0 \leq i \leq n-1)$, portanto $\{S_n^1(e, y)\}(\vec{x}) = \{e\}(y, \vec{x})$. Claramente, S_n^1 é recursiva primitiva.

A função recursiva primitiva S_n^m é obtida aplicando-se S_n^1 m vezes. Da nossa definição segue que S_n^m é também recursiva. □

A função S_n^m nos permite considerar algumas entradas como parâmetros, e o restante como entradas própriamente ditas. Essa é uma consideração de rotina no cotidiano da matemática: 'considere $f(x, y)$ como uma função de y'. A notação lógica para essa especificação de entradas faz uso do *operador lambda*. Digamos que $t(x, y, z)$ seja um termo (em alguma linguagem), então $\lambda x \cdot t(x, y, z)$ é para cada escolha de y, z a função $x \mapsto t(x, y, z)$. Dizemos que y e z são parâmetros nessa função. O cálculo do valor desses termos lambda é simples: $\lambda x \cdot t(x, y, z)(n) = t(n, y, z)$. Esse tópico pertence ao chamado *lambda-cálculo*, e para nós a notação é apenas uma ferramenta conveniente para nos expressarmos sucintamente.

O teorema S_n^m expressa uma propriedade de *uniformidade* das funções recursivas parciais. É de fato óbvio que, digamos para uma função recursiva parcial $\varphi(x, y)$, cada indivíduo $\varphi(n, y)$ seja recursivo parcial (substitua x pela função constante n), mas isso ainda não nos mostra que o índice de $\lambda y \cdot \varphi(x, y)$ seja, de uma forma sistemática, uniforme, computável a partir do índice de φ e x. Pelo teorema S_n^m, sabemos que o índice de $\{e\}(x, y, z)$, considerado como uma função de, digamos, z depende primitiva-recursivamente de x e y: $\{h(x, y)\}(z) = \{e\}(x, y, z)$. Veremos um número de aplicações do teorema S_n^m.

A seguir provaremos um poderoso teorema sobre funções recursivas parciais, que nos permite introduzir funções recursivas parciais por definições indutivas, ou por definições implícitas. Funções recursivas parciais podem, por esse teorema, ser dadas como soluções de certas equações.

Exemplo.

$$\varphi(n) = \begin{cases} 0 & \text{se } n \text{ é um primo, ou } 0, \text{ ou } 1\\ \varphi(2n+1)+1 & \text{caso contrário.}\end{cases}$$

Então $\varphi(0) = \varphi(1) = \varphi(2) = \varphi(3) = 0$, $\varphi(4) = \varphi(9) + 1 = \varphi(19) + 2 = 2$, $\varphi(5) = 0$, e, e.g. $\varphi(85) = 6$. À primeira vista, não podemos dizer muita coisa sobre tal seqüência. O teorema a seguir (de Kleene) mostra que podemos sempre encontrar uma solução recursiva parcial para uma tal equação para φ.

Teorema 7.2.5 (O Teorema da Recursão) *Existe uma função recursiva primitiva* rc *tal que para cada* e *e* $\vec{x}$, $\{\mathrm{rc}(e)\}(\vec{x}) = \{e\}(\mathrm{rc}(e), \vec{x})$.

Vamos notar primeiro que o teorema de fato dá a solução r da seguinte equação: $\{r\}(\vec{x}) = \{e\}(r, \vec{x})$. De fato, a solução depende primitiva-recursivamente do índice dado e: $\{f(e)\}(x) = \{e\}(f(e), x)$. Se não estivermos interessados na dependência (recursiva primitiva) do índice da solução em relação ao índice antigo, podemos até nos contentar com a solução de $\{f\}(x) = \{e\}(f, x)$.

Demonstração. Seja $\varphi(m, e, \vec{x}) = \{e\}(S^2_{n+2}(m, m, e), \vec{x})$ e suponha que p seja um índice de φ. Faça $\mathrm{rc}(e) = S^2_{n+2}(p, p, e)$, então

$$\begin{aligned}\{\mathrm{rc}(e)\}(\vec{x}) &= \{S^2_{n+2}(p, p, e)\}(\vec{x}) = \{p\}(p, e, \vec{x}) = \varphi(p, e, \vec{x}) \\ &= \{e\}(S^2_{n+2}(p, p, e), \vec{x}) = \{e\}(\mathrm{rc}(e), \vec{x}).\end{aligned}$$

□

Como um caso especial obtemos

Corolário 7.2.6 *Para cada* e *existe um* n *tal que* $\{n\}(\vec{x}) = \{e\}(n, \vec{x})$.

Corolário 7.2.7 *Se* $\{e\}$ *for recursiva primitiva, então a solução da equação* $\{f(e)\}(\vec{x}) = \{e\}(f(e), \vec{x})$ *dada pelo teorema da recursão também é recursiva primitiva.*

Demonstração. Imediata da definição explícita da função rc. □

Daremos um número de exemplos assim que tivermos mostrado que podemos obter todas as funções recursivas primitivas. Por agora temos um amplo estoque de funções com as quais experimentar. Primeiro temos que provar alguns teoremas a mais.

As funções recursivas parciais são fechadas sob uma forma geral de minimização, às vezes chamada de *busca ilimitada*, que para uma dada função recursiva $f(y, \vec{x})$ e argumentos $\vec{x}$ percorre todos os valores de y e procura pelo primeiro que faz com que $f(y, \vec{x})$ seja igual a zero.

Teorema 7.2.8 *Seja* f *uma função recursiva, então* $\varphi(\vec{x}) = \mu y[f(y, \vec{x}) = 0]$ *é recursiva parcial.*

Demonstração. A idéia é computar consecutivamente $f(0, \vec{x})$, $f(1, \vec{x})$, $f(2, \vec{x})$, ... até que encontremos um valor 0. Isso não precisa forçosamente acontecer, mas se acontece, chegaremos a ele. Enquanto estamos computando esses valores, contamos o número de passos. Quem cuida disso é uma função recursiva. Portanto desejamos uma função ψ com índice e, operando sobre y e $\vec{x}$, que faz o trabalho para nós, i.e. uma ψ que após computar um valor positivo para $f(y, \vec{x})$ vai para o próximo valor de entrada y e adiciona 1 ao contador. Como quase não temos ferramentas aritméticas no momento, a construção é um tanto enrolada e artificial.

Na tabela a seguir computamos $f(y, \vec{x})$ passo a passo (as saídas estão na terceira linha), e na última linha computamos $\psi(y, \vec{x})$ de trás para frente, como se fosse.

y	0	1	2	3	$\ldots$	$k-1$	k
$f(y,\vec{x})$	$f(0,\vec{x})$	$f(1,\vec{x})$	$f(2,\vec{x})$	$f(3,\vec{x})$	$\ldots$	$f(k-1,\vec{x})$	$f(k,\vec{x})$
	2	7	6	12	$\ldots$	3	0
$\psi(y,\vec{x})$	k	$k-1$	$k-2$	$k-3$	$\ldots$	1	0

$\psi(0,\vec{x})$ é o k requerido. A instrução para ψ é simples:

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

De modo a encontrar um índice para ψ, faça $\psi(y,\vec{x}) = \{e\}(y,\vec{x})$ e procure por um valor para e. Introduzimos duas funções auxiliares ψ_1 e ψ_2 com índices b e c tais que $\psi_1(e,y,\vec{x}) = 0$ e $\psi_2(e,y,\vec{x}) = \psi(y+1,\vec{x})+1$. O índice c segue facilmente aplicando-se R3, R7 e o teorema S_n^m. Se $f(y,\vec{x}) = 0$ então consideramos ψ_1, senão ψ_2. Agora introduzimos, pela cláusula R4, uma nova função χ_0 que computa um índice:

$$\chi_0(e,y,\vec{x}) = \begin{cases} b & \text{se } f(y,\vec{x}) = 0 \\ c & \text{caso contrário} \end{cases}$$

e fazemos $\chi(e,y,\vec{x}) = \{\chi_0(e,y,\vec{x})\}(e,y,\vec{x})$. O teorema da recursão nos fornece um índice e_0 tal que $\chi(e_0,y,\vec{x}) = \{e_0\}(y,\vec{x})$.

Afirmamos que $\{e_0\}(0,\vec{x})$ produz o valor desejado k, se por acaso ele existe, i.e. e_0 é o índice da ψ pelo qual estávamos procurando, e $\varphi(\vec{x}) = \{e\}(0,\vec{x})$.

Se $f(y,\vec{x}) \neq 0$ então $\chi(e,y,\vec{x}) = \{c\}(e_0,y,\vec{x}) = \psi_2(e_0,y,\vec{x}) = \psi(y+1,\vec{x})+1$, e se $f(y,\vec{x}) = 0$ então $\chi(e_0,y,\vec{x}) = \{b\}(e_0,y,\vec{x}) = 0$.

Se, por outro lado, k for o primeiro valor y tal que $f(y,\vec{x}) = 0$, então $\psi(0,\vec{x}) = \psi(1,\vec{x})+1 = \psi(2,\vec{x})+2 = \cdots = \psi(y_0,\vec{x})+y_0 = k$. □

Note que a função dada não precisa ser recursiva, e que o argumento acima também funciona para f recursiva parcial. Temos então que reformular $\mu y[f(x,\vec{y}) = 0]$ como o y tal que $f(y,\vec{x}) = 0$ e para todo $z < y$ o valor de $f(z,\vec{x})$ está definido e é positivo.

Lema 7.2.9 *A função predecessor é recursiva.*

Demonstração. Defina

$$x \dot{-} 1 = \begin{cases} 0 & \text{se } x = 0 \\ \mu y[y+1 = x] & \text{caso contrário} \end{cases}$$

onde $\mu y[y+1 = x] = \mu y[f(y,x) = 0]$ com

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

□

Teorema 7.2.10 *As funções recursivas são fechadas sob recursão primitiva.*

Demonstração. Sejam g e h recursivas, e suponha que f seja dada por

$$\begin{cases} f(0,\vec{x}) = g(\vec{x}) \\ f(y+1,\vec{x}) = h(f(y,\vec{x}),\vec{x},y) \end{cases}$$

Reescrevemos o esquema como

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

No lado direito temos uma definição por casos. Portanto, ela define uma função recursiva parcial com índice, digamos, a de $y, \vec{x}$ e o índice e da função f que estamos procurando. Isso leva a uma equação $\{e\}(y,\vec{x}) = \{a\}(y,\vec{x},e)$. Pelo teorema da recursão a equação tem uma solução e_0. Uma indução fácil sobre y mostra que $\{e_0\}$ é total, portanto f é uma função recursiva. □

Obtemos agora o obrigatório

Corolário 7.2.11 *Todas as funções recursivas primitivas são recursivas.*

Agora que recuperamos as funções recursivas primitivas, podemos obter muitas funções recursivas parciais.

Exemplos.

1. Defina $\varphi(x) = \{e\}(x) + \{f\}(x)$, então, por 7.2.11 e R5, φ é recursiva parcial e gostaríamos de expressar o índice de φ como uma função de e e f. Considere $\psi(e,f,x) = \{e\}(x) + \{f\}(x)$. ψ é recursiva parcial, portanto ela tem um índice n, i.e. $\{n\}(e,f,x) = \{e\}(x) + \{f\}(x)$. Pelo teorema S_n^m existe uma função recursiva primitiva h tal que $\{n\}(e,f,x) = \{h(n,e,f)\}(x)$. Por conseguinte, $g(e,f) = h(n,e,f)$ é a função requerida.

2. Existe uma função recursiva parcial φ tal que $\varphi(n) = (\varphi(n+1)+1)^2$: Considere $\{z\}(n) = \{e\}(z,n) = (\{z\}(n+1)+1)^2$. Pelo teorema da recursão existe uma solução $\mathrm{rc}(e)$ para z, logo φ existe. Um argumento simples mostra que φ não pode estar definida para nenhum n, portanto a solução é a função vazia (a máquina que nunca dá uma saída).

3. A *função de Ackermann*, ver [Smorynski 1991], p.70. Considere a seguinte seqüência de funções:

$$\begin{aligned} &\varphi_0(m,n) = n + m \\ &\varphi_1(m,n) = n \cdot m \\ &\varphi_2(m,n) = n^m \\ &\qquad\vdots \\ &\begin{cases} \varphi_{k+1}(0,n) & = n \\ \varphi_{k+1}(m+1,n) & = \varphi_k(\varphi_{k+1}(m,n),n) \quad (k \geq 2) \end{cases} \end{aligned}$$

   Essa seqüência consiste de funções mais e mais rapidamente crescentes. Podemos juntar todas essas funções numa função:

$$\varphi(k,m,n) = \varphi_k(m.n).$$

   As equações acima podem ser resumidas:

$$\begin{cases} \varphi(0,m,n) = n + m \\ \varphi(k+1,0,n) = \begin{cases} 0 & \text{se } k = 0 \\ 1 & \text{se } k = 1 \\ n & \text{caso contrário} \end{cases} \\ \varphi(k+1,m+1,n) = \varphi(k,\varphi(k+1),m,n),n). \end{cases}$$

Note que a segunda equação tem que distinguir casos de acordo com φ_{k+1} ser a multiplicação, a exponenciação, ou o caso geral ($k \geq 2$). Usando o fato de que todas as funções recursivas primitivas são recursivas, reescrevemos os três casos numa equação da forma $\{e\}(k, m, n) = f(e, k, m, n)$ para uma f recursiva adequada (exercício 3). Daí, pelo teorema da recursão, existe uma função recursiva com índice e que satisfaz as equações acima. Ackermann mostrou que a função $\varphi(n, n, n)$ cresce, a partir de um certo momento, mais rapidamente que qualquer função recursiva primitiva.

4. O teorema da recursão pode também ser usado para definições indutivas de conjuntos ou relações; isso é visto mudando para funções características, e.g. suponha que desejemos uma relação $R(x, y)$ tal que

$$R(x, y) \Leftrightarrow (x = 0 \wedge y \neq 0) \vee ((x \neq 0 \wedge y \neq 0) \wedge R(x \dot{-} 1, y \dot{-} 1)).$$

Então escrevemos

$$K_R(x, y) = sg(\overline{sg}(x) \cdot sg(y) + sg(x) \cdot sg(y) \cdot K_R(x \dot{-} 1, y \dot{-} 1)),$$

de modo que existe um e tal que

$$K_R(x, y) = \{e\}(K_R(x \dot{-} 1, y \dot{-} 1), x, y).$$

Agora suponha que K_R tenha índice z, então temos

$$\{z\}(x, y) = \{e'\}(z, x, y).$$

A solução $\{n\}$ como é dada pelo teorema da recursão é a função característica requerida. Vê-se imediatamente que R é a relação 'menor que'. Por conseguinte, $\{n\}$ é total, e portanto recursiva; isso mostra que R é também recursiva. Note que pela observação seguinte ao teorema da recursão, obtemos até mesmo a recursividade primitiva de R.

O teorema fundamental a seguir é extremamente útil para muitas aplicações. Sua importância teórica é que ele mostra que todas as funções recursivas parciais podem ser obtidas a partir de uma relação recursiva primitiva por *uma* minimização.

Portanto minimização é a ligação que faltava entre recursiva primitiva e recursiva (parcial).

Teorema 7.2.12 (Teorema da Forma Normal) *Existe um predicado recursivo primitivo T tal que $\{e\}(\vec{x}) = ((\mu z)T(e, \langle\vec{x}\rangle, z))_1$.*

Demonstração. Nossa heurística para funções recursivas parciais foi baseada na metáfora da máquina: pense numa máquina abstrata com ações prescritas pelas cláusulas R1 a R7. Ao reconstituir o índice e de uma dessas máquinas, damos por assim dizer um procedimento de computação. Agora é um trabalho braçal especificar todos os passos envolvidos em tal 'computação'. Uma vez que tivermos chegado a isso, teremos tornado nossa noção de 'computação' precisa, e da forma da especificação, podemos imediatamente concluir que "c é o código de uma computação"é de fato um predicado recursivo primitivo. Buscamos por um predicado $T(e, u, z)$ que formaliza o enunciado heurístico 'z é uma computação (codificada) que é realizada por uma função recursiva parcial com índice e sobre a entrada u' (i.e. $\langle\vec{x}\rangle$). A 'computação' foi arranjada de tal maneira que a primeira projeção de z é sua saída.

A prova é uma questão de perseverança burocrática—não difícil, mas também não empolgante. Para o leitor é melhor destrinchar alguns poucos casos por si só e deixar o restante, do que explicitar todos os detalhes seguintes.

Primeiro, dois exemplos.

1. A função sucessor aplicada a $(1, 2, 3)$:

   $S_1^3(1, 2, 3) = 2 + 1 = 3$. Uma advertência: aqui S_1^3 é usada para a função sucessor operando sobre o segundo item da cadeia de entrada de comprimento 3. A notação é usada somente aqui.

   O índice é $e = \langle 2, 3, 1\rangle$, a entrada é $u = \langle 1, 2, 3\rangle$, e o passo é a computação direta $z = \langle 3, \langle 1, 2, 3\rangle, \langle 2, 3, 1\rangle\rangle = \langle 3, u, e\rangle$.

2. A composição da função constante com a função de projeção.

   $P_2^3(C_0^2(7, 0), 5, 1) = 1$.

   Pela cláusula R5, a entrada dessa função tem que ser uma cadeia de números, portanto temos que introduzir uma entrada apropriada. A solução mais simples é usar $(7, 0)$ como entrada e 'fabricar' os argumentos remanescentes 5 e 1 a partir deles. Daí, vamos fazer

   $P_2^3(C_0^2(7, 0), 5, 1) = P_2^3(C_0^2(7, 0), C_5^2(7, 0), C_1^2(7, 0))$.

   De modo a manter a notação legível, usaremos variáveis ao invés das entradas numéricas.

   $\varphi(y_0, y_1) = P_2^3(C_0^2(y_0, y_1), C_5^2(y_0, y_1), C_1^2(y_0, y_1)) = P_2^3(C_0^2(y_0, y_1), x_1, x_2)$.

   Vamos primeiro escrever os dados para as funções componentes:

	índice	entrada	passo
C_0^2	$\langle 0, 2, 0\rangle = e_0$	$\langle y_0, y_1\rangle = u$	$\langle 0, u, e_0\rangle = z_0$
$C_{x_1}^2$	$\langle 0, 2, x_1\rangle = e_1$	$\langle y_0, y_1\rangle = u$	$\langle x_1, u, e_1\rangle = z_1$
$C_{x_2}^2$	$\langle 0, 2, x_2\rangle = e_2$	$\langle y_0, y_1\rangle = u$	$\langle x_2, u, e_2\rangle = z_2$
P_2^3	$\langle 1, 3, 2\rangle = e_3$	$\langle 0, x_1, x_2\rangle = u'$	$\langle x_2, u', e_3\rangle = z_3$

   Agora a composição:

	índice	entrada	passo
$f(y_0, y_1)$	$\langle 4, 2, e_3, e_0, e_1, e_2\rangle = e$	$\langle y_0, y_1\rangle = u$	$\langle x_2, \langle y_0, y_1\rangle, e, z_3, \langle z_0, z_1, z_2\rangle\rangle = z$

   Como vemos nesse exemplo, 'passo' significa o último passo na cadeia de passos que leva à saída. Agora para uma computação real sobre entradas numéricas, tudo o que se tem que fazer é substituir y_0, y_1, x_1, x_2 por números e escrever os dados para $\varphi(y_0, y_1)$.

Tentamos organizar a prova de uma forma legível acrescentando sempre um comentário auxiliar.

Os ingredientes para, e as condições sobre, computações são listadas abaixo. O índice contém a informação dada nas cláusulas Ri. A computação codifica os seguintes itens:

(1) a saída (3) o índice
(2) a entrada (4) subcomputações

Note que z na tabela abaixo é o 'número mestre', i.e. podemos obter os dados remanescentes a partir de z, e.g. $e = (z)_2$, $comp(u) = (e)_1 = (z)_{2,1}$, e a saída (se houver alguma) da computação, $(z)_0$. Em particular, podemos extrair os 'números mestre' das subcomputações. Portanto, ao decodificar o código para uma computação, podemos efetivamente encontrar os códigos para as subcomputações, etc. Isso sugere um algoritmo recursivo primitivo para a extração da 'história' total de uma computação a partir do seu código. Na realidade, isso é essencialmente o conteúdo do teorema da forma normal.

	Índice	Entrada	Passo	Condições sobre Subcomputações
	e	u	z	
R1	$\langle 0, n, q\rangle$	$\langle \vec{x}\rangle$	$\langle q, u, e\rangle$	
R2	$\langle 1, n, i\rangle$	$\langle \vec{x}\rangle$	$\langle x_i, u, e\rangle$	
R3	$\langle 2, n, i\rangle$	$\langle \vec{x}\rangle$	$\langle x_i + 1, u, e\rangle$	
R4	$\langle 3, n+4\rangle$	$\langle p, q, r, s, \vec{x}\rangle$	$\langle p, u, e\rangle$ se $r = s$ $\langle q, u, e\rangle$ se $r \neq s$	
R5	$\langle 4, n, b, c_0, \ldots, c_{k-1}\rangle$	$\langle \vec{x}\rangle$	$\langle (z')_0, u, e, z',$ $\langle z''_0, \ldots, z''_{k-1}\rangle\rangle$	$z', z''_0, \ldots, z''_{k-1}$ são computações com índices $b, c_0, \ldots, c_{k-1}$. z' tem entrada $\langle (z''_0)_0, \ldots, (z''_{k-1})_0\rangle$.
R6	$\langle 5, n+2\rangle$	$\langle p, q, \vec{x}\rangle$	$\langle s, u, e\rangle$	(cf. 7.2.4)
R7	$\langle 6, n+1\rangle$	$\langle b, \vec{x}\rangle$	$\langle (z')_0, u, e, z'\rangle$	z' é uma computação com entrada $\vec{x}$ e índice b.

Prosseguiremos agora de uma maneira (levemente) mais formal, definindo um predicado $C(z)$ (para z uma computação), usando a informação da tabela anterior. Por conveniência, assumimos que nas cláusulas abaixo, as seqüências u (em $Seq(u)$) têm comprimento positivo.

$C(z)$ é definido por casos da seguinte forma:

$$
C(z) := \begin{cases}
\exists q, u, e < z(z = \langle q, u, e\rangle \wedge Seq(u) \wedge e = \langle 0, comp(u), q\rangle) & (1)\\
\text{ou} & \\
\exists u, e, i < z(z = \langle (u)_i, u, e\rangle \wedge Seq(u) \wedge e = \langle 1, comp(u), i\rangle) & (2)\\
\text{ou} & \\
\exists u, e, i < z(z = \langle (u)_i + 1, u, e\rangle \wedge Seq(u) \wedge e = \langle 2, comp(u), i\rangle) & (3)\\
\text{ou} & \\
\exists u, e < z(Seq(u) \wedge e = \langle 3, comp(u)\rangle \wedge comp(u) > 4 \wedge ((z = \langle (u)_0, u, e\rangle \wedge & \\
\quad \wedge (u)_2 = (u)_3) \vee (z = \langle (u)_1, u, e\rangle \wedge (u)_2 \neq (u)_3))) & (4)\\
\text{ou} & \\
Seq(z) \wedge comp(z) = 5 \wedge Seq((z)_2) \wedge Seq((z)_4) \wedge comp((z)_2) = & \\
\quad = 3 + comp((z)_4) \wedge (z)_{2,0} = 4 \wedge C((z)_3) \wedge (z)_{3,0} = (z)_0 \wedge (z)_{3,1} = & \\
\quad = \langle (z)_{4,0,0}, \cdots, (z)_{4,comp((z)_4),0}\rangle \wedge (z)_{3,2} = (z)_{2,2} \wedge & \\
\quad \wedge \bigwedge_{i=0}^{comp((z)_4)-1} (C((z)_{4,i}) \wedge (z)_{4,i,2} = (z)_{0,2+i} \wedge (z)_{4,i,1} = (z)_1) & (5)\\
\text{ou} & \\
\exists s, u, e < z(z = \langle s, u, e\rangle \wedge Seq(u) \wedge e = \langle 5, comp(u)\rangle \wedge & \\
\quad s = \langle 4, (u)_{0,1} \dot{-} 1, (u)_0, \langle 0, (u)_{0,1} \dot{-} 1, (u)_1\rangle, \langle 1, (u)_{0,1} \dot{-} 1, 0\rangle, \dots & \\
\quad \dots, \langle 1, (u)_{0,1} \dot{-} 1, (e)_1 \dot{-} 2\rangle\rangle), & (6)\\
\text{ou} & \\
\exists u, e, w < z(Seq(u) \wedge e = \langle 6, comp(y)\rangle \wedge z = \langle (w)_0, u, e, w\rangle \wedge C(w) \wedge & \\
\quad \wedge (w)_2 = (u)_0 \wedge (w)_1 = \langle (u)_1, \dots, (u)_{comp(u)-1}\rangle) & (7)
\end{cases}
$$

Observamos que o predicado C ocorre no lado direito apenas para argumentos menores; além do mais, todas as operações envolvidas nessa definição de $C(z)$ são recursivas primitivas. Agora aplicamos o teorema da recursão, como no exemplo 4 após o corolário 7.2.11, e concluímos que $C(z)$ é recursivo primitivo.

Agora fazemos $T(e, \vec{x}, z) := C(z) \wedge e = (z)_2 \wedge \langle \vec{x}\rangle = (z)_1$. Portanto, o predicado $T(e, \vec{x}, z)$ formaliza o enunciado 'z é a computação da função recursiva parcial (máquina) com índice e operando sobre a entrada $\vec{x}$'. A saída da computação, se ela existe, é $U(z) = (z)_0$; logo, temos que $\{e\}(\vec{x}) = (\mu z T(e, \vec{x}, z))_0$.

Para aplicações a estrutura precisa de T não é importante, é suficiente saber que ele é recursivo primitivo. □

Exercícios

1. Mostre que a função vazia (isto é, a função que diverge para todas as entradas) é recursiva parcial. Indique um índice para a função vazia.

2. Mostre que cada função recursiva parcial tem uma quantidade infinita de índices.

3. Faça a conversão das três equações da função de Ackermann numa função, veja o exemplo 3 após o corolário 7.2.11.

7.3 Conjuntos recursivamente enumeráveis

Se um conjunto A tem uma função característica, então essa função age como um teste efetivo de pertinência. Podemos decidir quais elementos estão em A e quais não estão. Conjuntos decidíveis, embora convenientes, demandam demais; usualmente não é necessário decidir o que está num conjunto, desde que possamos gerá-lo efetivamente. Equivalentemente, como veremos, é suficiente se ter uma máquina abstrata que apenas aceite elementos, e não os rejeite. Se você a alimenta com um elemento, ela

pode eventualmente mostrar uma luz verde de aceitação, mas não existe luz vermelha para a rejeição.

Definição 7.3.1 *1. Um conjunto (relação) é (recursivamente)* decidível *se ele for recursivo.*

2. Um conjunto é recursivamente enumerável (RE) se ele for o domínio de uma função recursiva parcial.

3. $W_e^k = \{\vec{x} \in \mathbb{N}^k \mid \exists y(\{e\}(\vec{x}) = y)\}$*, i.e. o domínio da função recursiva parcial* $\{e\}$*. Chamamos* e *de 'o índice RE de* W_e^k*'. Se nenhuma confusão surge omitiremos o expoente.*

Notação: escrevemos $\varphi(\vec{x}) \downarrow$ (respectivamente, $\varphi(\vec{x}) \uparrow$) para $\varphi(\vec{x})$ converge (resp. $\varphi(\vec{x})$ diverge).

É uma boa heurística pensar em conjuntos RE como sendo aceitos por máquinas, e.g. se A_i for aceito pela máquina M_i ($i = 0, 1$), então construimos uma nova máquina que simula M_0 e M_1 simultaneamente, e.g. você alimenta M_0 e M_1 com uma entrada, e realiza a computação alternantemente – um passo para M_0 e depois um passo para M_1, e assim n é aceito por M se for aceito por M_0 ou M_1. Daí, a união de dois conjuntos RE também é RE.

Exemplo 7.3.2 *1.* $\mathbb{N} =$ *o domínio da função constante* 1.

2. $\emptyset =$ *o domínio da função vazia. Essa função é recursiva parcial, como já vimos.*

3. Todo conjunto recursivo é RE. Seja A *recursivo, faça*

$$\psi(\vec{x}) = \mu y(K_A(\vec{x}) = y \wedge y \neq 0)$$

Então $Dom(\psi) = A$*.*

Os conjuntos recursivamente enumeráveis derivam sua importância do fato de que eles são efetivamente dados, no sentido preciso do teorema que vem a seguir. Além do mais, verifica-se que a maioria das relações importantes em lógica são RE. Por exemplo, o conjunto de (códigos de) sentenças demonstráveis da aritmética ou da lógica de predicados é RE. Os conjuntos RE representam o primeiro passo além dos conjuntos decidíveis, como mostraremos adiante.

Teorema 7.3.3 *Os seguintes enunciados são equivalentes, (*$A \subseteq \mathbb{N}$*):*

1. $A = Dom(\varphi)$ *para alguma* φ *recursiva parcial,*

2. $A = Ran(\varphi)$ *para alguma* φ *recursiva parcial,*

3. $A = \{x \mid \exists y R(x, y)\}$ *para alguma* R *recursiva.*

Demonstração. (1) ⇒ (2). Defina $\psi(x) = x \cdot sg(\varphi(x) + 1)$. Se $x \in Dom(\varphi)$, então $\psi(x) = x$, portanto $x \in Ran(\psi)$, e se $x \in Ran(\psi)$, então $\varphi(x) \downarrow$, logo $x \in Dom(\varphi)$.

(2) $\Rightarrow$ (3). Seja $A = Ran(\varphi)$, com $\{g\} = \varphi$, então

$$x \in A \Leftrightarrow \exists w(T(g, (w)_0, (w)_1) \wedge x = (w)_{1,0}).$$

A relação no escopo do quantificador é recursiva.

Note que w 'simula' um par: primeira coordenada—entrada, segunda coordenada—computação, tudo no sentido do teorema da forma normal.

(3) $\Rightarrow$ (1). Defina $\varphi(x) = \mu y R(x, y)$. φ é recursiva parcial e $Dom(\varphi) = A$. Observe que (1) $\Rightarrow$ (3) também se verifica para $A \subseteq \mathbb{N}^k$.

□

Como definimos conjuntos recursivos por meio de funções características, e, dado que estabelecemos o fecho sob recursão primitiva, podemos copiar todas as propriedades de fechamento dos conjuntos (e relações) recursivos primitivos para conjuntos (e relações) recursivos.

A seguir listamos um número de propriedades de fechamento de conjuntos RE. Escreveremos conjuntos e relações também como predicados, quando isso acontece de ser conveniente.

Teorema 7.3.4 *1. Se A e B forem RE, então o mesmo acontece com $A \cup B$ e $A \cap B$.*

2. Se $R(x, \vec{y})$ for RE, então $\exists x R(x, \vec{y})$ também o é.

3. Se $R(x, \vec{y})$ for RE e φ for recursiva parcial, então $R(\varphi(\vec{y}, \vec{z}), \vec{y})$ é RE.

4. Se $R(x, \vec{y})$ for RE, então o mesmo acontece com $\forall x < z R(x, \vec{y})$ e $\exists x < z R(x, \vec{y})$.

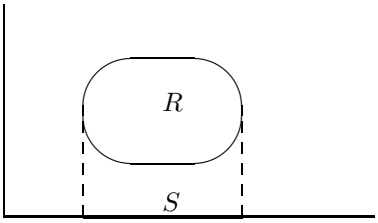
Demonstração. 1. Existem R e S recursivas tais que

$$A\vec{y} \Leftrightarrow \exists x R(x, \vec{y}),$$
$$B\vec{y} \Leftrightarrow \exists x S(x, \vec{y}).$$

$$\begin{aligned} \text{Então } A\vec{y} \wedge B\vec{y} &\Leftrightarrow \exists x_1 x_2 (R(x_1, \vec{y}) \wedge S(x_2, \vec{y})) \\ &\Leftrightarrow \exists z (R((z)_0, \vec{y}) \wedge S((z)_1, \vec{y})) \end{aligned}$$

A relação no escopo do quantificador é recursiva, portanto $A \cap B$ é RE. Um argumento semelhante estabelece a enumerabilidade recursiva de $A \cup B$. O truque de substituir x_1 e x_2 por $(z)_0$ e $(z)_1$, e $\exists x_1 x_2$ por $\exists z$ é chamado de *contração de quantificadores.*

2. Seja $R(x, \vec{y}) \Leftrightarrow \exists z S(z, x, \vec{y})$ para uma S recursiva, então $\exists x R(x, \vec{y}) \Leftrightarrow \exists x \exists z S(z, x, \vec{y})$ $\Leftrightarrow \exists u S((u)_0, (u)_1, \vec{y})$. Portanto, a *projeção* $\exists x R(x, \vec{y})$ de R é RE. Geometricamente falando, $\exists x R(x, \vec{y})$ é de fato uma projeção. Considere o caso bidimensional,



A projeção vertical S de R é dada por $Sx \Leftrightarrow \exists y R(x, y)$.

3. Seja R o domínio de uma ψ recursiva parcial, então $R(\varphi(\vec{y}, \vec{z}), \vec{y})$ é o domínio de $\psi(\varphi(\vec{y}, \vec{z}), \vec{y})$.

4. Deixo ao leitor.

□

Teorema 7.3.5 *O grafo de uma função parcial é RE sse a função é recursiva parcial.*

Demonstração. $G = \{(\vec{x}, y) \mid y = \{e\}(\vec{x})\}$ é o grafo de $\{e\}$. Agora $(\vec{x}, y) \in G \Leftrightarrow \exists z(T(e, \langle \vec{x} \rangle, z) \wedge y = (z)_0)$, portanto G é RE.

Para a recíproca, se G for RE, então $G(\vec{x}, y) \Leftrightarrow \exists z R(\vec{x}, y, z)$ para alguma R recursiva. Daí, $\varphi(\vec{x}) = (\mu w R(\vec{x}, (w)_0, (w)_1))_0$, portanto φ é recursiva parcial. □

Podemos também caracterizar conjuntos recursivos em termos de conjuntos RE. Suponha que tanto A quanto seu complemento A^c sejam RE, então (heuristicamente) temos duas máquinas enumerando A e A^c. Agora, o teste de pertinência de A é simples: ligue ambas as máquinas e espere até n aparecer como saída da primeira ou da segunda máquina. Isso tem necessariamente que ocorrer em um número finito de passos, pois $n \in A$ ou $n \in A^c$ (princípio do terceiro excluído). Logo, temos um teste efetivo. Formalizamos isso da seguinte forma:

Teorema 7.3.6 *A é recursivo $\Leftrightarrow$ A e A^c forem RE.*

Demonstração. $\Rightarrow$ Trivial: $A(\vec{x}) \Leftrightarrow \exists y A(\vec{x})$, onde y é uma variável *dummy*. Igualmente para A^c.

$\Leftarrow$ Seja $A(\vec{x}) \Leftrightarrow \exists y A(\vec{x}, y)$, $\neg A(\vec{x}) \Leftrightarrow \exists z S(v, z)$. Como $\forall \vec{x}(A(\vec{x}) \vee \neg A(\vec{x}))$, temos $\forall \vec{x} \exists y(R(\vec{x}, y) \vee S(\vec{x}, y))$, portanto $f(\vec{x}) = \mu y[R(\vec{x}, y) \vee S(\vec{x}, y)]$ é recursiva e se inserirmos o y que encontramos em $R(\vec{x}, y)$, então sabemos que se $R(\vec{x}, f(\vec{x}))$ for verdadeira, o $\vec{x}$ pertence a A. Portanto, $A(\vec{x}) \Leftrightarrow R(\vec{x}, f(\vec{x}))$, i.e. A é recursivo.

□

Para funções recursivas parciais temos uma forma forte de definição por casos:

Teorema 7.3.7 *Suponha que $\psi_0, \ldots, \psi_{k-1}$ sejam recursivas parciais, $R_0, \ldots, R_{k-1}$ relações RE mutuamente disjuntas, então a seguinte função é recursiva parcial:*

$$\varphi(\vec{x}) = \begin{cases} \psi_0(\vec{x}) & \textit{se } R_0(\vec{x}) \\ \psi_1(\vec{x}) & \textit{se } R_1(\vec{x}) \\ \vdots & \vdots \\ \psi_{k-1}(\vec{x}) & \textit{se } R_{k-1}(\vec{x}) \\ \uparrow & \textit{caso contrário} \end{cases}$$

Demonstração. Consideramos o grafo da função φ.

$$G(\vec{x}, y) \Leftrightarrow (R_0(\vec{x}) \wedge y = \psi_1(\vec{x})) \vee \cdots \vee (R_{k-1}(\vec{x}) \wedge y = \psi_{k-1}(\vec{x})).$$

Pelas propriedades de conjuntos RE, $G(\vec{x}, y)$ é RE e, por conseguinte, $\varphi(\vec{x})$ é recursiva parcial. (Note que o último caso na definição de φ é apenas um pouco de decoração.) □

Agora podemos mostrar a existência de conjuntos RE indecidíveis.

Exemplos.
(1) (**O Problema da Parada (Turing)**)
Considere $K = \{x \mid \exists z T(x,x,z)\}$. K é a projeção de uma relação recursiva, portanto é RE. Suponha que K^c também seja RE, então $x \in K^c \Leftrightarrow \exists z T(e,x,z)$ para algum índice e. Agora $e \in K \Leftrightarrow \exists z T(e,e,z) \Leftrightarrow e \in K^c$. Contradição. Daí, K não é recursivo pelo teorema 7.3.6. Isso nos diz que existem conjuntos recursivamente enumeráveis que não são recursivos. Em outras palavras, o fato de que se pode efetivamente enumerar um conjunto não garante que ele seja decidível.

O problema da decisão para K é chamado *problema da parada*, porque ele pode ser parafraseado como 'decida se a máquina com índice x realiza uma computação que pára após um número finito de passos quando apresentada com x como entrada. Note que é *ipso facto* indecidível se 'a máquina com índice x no final das contas pára quando executa sobre a entrada y'.

Trata-se de uma característica específica de problemas de decisão em teoria da recursão, que eles concernem testes para entradas tomadas de algum domínio. Não faz sentido pedir por um procedimento de decisão para, digamos, a hipótese de Riemann, pois existe trivialmente uma função recursiva f que testa o problema no sentido de que $f(0) = 0$ se a hipótese de Riemann se verifica e $f(0) = 1$ se a hipótese de Riemann for falsa. A saber, considere as funções f_0 e f_1, que são as funções constantes 0 e 1 respectivamente. Agora, lógica nos diz que uma das duas é a função requerida (essa é a lei do terceiro excluído), infelizmente não sabemos qual função ela é. Portanto, para problemas sem parâmetros, não faz sentido, no arcabouço da teoria da recursão, discutir decidibilidade. Como vimos, lógica intuicionística vê esse 'exemplo patológico' sob uma luz diferente.

(2) Não é decidível se $\{x\}$ é uma função total.

Suponha que fosse decidível, então teríamos uma função recursiva f tal que $f(x) = 0 \Leftrightarrow \{x\}$ é total. Agora considere

$$\varphi(x,y) := \begin{cases} 0 & \text{se } x \in K \\ \uparrow & \text{caso contrário} \end{cases}$$

Pelo teorema S_n^m existe uma função recursiva h tal que $\{h(x)\}(y) = \varphi(x,y)$. Agora, $\{h(x)\}$ é total $\Leftrightarrow x \in K$, portanto $f(h(x)) = 0 \Leftrightarrow x \in K$, i.e. temos uma função característica recursiva $\overline{sg}(f(h(x)))$ para K. Contradição. Daí, tal f não existe, ou seja, $\{x \mid \{x\}$ é total$\}$ não é recursivo.

(3) O problema 'W_e é finito' não é recursivamente solúvel.

Em palavras, 'não é decidível se um conjunto recursivamente enumerável é finito'. Suponha que existisse uma função recursiva f tal que $f(e) = 0 \Leftrightarrow W_e$ é finito. Considere a $h(x)$ definida no exemplo (2). Claramente, $W_{h(x)} = Dom\{h(x)\} = \emptyset \Leftrightarrow x \notin K$, e $W_{h(x)}$ é infinito para $x \in K$. $f(h(x)) = 0 \Leftrightarrow x \notin K$, e portanto $sg(f(h(x)))$ é uma função característica recursiva para K. Contradição.

Note que $x \in K \Leftrightarrow \{x\}x \downarrow$, portanto podemos reformular as soluções acima da seguinte forma: em (2) tome $\varphi(x,y) = 0 \cdot \{x\}(x)$ e em (3) $\varphi(x,y) = \{x\}(x)$.

(4) A igualdade de conjuntos RE é indecidível.

Ou seja, $\{(x,y) \mid W_x = W_y\}$ não é recursivo. Reduzimos o problema à solução de (3) escolhendo $W_y = \emptyset$.

(5) Não é decidível se W_e é recursivo.

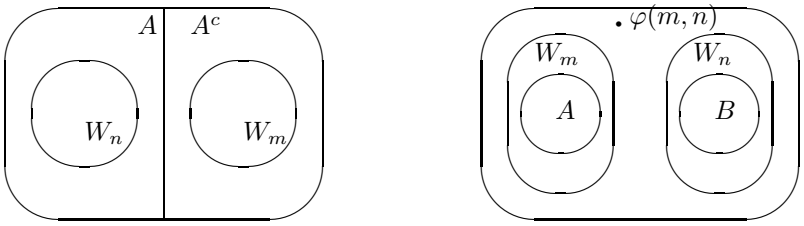
Ponha $\varphi(x,y) = \{x\}(x) \cdot \{y\}(y)$, então $\varphi(x,y) = \{h(x)\}(y)$ para uma certa h recursiva, e

$$Dom\{h(x)\} = \begin{cases} K & \text{se } x \in K \\ \emptyset & \text{caso contrário} \end{cases}$$

Suponha que houvesse uma função recursiva f tal que $f(x) = 0 \Leftrightarrow W_x$ é recursivo, então $f(h(x)) = 0 \Leftrightarrow x \notin K$ e, portanto, K seria recursivo. Contradição.

Além dessas, existem diversas técnicas para se estabelecer indecidibilidade. Trataremos o método da inseparabilidade.

Definição 7.3.8 *Dois conjuntos RE disjuntos W_m e W_n são* recursivamente separáveis *se existe um conjunto recursivo A tal que $W_n \subseteq A$ e $W_m \subseteq A^c$. Conjuntos disjuntos A e B são* efetivamente inseparáveis *se existe uma função recursiva parcial φ tal que para todos m, n com $A \subseteq W_m$, $B \subseteq W_n$, $W_m \cap W_n = \emptyset$, temos $\varphi(m,n) \downarrow$ e $\varphi(m,n) \notin W_m \cup W_n$.*



Imediatamente vemos que conjuntos RE efetivamente inseparáveis são recursivamente inseparáveis, i.e. não recursivamente separáveis.

Teorema 7.3.9 *Existem conjuntos RE efetivamente inseparáveis.*

Demonstração. Defina $A = \{x \mid \{x\}(x) = 0\}$, $B = \{x \mid \{x\}(x) = 1\}$. Claramente $A \cap B = \emptyset$ e ambos os conjuntos são RE.

Suponha que $W_m \cap W_n = \emptyset$ e $A \subseteq W_m$, $B \subset W_n$. Para definir φ começamos testando $x \in W_m$ ou $x \in W_n$; se primeiro encontrarmos $x \in W_m$, fazemos com que uma função auxiliar $\sigma(x)$ seja igual a 1, e se x aparece primeiro em W_n colocamos $\sigma(x) = 0$.

Formalmente

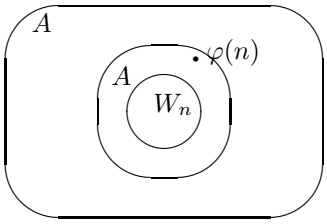
$$\sigma(m,n,x) = \begin{cases} 1 & \text{se } \exists z(T(m,x,z) \text{ e } \forall y < z \neg T(n,x,y)) \\ 0 & \text{se } \exists z(T(n,x,z) \text{ e } \forall y \leq z \neg T(m,x,y)) \\ \uparrow & \text{caso contrário.} \end{cases}$$

Pelo teorema S_n^m, $\{h(m,n)\}(x) = \sigma(m,n,x)$ para alguma h recursiva.

$$\begin{array}{rcl} h(m,n) \in W_m & \Rightarrow & h(m,n) \notin W_n. \text{ Portanto } \exists z(T(m,h(m,n),z) \wedge \\ & & \forall y < z \neg T(n,h(m,n),y)) \\ & \Rightarrow & \sigma(m,n,h(m,n)) = 1 \Rightarrow \{h(m,n)\}(h(m,n)) = 1 \\ & \Rightarrow & h(m,n) \in B \Rightarrow h(m,n) \in W_n. \end{array}$$

Contradição. Daí, $h(m,n) \notin W_m$. Igualmente, $h(m,n) \notin W_n$. Por conseguinte, h é a φ requerida. $\square$

Definição 7.3.10 *Um subconjunto A de $\mathbb{N}$ é* produtivo *se existe uma função recursiva parcial φ, tal que para cada $W_n \subseteq A$, $\varphi(n) \downarrow$ e $\varphi(n) \in A - W_n$.*



O teorema acima nos dá o seguinte:

Corolário 7.3.11 *Existem conjuntos produtivos.*

Demonstração. O conjunto A^c definido na demonstração acima é *produtivo*. Seja $W_k \subseteq A^c$. Faça $W_\ell = B \cup W_k = W_n \cup W_k = W_{h(n,k)}$ para uma função recursiva apropriada h. Agora aplique a função de separação da demonstração do teorema precedente a $A = W_m$ e $W_{h(n,k)}$: $\varphi(m, h(n,k)) \in A^c - W_m$. □

Conjuntos produtivos são, num sentido forte, não RE: independentemente de como se tenta encaixar um conjunto RE neles, pode-se uniformemente e efetivamente indicar um ponto que não é levado em conta por esse conjunto RE.

Exercícios

1. A projeção de um conjunto RE é RE, i.e. se $R(\vec{x}, y)$ for RE então o mesmo acontece com $\exists y R(\vec{x}, y)$.

2. (i) Se A for enumerado por uma função estritamente monótona, então A é recursivo.

   (ii) Se A é infinito e recursivo, então A é enumerado por uma função recursiva estritamente monótona.

   (iii) Um conjunto RE infinito contém um subconjunto recursivo infinito.

3. Todo conjunto RE não-vazio é enumerado por uma função recursiva total.

4. Se A for um conjunto RE e f uma função parcialmente recursiva, então $f^{-1}(A)$ $(= \{x \mid f(x) \in A\})$ e $f(A)$ são RE.

5. Mostre que os seguintes conjuntos não são recursivos:

   (i) $\{(x, y) \mid W_x = W_y\}$

   (ii) $\{x \mid W_x$ é recursivo$\}$

   (iii) $\{x \mid 0 \in W_x\}$

7.4 Um pouco de aritmética

Na seção sobre funções recursivas estivemos trabalhando no modelo padrão da aritmética; como estamos agora lidando com demonstrabilidade *na* aritmética temos que evitar argumentos semânticos, e nos basearmos exclusivamente em derivações dentro do sistema formal da aritmética. A teoria geralmente aceita para a aritmética vai lá atrás a Peano, e portanto falamos da *aritmética de Peano*, $\mathbf{PA}$ (cf. 2.7).

Uma grande questão no final dos anos 1920s era a completude de $\mathbf{PA}$. Gödel pôs um fim às grandes esperanças prevalentes da época mostrando que $\mathbf{PA}$ é incompleta (1931). De forma a realizar os passos necessários para a prova de Gödel, temos que provar um número de teoremas em $\mathbf{PA}$. A maioria desses fatos podem ser encontrados nos textos sobre teoria dos números, ou sobre os fundamentos da aritmética. Deixaremos um número considerável de provas ao leitor. A maior parte do tempo tem-se que aplicar uma forma apropriada de indução. Por mais importante que essas reais demonstrações sejam, o coração do argumento de Gödel reside na sua engenhosa incorporação de argumentos recursão-teóricos dentro de $\mathbf{PA}$.

Um das óbvias pedras-no-caminho para uma imitação imediata da 'auto-referência' é a aparente pobreza da linguagem de $\mathbf{PA}$. Ela não nos permite falar de, e.g., uma cadeia finita de números. Uma vez que temos exponenciação podemos simplesmente codificar seqüências finitas de números. Gödel mostrou que pode-se de fato definir a exponencial (e muito mais) ao custo de um pouco de aritmética adicional, levando à sua famosa β-função. Em 1971 Matiyashevich mostrou por outros meios que a exponencial é definível em $\mathbf{PA}$, portanto nos permitindo manusear codificação de seqüências em $\mathbf{PA}$ diretamente. A aritmética de Peano mais exponenciação é *prima facie* mais forte que $\mathbf{PA}$, mas os resultados supracitados mostram que a exponenciação pode ser eliminada. Vamos chamar o sistema estendido de $\mathbf{PA}$; nenhuma confusão surgirá.

Repetimos os axiomas:

– $\forall x(S(x) \neq 0)$,
– $\forall xy(S(x) = S(y) \to x = y)$,
– $\forall x(x + 0 = x)$,
– $\forall xy(x + S(y)) = S(x + y))$,
– $\forall x(x \cdot 0 = 0)$,
– $\forall xy(x \cdot S(y) = x \cdot y + x)$,
– $\forall x(x^0 = 1)$,
– $\forall xy(x^{S(y)} = x^y \cdot x)$,
– $\varphi(0) \wedge \forall x(\varphi(x) \to \varphi(S(x))) \to \forall x\varphi(x)$.

Como $\vdash \overline{1} = S(0)$, usaremos ambos $S(x)$ e $x + \overline{1}$, o que quer que seja mais conveniente. Usaremos também as abreviações usuais. De modo a simplificar a notação, omitiremos tacitamente o '$\mathbf{PA}$' em frente de '$\vdash$' sempre que possível. Como uma outra simplificação inofensiva da notação, freqüentemente escreveremos simplesmente n para $\overline{n}$ quando nenhuma confusão possa surgir.

No que se segue daremos um número de teoremas de $\mathbf{PA}$; de forma a melhorar a legibilidade, omitiremos os quantificadores universais precedendo as fórmulas. O leitor deve sempre pensar no 'fecho universal de ...'.

Além do mais, usaremos as abreviações padrão da álgebra, i.e. deixando de fora o ponto da multiplicação, os parênteses supérfluos, etc., quando nenhuma confusão surge. Escreveremos também 'n' ao invés de '$\overline{n}$'.

As operações básicas satisfazem as leis bem-conhecidas:

Lema 7.4.1 *Adição e multiplicação são associativas e comutativas, e $\cdot$ distribui sobre $+$.*

(i) $\vdash (x+y)+z = x+(y+z)$
(ii) $\vdash x+y = y+x$
(iii) $\vdash x(yz) = (xy)z$
(iv) $\vdash xy = yx$
(v) $\vdash x(y+z) = xy+xz$
(vi) $\vdash x^{y+z} = x^y x^z$
(vii) $\vdash (x^y)^z = x^{yz}$

Demonstração. Rotina. □

Lema 7.4.2

(i) $\vdash x = 0 \vee \exists y(x = Sy)$
(ii) $\vdash x+z = y+z \to x = y$
(iii) $\vdash z \neq 0 \to (xz = yz \to x = y)$
(iv) $\vdash x \neq 0 \to (x^y = x^z \to y = z)$
(v) $\vdash y \neq 0 \to (x^y = z^y \to x = z)$

Demonstração. Rotina. □

Um número de fatos úteis são listados nos exercícios.

Embora a linguagem de **PA** seja modesta, muitas das relações e funções usuais podem ser definidas. A ordenação é um exemplo importante.

Definição 7.4.3 $x < y := \exists z(x + Sz = y)$.

Usaremos as seguintes abreviações:

$x < y < z$	significa	$x < y \wedge y < z$
$\forall x < y \varphi(x)$	significa	$\forall x(x < y \to \varphi(x))$
$\exists x < y \varphi(x)$	significa	$\exists x(x < y \wedge \varphi(x))$
$x > y$	significa	$y < x$
$x \leq y$	significa	$x < y \vee x = y$.

Teorema 7.4.4

(i) $\vdash \neg x < x$
(ii) $\vdash x < y \wedge y < z \to x < z$
(iii) $\vdash x < y \vee x = y \vee y < x$
(iv) $\vdash 0 = x \vee 0 < x$
(v) $\vdash x < y \to Sx = y \vee Sx < y$
(vi) $\vdash x < Sx$
(vii) $\vdash \neg x < y \wedge y < Sx$
(viii) $\vdash x < Sy \leftrightarrow (x = y \vee x < y)$
(ix) $\vdash x < y \leftrightarrow x + z < y + z$
(x) $\vdash z \neq 0 \to (x < y \leftrightarrow xz < yz)$
(xi) $\vdash x \neq 0 \to (0 < y < z \to x^y < x^z)$
(xii) $\vdash z \neq 0 \to (x < y \to x^z < y^z)$
(xiii) $\vdash x < y \leftrightarrow Sx < Sy$

Demonstração. Rotina. □

Quantificação com um limitante explícito pode ser substituída por uma disjunção (ou conjunção) repetida.

Teorema 7.4.5
$\vdash \forall x < \overline{n}\varphi(x) \leftrightarrow \varphi(0) \wedge \ldots \wedge \varphi(\overline{n-1})$, $(n > 0)$,
$\vdash \exists x < \overline{n}\varphi(x) \leftrightarrow \varphi(0) \vee \ldots \vee \varphi(\overline{n-1})$, $(n > 0)$.

Demonstração. Indução sobre n. □

Teorema 7.4.6 *(i) indução bem-fundada*

$\vdash \forall x(\forall y < x\ \varphi(y) \rightarrow \varphi(x)) \rightarrow \forall x\varphi(x)$

(ii) princípio do menor número (PMN)

$\vdash \exists x\varphi(x) \rightarrow \exists x(\varphi(x) \wedge \forall y < x \neg\varphi(x))$

Demonstração. (i) Vamos fazer $\psi(x) := \forall y < x\varphi(y)$. Assumimos que $\forall x(\psi(x) \rightarrow \varphi(x))$ e procedemos para aplicar indução sobre $\psi(x)$.

Claramente $\vdash \psi(0)$.

Assim, assuma, pela hipótese da indução, $\psi(x)$.

Agora, $\psi(Sx) \leftrightarrow (\forall y < Sx\varphi(y)) \leftrightarrow (\forall y((y = x \vee y < x) \rightarrow \varphi(y))) \leftrightarrow (\forall y((y = x \rightarrow \varphi(y)) \wedge (y < x \rightarrow \varphi(y)))) \leftrightarrow (\forall y(\varphi(x) \wedge (y < x \rightarrow \varphi(y)))) \leftrightarrow (\varphi(x) \wedge \forall y < x\varphi(y)) \leftrightarrow (\varphi(x) \wedge \psi(x))$.

Agora, $\psi(x)$ foi dada em $\psi(x) \rightarrow \varphi(x)$. Logo, obtemos $\psi(Sx)$. Isso mostra $\forall x\psi(x)$, e, por conseguinte, derivamos $\forall x\varphi(x)$.

(ii) Considere a contraposição e reduza-a a (i).

□

Em nossas considerações adicionais as seguintes noções terão seu papel a desempenhar.

Definição 7.4.7 *(i) Divisibilidade*

$x|y := \exists z(xz = y)$

(ii) Subtração com ponto de corte

$z = y \dot{-} x := (x < y \wedge x + z = y) \vee (y \leq x \wedge z = 0)$

(iii) Resto após a divisão

$z = resto(x, y) := (x \neq 0 \wedge \exists u(y = ux + z) \wedge z < x) \vee (x = 0 \wedge z = y)$

(iv) x é primo

$Primo(x) := x > 1 \wedge \forall yz(x = yz \rightarrow y = x \vee y = 1)$

Os lados direitos de (ii) e (iii) de fato determinam funções, como mostrado em

Lema 7.4.8 *(i)* $\vdash \forall xy\exists!z((x < y \wedge z + x = y) \vee (y \leq x \wedge z = 0))$

(ii) $\vdash \forall xy\exists!z((x \neq 0 \wedge \exists u(y = ux + z) \wedge z < y) \vee (x = 0 \wedge z = 0))$.

Demonstração. Em ambos os casos, indução sobre y. □

Existe uma outra caracterização dos números primos.

Lema 7.4.9 *(i)* $\vdash Primo(x) \leftrightarrow x > 1 \wedge \forall y(y|x \rightarrow y = 1 \vee y = x)$

(ii) $\vdash Primo(x) \leftrightarrow x > 1 \wedge \forall yz(x|yz \rightarrow x|y \vee x|z)$

Demonstração. (i) é uma mera reformulação da definição.

(ii) $\rightarrow$ é um pouco complicado. Introduzimos um limitante no produto yz, e fazemos uma indução bem-fundada sobre o limitante. Faça $\varphi(w) = \forall yz \leq w(x|yz \rightarrow x|y \vee x|z)$. Agora mostramos $\forall w(\forall v < w\varphi(v) \rightarrow \varphi(w))$.

Suponha que $\forall v < w\varphi(v)$ e assuma que $\neg\varphi(w)$, i.e. existem $y, z \leq w$ tais que $x|yz$, $\neg x|y$, $\neg x|z$. Vamos 'diminuir' o y tal que o w é também diminuído. Como $\neg x|y$, $\neg x|z$, temos $z \neq 0$. Se $y \geq z$, então podemos substituí-lo por $y = resto(x, y)$ e prosseguir com o argumento. Portanto, suponha que $y < x$. Agora uma vez mais obtemos o resto, $x = ay + b$ com $b < y$. Consideramos $b = 0$ e $b > 0$.

Se $b = 0$, então $x = ay$; logo, $y = 1 \vee y = x$. Se $y = 1$, então $x|z$. Contradição.

Se $y = x$ então $x|y$. Contradição.

Agora, se $b > 0$, então $bz = (x - ay)z = xz - ayz$. Como $x|yz$, obtemos $x|bz$. Observe que $bz < yz < w$, portanto temos uma contradição com $\forall v < w\varphi(v)$. Daí, por *RAA*, estabelecemos o enunciado requerido.

Para a recíproca ($\leftarrow$), temos somente que aplicar os fatos estabelecidos sobre divisibilidade.

□

Podemos provar na aritmética de Peano que existe algum primo? Sim, por exemplo, $\mathbf{PA} \vdash \forall x(x > 1 \rightarrow \exists y(Primo(y) \wedge y|x)$.

Demonstração. Observe que $\exists y(y > 1 \wedge y|x)$. Pelo princípio PML existe um desses menores y: $\exists y(y > 1 \wedge y|x \wedge \forall z < y(z > 1 \rightarrow \neg z|y))$. Agora é fácil mostrar que esse mínimo y é um primo. □

Primos e e expoentes são muito úteis para codificar seqüências finitas de números naturais, e, portanto, para codificação em geral. Existem muito mais codificações, e algumas delas são mais realistas no sentido de que elas têm uma complexidade menor. Para nosso propósito, entretanto, primos e expoentes bastarão.

Como vimos, podemos codificar uma seqüência finita $(n_0, \ldots, n_{k-1})$ como o número $2^{n_0+1} \cdot 3^{n_1+1} \cdot \ldots \cdot p_{k-1}^{n_{k-1}+1}$.

Introduziremos alguns predicados auxiliares.

Definição 7.4.10 (Primos sucessivos) $Primossuc(x, y) := x < y \wedge Primo(x) \wedge Primo(y) \wedge \forall z(x < z < y \rightarrow \neg Primo(z))$.

O próximo passo é definir a seqüência de números primos 2, 3, 5, ..., p_n, ... O truque básico aqui é que consideramos todos os primos sucessivos com expoentes em ascensão: 2^0, 3^1, 5^2, 7^3, ..., p_x^x. Formamos o produto e aí pegamos o último fator.

Definição 7.4.11 (O x-ésimo número primo, p_x) $p_x = y := \exists z(\neg 2|z \wedge \forall v < y \forall u \leq y(Primossuc(v, u) \rightarrow \forall w < z(v^w|z \rightarrow u^{w+1}|z)) \wedge y^x|z \wedge \neg y^{x+1}|z)$.

Observe que, como a definição dá origem a uma função, temos que mostrar que

Lema 7.4.12 $\vdash \exists z(\neg 2|z \wedge \forall v < y_0 \forall u(Primossuc(v,u) \rightarrow \forall w < z(v^w|z \rightarrow u^{w+1}|z)) \wedge y_0^x|z \wedge \neg y_0^{x+1}|z) \wedge \exists z(\neg 2|z \wedge \forall v < y_1 \forall u \leq y_1(Primossuc(v,u) \rightarrow \forall w < z(v^w|z \rightarrow u^{w+1}|z)) \wedge y_1^x|z \wedge \neg y_1^{x+1}|z) \rightarrow y_0 = y_1$.

A definição acima simplesmente imita a descrição informa. Note que podemos limitar o quantificador existencial como $\exists z < y^{x^2}$. Agora acabamos de justificar a notação de números de seqüências como produtos da forma

$$p_0^{n_0+1} \cdot p_1^{n_1+1} \cdot \ldots \cdot p_{k-1}^{n_{k-1}+1}$$

O leitor deve verificar isso de acordo com a definição $p_0 = 2$. A decodificação também pode ser definida. Em geral pode-se definir a potência de um fator primo.

Definição 7.4.13 (decodificação) $(z)_k = v := p_k^{v+1}|z \wedge \neg p_k^{v+2}|z$.

O comprimento de uma seqüência codificada pode também ser extraída do código:

Definição 7.4.14 (comprimento) $comp(z) = x := p_x|z \wedge \forall y < z(p_y|z \rightarrow y < x)$.

Lema 7.4.15 $\vdash Seq(z) \rightarrow (comp(z) = x \leftrightarrow (p_x|z \wedge \neg p_{x+1}|z))$.

Definimos separadamente a codificação da seqüência vazia: $\langle\ \rangle := 1$.

A codificação da seqüência $(x_0, \ldots, x_{n-1})$ é denotada por $\langle x_0, \ldots, x_{n-1}\rangle$.

Operações como concatenação e restrição de seqüências codificadas podem ser definidas tais que

$\langle x_0, \ldots, x_{n-1}\rangle * \langle y_0, \ldots, y_{m-1}\rangle = \langle x_0, \ldots, x_{n-1}, y_0, \ldots, y_{m-1}\rangle$

$\langle x_0, \ldots, x_{n-1}\rangle | m = \langle x_0, \ldots, x_{m-1}\rangle$, onde $m \leq n$ (atenção: aqui | é usado para a relação de restrição, não confunda com divisibilidade).

A cauda de uma seqüência é definida da seguinte forma:

$$cauda(y) = z \leftrightarrow (\exists x(y = \langle x\rangle * z) \vee (comp(y) = 0 \wedge z = 0)).$$

Termos fechados da **PA** podem ser calculados em **PA**:

Lema 7.4.16 *Para qualquer termo fechado t existe um número n tal que $\vdash t = \overline{n}$.*

Demonstração. Indução externa sobre t, cf. lema 2.3.3. Observe que n é univocamente determinado. □

Corolário 7.4.17 $\mathbb{N} \models t_1 = t_2 \Rightarrow \vdash t_1 = t_2$ *para t_1, t_2 ambos fechados.*

O teorema de Gödel mostrará que em geral '*verdadeiro no modelo padrão*' (de agora em diante simplesmente diremos 'verdadeiro') e demonstrável em **PA** não são o mesmo. Entretanto, para uma classe de sentenças simples, isso está correto.

Definição 7.4.18

(i) A classe Δ_0 de fórmulas é indutivamente definida por

$\varphi \in \Delta_0$ *para φ atômica*
$\varphi, \psi \in \Delta_0 \Rightarrow \neg\varphi$, $\varphi \wedge \psi$, $\varphi \vee \psi$, $\varphi \rightarrow \psi \in \Delta_0$
$\varphi \in \Delta_0 \Rightarrow \forall x < y\ \varphi$, $\exists x < y\ \varphi\ \exists x\ \varphi \in \Delta_0$

(ii) A classe Σ_1 é dada por

$\varphi, \neg\varphi \in \Sigma_1$ *para* φ *atômica*
$\varphi, \psi \in \Sigma_1 \Rightarrow \varphi \vee \psi$, $\varphi \wedge \psi \in \Sigma_1$
$\varphi \in \Sigma_1 \Rightarrow \forall x < y\ \varphi$, $\exists x < y\ \varphi \in \Sigma_1$

Uma fórmula é chamada Σ_1-estrita se ela for da forma $\exists \vec{x} \varphi(\vec{x})$, onde φ é Δ_0.

Chamaremos fórmulas nas classes Δ_0 e Σ_1 de Δ_0-fórmulas e Σ_1-fórmulas, respectivamente. Fórmulas demonstravelmente equivalentes a Σ_1-fórmulas também serão chamadas de Σ_1-fórmulas.

Para Σ_1-fórmulas temos que 'verdadeiro = demonstrável'.

Lema 7.4.19 *$\vdash \varphi$ ou $\vdash \neg\varphi$, para Δ_0-sentenças φ.*

Demonstração. Indução sobre φ.

(i) φ atômica. Se $\varphi \equiv t_1 = t_2$ e $t_1 = t_2$ for verdadeira, veja o corolário 7.4.17.

Se $t_1 = t_2$ for falsa, então $t_1 = n$ e $t_2 = m$, onde, digamos $n = m + k$ com $k > 0$. Agora assuma (em $\mathbf{PA}$) $t_1 = t_2$, então $\overline{m} = \overline{m} + \overline{k}$. Pelo lema 7.4.2, obtemos $0 = \overline{k}$. Mas como $k = S(l)$ para algum l, obtemos uma contradição. Daí, $\vdash \neg t_1 = t_2$.

(ii) Os casos de indução são óbvios. Para $\forall x < t\ \varphi(x)$, onde t é um termo fechado, use a identidade $\forall x < \overline{n}\ \varphi(x) \leftrightarrow \varphi(0) \wedge \ldots \wedge \varphi(\overline{n-1})$. Igualmente para $\exists x < t\ \varphi(x)$.

□

Teorema 7.4.20 (Σ_1-completude) *$\models \varphi \Leftrightarrow \mathbf{PA} \vdash \varphi$, para Σ_1-sentenças φ.*

Demonstração. Como a veracidade de $\exists x \varphi(x)$ remete à veracidade de $\varphi(\overline{n})$ para algum n, podemos aplicar o lema acima. □

7.5 Representabilidade

Nesta seção daremos a formalização de tudo isso em $\mathbf{PA}$, i.e. mostraremos que predicados definíveis existem correspondendo com os predicados introduzidos acima (no modelo padrão) – e que suas propriedades são demonstráveis.

Definição 7.5.1 (representabilidade)

– *Uma fórmula $\varphi(x_0, \ldots, x_{n-1}, y)$ representa uma função n-ária f se para todos $k_0, \ldots, k_{n-1}$*

$$f(k_0, \ldots, k_{n-1}) = p \quad \Rightarrow \quad \vdash \forall y(\varphi(\overline{k_0}, \ldots, \overline{k_{n-1}}, y) \leftrightarrow y = \overline{p})$$

– *Uma fórmula $\varphi(x_0, \ldots, x_{n-1})$ representa um predicado P se para todos $k_0, \ldots, k_{n-1}$*

$$P(k_0, \ldots, k_{n-1}) \quad \Rightarrow \quad \vdash \varphi(\overline{k_0}, \ldots, \overline{k_{n-1}})$$

e

$$\neg P(k_0, \ldots, k_{n-1}) \quad \Rightarrow \quad \vdash \neg\varphi(\overline{k_0}, \ldots, \overline{k_{n-1}})$$

– *Um termo $t(k_0, \ldots, k_{n-1})$ representa f se para todos $k_0, \ldots, k_{n-1}$*

$$f(k_0, \ldots, k_{n-1}) = p \quad \Rightarrow \quad \vdash t(\overline{k_0}, \ldots, \overline{k_{n-1}}) = \overline{p}$$

Lema 7.5.2 *Se f for representável por um termo, então f é representável por uma fórmula.*

Demonstração. Suponha que f seja representável por t. Faça $f(\mathbf{k}) = p$. Então $\vdash t(\overline{\mathbf{k}}) = \overline{p}$. Agora defina a fórmula $\varphi(\vec{x}, y) := t(\vec{x}) = y$. Então temos $\vdash \varphi(\overline{\mathbf{k}}, \overline{p})$. E, portanto, $\overline{p} = y \rightarrow \varphi(\overline{\mathbf{k}}, \overline{p})$. Isso prova $\vdash \varphi(\overline{\mathbf{k}}, \overline{p}) \leftrightarrow \overline{p} = y$. □

Às vezes é conveniente dividir a representabilidade de funções em duas cláusulas.

Lema 7.5.3 *Uma função k-ária é representável por φ sse*

$$f(n_0, \ldots, n_{k-1}) = m \;\Rightarrow\; \vdash \varphi(\overline{n_0}, \ldots, \overline{n_{k-1}}, \overline{m}) \quad \textit{e} \quad \vdash \exists! z \varphi(\overline{n_0}, \ldots, \overline{n_{k-1}}, z).$$

Demonstração. Imediata. Note que a última cláusula pode ser substituída por $\vdash \varphi(\overline{n_0}, \ldots, \overline{n_{k-1}}, z) \rightarrow z = \overline{m}$. □

As funções básicas da aritmética têm seus termos representantes óbvios. Funções bastante simples podem, no entanto, não ser representadas por temos. E.g., a função *signum* é representada por $\varphi(x, y) := (x = 0 \wedge y = 0) \vee (\neg x = 0 \wedge y = 1)$, mas não por um termo. Entretanto podemos facilmente mostrar $\vdash \forall x \exists! y \varphi(x, y)$, e por conseguinte poderíamos conservativamente adicionar a sg a **PA** (cf. teorema 3.4.6). Note que um bom número de predicados e funções úteis têm fórmulas Δ_0 como uma representação.

Lema 7.5.4 *P é representável $\Leftrightarrow$ K_P é representável.*

Demonstração. Suponha que $\varphi(\vec{x})$ represente P. Defina $\psi(\vec{x}, y) = (\varphi(\vec{x}) \wedge (y = 1)) \vee (\neg\varphi(\vec{x}) \wedge (y = 0))$. Então ψ representa K_P, pois se $K_P(\mathbf{k}) = 1$, então $P(\mathbf{k})$, portanto $\vdash \varphi(\overline{\mathbf{k}})$ e $\vdash \psi(\overline{\mathbf{k}}, y) \leftrightarrow (y = 1)$, e se $K_P(\mathbf{k}) = 0$, então $\neg P(\mathbf{k})$, portanto $\vdash \neg\varphi(\overline{\mathbf{k}})$ e $\vdash \psi(\overline{\mathbf{k}}, y) \leftrightarrow (y = 0)$. Reciprocamente, suponha que $\psi(\vec{x}, y)$ represente K_P. Defina $\varphi(\vec{x}) := \psi(\vec{x}, 1)$. Então φ representa P. □

Existe uma classe grande de funções representáveis, que inclui as funções recursivas primitivas.

Teorema 7.5.5 *As funções recursivas primitivas são representáveis.*

Demonstração. Indução sobre a definição de função recursiva primitiva. É simples mostrar que funções iniciais são representáveis. A função constante C_w^k é representada pelo termo $\overline{m}$, a função sucessor S é representada por $x + 1$, e a função de projeção P_i^k é representada por x_i.

As funções representáveis são fechadas sob substituição e recursão primitiva. Indicaremos a prova para o fecho sob recursão primitiva.

Considere

$$\begin{cases} f(\vec{x}, 0) &= g(\vec{x}) \\ f(\vec{x}, y+1) &= h(f(\vec{x}, y), \vec{x}, y) \end{cases}$$

g é representada por φ, h é representada por ψ:

$$g(\vec{n}) = m \Rightarrow \begin{cases} \vdash \varphi(\vec{n}, m) & \text{e} \\ \vdash \varphi(\vec{n}, y) \rightarrow y = m \end{cases}$$

$$h(p,\vec{n},q)=m \Rightarrow \begin{cases} \vdash \psi(p,\vec{n},q,m) & \text{e} \\ \vdash \psi(p,\vec{n},q,y)\rightarrow y=m \end{cases}$$

Afirmação: f é representada por $\sigma(\vec{x},y,z)$, que está imitando $\exists w \in Seq(comp(w) = y+1 \wedge ((w)_0 = g(\vec{x}) \wedge \forall i \leq y((w)_{i+1} = h((w)_i,\vec{x},i) \wedge z = (w)_y)$

$$\begin{aligned} \sigma(\vec{x},y,z) := \quad & \exists w \in Seq(comp(w) = y+1 \wedge \varphi(\vec{x},(w)_0) \wedge \\ & \forall i \leq y(\psi((w)_i,\vec{x},i(w)_{i+1}) \wedge z = (w)_y) \end{aligned}$$

Agora faça $f(\vec{n},p)=m$, então

$$\begin{cases} f(\vec{n},0) = g(\vec{n}) & = a_0 \\ f(\vec{n},1) = h(f(\vec{n},0),\vec{n},0) & = a_1 \\ f(\vec{n},2) = h(f(\vec{n},1),\vec{n},1) & = a_2 \\ \vdots & \\ f(\vec{n},p) = h(f(\vec{n},p-1),\vec{n},p-1) & = a_p = m \end{cases}$$

Ponha $w = \langle a_0,\ldots,a_p\rangle$; note que $comp(w) = p+1$.

$$\begin{aligned} g(\vec{n}) = \quad & f(\vec{n},0)=a_0 \quad \Rightarrow \quad \vdash \varphi(\vec{n},a_0) \\ & f(\vec{n},1)=a_1 \quad \Rightarrow \quad \vdash \psi(a_0,\vec{n},0,a_1) \\ & \qquad \vdots \\ & f(\vec{n},p)=a_p \quad \Rightarrow \quad \vdash \psi(a_{p-1},\vec{n},p-1,a_p) \end{aligned}$$

Por conseguinte, temos $\vdash comp(w) = p+1 \wedge \varphi(\vec{n},a_0) \wedge \psi(a_0,\vec{n},0,a_1) \wedge \ldots \wedge \psi(a_{p-1},\vec{n},p-1,a_p) \wedge (w)_p = m$ e portanto $\vdash \sigma(\vec{n},p,m)$.

Agora temos que provar a segunda parte: $\vdash \sigma(\vec{n},p,z) \rightarrow z = m$. Provamos isso por indução sobre p.

(1) $p = 0$. Observe que $\vdash \sigma(\vec{n},0,z) \leftrightarrow \varphi(\vec{n},z)$, e como φ representa g, obtemos $\vdash \varphi(\vec{n},z) \rightarrow z = \overline{m}$.

(2) $p = q+1$. Hipótese da indução: $\vdash \sigma(\vec{n},q,z) \rightarrow z = \overline{f(\vec{n},q)} (= m)$.

$$\begin{aligned} \sigma(\vec{n},q+1,z) := \quad & \exists w \in Seq(comp(w) = q+2 \wedge \varphi(\vec{n},(w)_0) \wedge \\ & \forall i \leq y(\psi((w)_i,\vec{n},i(w)_{i+1}) \wedge z = (w)_{q+1}) \end{aligned}$$

Agora vemos que

$$\vdash \sigma(\vec{n},q+1,z) \rightarrow \exists u(\sigma(\vec{n},q,u) \wedge \psi(u,\vec{n},q,z)).$$

Usando a hipótese da indução obtemos

$$\vdash \sigma(\vec{n},q+1,z) \rightarrow \exists u(u = f(\vec{n},q) \wedge \psi(u,\vec{n},q,z)).$$

E portanto $\vdash \sigma(\vec{n},q+1,z) \rightarrow \psi(f(\vec{n},q),\vec{n},q,z)$.

Logo, pela propriedade de ψ: $\vdash \sigma(\vec{n},q+1,z) \rightarrow z = f(\vec{n},q+1)$.

□

Agora é mais um passo para mostrar que todas as funções recursivas são representáveis, pois vimos que todas as funções recursivas podem ser obtidas por uma única minimização a partir de um predicado recursivo primitivo.

Teorema 7.5.6 *Todas as funções recursivas são representáveis.*

Demonstração. Mostramos que as funções representáveis são fechadas sob minimização. Como representabilidade para predicados é equivalente a representabilidade para funções, consideramos o caso $f(\vec{x}) = \mu y(\vec{x},y)$ para um predicado P representado por φ, onde $\forall\vec{x}\exists y P(\vec{x},y)$.

Afirmação. $\psi(\vec{x}, y) := \varphi(\vec{x}, y) \wedge \forall z < y \neg\varphi(\vec{x}, y)$ representa $\mu y P(\vec{x}, y)$.

$$\begin{aligned} m = \mu y P(\vec{n}, y) &\Rightarrow P(\vec{n}, m) \wedge \neg P(\vec{n}, 0) \wedge \ldots \wedge \neg P(\vec{n}, m-1) \\ &\Rightarrow \vdash \varphi(\vec{n}, m) \wedge \neg\varphi(\vec{n}, 0) \wedge \ldots \wedge \neg\varphi(\vec{n}, m-1) \\ &\Rightarrow \vdash \varphi(\vec{n}, m) \wedge \forall z < m \neg\varphi(\vec{n}, z) \\ &\Rightarrow \vdash \psi(\vec{n}, m) \end{aligned}$$

Agora, suponha que $\varphi(\vec{n}, y)$ seja dada, então temos $\varphi(\vec{n}, y) \wedge \forall z < y \neg\varphi(\vec{n}, z)$. Isso imediatamente leva a $m \geq y$. Reciprocamente, como $\varphi(\vec{n}, m)$, vemos que $m \leq y$. Logo, $y = m$. Esse argumento informal é facilmente formalizado como $\vdash \varphi(\vec{n}, y) \rightarrow y = m$. □

Estabelecemos que conjuntos recursivos são representáveis. Poder-se-ia talvez esperar que isso pode ser estendido a conjuntos recursivamente enumeráveis. Acontece que isso não é o caso. Consideraremos os conjuntos RE agora.

Definição 7.5.7 *$R(\vec{x})$ é semi-representável em T se $R(\vec{n}) \Leftrightarrow T \vdash \varphi(\vec{n})$ para uma $\varphi(\vec{x})$.*

Teorema 7.5.8 *R é semi-representável $\Leftrightarrow$ R é recursivamente enumerável.*

Corolário 7.5.9 *R é representável $\Leftrightarrow$ R é recursiva.*

Exercícios

1. Mostre
   $\vdash x + y = 0 \rightarrow x = 0 \wedge y = 0$
   $\vdash xy = 0 \rightarrow x = 0 \vee y = 0$
   $\vdash xy = 1 \rightarrow x = 1 \wedge y = 1$
   $\vdash x^y = 1 \rightarrow y = 0 \vee x = 1$
   $\vdash x^y = 0 \rightarrow x = 0 \wedge y \neq 0$
   $\vdash x + y = 1 \rightarrow (x = 0 \wedge y = 1) \vee (x = 1 \wedge y = 0)$

2. Mostre que todas as Σ_1-fórmulas são equivalentes a fórmulas prenex com quantificadores existenciais precedendo os universais limitados. (Dica: Considere a combinação $\forall x < t \exists y \varphi(x, y)$, isso leva a uma seqüência codificada z tal que $\forall x < t \varphi(x, (z)_x)$). I.e., em **PA** fórmulas Σ_1 são equivalentes a fórmulas Σ_1 estritas.)

3. Mostre que se pode contrair quantificadores similares. E.g., $\forall x \forall y \varphi(x, y) \leftrightarrow \forall z \varphi((z)_0, (z)_1)$.

7.6 Derivabilidade

Nesta seção definimos uma codificação para um predicado recursivamente enumerável $Teo(x)$, que diz que "x é um teorema". Devido à minimização e aos limitantes superiores sobre quantificadores, todos os predicados e funções definidos ao longo do caminho são recursivos primitivos. Observe que estamos de volta à teoria da recursão, isto é, à aritmética informal.

Codificação da sintaxe

A função $\ulcorner - \urcorner$ codifica a sintaxe. Para o alfabeto, é dada por

$\wedge$	$\rightarrow$	$\forall$	0	S	$+$	$\cdot$	exp	$=$	$($	$)$	x_i
2	3	5	7	11	13	17	19	23	29	31	p_{11+i}

A seguir codificamos os termos.

$$\ulcorner f(t_1,\ldots,t_n)\urcorner \quad := \quad \langle \ulcorner f\urcorner, \ulcorner(\urcorner, \ulcorner t_1\urcorner, \ldots, \ulcorner t_n\urcorner, \ulcorner)\urcorner\rangle$$

Finalmente codificamos as fórmulas. Note que $\{\wedge, \rightarrow, \forall\}$ é um conjunto funcionalmente completo, portanto os conectivos remanescentes podem ser definidos.

$$\begin{array}{rcl}
\ulcorner(t=s)\urcorner & := & \langle \ulcorner(\urcorner, \ulcorner t\urcorner, \ulcorner=\urcorner, \ulcorner s\urcorner, \ulcorner)\urcorner\rangle \\
\ulcorner(\varphi\wedge\psi)\urcorner & := & \langle \ulcorner(\urcorner, \ulcorner \varphi\urcorner, \ulcorner\wedge\urcorner, \ulcorner \psi\urcorner, \ulcorner)\urcorner\rangle \\
\ulcorner(\varphi\rightarrow\psi)\urcorner & := & \langle \ulcorner(\urcorner, \ulcorner \varphi\urcorner, \ulcorner\rightarrow\urcorner, \ulcorner \psi\urcorner, \ulcorner)\urcorner\rangle \\
\ulcorner(\forall x_i\varphi)\urcorner & := & \langle \ulcorner(\urcorner, \ulcorner\forall\urcorner, \ulcorner x_i\urcorner, \ulcorner \varphi\urcorner, \ulcorner)\urcorner\rangle
\end{array}$$

$Const(x)$ e $Var(x)$ caracterizam os códigos de constantes e variáveis, respectivamente:

$$\begin{array}{rcl}
Const(x) & := & x = \ulcorner 0\urcorner \\
Var(x) & := & \exists i \leq x (p_{11+i} = x) \\
Fnc1(x) & := & x = \ulcorner S\urcorner \\
Fnc2(x) & := & x = \ulcorner +\urcorner \vee x = \ulcorner \cdot\urcorner \vee x = \ulcorner exp\urcorner
\end{array}$$

$Term(x)$ – *x é um termo* – e $Form(x)$ – *x é uma fórmula* – são predicados recursivos primitivos conforme a versão recursiva primitiva do teorema da recursão. Note que codificaremos conforme a notação padrão de função, e.g. $+(x,y)$ ao invés de $x+y$.

$$\begin{array}{rcl}
Term(x) & := & Const(x) \vee Var(x) \vee \\
 & & \Big(Seq(x) \wedge comp(x) = 4 \wedge Fnc1((x)_0) \wedge \\
 & & (x)_1 = \ulcorner(\urcorner \wedge Term((x)_2) \wedge (x)_3 = \ulcorner)\urcorner\Big) \vee \\
 & & \Big(Seq(x) \wedge comp(x) = 5 \wedge Fnc2((x)_0) \wedge \\
 & & (x)_1 = \ulcorner(\urcorner \wedge Term((x)_2) \wedge Term((x)_3) \wedge (x)_4 = \ulcorner)\urcorner\Big)
\end{array}$$

$$Form(x) := \left\{ \begin{array}{l}
Seq(x) \wedge comp(x) = 5 \wedge (x)_0 = \ulcorner(\urcorner \wedge (x)_4 = \ulcorner)\urcorner \wedge \\
\Big(\big(Term((x)_1)\big) \wedge (x)_2 = \ulcorner=\urcorner \wedge Term((x)_3)\Big) \vee \\
\Big(Form((x)_1) \wedge (x)_2 = \ulcorner\wedge\urcorner \wedge Form((x)_3)\Big) \vee \\
\Big(Form((x)_1) \wedge (x)_2 = \ulcorner\rightarrow\urcorner \wedge Form((x)_3)\Big) \vee \\
\Big((x)_1 = \ulcorner\forall\urcorner \wedge Var((x)_2) \wedge Form((x)_3)\Big)\Big)
\end{array} \right.$$

Todos os tipos de noções sintáticas podem ser codificadas em predicados recursivos primitivos, por exemplo, $Livre(x,y)$ – *x é uma variável livre em y*, e $LivrePara(x,y,z)$ – *x é livre para y em z*.

$$Livre(x,y) := \left\{ \begin{array}{l}
\Big(Var(x) \wedge Term(y) \wedge \neg Const(y) \wedge \\
(Var(y) \rightarrow x = y) \wedge \\
(Fnc1((y)_0) \rightarrow Livre(x,(y)_2)) \wedge \\
(Fnc2((y)_0 \rightarrow (Livre(x,(y)_2) \vee Livre(x,(y)_3)))\Big) \\
\qquad \text{ou} \\
\Big(Var(x) \wedge Form(y) \wedge \\
((y)_1 \neq \ulcorner\forall\urcorner \rightarrow (Livre(x,(y)_1) \vee Livre(x,(y)_3)))) \wedge \\
((y)_1 = \ulcorner\forall\urcorner \rightarrow (x \neq (y)_2 \wedge Livre(x,(y)_4)))\Big)
\end{array} \right.$$

$$LivrePara(x,y,z) := \begin{cases} Term(x) \wedge Var(y) \wedge Form(z) \wedge \\ \Big(\big((z)_2 = \ulcorner = \urcorner\big) \vee \\ \big((z)_1 \neq \ulcorner \forall \urcorner \wedge LivrePara(x, y(z)_1) \wedge LivrePara(x, y, (z)_3\big) \vee \\ \big((z)_1 = \ulcorner \forall \urcorner \wedge \neg Livre((z)_2, x) \wedge \\ (Livre(y,z) \rightarrow (Livre((z)_2, x) \wedge Livre(x, y, (z)_3)))\Big)\Big) \end{cases}$$

Tendo codificado esses predicados, podemos definir um operador de substituição Sub tal que $Sub(\ulcorner\varphi\urcorner, \ulcorner x\urcorner, \ulcorner t\urcorner) = \ulcorner\varphi[t/x]\urcorner$.

$$Sub(x,y,z) := \begin{cases} x & \text{se } Const(x) \\ x & \text{se } Var(x) \wedge x \neq y \\ z & \text{se } Var(x) \wedge x = y \\ \langle (x)_, \ulcorner(\urcorner, Sub((x)_2, y, z), \ulcorner)\urcorner\rangle & \text{se } Term(x) \wedge \\ & \quad Fnc1((x)_0) \\ \langle (x)_0, \ulcorner(\urcorner, Sub((x)_2, y, z), Sub((x)_3, y, z), \ulcorner)\urcorner\rangle & \text{se } Term(x) \wedge \\ & \quad Fnc2((x)_0) \\ \langle \ulcorner(\urcorner, Sub((x)_1, y, z), (x)_2, Sub((x)_3, y, z), \ulcorner)\urcorner\rangle & \text{se } Form(x) \wedge \\ & \quad LivrePara(x,y,z) \wedge \\ & \quad (x)_0 \neq \ulcorner \forall \urcorner \\ \langle \ulcorner(\urcorner, (x)_1, Sub((x)_3, y, z), \ulcorner)\urcorner\rangle & \text{se } Form(x) \wedge \\ & \quad LivrePara(z,y,x) \wedge \\ & \quad (x)_0 = \ulcorner \forall \urcorner \\ & \text{caso contrário} \end{cases}$$

Claramente Sub é recursiva primitiva (recursão de curso de valor).

Codificação da derivabilidade

Nosso próximo passo é obter um predicado recursivo primitivo Der que diz que x é derivável com hipóteses $y_0, \ldots, y_{comp(y)-1}$ e conclusão z. Antes disso damos uma codificação de derivações.

derivação inicial

$$[\varphi] = \langle 0, \varphi\rangle$$

$\wedge$ I

$$\begin{bmatrix} D_1 & D_2 \\ \varphi & \psi \\ \hline (\varphi \wedge \psi) \end{bmatrix} = \langle\langle 0, \ulcorner\wedge\urcorner\rangle, \begin{bmatrix} D_1 \\ \varphi \end{bmatrix}, \begin{bmatrix} D_2 \\ \psi \end{bmatrix}, \ulcorner(\varphi \wedge \psi)\urcorner\rangle$$

$\wedge$ E

$$\begin{bmatrix} D \\ \hline (\varphi \wedge \psi) \\ \hline \varphi \end{bmatrix} = \langle\langle 1, \ulcorner\wedge\urcorner\rangle, \begin{bmatrix} D \\ (\varphi \wedge \psi) \end{bmatrix}, \ulcorner\varphi\urcorner\rangle$$

$\rightarrow$ I

$$\begin{bmatrix} \varphi \\ D \\ \psi \\ \hline (\varphi \rightarrow \psi) \end{bmatrix} = \langle\langle 0, \ulcorner\rightarrow\urcorner\rangle, \begin{bmatrix} D \\ \psi \end{bmatrix}, \ulcorner(\varphi \rightarrow \psi)\urcorner\rangle$$

$\rightarrow$ E

$$\begin{bmatrix} D_1 & D_2 \\ \varphi & (\varphi \rightarrow \psi) \\ \hline & \psi \end{bmatrix} = \langle\langle 1, \ulcorner \rightarrow \urcorner\rangle, \begin{bmatrix} D_1 \\ \varphi \end{bmatrix}, \begin{bmatrix} D_2 \\ (\varphi \rightarrow \psi) \end{bmatrix}, \ulcorner \psi \urcorner\rangle$$

RAA

$$\begin{bmatrix} (\varphi \rightarrow \bot) \\ D \\ \bot \\ \hline \varphi \end{bmatrix} = \langle\langle 1, \ulcorner \bot \urcorner\rangle, \begin{bmatrix} D \\ \bot \end{bmatrix}, \ulcorner \varphi \urcorner\rangle$$

$\forall$ I

$$\begin{bmatrix} D \\ \varphi \\ \hline (\forall x \varphi) \end{bmatrix} = \langle\langle 0, \ulcorner \forall \urcorner\rangle, \begin{bmatrix} D \\ \varphi \end{bmatrix}, \ulcorner (\forall x \varphi) \urcorner\rangle$$

$\forall$ E

$$\begin{bmatrix} D \\ (\forall x \varphi) \\ \hline \varphi[t/x] \end{bmatrix} = \langle\langle 1, \ulcorner \forall \urcorner\rangle, \begin{bmatrix} D \\ (\forall x \varphi) \end{bmatrix}, \ulcorner \varphi[t/x] \urcorner\rangle$$

Para Der precisamos de um dispositivo para cancelar hipóteses de uma derivação. Consideramos uma seqüência y de (códigos de) hipóteses e sucessivamente remover os itens u.

$$Cancela(u, y) := \begin{cases} y & \text{se } comp(y) = 0 \\ Cancela(u, cauda(y)) & \text{se } (y)_0 = u \\ \langle (y)_0, Cancela(u, cauda(y))\rangle & \text{se } (y)_0 \neq u \end{cases}$$

Aqui $cauda(y) = z \Leftrightarrow (comp(y) > 0 \wedge \exists x(y = \langle x\rangle * z) \vee (comp(y) = 0 \wedge z = 0)$.

Agora podemos codificar Der, onde $Der(x, y, z)$ significa 'x é o código de uma derivação de uma fórmula com código z de uma seqüência codificada de hipóteses y'.

Na definição de Der, $\bot$ é definida como $(0 = 1)$.

$$
\begin{array}{rl}
Der(x,y,z) := & Form(z) \wedge \bigwedge_{i=0}^{comp(y)-1} Form((i)_v) \wedge \\
& \Big(\big(\exists i < comp(y)(z = (y)_i \wedge x = \langle 0, z \rangle \big) \\
& \qquad \text{ou} \\
& \big(\exists x_1 x_2 \leq x \exists y_1 y_2 \leq x \exists z_1 z_2 \leq x \; y = y_1 * y_2 \wedge \\
& Der(x_1, y_1, z_1) \wedge Der(x_2, y_2, z_2) \wedge \\
& z = \langle \ulcorner(\urcorner, z, \ulcorner\wedge\urcorner, z_2, \ulcorner)\urcorner \rangle \wedge x = \langle \langle 0, \ulcorner\wedge\urcorner \rangle, x_1, x_2, z \rangle \big) \\
& \qquad \text{ou} \\
& \big(\exists u \leq x \exists x_1 \leq x \exists z_1 \leq x Der(x_1, y, z_1) \wedge \\
& (z_1 = \langle \ulcorner(\urcorner, z, \ulcorner\wedge\urcorner, u, \ulcorner)\urcorner \rangle \vee (z_1 = \langle \ulcorner(\urcorner, u, \ulcorner\wedge\urcorner, z, \ulcorner)\urcorner \rangle) \wedge \\
& x = \langle \langle 1, \ulcorner\wedge\urcorner \rangle, x_1, z \rangle \big) \\
& \qquad \text{ou} \\
& \big(\exists x_1 \leq x \exists y_1 \leq x \exists u \leq x \exists z_1 \leq x (y = Cancela(u, y_1) \vee \\
& y = y_1) \wedge Der(x_1, y_1, z_1) \wedge z = \langle \ulcorner(\urcorner, u, \ulcorner\to\urcorner, z_1, \ulcorner)\urcorner \rangle \wedge \\
& x = \langle \langle 0, \ulcorner\to\urcorner \rangle, x_1, z_1 \rangle \big) \\
& \qquad \text{ou} \\
& \big(\exists x_1 x_2 \leq x \exists y_1 y_2 \leq x \exists z_1 z_2 \leq x (y = y_1 * y_2 \wedge \\
& Der(x_1, y_1, z_1) \wedge Der(x_2, y_2, z_2) \wedge \\
& z_2 = \langle \ulcorner(\urcorner, z_1, \ulcorner\to\urcorner, \ulcorner)\urcorner \rangle \wedge x = \langle \langle 1, \ulcorner\to\urcorner \rangle, x_1, x_2, z \rangle) \big) \\
& \qquad \text{ou} \\
& \big(\exists x_1 \leq x \exists z_1 \leq x \exists v \leq x (Der(x_1, y, z_1) \wedge Var(v) \wedge \\
& \bigwedge_{i=0}^{comp(y)-1} \neg Livre(v, (y)_i)) \wedge z = \langle \ulcorner\forall\urcorner, v, \ulcorner(\urcorner, z_1, \ulcorner)\urcorner \rangle \wedge \\
& x = \langle \langle 0, \ulcorner\forall\urcorner, x_1, z_1 \rangle) \big) \\
& \qquad \text{ou} \\
& \big(\exists t \leq x \exists v \leq x \exists x_1 \leq x \exists z_1 \leq x (Var(v) \wedge Term(t) \wedge \\
& LivrePara(t, v, z_1) \wedge z = Sub(z_1, v, t) \wedge \\
& Der(x_1, y, \langle \ulcorner\forall\urcorner, v, \ulcorner(\urcorner, z_1, \ulcorner)\urcorner \rangle \wedge x = \langle \langle 1, \ulcorner\forall\urcorner \rangle, y, z \rangle)) \big) \\
& \qquad \text{ou} \\
& \big(\exists x_1 \leq x \exists y_1 \leq x \exists z_1 \leq x (Der(x_1, y_1, \langle \ulcorner\bot\urcorner \rangle) \wedge \\
& y = Cancela(\langle z, \ulcorner\to\urcorner, \ulcorner\bot\urcorner \rangle, y_1) \wedge x = \langle \langle 1, \ulcorner\bot\urcorner, x_1, z_1 \rangle) \big) \big)
\end{array}
$$

Codificação da demonstrabilidade

Os axiomas da aritmética de Peano são listados no início da seção 7.4. Entretanto, para o propósito de se codificar derivabilidade temos que ser precisos; devemos incluir os axiomas para identidade. Eles são os usuais (veja 2.6 e 2.10.2), incluindo os 'axiomas de congruência' para as operações:

$$
\begin{array}{rcl}
(x_1 = y_1 \wedge x_2 = y_2) & \to & \big(S(x_1) = S(y_1) \wedge x_1 + x_2 = y_1 + y_2 \wedge \\
& & x_1 \cdot x_2 = y_1 \cdot y_2 \wedge x_1^{x_2} = y_1^{y_2} \big)
\end{array}
$$

Esses axiomas podem ser facilmente codificados e reunidos num predicado recursivo $Ax(x)$ – *x é um axioma*. O predicado de demonstrabilidade $Prova(x, z)$ – *x é uma derivação de z a partir dos axiomas de* **PA** – segue imediatamente.

$$
Prova(x,z) \quad := \quad \exists y \leq x \big(Der(x,y,z) \wedge \bigwedge_{i=0}^{comp(y)-1} Ax((y)_i) \big)
$$

Finalmente, podemos definir $Teo(x)$ – *x é um teorema*. Teo é recursivamente enumerável.

$$Teo(z) \quad := \quad \exists x Prova(x, z)$$

Tendo à nossa disposição o predicado da demonstrabilidade, que é Σ_1^0, podemos concluir a prova de 'semi-representável = RE' (Teorema 7.4.9).

Demonstração. Por conveniência, suponha que R seja unária, i.e. um conjunto, e recursivamente enumerável.

- $\Rightarrow$ R é semi-representável por φ. $R(n) \Leftrightarrow \vdash \varphi(\overline{n}) \Leftrightarrow \exists y Prova\,(\ulcorner\varphi(\overline{n})\urcorner, y)$. Note que $\ulcorner\varphi(\overline{n})\urcorner$ é uma função recursiva de n. $Prova$ é recursivo primitivo, portanto R é recursivamente entumerável.

- $\Leftarrow$ R é recursivamente enumerável $\Rightarrow R(n) = \exists x P(n, x)$ para um P recursivo primitivo. $P(n, m) \Leftrightarrow \vdash \varphi(\overline{n}, \overline{m})$ para algum φ. $R(n) \Leftrightarrow P(n, m)$ para algum $m \Leftrightarrow \vdash \varphi(\overline{n}, \overline{m})$ para algum $m \Rightarrow \vdash \exists y \varphi(\overline{n}, y)$. Por conseguinte, temos também que $\vdash \exists y \varphi(\overline{n}, y) \Rightarrow R(n)$. Portanto $\exists y \varphi(\overline{n}, y)$ semi-representa R.

□

7.7 Incompletude

Teorema 7.7.1 (Teorema do ponto fixo) *Para cada fórmula $\varphi(x)$ (com $VL(\varphi) = \{x\}$) existe uma sentença ψ tal que $\vdash \varphi(\overline{\ulcorner\psi\urcorner}) \leftrightarrow \psi$.*

Demonstração. Versão popular: considere uma função de substituição simplificada $s(x, y)$ que é a velha função de substituição para uma variável fixada: $s(x, y) = Sub(x, \ulcorner x_0 \urcorner, y)$. Então defina $\theta(x) := \varphi(s(x, x))$. Seja $m := \ulcorner\theta(x)\urcorner$, então faça $\psi := \theta(\overline{m})$. Note que $\psi \leftrightarrow \theta(\overline{m}) \leftrightarrow \varphi(\overline{s(\overline{m}, \overline{m})}) \leftrightarrow \varphi(\overline{s(\overline{\ulcorner\theta(x)\urcorner}, \overline{m})}) \leftrightarrow \varphi(\overline{\ulcorner\theta(\overline{m})\urcorner})$ $\leftrightarrow \varphi(\overline{\ulcorner\psi\urcorner})$.

Esse argumento funcionaria se houvesse uma função (ou termo) para s na linguagem. Isso poderia ser feito estendendo-se a linguagem com uma quantidade suficiente de funções ("todas as funções recursivas primitivas" certamente bastará). Agora temos que usar fórmulas representantes.

Versão formal: suponha que $\sigma(x, y, z)$ represente a funccão recursiva primitiva $s(x, y)$. Agora suponha que $\theta(x) := \exists y(\varphi(y) \wedge \sigma(x, x, y))$, $m = \ulcorner\theta(x)\urcorner$ e $\psi = \theta(\overline{m})$. Então

$$\psi \leftrightarrow \theta(\overline{m}) \leftrightarrow \exists y(\varphi(y) \wedge \overline{\sigma(\overline{m}, \overline{m}, y)}) \tag{7.1}$$
$$\vdash \forall y(\sigma(\overline{m}, \overline{m}, y) \leftrightarrow y = \overline{s(m, m)})$$
$$\vdash \forall y(\sigma(\overline{m}, \overline{m}, y) \leftrightarrow y = \overline{\ulcorner\theta(\overline{m})\urcorner}) \tag{7.2}$$

Por lógica (7.1) e (7.2) dão $\psi \leftrightarrow \exists y(\varphi(y) \wedge y = \overline{\ulcorner\theta(\overline{m})\urcorner})$ portanto $\psi \leftrightarrow \varphi(\overline{\ulcorner\theta(\overline{m})\urcorner})$ $\leftrightarrow \varphi(\overline{\ulcorner\psi\urcorner})$. □

Definição 7.7.2 *(i)* **PA** *(ou qualquer outra teoria T da aritmética) é chamada de* ω-completa *se $\vdash \exists x \varphi(x) \Rightarrow \vdash \varphi(\overline{n})$ para algum $n \in \mathbb{N}$.*

(ii) T é ω-consistente *se não existe φ tal que ($\vdash \exists x \varphi(x)$ e $\vdash \neg\varphi(\overline{n})$ para todo n) para todo φ.*

Teorema 7.7.3 (Primeiro teorema da incompletude de Gödel) *Se* $\mathbf{PA}$ *for* ω*-consistente então* $\mathbf{PA}$ *é incompleta.*

Demonstração. Considere o predicado $Prova(x, y)$ representado pela fórmula $\overline{Prova}(x, y)$. Seja $\overline{Teo}(x) := \exists y\overline{Prova}(x, y)$. Aplique o teorema do ponto fixo a $\neg\overline{Teo}(x)$: existe uma φ tal que $\vdash \varphi \leftrightarrow \neg\overline{Teo}(\overline{\ulcorner\varphi\urcorner})$. φ, a chamada *sentença de Gödel*, diz em $\mathbf{PA}$: "Eu não sou demonstrável."

Afirmação 1: Se $\vdash \varphi$ então $\mathbf{PA}$ é inconsistente.

Demonstração. $\vdash \varphi \Rightarrow$ existe um n tal que $Prova(\ulcorner\varphi\urcorner, n)$, daí $\vdash \overline{Prova}(\overline{\ulcorner\varphi\urcorner}, \overline{n}) \Rightarrow$ $\vdash \exists y\overline{Prova}(\overline{\ulcorner\varphi\urcorner}, y) \Rightarrow \vdash \overline{Prova}(\overline{\ulcorner\varphi\urcorner}) \Rightarrow \vdash \neg\varphi$. Logo, $\mathbf{PA}$ é inconsistente. □

Afirmação 2: Se $\vdash \neg\varphi$ então $\mathbf{PA}$ é ω-inconsistente.

Demonstração. $\vdash \neg\varphi \Rightarrow \vdash \overline{Teo}(\overline{\ulcorner\varphi\urcorner}) \Rightarrow \vdash \exists x\overline{Prova}(\overline{\ulcorner\varphi\urcorner}, x)$. Suponha que $\mathbf{PA}$ seja ω-consistente; como ω-consistência implica consistência, temos que $\nvdash \varphi$ e, por conseguinte, $\neg Prova(\ulcorner\varphi\urcorner, n)$ para todo n. Logo, $\vdash \neg\overline{Prova}(\overline{\ulcorner\varphi\urcorner}, \overline{n})$ para todo n. Contradiccão. □

□

Observações. Na demonstração acima fizemos uso da representabilidade do predicado de demonstrabilidade, que por sua vez dependeu da representabilidade de todas as funções e predicados recursivos.

Para a representabilidade de $Prova(x, y)$, o conjunto de axiomas tem que ser recursivamente enumerável. Portanto o primeiro teorema da incompletude de Gödel se verifica para todas as teorias recursivamente enumeráveis nas quais as funções recursivas são representáveis. Assim, não se pode tornar $\mathbf{PA}$ completa adicionando-se a sentença de Gödel, pois a teoria resultante seria novamente incompleta.

No modelo padrão $\mathbb{N}$ ou φ ou $\neg\varphi$ é verdadeira. A definição nos habilita a determinar qual delas. Note que os axiomas de $\mathbf{PA}$ são verdadeiros em $\mathbb{N}$, portanto $\models \varphi \leftrightarrow \neg\overline{Teo}(\overline{\ulcorner\varphi\urcorner})$. Suponha que $\mathbb{N} \models \overline{Prova}(\overline{\ulcorner\varphi\urcorner}, \overline{n})$ para algum $n \Leftrightarrow \vdash \overline{Prova}(\ulcorner\varphi\urcorner, \overline{n})$ para algum $n \Leftrightarrow \vdash \varphi \Rightarrow \vdash \neg\overline{Teo}(\overline{\ulcorner\varphi\urcorner}) \Rightarrow \mathbb{N} \models \neg\overline{Teo}(\overline{\ulcorner\varphi\urcorner})$. Contradição. Logo, φ é verdadeira em $\mathbb{N}$. Isso é usualmente expresso como existe um enunciado verdadeiro da aritmética que não é demonstrável'.

Observações. É geralmente aceito que $\mathbf{PA}$ é uma teoria *verdadeira*, ou seja, $\mathbb{N}$ é um modelo de $\mathbf{PA}$, e portanto as condições sobre o teorema de Gödel parecem ser supérfluos. Entretanto, o fato de que $\mathbf{PA}$ é uma teoria verdadeira é baseado num argumento semântico. O refinamento consiste em se considerar teorias arbitrárias, sem o uso de semântica.

O teorema da incompletude pode ser liberado da condição de ω-consistência. Introduzimos para esse propósito o predicado de Rosser:

$$Ros(x) \quad := \quad \exists y(Prova(neg(x), y) \wedge \forall z < y \neg Prova(x, z)),$$

com $neg(\ulcorner\varphi\urcorner) = \ulcorner\neg\varphi\urcorner$. O predicado após o quantificador é representado por $\overline{Prova}(\overline{neg}(x, y), y) \wedge$ $\forall z < y\neg\overline{Prova}(x, z)$. Uma aplicação do teorema do ponto fixo produz um ψ tal que

$$\psi \leftrightarrow \exists y(\overline{Prova}(\overline{\neg\psi}, y) \wedge \forall z < y\neg\overline{Prova}((\overline{\ulcorner\varphi\urcorner}, z)) \qquad (1)$$

Afirmação: **PA** é consistente $\Rightarrow \nvdash \psi$ e $\nvdash \neg\psi$.

Demonstração.

(i) Suponha que $\vdash \psi$. Então existe um n tal que $Prova(\ulcorner\psi\urcorner, n)$, portanto $\vdash \overline{Prova}(\overline{\ulcorner\psi\urcorner}, \overline{n})$ (2)

De (1) e (2) segue que $\vdash \exists y < \overline{n}\overline{Prova}(\overline{\ulcorner\psi\urcorner}, y)$, i.e. $\vdash \overline{Prova}(\overline{\ulcorner\psi\urcorner}, \overline{0}) \vee \ldots \vee \overline{Prova}(\overline{\ulcorner\neg\psi\urcorner}, \overline{n-1})$. Note que $\overline{Prova}$ é Δ_0, daí o seguinte se verifica: $\vdash \sigma \vee \tau \Leftrightarrow \vdash \sigma$ ou $\vdash \tau$, portanto $\vdash \overline{Prova}(\overline{\ulcorner\neg\psi\urcorner}, \overline{0})$ ou $\ldots$ ou $\vdash \overline{Prova}(\overline{\ulcorner\neg\psi\urcorner}, \overline{n-1})$. Logo, $Prova(\ulcorner\neg\psi\urcorner, i)$ para algum $i < n \Rightarrow \vdash \neg\psi \Rightarrow$ **PA** é inconsistente.

(ii) Suponha que $\vdash \neg\psi$. Então $\vdash \forall y(\overline{Prova}(\overline{\ulcorner\neg\psi\urcorner}, y) \rightarrow \exists z < y\overline{Prova}(\overline{\ulcorner\psi\urcorner}, z))$. Também, $\vdash \neg\psi \Rightarrow Prova(\ulcorner\neg\urcorner, n)$ para algum $n \Rightarrow \vdash \overline{Prova}(\overline{\ulcorner\neg\psi\urcorner}, \overline{n})$ para algum $n \Rightarrow \vdash \exists z < \overline{n}\overline{Prova}(\overline{\ulcorner\psi\urcorner}, z) \Rightarrow$ (como acima) $Prova(\ulcorner\psi\urcorner, k)$ para algum $k < n$, portanto $\vdash \psi \Rightarrow$ **PA** é inconsistente.

□

Vimos que verdade em $\mathbb{N}$ não necessariamente implica demontrabilidade em **PA** (ou qualquer outra extensão axiomatizável (recursivamente enumerável)). Entretanto, vimos que **PA** É Σ_1^0-completa, portanto verdade e demonstrabilidade ainda coincidem para enunciados simples.

Definição 7.7.4 *Uma teoria T (na linguagem de* **PA***) é chamada de* Σ_1^0-correta *se* $T \vdash \varphi \Rightarrow \mathbb{N} \models \varphi$ *para* Σ_1^0*-sentenças* φ.

Não entraremos nas questões intrigantes dos fundamentos ou da filosofia da matemática e da lógica. Aceitando o fato de que o modelo padrão $\mathbb{N}$ é um modelo de **PA**, obtemos consistência, corretude e Σ_1^0-corretude de graça. É uma velha tradição em teoria da prova enfraquecer suposições tanto quanto possível, de forma que faça sentido ver o que se pode fazer sem quaisquer noções semânticas. O leitor interessado é remetido à literatura.

Agora apresentamos uma prova alternativa do teorema da incompletude. Aqui usamos o fato de que **PA** é Σ_1^0-correta.

Teorema 7.7.5 **PA** *é incompleta.*

Demonstração. Considere um conjunto RE X que não é recursivo. Ele é semi-representado por uma Σ_1^0-fórmula φ. Seja $Y = \{n \mid \mathbf{PA} \vdash \varphi(\overline{n})\}$.

Pela Σ_1^0-completude obtemos $n \in X \Rightarrow \mathbf{PA} \vdash \varphi(\overline{n})$. Como Σ_1^0-corretude implica consistência, obtemos também $\mathbf{PA} \vdash \neg\varphi(\overline{n}) \Rightarrow n \notin X$, logo, $Y \subseteq X^c$. O predicado de demonstrabilidade nos diz que Y é RE. Agora, X^c não é RE, portanto existe um número k com $k \in (X \cup Y)^c$. Para esse número k sabemos que $\mathbf{PA} \nvdash \neg\varphi(\overline{k})$ e também $\mathbf{PA} \nvdash \varphi(\overline{k})$, pois $\mathbf{PA} \vdash \varphi(\overline{k})$ implicaria, pela Σ_1^0-corretude, que $k \in X$. Como resultado, estabelecemos que $\neg\varphi(\overline{k})$ é verdadeira mas não demonstrável em **PA**, i.e. **PA** é incompleta. □

Quase que imediatamente obtemos a indecidibilidade de **PA**.

Teorema 7.7.6 **PA** *é indecidível.*

Demonstração. Considere o mesmo conjunto $X = \{n \mid \mathbf{PA} \vdash \varphi(\overline{n})\}$ como acima. Se $\mathbf{PA}$ fosse decidível, o conjunto X seria recursivo. Logo, $\mathbf{PA}$ é indecidível. □

Note que obtemos o mesmo resultado para qualquer extensão Σ_1^0-correta axiomatizável de $\mathbf{PA}$. Para resultados mais fortes veja no livro de Smorynski *Logical Number Theory*

que f com $f(n) = \ulcorner \varphi(\overline{n}) \urcorner$ é recursiva primitiva.

Observações. A sentença de Gödel γ "Eu não sou demonstrável" é a negação de uma Σ_1^0-sentença estrita (uma assim-chamada Π_1^0-sentença). Sua negação não pode ser verdadeira (por que?). Portanto $\mathbf{PA} + \neg\gamma$ não é Σ_1^0-correta.

Agora apresentaremos uma outra abordagem à indecidibilidade da aritmética, baseada em conjuntos efetivamente inseparáveis.

Definição 7.7.7 *Sejam φ e ψ fórmulas existenciais: $\varphi = \exists x\varphi'$ e $\psi = \exists x\psi'$. As fórmulas de* comparação de testemunhas *para φ e ψ são dadas por:*

$$\begin{array}{rcl} \varphi \leq \psi & := & \exists x(\varphi'(x) \wedge \forall y < x \neg\psi'(x)) \\ \varphi < \psi & := & \exists x(\varphi'(x) \wedge \forall y \leq x \neg\psi'(x)). \end{array}$$

Lema 7.7.8 (Lema da Redução Informal) *Suponha que φ e ψ sejam Σ_1^0 estritas, $\varphi_1 := \varphi \leq \psi$ e $\psi_1 := \psi < \varphi$. Então*
(i) $\mathbb{N} \models \varphi_1 \to \varphi$
(ii) $\mathbb{N} \models \psi_1 \to \psi$
(iii) $\mathbb{N} \models \varphi \vee \psi \leftrightarrow \varphi_1 \vee \psi_1$
(iv) $\mathbb{N} \models \neg(\varphi_1 \wedge \psi_1)$.

Demonstração. Imediata da definição. □

Lema 7.7.9 (Lema da Redução Formal) *Sejam φ, ψ, φ_1 e ψ_1 como no lema acima.*
(i) $\vdash \varphi_1 \to \varphi$
(ii) $\vdash \psi_1 \to \psi$
(iii) $\mathbb{N} \models \varphi_1 \Rightarrow \vdash \varphi_1$
(iv) $\mathbb{N} \models \psi_1 \Rightarrow \vdash \psi_1$
(v) $\mathbb{N} \models \varphi_1 \Rightarrow \vdash \neg\psi_1$
(vi) $\mathbb{N} \models \psi_1 \Rightarrow \vdash \neg\varphi_1$
(vii) $\vdash \neg(\varphi_1 \wedge \psi_1)$.

Demonstração. (i)–(iv) são conseqüências diretas da definição e da Σ_1^0-completude.
(v) e (vi) são exercícios em dedução natural (use $\forall uv(u < v \vee v \leq u)$); e (vii) segue de (v) (ou (vi)). □

Teorema 7.7.10 (Indecidibilidade de PA) *A relação $\exists y Prova(x, y)$ não é recursiva. Versão popular: $\vdash$ não é decidível para* $\mathbf{PA}$.

Demonstração. Considere dois conjuntos recursivamente enumeráveis efetivamente inseparáveis A e B com fórmulas Σ_1^0-definidas estritas $\varphi(x)$ e $\psi(x)$. Defina $\varphi_1(x) := \varphi(x) \leq \psi(x)$ e $\psi_1(x) := \psi(x) < \varphi(x)$.

$$\begin{array}{rcl}
\text{então } n \in A & \Rightarrow & \mathbb{N} \models \varphi(\overline{n}) \wedge \neg\psi(\overline{n}) \\
& \Rightarrow & \mathbb{N} \models \varphi_1(\overline{n}) \\
& \Rightarrow & \vdash \varphi_1(\overline{n}) \\
\text{e } n \in B & \Rightarrow & \mathbb{N} \models \psi(\overline{n}) \wedge \neg\varphi(\overline{n}) \\
& \Rightarrow & \mathbb{N} \models \psi_1(\overline{n}) \\
& \Rightarrow & \vdash \neg\varphi_1(\overline{n}).
\end{array}$$

Seja $\hat{A} = \{n \mid\vdash \varphi_1(\overline{n})\}$, $\hat{B} = \{n \mid\vdash \neg\varphi_1(\overline{n})\}$, então $A \subseteq \hat{A}$ e $B \subseteq \hat{B}$. **PA** é consistente, portanto $\hat{A} \cap \hat{B} = \emptyset$. $\hat{A}$ é recursivamente enumerável, mas, em razão da inseparatividade efetiva de A e B, não recursivo. Suponha que $\{\ulcorner\sigma\urcorner \mid \vdash \sigma\}$ seja recursivo, i.e. $X = \{k \mid Form(k) \wedge \exists z Prova(k, z)\}$ é recursivo. Considere f com $f(n) = \ulcorner\varphi_1(\overline{n})\urcorner$, então $\{n \mid \exists z Prova(\ulcorner\varphi_1(\overline{n})\urcorner, z)\}$ também é recursivo, i.e. $\hat{A}$ é um separador recursivo de A e B. Contradição. Por conseguinte, X não é recursivo. □

Da indecidibilidade de **PA** imediatamente obtemos uma vez mais o teorema da incompletude:

Corolário 7.7.11 **PA** *é incompleta.*

Demonstração. (a) Se **PA** fosse completa, então do teorema geral "teorias completas axiomatizáveis são decidíveis" seguiria que **PA** era decidível.
(b) Em razão de $\hat{A}$ e $\hat{B}$ serem ambos recursivamente enumeráveis, existe um n com $n \notin \hat{A} \cup \hat{B}$, i.e. $\nvdash \varphi(\overline{n})$ e $\nvdash \neg\varphi(\overline{n})$. □

Observação. Os resultados acima não são de forma alguma ótimos; pode-se representar as funções recursivas em sistemas consideravelmente mais fracos, e daí provar sua incompletude. Existe um número de subsistemas de **PA** que são finitamente axiomatizáveis, como, por exemplo, o sistema Q de Raphael Robinson (cf. Smorynski, *Logical number theory*, p. 368ff), que é incompleto e indecidível. Usando esse fato obtém-se facilmente:

Corolário 7.7.12 (Teorema de Church) *A lógica de predicados é indecidível.*

Demonstração. Sejam $\{\sigma_1, \ldots, \sigma_n\}$ os axiomas de Q, então $\sigma_1, \ldots, \sigma_n \vdash \varphi \Leftrightarrow \vdash (\sigma_1 \wedge \ldots \wedge \sigma_n) \rightarrow \varphi$. Um método de decisão para a lógica de predicados nos forneceria, portanto, um método para Q. □

Observações.

1. Como **HA** é um subsistema de **PA**, a sentença de Gödel γ é certamente independente de **HA**. Portanto, $\gamma \vee \neg\gamma$ não é um teorema de **HA**. Pois se $\mathbf{HA} \vdash \gamma \vee \neg\gamma$, então pela propriedade da disjunção para **HA**, teríamos $\mathbf{HA} \vdash \gamma$ ou $\mathbf{HA} \vdash \neg\gamma$, o que é impossível para a sentença de Gödel. Logo, temos um teorema específico de **PA** que não é demonstrável em **HA**.
2. Como **HA** tem a propriedade da existência, pode-se percorrer a primeira versão da prova do teorema da incompletude, evitando-se o uso de ω-consistência.

Exercícios

1. Mostre que f com $f(n) = \ulcorner t(\overline{n})\urcorner$ é recursiva primitiva.
2. Mostre que f com $f(n) = \ulcorner\varphi(\overline{n})\urcorner$ é recursiva primitiva.
3. Encontre o que $\varphi \rightarrow \varphi \leq \varphi$ significa para um φ como acima.