Expresando las funciones recursivas primitivas en un lenguaje de primer orden para la aritmética

Marcelo Lynch*

Introducción

La idea de este documento es definir un lenguaje y una interpretación adecuada para expresar propiedades aritméticas, es decir, propiedades de los números naturales con la suma y el producto. Trabajaremos con la interpretación $(\mathbb{N}, +, \cdot, 0, 1)$, y nos interesa ver allí qué tipo de funciones y relaciones son expresables, y en particular veremos que podemos expresar cualquier función recursiva primitiva en esa interpretación. **Este resultado es central en la demostración de los teoremas de incompletitud de Gödel**, que fueron posiblemente el acontecimiento más importante en lógica en el siglo XX.

El documento está estructurado como sigue: primero se define qué significa expresar una relación o una función en un lenguaje de primer orden, y se dan ejemplos. Luego se define el lenguaje e intepretación que vamos a usar. Después se demuestra que las funciones iniciales y las operaciones constructivas de funciones recursivas primitivas (composición y recursión primitiva) son expresables en el lenguaje. Esto demuestra que todas las funciones RP son expresables. Finalmente, en un apéndice se exhibe la demostración de que existe una sorprendente codificación expresable en nuestro lenguaje, (y fue utiizada en las demostraciones previas asumiendo su existencia).

Expresando relaciones y funciones

Si $\mathcal{L}$ es un lenguaje de primer orden y $\mathcal{I}$ una interpretación de $\mathcal{L}$ con universo $\mathcal{U}$ sabemos que expresar un conjunto $A \subset \mathcal{U}$ es dar una fórmula con una sola variable libre $\alpha(x)$ tal que

$$V_{\mathcal{I},v_{x=a}}[\alpha(x)] = 1 \iff a \in A$$

.

Podemos extender naturalmente este concepto para expresar **relaciones** en $\mathcal{U}$, utilizando más variables libres:

Definición. Sea $A \subset \mathcal{U}^n$ una relacion n-aria, decimos que una fórmula con exactamente n variables libres $\alpha(x_1, \cdots x_n)$ expresa la relación A si se cumple:

$$V_{\mathcal{I},v\substack{x_1=a_1\\x_2=a_2\\\cdots\\x_n=a_n}}[\alpha(x_1, \cdots, x_n)] = 1 \iff (a_1, a_2, \cdots, a_n) \in A$$

Observemos que el caso de expresar un subconjunto de $\mathcal{U}$ se vuelve el caso particular $n = 1$.

*mlynch@itba.edu.ar - son bienvenidas consultas, comentarios y correcciones

Recordando que una función n-aria $f : \mathcal{U}^n \to \mathcal{U}$ es en definitiva una relación incluida en $\mathcal{U}^{n+1}$, vemos que podemos decir que $\alpha(x_1, \cdots, x_n, y)$, con $n+1$ variables libres, **expresa la función** f si

$$V_{\substack{\mathcal{I},v \\ x_1=a_1 \\ x_2=a_2 \\ \cdots \\ x_n=a_n \\ y=y_0}}[\alpha(x_1, \cdots, x_n, y)] = 1 \iff y_0 = f(a_1, a_2, \cdots, a_n)$$

En lo que sigue vamos a abusar la notación para mayor claridad de escritura, omitiendo $\mathcal{I}$ y v del subíndice y usando los mismos símbolos para las variables de la fórmula y los elementos del universo que se asocian a esa variable por la valuación (como si las igualdades en la valuación modificada fueran "$\underbrace{x_i}_{\text{variable}} = \underbrace{x_i}_{\in\mathbb{N}}$" en vez de "$x_i = a_i$") : así, la condición anterior se escribiría:

$$V[\alpha(x_1, \cdots, x_n, y)] = 1 \iff y = f(x_1, x_2 \cdots, x_n)$$

Así, será claro por el contexto cuando los símbolos en el papel se refieren a una variable (cuando son usados en una fórmula) o al elemento del dominio asociado a la variable por la valuación (cuando son usados fuera de las fórmulas).

Un ejemplo

Sea $\mathcal{L}$ con con igualdad y un símbolo de función binario $\boldsymbol{f}$ y la interpretación $\mathcal{I} = (\mathbb{N}, +)$. Consideremos la relación binaria $\leq$, el orden usual entre los números naturales. Para expresarla necesitamos una fórmula $\alpha(x, y)$ tal que

$$V[\alpha(x, y)] = 1 \iff x \leq y$$

La fórmula[1] $\alpha(x, y) := \exists z \boldsymbol{f}(x, z) = y$ lo cumple:

$$\begin{aligned} & V[\alpha(x, y)] = 1 \\ & \iff \exists z \in \mathbb{N} \text{ tal que } x + z = y \\ & \iff x \leq y \end{aligned}$$

Consideremos ahora la función $g : \mathbb{N}^2 \to \mathbb{N}$ con $g(n, m) = 2n + m$. Para expresarla necesitamos una fórmula con tres variables libres $\alpha(x_1, x_2, y)$ tal que

$$V[\alpha(x_1, x_2, y)] = 1 \iff y = g(x_1, x_2)$$

La fórmula $\gamma(x_1, x_2, y) := \boldsymbol{f}(\boldsymbol{f}(x_1, x_1), x_2) = y$ lo cumple:

$$\begin{aligned} & V[\gamma(x_1, x_2, y)] = 1 \\ & \iff (x_1 + x_1) + x_2 = y \\ & \iff 2x_1 + x_2 = x \\ & \iff g(x_1, x_2) = y \end{aligned}$$

Una observación importante: si quisieramos ahora escribir un enunciado que exprese "para todo x, y se cumple $x \leq g(x, y)''$, la manera de usar "lo que vale la función" en el predicado es utilizando una variable auxiliar z que se "transforme" en $g(x, y)$:

$$\forall x \forall y \forall z \ (\gamma(x, y, z) \to \alpha_{\leq}(x, z)))$$

Esto dice "si $z = g(x, y)$ entonces $x \leq z$", que es otra manera de decir lo que queríamos. Tenemos que hacerlo así porque la unica forma de "usar la funcion g" es con la fórmula $\gamma(x, y, z)$, que lo que dice es "$g(x, y)$ es igual a z": no podemos simplemente obtener un término que denote el valor "suelto" $f(x, y)$.

[1] Vamos a usar := cuando definamos fórmulas o términos de $\mathcal{L}$, para diferenciarlo en ese contexto del símbolo = del lenguaje. Recordemos que siempre que definimos términos o fórmulas estamos definiendo "cadenas de texto", es decir toda referencia a otras fórmulas o términos previamente definidos debe tomarse como una "macroexpansión".

Un lenguaje para la aritmética

Como queremos un lenguaje para hablar de los números naturales con la suma y el producto, en lo que sigue vamos a trabajar con un lenguaje $\mathcal{L}$ con igualdad, dos símbolos de función binaria $\boldsymbol{s}$ y $\boldsymbol{p}$, y dos constantes $\mathbf{0}$ y $\mathbf{1}$. La interpretación será $\mathcal{I} = (\mathbb{N}, +, \cdot, 0, 1)$ es decir tiene universo $\mathbb{N}$, $\boldsymbol{s}$ se interpreta como la suma, $\boldsymbol{p}$ se interpreta como el producto y las constantes $\mathbf{0}$ y $\mathbf{1}$ se interpretan como los números 0 y 1.

Observación: los números 0 y 1 son distinguibles solo usando $\boldsymbol{s}$ y $\boldsymbol{p}$: la fórmula $\forall y\ \boldsymbol{s}(x, y) = y$ distingue al 0 y $\forall y\ \boldsymbol{p}(x, y) = y$ distingue al 1. En este sentido, las constantes son redundantes y podríamos no tenerlas, pero las agregamos para que las fórmulas sean más sucintas, y para lograr términos que se interpreten como cualquier número natural (ver la sección que sigue).

Distinguiendo a cada número natural

Para entrar en calor, veamos como podemos distinguir a cualquier número natural en esta interpretación. Consideremos esta serie de **términos de $\mathcal{L}$**:

$$t_0 := \mathbf{0}$$

$$t_1 := \mathbf{1}$$

$$t_n := \boldsymbol{s}(t_{n-1}, \mathbf{1}) \text{ si } n \geq 1$$

Si $\overline{v} : Term_{\mathcal{L}} \to \mathbb{N}$ es una valuación (extendida) cualquiera para este lenguaje e interpretación, veamos que $\overline{v}(t_n) = n$ para todo $n \in \mathbb{N}$, por inducción en n.

Caso base: $n = 0$ o $n = 1$. En este caso t_n coincide con las constantes del lenguaje, entonces es directo: si $n = 0$, $\overline{v}(t_n) = \overline{v}(\mathbf{0}) = 0$, si $n = 1$ entonces $\overline{v}(t_n) = \overline{v}(\mathbf{1}) = 1$.

Hipótesis inductiva: $\overline{v}(t_n) = n$

Tesis: $\overline{v}(t_{n+1}) = n + 1$. Pero tenemos:

$$\overline{v}(t_{n+1}) = \overline{v}(\boldsymbol{s}(t_n, 1) = \underbrace{\overline{v}(t_n)}_{=n \text{ por HI}} + \overline{v}(\mathbf{1}) = n + 1$$

□

Teniendo esto en cuenta, en lo que sigue vamos a escribir $\boldsymbol{n}$, el número en negrita, para denotar el término t_n (en los casos $n = 0, 1$ estamos escribiendo el símbolo de constante del lenguaje, en los otros lo podemos pensar como una "macro" para el término).

Con esto vemos que podemos distinguir a cualquier número natural n con la fórmula

$$\alpha_n(x) := x = \boldsymbol{n}$$

En efecto:

$$\begin{aligned} & V[x = \boldsymbol{n}] = 1 \\ \iff & x = \overline{v}(\boldsymbol{n}) \\ \iff & x = n \end{aligned}$$

Expresando las funciones recursivas primitivas

Nos disponemos a demostrar que toda función recursiva primitiva es expresable en esta interpretación. Para eso vamos a mostrar que:

1. las funciones iniciales son expresables,
2. si dos funciones son expresables, también lo es su composición, y
3. si dos funciones son expresables también lo es la obtenida por recursión primitiva a partir de ellas

Como es esperable, el paso (3) va a ser el que nos cueste más trabajo.

Dos relaciones útiles

Expresemos dos relaciones que nos van a resultar útiles en lo que sigue. A cada relación $\mathcal{R}$ le haremos corresponder una fórmula $\alpha_{\mathcal{R}}$ que la expresa. Las demostraciones quedan como ejercicio para el lector o lectora:

La relación de orden $\leq$ es expresada por $\alpha_{\leq}(x, y) := \exists z(x + z = y)$

La relación de orden estricto $<$ es expresada por $\alpha_{<}(x, y) := (\alpha_{<}(x, y) \wedge \neg(x = y))$

Las funciones iniciales son expresables

Cero

La función $cero : \mathbb{N} \to \mathbb{N}$ tal que $cero(x) = 0$ es expresable mediante la fórmula

$$\alpha_{cero}(x, y) := (y = \mathbf{0} \wedge x = x)$$

Demostración:

$$\begin{aligned} & V[\alpha_{cero}(x, y)] = 1 \\ \iff & y = 0 \wedge x = x \\ \iff & y = 0 \\ \iff & y = cero(x) \end{aligned}$$

La parte "$x = x$" aparece únicamente para que la variable x aparezca libre en la fórmula (definimos que para que la fórmula exprese a la función n-aria debe tener $n + 1$ variables libres).

Proyecciones

Cualquier función proyección $\pi_k^n : \mathbb{N}^n \to \mathbb{N}$ / $\pi_k^n(a_1 \cdots a_n) = a_k$ es expresable mediante la fórmula:

$$\alpha_{\pi_k^n}(x_1, \cdots x_n, y) := (y = x_k \wedge \bigwedge_{1 \leq i \leq n} x_i = x_i)$$

porque:

$$\begin{aligned} & V[\alpha_{\pi_k^n}(x_1, \cdots, x_n, y)] = 1 \\ & \iff y = x_k \text{ y } (x_1 = x_1 \text{ y } \cdots \text{ y } x_n = x_n) \\ & \iff y = x_k \\ & \iff y = \pi_k^n(x_1, \cdots, x_n) \end{aligned}$$

Donde de nuevo la serie de igualdades "$x_i = x_i$" está solamente para asegurar que figuren como variables libres.

Sucesor

Y la función $suc : \mathbb{N} \to \mathbb{N}$ tal que $suc(x) = x + 1$ es expresable mediante

$$\alpha_{suc}(x, y) := y = \boldsymbol{s}(x, \mathbf{1})$$

porque

$$\begin{aligned} & V[\alpha_{suc}(x, y)] = 1 \\ & \iff y = x + 1 \\ & \iff y = suc(x) \end{aligned}$$

Las composición es expresable

Composición "simple"

Supongamos que $f : \mathbb{N} \to \mathbb{N}$ y $g : \mathbb{N} \to \mathbb{N}$ son expresables mediante las fórmulas $\alpha_f(x, y)$ y $\alpha_g(x, y)$ respectivamente. Entonces $f \circ g$ es expresable mediante

$$\alpha_{f \circ g}(x, y) := \exists z(\alpha_g(x, z) \wedge \alpha_f(z, y))$$

Demostración: la idea intuitiva es que usamos la variable z para "guardar" el resultado de aplicar g y despues "evaluarla" en f. Más formalmente tenemos:

$$\begin{aligned} & V[\alpha_{f \circ g}(x, y)] = 1 \\ & \iff \exists z \in \mathbb{N} \text{ tal que } V[\alpha_g(x, z)] = 1 \text{ y } V[\alpha_f(z, y)] = 1 \\ & \iff \exists z \in \mathbb{N} \text{ tal que } z = g(x) \text{ y } y = f(z) \\ & \iff y = f(g(x)) \\ & \iff y = (f \circ g)(x) \end{aligned}$$

Composición "general"

Supongamos que $f : \mathbb{N}^m \to \mathbb{N}$ es una función expresable y $\alpha_f(x_1, \cdots, x_m, y)$ una fórmula que la expresa, y para todo $1 \leq i \leq m$ sean $g_i : \mathbb{N}^k \to \mathbb{N}$ funciones expresables mediante fórmulas $\alpha_{g_i}(x_1, \cdots, x_k, y)$

Entonces la composición $h : \mathbb{N}^k \to \mathbb{N}$ con $h = f \circ (g_1 \times \cdots \times g_m)$ es expresable mediante la fórmula

$$\alpha_h(x_1, \cdots, x_k, y) := \exists z_1 \cdots \exists z_m(\underbrace{\bigwedge_{1 \leq i \leq m} \alpha_{g_i}(x_1, \cdots, x_k, z_i)}_{\text{"Cada } z_i \text{ es el resultado de aplicar } g_i\text{"}} \wedge \underbrace{\alpha_f(z_1, \cdots, z_m, y))}_{\text{"}y \text{ es el resultado de aplicar } f \text{ con las } z_i\text{''}}$$

La idea es la misma que antes: cuando interpretamos, el valor ligado a z_i (que participa de la fórmula α_{g_i}) "guarda" el resultado de aplicar g_i, y luego todas las z_i "entran a f" en la fórmula α_f. Más formalmente:

$$\begin{aligned}
&V[\alpha_h(x_1, \cdots, x_k, y)] = 1 \\
&\iff \exists z_1, z_2, \cdots, z_m \in \mathbb{N} \text{ tal que } y = f(z_1 \cdots z_m) \text{ y para } 1 \leq i \leq m \text{ es } z_i = g_i(x_1, \cdots, x_k) \\
&\iff y = f(g_1(x_1 \cdots x_k), \cdots, g_m(x_1 \cdots, x_k)) \\
&\iff y = (f \circ (g_1 \times \cdots \times g_m))(x_1, \cdots, x_k) \\
&\iff y = h(x_1, \cdots, x_k)
\end{aligned}$$

Expresando la recursión primitiva

Por qué se nos va a hacer difícil

Recordemos que $f : \mathbb{N} \to \mathbb{N}$ se obtiene por recursión primitiva (de tipo I) a partir de $g : \mathbb{N}^2 \to \mathbb{N}$ si

$$f(0) = k \in \mathbb{N}$$
$$f(n+1) = g(n, f(n))$$

La dificultad de expresar la recursión primitiva es que de alguna manera necesitamos, para encontrar $f(n)$, tener ya disponible el valor de $f(n-1)$, y para eso el de $f(n-2)$, etcétera: en definitiva necesitamos una *secuencia de $n-1$ valores* que logramos aplicando g con el anterior (empezando desde el primero que es el valor de $f(0)$), y por ultimo aplicar g una vez más y obtener $f(n)$.

En definitiva nos encantaría tener una fórmula para f que haga algo así (recordar que los números en negrita son términos que se interpretan como el correspondiente números natural):

$$\begin{aligned}
\alpha_f(x, y) := \exists z_0 \cdots z_{x-1} \\
&(z_0 = \boldsymbol{k} && [\textit{en } z_0 \textit{ "queda" } f(0)] \\
&\wedge \alpha_g(\mathbf{0}, z_0, z_1) && [z_1 = g(0, z_0) = g(0, f(0)) = f(1)] \\
&\wedge \alpha_g(\mathbf{1}, z_1, z_2) && [z_2 = g(0, z_1) = g(1, f(1)) = f(2)] \\
&\cdots \\
&\wedge \alpha_g(\boldsymbol{x-2}, z_{x-2}, z_{x-1}) \\
&\wedge \alpha_g(\boldsymbol{x-1}, z_{x-1}, y)) && [y = g(x-1, z_{x-1}) = g(x-1, f(x-1)) = f(x)]
\end{aligned}$$

A la derecha entre corchetes se explica como se interpreta cada cláusula: en esta "fórmula" es como si los valores fueran construyéndose usando las variables z_i, donde z_i guarda el valor de $f(i-1)$, y el valor "final" en y (que tiene que tener $f(x)$) queda como $g(x-1, f(x-1))$, que es lo que exige la recursión primitiva.

Pero **esta fórmula está increiblemente mal definida**, porque x es una variable de la fórmula, no podemos usarla "elegir" la cantidad de variables z y construir estas "cadenas de $\wedge$ con x elementos". **¡x ni siquiera es un número, es una variable del lenguaje $\mathcal{L}$!** Solamente lo interpretamos como un número cuando trabajamos en la interpretación $\mathcal{I}$.

La salvación

En suma: cuando definamos la fórmula para la función f solo vamos a tener **una cantidad fija de variables** (libres o ligadas) para usar, pero **nos gustaría de alguna manera tener acceso a una secuencia de valores de longitud arbitraria y variable** (donde van a "aparecer", cuando interpretemos la fórmula, los valores "anteriores" de la función). Para eso tenemos que poder **expresar algún tipo de codificación**. Por ahora vamos a enunciar un lema que dice que existe una función que nos va a salvar: nos dejará hacer exactamente lo que necesitamos. En el apéndice se muestra esa función efectivamente existe. Enunciamos entonces el siguiente

> **Lema:** Existe una función $B : \mathbb{N}^3 \to \mathbb{N}$ tal que dada **cualquier** secuencia (finita) de valores $a_0, a_1, \cdots, a_n$, existen dos valores $b, c \in \mathbb{N}$ tal que $B(b, c, i) = a_i$ para $1 \leq i \leq n$. Más aún, la función es expresable mediante una fórmula $\beta(x_1, x_2, x_3, y)$ en el lenguaje para la aritmética que definimos.

Esta función mágica nos garantiza que **cualquier secuencia finita de números puede ser codificada eligiendo convenientemente los primeros dos argumentos de** B, y una vez encontrados estos dos argumentos, la función nos da el i-ésimo elemento de la secuencia si el último argumento es i (para $1 \leq i \leq n$). Por ejemplo, si tenemos la secuencia de números $21, 32, 1, 3, 1$ este lema nos dice que existen dos valores b_0 y c_0 tales que:

$$B(b_0, c_0, 0) = 21$$
$$B(b_0, c_0, 1) = 32$$
$$B(b_0, c_0, 2) = 1$$
$$B(b_0, c_0, 3) = 3$$
$$B(b_0, c_0, 4) = 1$$

Y además podemos expresar en nuestro lenguaje el hecho "$y = B(x_1, x_2, x_3)$" mediante la fórmula $\beta(x_1, x_2, x_3, y)$.

Expresando la recursión primitiva de tipo I

Con la fórmula β que expresa la función B estamos en condiciones de expresar la recursión primitiva. Entonces, sea $g : \mathbb{N} \to \mathbb{N}$ una función expresable mediante la fórmula $\alpha_g(x, y)$, y sea f una función que se obtiene por recursión primitiva a partir de g, es decir que cumple

$$f(0) = k \in \mathbb{N}$$
$$f(n+1) = g(n, f(n))$$

Vamos a construir una fórmula $\alpha_f(x, y)$ que expresará lo siguiente cuando la interpretemos:

$$\begin{aligned}
&(x = 0) \Rightarrow y = k \\
&(x \neq 0) \Rightarrow \exists b\ \exists c\ [B(b, c, 0) = k \\
&\qquad\qquad \wedge\ \forall i < x \text{ se cumple } B(b, c, i+1) = g(i, B(b, c, i)) \\
&\qquad\qquad \wedge\ B(b, c, x) = y\ ]
\end{aligned}$$

Es decir:

- si x es 0 entonces y tiene que ser k, que es $f(0)$
- si $x \neq 0$ (o sea $x > 0$) usamos la función B, condicionando b y c para que se cumpla:

$$B(b, c, 0) = k$$

$$B(b, c, i + 1) = g(i, B(b, c, i))$$

Con estas condiciones estamos invocando exactamente los b y c que codifican la secuencia de $x + 1$ elementos $f(0), f(1), \cdots f(x)$:

$$B(b, c, 0) = k = f(0)$$

$$B(b, c, 1) = g(0, B(b, c, 0) = g(0, f(0)) = f(1)$$

$$\cdots$$

$$B(b, c, x - 1) = g(0, B(b, c, x - 2) = g(x - 2, f(x - 2)) = f(x - 1)$$

$$B(b, c, x) = g(0, B(b, c, x - 1) = g(x - 1, f(x - 1)) = f(x)$$

.

Finalmente, imponemos que $B(b, c, x) = y$: esto implica que y tiene que ser $B(b, c, x) = f(x)$, que es lo que queremos.

Para escribirla en el lenguaje $\mathcal{L}$ tenemos que usar $\alpha_g(x_1, x_2, y)$ para expresar "$y = g(x_1, x_2)$", $\beta(x_1, x_2, x_3, y)$ para expresar "$y = B(x_1, x_2, x_3)$", $\alpha_<(x, y)$ para expresar "$x < y$", y $\boldsymbol{s}(i, \mathbf{1})$ para expresar "$i + 1$", y además hay que pasar por variables auxiliares (z_1 y z_2 en la fórmula de abajo) para componer funciones, entonces queda esta fórmula terrible:

$$\begin{aligned}
\alpha_f(x, y) := ((x = \mathbf{0} \to y = \boldsymbol{k}) \\
\wedge (\neg\, x = \mathbf{0} \\
\to \exists b\, \exists c\, (&\underbrace{\beta(b, c, \mathbf{0}, \boldsymbol{k})}_{B(b,c,0)=k} \\
\wedge \forall i\, (&\underbrace{\alpha_<(i, x)}_{i<x} \to \underbrace{(\exists z_1 \exists z_2 (\underbrace{\beta(b, c, i, z_1)}_{z_1=B(b,c,i)} \wedge \underbrace{\alpha_g(i, z_1, z_2)}_{z_2=g(i,z_1))} \wedge \underbrace{\beta(b, c, \boldsymbol{s}(i, \mathbf{1}), z_2)}_{B(b,c,i+1)=z_2}))}_{B(b,c,i+1)=g(i,B(b,c,i))} \\
\wedge &\underbrace{\beta(b, c, x, y)}_{y=B(b,c,x)}))
\end{aligned}$$

Para decir que α_f expresa f debemos mostrar que $V_{x=a,y=b}[\alpha_f(x, y)] = 1 \iff y = f(x)$. La demostración a partir de las sucesivas aplicaciones de V a la fórmula es muy engorrosa, así que no la haremos aquí, pero la idea es:

- para ($\Rightarrow$): como las fórmulas $\beta, \alpha_g, \alpha_<$, α_{suc} expresan B, g, $<$ y suc respectivamente, si la fórmula α_f "es verdadera" (o sea: si $V[\alpha_f] = 1$) entonces esos b y c "que existen" codifican exactamente la secuencia $f(0), \cdots, f(x)$ y entonces y siempre vale $f(x)$, y entonces queda demostrado
- para ($\Leftarrow$): partimos de que $y = f(x)$, si $x = 0$ entonces $y = f(0) = k$ entonces $V_{v_{\cdots}}(\alpha_f) = 1$, y si $x > 0$ el lema que enunciamos nos dice que **sí existen** b **y** c que cumplen lo que dice la fórmula (o sea que $B(b, c, i) = f(i)$ para $1 \leq i \leq x$), entonces $V[\alpha_f] = 1$, y queda demostrado.

Expresando la recursión primitiva de tipo II

La generalización para más de una variable usa exactamente la misma idea, lo único que cambia es que ahora las funciones tienen más variables entonces la fórmula se escribe un poquito distinto.

Así, sea $k > 1$ y $f : \mathbb{N}^k \to \mathbb{N}$ una función que se obtiene por recursión primitiva a partir de $g : \mathbb{N}^{k-1} \to \mathbb{N}$ y $h : \mathbb{N}^{k+1} \to \mathbb{N}$, es decir:

$$f(x_1, \cdots, x_{k-1}, 0) = g(x_1, \cdots x_{k-1})$$
$$f(x_1, \cdots, x_{k-1}, n+1) = h(x_1, \cdots, x_{k-1}, n, f(x_1, \cdots, n))$$

Entonces si g y h son expresables mediante fórmulas $\alpha_g(x_1, \cdots, x_{k-1}, y)$ y $\alpha_h(x_1, \cdots, x_k, x_{k+1}, y)$ respectivamente, f también lo es, mediante la fórmula $\alpha_f(x_1, \cdots, x_k, y)$ que exprese esto:

$$\begin{aligned}
&(x_k = 0) \Rightarrow y = g(x_1, \cdots x_{k-1}) \\
&(x_k \neq 0) \Rightarrow \exists b\ \exists c\ [B(b, c, 0) = g(x_1, \cdots, x_{k-1}) \\
&\qquad\qquad \wedge \forall i < x \text{ se cumple } B(b, c, i+1) = h(x_1, \cdots, x_{k-1}, i, B(b, c, i)) \\
&\qquad\qquad \wedge B(b, c, x) = y\ ]
\end{aligned}$$

La fórmula en $\mathcal{L}$ es:

$$\begin{aligned}
\alpha_f(x_1, \cdots, x_k, y) := (x_k = \mathbf{0} &\to \alpha_g(x_1, \cdots, x_{k-1}, y) \\
\wedge (\neg\ x_k = \mathbf{0} \\
&\to \exists b\ \exists c\ (\exists z(\alpha_g(x_1, \cdots, x_{k-1}, z) \wedge \beta(b, c, \mathbf{0}, z)) \\
&\quad \wedge \forall i\ (\alpha_<(i, x_k) \to (\exists z_1 \exists z_2(\beta(b, c, i, z_1) \wedge \alpha_h(x_1, \cdots, x_{k-1}, i, z_1, z_2) \wedge \beta(b, c, \boldsymbol{s}(i, \mathbf{1}), z_2))) \\
&\quad \wedge \beta(b, c, x, y)))
\end{aligned}$$

Y la demostración de que efectivamente α_f expresa f sigue la misma línea que en la de recursión de tipo I que describimos en la sección anterior.

Conclusión: las funciones recursivas primitivas son expresables en nuestro lenguaje

Si f es una función RP podemos definir su complejidad como la mínima cantidad de pasos (en el sentido: composiciones o recursión primitiva con otras funciones RP) necesarios para construirla a partir de funciones iniciales. Por definición de las funciones RP este numero siempre existe y es un número natural, entonces podemos demostrar nuestra proposición por inducción en la complejidad.

> **Proposición**: Si f es una función recursiva primitiva entonces f es expresable en nuestro lenguaje $\mathcal{L}$ para la aritmética con la interpretación $\mathcal{I} = (\mathbb{N}, +, \cdot, 0, 1)$

Demostración: *Caso base*: Si la complejidad de f es 0 entonces f es una función inicial, y demostramos que las funciones iniciales son expresables.

Hipótesis inductiva: Si f es una función RP de complejidad $k < n$ entonces f es expresable.
Tesis: Si f es una función RP de complejidad $n + 1$ entonces f es expresable.

Ciertamente: sea f una función RP de complejidad $n + 1$, entonces pasa alguno de los siguientes:

1. f se obtuvo por composición de dos funciones g_1 y g_2

2. f se obtuvo por recursión primitiva de tipo I a partir de una g recursiva primitiva

3. f se obtuvo por recursión primitiva de tipo II a partir de dos g_1 y g_2 que también son RP.

En los tres casos, las funciones con las que se construye f son funciones RP con complejidad menor a $n+1$ (pues la construcción de f constituye un último paso, entonces pueden construirse con al menos un paso menos). Entonces por hipótesis inductiva son expresables, y con lo que ya demostramos podemos en los tres casos afirmar que entonces f también es expresable.

□

Corolario: Cualquier conjunto recursivo primitivo es expresable en $(\mathbb{N}, +, \cdot, 0, 1)$.

Demostración: Un conjunto $A \subset \mathbb{N}$ es RP si su función característica $\chi_A : \mathbb{N} \to \{0, 1\}$ es RP. Pero si χ_A es RP entonces, por lo que acabamos de demostrar, existe una fórmula $\alpha_{\chi_A}(x, y)$ que la expresa. Entonces la fórmula

$$\alpha_A(x) := \alpha_{\chi_A}(x, \mathbf{1})$$

expresa al conjunto A:

$$V[\alpha_{\chi_A}(x, \mathbf{1})] = 1 \iff 1 = \chi_A(x) \iff x \in A$$

□

Apéndice

Existencia y expresión de la función B

Para terminar, vamos a definir la función B del lema y probar que tiene las propiedades que dijimos, es decir, demostraremos el lema. Esta función se llama en la literatura "función beta de Gödel".

Sea entonces

$$B : \mathbb{N}^3 \to \mathbb{N}$$

$$B(x, y, i) = resto(x, (i + 1) \cdot y + 1)$$

donde $resto(x, y)$ es el resto de dividir x por y.

Afirmación: dada cualquier secuencia de numeros naturales $a_0, a_1, \cdots, a_k$ existen b y c tales que $B(b, c, i) = a_i$ para $0 \leq i \leq k$.

Demostración: Sea $a_0, a_1, \cdots, a_k$ una secuencia de $k + 1$ numeros naturales, y sea s un número mayor que todos los a_k y que k, por ejemplo $s = 1 + max\{a_0, \cdots a_k, k\}$.

Ahora definimos los números $m_i = (i+1)s! + 1$, con $0 \leq i \leq k$, (notar que tiene el formato del segundo argumento a $resto$ que aplica B), es decir:

$$\begin{aligned}
m_0 &= s! + 1 \\
m_1 &= 2 \cdot s! + 1 \\
m_2 &= 3 \cdot s! + 1 \\
&\cdots \\
m_{k-1} &= k \cdot s! + 1 \\
m_k &= (k + 1) \cdot s! + 1
\end{aligned}$$

Afirmación: Estos números son coprimos dos a dos: $i \neq j \Rightarrow m_i \perp m_j$ para $0 \leq i, j \leq k$.

Demostración: Supongamos que no para llegar a un absurdo, es decir, supongamos que existe p primo tal que ${}^{p}|_{m_i}$ y ${}^{p}|_{m_j}$ con $0 \leq i < j \leq k$. Vamos a usar estas propiedades de la divisibilidad:

1. ${}^{a}|_{b}$ y ${}^{a}|_{c} \Rightarrow {}^{a}|_{b-c}$
2. ${}^{a}|_{b} \Rightarrow {}^{a}|_{c \cdot b}$ para cualquier $c \in \mathbb{Z}$
3. ${}^{a}|_{b \cdot c} \Rightarrow {}^{a}|_{b}$ ó ${}^{a}|_{c}$ para todo $b, c \in \mathbb{Z}$
4. ${}^{a}|_{b}$ y ${}^{b}|_{c} \Rightarrow {}^{a}|_{c}$

Si p divide a m_i y m_j entonces (por (1)) p divide a la diferencia $m_j - m_i$, es decir ${}^{p}|_{(j-i)s!}$, y entonces (por (3)) ${}^{p}|_{(j-i)}$ o ${}^{p}|_{s!}$.

Pero $j - i < k < s$, entonces $(j-i)$ es un factor de $s!$, entonces ${}^{j-i}|_{s!}$ y entonces forzosamente tenemos que ${}^{p}|_{s!}$ (porque si p divide a un factor de $s!$ entonces también divide a $s!$, la propiedad (4) que escribimos arriba). Entonces también ${}^{p}|_{(i+1)\cdot s!}$, porque $i+1$ es un número entero (propiedad (2)).

Pero habíamos dicho que ${}^{p}|_{m_i=(i+1)s!+1}$, entonces ${}^{p}|_{(i+1)s!+1-((i+1)\cdot s!)}$, es decir ${}^{p}|_{1}$. ¡**Absurdo**! p es primo. El absurdo vino de suponer que existian m_i, y m_j distintos que no eran coprimos, entonces debe ser que todos son coprimos dos a dos.

□

Consideremos ahora el sistema de congruencias

$$\begin{aligned} x &\equiv a_0 \ (mod\ m_0) \\ x &\equiv a_1 \ (mod\ m_1) \\ x &\equiv a_2 \ (mod\ m_2) \\ &\cdots \\ x &\equiv a_k \ (mod\ m_k) \end{aligned}$$

Como los números $m_0, \cdots, m_k$ son coprimos dos a dos, entonces por el Teorema Chino del Resto existe una única solución módulo M, donde $M = m_0 \cdot m_1 \cdots m_k$. Sea $x_0 \in \mathbb{N}$ la única solución que cumple $0 \leq x_0 < M$. Entonces:

$$\begin{aligned} x_0 &\equiv a_0 \ (mod\ m_0) \\ x_0 &\equiv a_2 \ (mod\ m_1) \\ x_0 &\equiv a_3 \ (mod\ m_2) \\ &\cdots \\ x_0 &\equiv a_k \ (mod\ m_k) \end{aligned}$$

Notemos que como $0 \leq a_i < m_i$ (porque $s > a_i$ y la construcción de m_i), entonces a_i **es efectivamente el resto de dividir** x_0 **por** $m_i = (i+1)s! + 1$, entonces tenemos que:

$$\begin{aligned} a_0 &= resto(x_0, s! + 1) = B(x_0, s!, 0) \\ a_1 &= resto(x_0, 2 \cdot s!) = B(x_0, s!, 1) \\ &\cdots \\ a_{k-1} &= resto(x_0, k \cdot s! + 1) = B(x_0, s!, k-1) \\ a_k &= resto(x_0, (k+1) \cdot s! + 1) = B(x_0, s!, k) \end{aligned}$$

Y así demostramos lo que queríamos: existen b y c naturales tales que $B(b, c, i) = a_i$ para todo $0 \leq i \leq k$ (con los nombres que elegimos en la demostración, $b = x_0$ y $c = s!$).

□

Existe una fórmula β que expresa a B en nuestra interpretación

Finalmente, vemos que la función B puede expresarse con una fórmula $\beta(x_1, x_2, x_3, y)$ de nuestro lenguaje. Para eso primero expresaremos la función *resto*. Recordemos que el resto de dividir a por b es el único número r con $0 \leq r < b$ tal que $a = bq + r$ para algún $q \in \mathbb{N}$. Así, la fórmula que expresa la función *resto* es:

$$\alpha_{resto}(x_1, x_2, y) := (\alpha_{\leq}(\mathbf{0}, y) \wedge \alpha_{<}(y, x_2) \wedge \exists q \; \boldsymbol{s}(\boldsymbol{p}(q, x_2), y) = x_1)$$

α_{resto} expresa *resto*:

$$\begin{aligned} & V[\alpha_{resto}(x_1, x_2, y)] = 1 \\ & \iff 0 \leq y,\ y < x_2, \text{ y existe } q \in \mathbb{N} \text{ tal que } qx_2 + y = x_1 \\ & \iff resto(x_1, x_2) = y \end{aligned}$$

Finalmente, la fórmula β que expresa B es

$$\beta(x_1, x_2, x_3, y) = \alpha_{resto}(x_1, \boldsymbol{s}(\boldsymbol{p}(\boldsymbol{s}(x_3, \mathbf{1}), x_2), \mathbf{1}), y)$$

En efecto:

$$\begin{aligned} & V[\beta(x_1, x_2, x_3, y)] = 1 \\ & \iff y = resto(x_1, (((x_3 + 1) \cdot x_2) + 1)) && \textit{porque } \alpha_{resto} \textit{ expresa la función resto} \\ & \iff y = resto(x_1, (x_3 + 1) \cdot x_2 + 1)) \\ & \iff y = B(x_1, x_2, x_3) \end{aligned}$$

□