Modelos de Computación y Complejidad

Grado en Ingeniería Informática. Tecnologías Informáticas

Tema 2: Funciones Computables

David Orellana Martín
Mario de J. Pérez Jiménez

Grupo de investigación en Computación Natural
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Curso 2021-2022

Objetivos

* Nueva metodologia para establecer la GOTO–computabilidad de funciones.
* Operaciones básicas con funciones y predicados.
  - Composición.
  - Recursión primitiva.
  - Minimización no acotada (μ-recursión) de funciones.
  - Minimización acotada de funciones.
  - Cuantificación de predicados.
* Funciones relevantes que son computables en el modelo GOTO.
* Codificación de sucesiones de números naturales.
  - La **codificación** derivada de la función **par**.
  - La **codificación de Gödel**.

Funciones GOTO–computables: nueva metodología

De acuerdo con la definición:

- ⋆ Para probar que una función es GOTO-computable, es suficiente proporcionar un programa GOTO que la calcule.

Nueva metodología:

- ⋆ Se van a introducir una serie de **operaciones con funciones** de tal manera que esas operaciones sean **estables** por GOTO-computabilidad.

- ⋆ Para probar que una función f es GOTO-computable será suficiente describir f a partir de alguna de las operaciones antes definidas, de manera que todas las funciones que intervienen sean GOTO-computables.

Composición de funciones

Sean las funciones parciales $g : \mathbb{N}^k \dashrightarrow \mathbb{N}$ y $h_1, .., h_k : \mathbb{N}^p \dashrightarrow \mathbb{N}$.

La función parcial $f : \mathbb{N}^p \dashrightarrow \mathbb{N}$ obtenida por **composición de g y $h_1, \ldots, h_k$** se define así: $f(\vec{x}) = g(h_1(\vec{x}), \ldots, h_k(\vec{x}))$, para cada $\vec{x} = (x_1, \ldots, x_p) \in \mathbb{N}^p$.

Notaremos $\boxed{f = \mathcal{C}(g; h_1, \ldots, h_k)}$

(a) La **composición** de funciones **parciales** es una función **parcial**.

(b) La **composición** de funciones **totales** es una función **total**.

(c) La **composición** de funciones GOTO–computables es una función GOTO–computable.

* En efecto, si $g : \mathbb{N}^k \dashrightarrow \mathbb{N}, h_1, ..., h_k : \mathbb{N}^p \dashrightarrow \mathbb{N}$ son funciones GOTO-computables, entonces el siguiente programa GOTO con macros, calcula $f = \mathcal{C}(g; h_1, ..., h_k)$:

$$\boxed{\begin{array}{c} Z_1 \longleftarrow h_1(X_1, ..., X_p) \\ \vdots \\ Z_k \longleftarrow h_k(X_1, ..., X_p) \\ Y \longleftarrow g(Z_1, ..., Z_k) \end{array}}$$

Ejemplo de composición de funciones

Vamos a probar que la aplicación f de $\mathbb{N}^3$ en $\mathbb{N}$ definida por $f(a,b,c) = b \cdot c + a$ es una función GOTO-computable.

En efecto: se verifica $f(a,b,c) = Sum(Prod(b,c),a)$.

Teniendo presente que $\Pi_1^{(3)}(a,b,c) = a; \Pi_2^{(3)}(a,b,c) = b$ y $\Pi_3^{(3)}(a,b,c) = c$ se tiene:

$$f(a,b,c) = Sum(\underbrace{Prod(\Pi_2^{(3)}(a,b,c), \Pi_3^{(3)}(a,b,c))}, \overbrace{\Pi_1^{(3)}(a,b,c)})$$

Por tanto: $f(a,b,c) = \mathcal{C}(Sum; \underbrace{\mathcal{C}(Prod; \Pi_2^{(3)}, \Pi_3^{(3)})}, \overbrace{\Pi_1^{(3)}})(a,b.c)$

En consecuencia, $f = \mathcal{C}(Sum; \underbrace{\mathcal{C}(Prod; \Pi_2^{(3)}, \Pi_3^{(3)})}, \overbrace{\Pi_1^{(3)}})$

Ahora bien, las funciones $Sum, Prod, \Pi_2^{(3)}, \Pi_3^{(3)}$ y $\Pi_1^{(3)}$ son GOTO-computables. Por tanto, la función f, obtenida a partir de éstas por composición, también será GOTO-computable.

Recursión Primitiva

Sean las funciones parciales $g : \mathbb{N}^k \dashrightarrow \mathbb{N}$ y $h : \mathbb{N}^{k+2} \dashrightarrow \mathbb{N}$.

La función parcial $f : \mathbb{N}^{k+1} \dashrightarrow \mathbb{N}$ obtenida por **recursión primitiva a partir de g y h** se define así: para cada $\vec{x} \in \mathbb{N}^k$ y cada $z \in \mathbb{N}$ se tiene que

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

Notaremos $\boxed{f = \mathcal{R}(g, h)}$.

- **Extensión al caso** $k = 0$. La función parcial $f : \mathbb{N} \dashrightarrow \mathbb{N}$ obtenida por **recursión primitiva a partir de la constante** $c \in \mathbb{N}$ **y de la función parcial** $h : \mathbb{N}^2 \dashrightarrow \mathbb{N}$, se define como sigue:

$$\begin{cases} f(0) = c \\ f(z+1) = h(z, f(z)), \text{para cada } z \in \mathbb{N} \end{cases}$$

En este caso escribiremos $\boxed{f = \mathcal{R}(c, h)}$.

Recursión Primitiva

(a) Toda función obtenida por **recursión primitiva** a partir de funciones **parciales** es una función **parcial**.

(b) Toda función obtenida por **recursión primitiva** a partir de funciones **totales** es una función **total**.

(c) Toda función obtenida por **recursión primitiva** a partir de funciones GOTO–computables es GOTO–computable.

* Sean $g : \mathbb{N}^k \dashrightarrow \mathbb{N}$, $h : \mathbb{N}^{k+2} \dashrightarrow \mathbb{N}$ funciones GOTO–computables. El siguiente programa GOTO con macros calcula $f = \mathcal{R}(g; h)$:

$$
\begin{array}{ll}
 & Y \longleftarrow g(X_1, ..., X_k) \\
[A] & IF\ X_{k+1} = 0\ GOTO\ E \\
 & Y \longleftarrow h(X_1, ..., X_k, Z, Y) \\
 & Z \longleftarrow Z + 1 \\
 & X_{k+1} \longleftarrow X_{k+1} - 1 \\
 & GOTO\ A
\end{array}
$$

Ejemplos de funciones definidas por recursión primitiva

1. La función suma en $\mathbb{N}$.

Sum es la única aplicación de $\mathbb{N} \times \mathbb{N}$ en $\mathbb{N}$ que verifica las condiciones siguientes:

(1) $Sum(a, 0) = a$, para cada $a \in \mathbb{N}$.

(2) $Sum(a, b+1) = Sum(a, b) + 1$, para cada $a, b \in \mathbb{N}$.

Veamos que la función *Sum* se puede describir por recursión primitiva, a partir de ciertas funciones g y h. En efecto: si $Sum = \mathcal{R}(g, h)$, entonces se verificaría:

(3) $Sum(a, 0) = g(a)$, para cada $a \in \mathbb{N}$.

(4) $Sum(a, b+1) = h(a, b, Sum(a, b))$, para cada $a, b \in \mathbb{N}$.

Comparando las relaciones (3) y (4) con (1) y (2) se deduce

* $g(a) = a$, para cada $a \in \mathbb{N}$.

* $h(a, b, Sum(a, b)) = Sum(a, b) + 1$, para cada $a, b \in \mathbb{N}$.

Considerando las funciones $g = Id_{\mathbb{N}}$ y $h(a, b, c) = c + 1$ se deduce que $Sum = \mathcal{R}(g, h)$.

Teniendo presente que g y h son funciones GOTO-computables, se concluye que la función *Sum* también es GOTO-computable.

Ejemplos de funciones definidas por recursión primitiva

2. La función producto en $\mathbb{N}$.

Prod es la única aplicación de $\mathbb{N} \times \mathbb{N}$ en $\mathbb{N}$ que verifica las condiciones siguientes:

(1) $Prod(a, 0) = 0$, para cada $a \in \mathbb{N}$.

(2) $Prod(a, b+1) = \mathrm{Prod(a, b)} + \mathrm{a}$, para cada $a, b \in \mathbb{N}$.

Veamos que la función *Prod* se puede describir por recursión primitiva, a partir de ciertas funciones g y h. En efecto: si $Prod = \mathcal{R}(g, h)$, entonces se verificaría:

(3) $Prod(a, 0) = \mathrm{g(a)}$, para cada $a \in \mathbb{N}$.

(4) $Prod(a, b+1) = \mathrm{h(a, b, Prod(a, b))}$, para cada $a, b \in \mathbb{N}$.

Comparando las relaciones (3) y (4) con (1) y (2) se deduce

* $\mathrm{g(a)} = 0$, para cada $a \in \mathbb{N}$.
* $\mathrm{h(a, b, Prod(a, b))} = \mathrm{Prod(a, b)} + \mathrm{a}$, para cada $a, b \in \mathbb{N}$.

Considerando las funciones $g = \mathcal{O}^{(1)}$ y $h(a, b, c) = c + a$ se deduce que $Prod = \mathcal{R}(g, h)$.

Teniendo presente que g y h son funciones GOTO-computables, se concluye que la función *Prod* también es GOTO-computable.

Ejemplos de funciones definidas por recursión primitiva

3. La función factorial en $\mathbb{N}$.

Fact es la única aplicación de $\mathbb{N}$ en $\mathbb{N}$ que verifica las condiciones siguientes:

(1) $Fact(0) = 1$.

(2) $Fact(b+1) = (b+1) \cdot Fact(b)$, para cada $b \in \mathbb{N}$.

Veamos que la función *Fact* se puede describir por recursión primitiva, a partir de una constante $c \in \mathbb{N}$ y una cierta función h. En efecto: si $Fact = \mathcal{R}(c, h)$, entonces se verificaría:

(3) $Fact(0) = c$.

(4) $Fact(b+1) = h(b, Fact(b))$, para cada $b \in \mathbb{N}$.

Comparando las relaciones (3) y (4) con (1) y (2) se deduce

* $c = 1$.
* $h(b, Fact(b)) = (b+1) \cdot Fact(b)$, para cada $b \in \mathbb{N}$.

Considerando la constante $c = 1$ y la función $h(b, c) = (b+1) \cdot c$, se deduce que $Fact = \mathcal{R}(c, h)$.

Teniendo presente que h es una función GOTO-computable, se concluye que la función *Fact* también es GOTO-computable.

Minimización no acotada de funciones

Sea $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$, $(k \geq 1)$. La función $f_\mu : \mathbb{N}^k \longrightarrow \mathbb{N}$ obtenida por **minimización no acotada** a partir de f se define así: para cada $\vec{x} \in \mathbb{N}^k$ se tiene

$$f_\mu(\vec{x}) = \begin{cases} \min\{y : f(\tilde{x}, y) = 0 \wedge \forall t < y\,(f(\tilde{x}, t)\downarrow)\} & \textit{si existe} \\ \uparrow & \textit{e.o.c.} \end{cases}$$

Notaremos $f_\mu(\vec{x}) = \mu y\,(f(\tilde{x}, y) = 0)$.

- Si una función f es de aridad $k+1$, entonces f_μ es de aridad k.
- En la definición se ha considerado el menor **y** tal que $f(\vec{x}, y) = 0$. En lugar de **0** se podría haber elegido **cualquier otro número natural**.
- En la definición, la variable **y** que se minimiza es la **última componente** de la $(k+1)$-tupla. Se podría haber considerado **cualquier otra componente**.
- La condición $\forall t < y\,(f(\vec{x}, t)\downarrow)$ es esencial para que la construcción de f_μ sea "**mecánica**".

Minimización no acotada de funciones

* Existen funciones **totales** cuya función obtenida por minimización no acotada **no** es **total**.
  - Consideremos la función suma, $Sum(x, y) = x + y$. Entonces

$$Sum_\mu(x) = \mu y\,(Sum(x, y) = 0) = \begin{cases} 0 & \text{si } x = 0 \\ \uparrow & \text{e.o.c.} \end{cases}$$

* Existen funciones **parciales no totales** cuya función obtenida por minimización no acotada **sí** es **total**.
  - Consideremos la función $g(x, y) = \begin{cases} x - y & \text{si } y \leq x \\ \uparrow & \text{si } y > x \end{cases}$

    Entonces $g_\mu(x) = \mu y\,(g(x, y) = 0) = x$, para cada $z \in \mathbb{N}$.
* Toda función obtenida por **minimización no acotada** a partir de una función `GOTO`–computable es `GOTO`–computable.
  - Sea $f : \mathbb{N}^{k+1} \dashrightarrow \mathbb{N}$ $(k \geq 1)$ una función `GOTO`–computable. El siguiente programa `GOTO` con macros calcula f_μ:

$[A]$	$Z \longleftarrow f(X_1, ..., X_k, Y)$
	$IF\ Z = 0\ GOTO\ E$
	$Y \longleftarrow Y + 1$
	$GOTO\ A$

Minimización no acotada de predicados

Un **predicado** de aridad $k \geq 1$ sobre $\mathbb{N}$ es una función total de $\mathbb{N}^k$ en $\{0,1\}$. Es decir, los predicados son casos particulares de funciones totales.

- Para predicados, la minimización no acotada se definirá de otra forma.

Sea θ un predicado $(k+1)$–ario. La función parcial k–aria $\theta_\mu : \mathbb{N}^k \dashrightarrow \mathbb{N}$ obtenida por **minimización no acotada** a partir de θ se define así:

$$\theta_\mu(\vec{x}) = \begin{cases} min\{z : \theta(\vec{x}, z) = 1\} & \text{si existe el mínimo} \\ \uparrow & \text{e.c.o.c.} \end{cases}$$

Notaremos $\theta_\mu(\vec{x}) = \mu z\,(\theta(\vec{x}, z))$.

* La función obtenida por minimización no acotada a partir de un predicado **no tiene poqué ser** un predicado.

* Toda función obtenida por minimización no acotada a partir de un predicado GOTO–computable, es una función GOTO–computable.

Minimización acotada de predicados

Sea $\theta(\vec{x}, y)$ un predicado $(k+1)$–ario. La función total $(k+1)$-aria $\theta^*_\mu : \mathbb{N}^{k+1} \to \mathbb{N}$ obtenida por **minimización acotada** a partir de θ es la siguiente:

$$\theta^*_\mu(\vec{x}, z) = \begin{cases} \min\{y \leq z : \theta(\tilde{x}, y) = 1\} & \text{si existe tal mínimo} \\ z + 1 & \text{e.c.o.c.} \end{cases}$$

- Notaremos $\mu y \leq z\ (\theta(\tilde{x}, y))$.

Proposición. Si θ es un predicado GOTO-computable de aridad $k+1$, entonces la función total θ^*_μ de aridad $k+1$ también es GOTO-computable.

- El siguiente programa GOTO con macros calcula θ^*_μ:

[A]	*IF* $\theta(X_1, \ldots, X_k, Y) \neq 0$ *GOTO E*
	$Y \longleftarrow Y + 1$
	IF $Y \leq X_{k+1}$ GOTO A
	$Y \longleftarrow X_{k+1} + 1$

Minimización acotada de funciones totales

Sea $f(\vec{x}, z)$ una función **total** de $\mathbb{N}^{k+1} \longrightarrow \mathbb{N}$.

* Consideramos el siguiente predicado $(k+1)$-ario sobre $\mathbb{N}$ asociado a f:

$$\theta_f^0(\vec{x}, y) \equiv \left[f(\vec{x}, y) = 0 \right]$$

* La función **total** $f_\mu^* : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ obtenida por **minimización acotada** a partir de f se define así: $f_\mu^*(\vec{x}, z) = \mu y \leq z \, (f(\vec{x}, y) = 0)$

Obsérvese que la minimización acotada de una función total ha sido definida a partir de la minimización acotada de un predicado.

Proposición. Si $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ $(k \geq 1)$ es una función **total** GOTO-computable, entonces la función **total** f_μ^* de aridad $(k+1)$ también es GOTO-computable.

- Basta tener presente que si f es una función **total** GOTO-computable, entonces el predicado $\theta_f^0(\vec{x}, y) \equiv \left[f(\vec{x}, y) = 0 \right]$ es GOTO-computable.

□

Predicados GOTO-computables

Recuérdese que un **predicado** sobre $\mathbb{N}$ de aridad $k \geq 1$ es una **función total booleana** sobre $\mathbb{N}^k$; es decir, una aplicación de $\mathbb{N}^k$ en $\{0, 1\}$.

Si $\theta(\vec{x})$ es un predicado sobre $\mathbb{N}$ de aridad $k \geq 1$ y $\theta(\vec{x}) = 1$ (resp. $\theta(\vec{x}) = 0$) entonces diremos que el predicado $\theta(\vec{x})$ es **verdadero** (resp. **falso**) sobre $\vec{x}$.

Un predicado es GOTO-computable **sii** lo es la función que lo define.

Ejemplos.

1.- El predicado 1-ario "**ser distinto de cero**" sobre $\mathbb{N}$: $\boxed{\theta(x) = 1 \text{ sii } x \neq 0}$.

El siguiente programa GOTO calcula dicho predicado.

$$
\begin{array}{ll}
 & \mathit{IF}\ X \neq 0\ \mathit{GOTO}\ A \\
 & Z \longleftarrow Z + 1 \\
 & \mathit{IF}\ Z \neq 0\ \mathit{GOTO}\ E \\
[A] & Y \longleftarrow Y + 1
\end{array}
$$

Predicados GOTO-computables

La **función signo**, **sg**, es el predicado 1-ario "**ser distinto de cero**" sobre $\mathbb{N}$. Es decir: sg es la aplicación de $\mathbb{N}$ en $\mathbb{N}$ definida como sigue:

$$\text{sg}(0) = 0 \text{ y } \text{sg}(x) = 1, \text{ si } x \neq 0$$

2.- El predicado 1-ario "**ser igual a cero**" sobre $\mathbb{N}$: $\boxed{\theta(x) = 1 \textbf{ sii } x = 0}$.

El siguiente programa GOTO calcula dicho predicado.

$$\boxed{\begin{array}{l} \mathit{IF}\ X \neq 0\ \mathit{GOTO}\ E \\ Y \longleftarrow Y + 1 \end{array}}$$

La **función signo inverso**, $\overline{\text{sg}}$, es el predicado 1-ario "**ser igual a cero**" sobre $\mathbb{N}$. Es decir: $\overline{\text{sg}}$ es la aplicación de $\mathbb{N}$ en $\mathbb{N}$ definida como sigue:

$$\overline{\text{sg}}(0) = 1 \text{ y } \overline{\text{sg}}(x) = 0, \text{ si } x \neq 0$$

Operaciones con predicados

Proposición. Sean θ, θ' predicados sobre $\mathbb{N}$, de aridad $k \geq 1$. Si θ y θ' son GOTO–computables, entonces los predicados $\neg\theta$, $\theta \wedge \theta'$, $\theta \vee \theta'$, $\theta \Rightarrow \theta'$, $\theta \Leftrightarrow \theta'$ son GOTO-computables.

Demostración: Basta tener presente que

- $\neg\theta(\vec{x}) = \overline{sg}(\theta(\vec{x}))$.
- $(\theta \wedge \theta')(\vec{x}) = \theta(\vec{x}) \cdot \theta'(\vec{x})$.
- $(\theta \vee \theta')(\vec{x}) = sg\big(\theta(\vec{x}) + \theta'(\vec{x})\big)$.
- $\theta \Rightarrow \theta' \equiv \neg\theta \vee \theta'$.
- $\theta \Leftrightarrow \theta' \equiv (\theta \Rightarrow \theta') \wedge (\theta' \Rightarrow \theta)$.

■

Generalización de los predicados $=$, $\leq$ y $<$

Proposición: Sea f una función total GOTO–computable de $\mathbb{N}^k$ en $\mathbb{N}$. Los siguientes predicados $(k+1)$-arios son GOTO–computables:

$$\theta(\vec{x}, y) \equiv f(\vec{x}) = y; \quad \theta'(\vec{x}, y) \equiv f(\vec{x}) \leq y; \quad \theta''(\vec{x}, y) \equiv f(\vec{x}) < y$$

Demostración: Se verifica lo siguiente:

* $\theta(x_1, \ldots, x_k, y) \equiv \big[f(x_1, \ldots, x_k) = y\big] \equiv \big(P_=(f(x_1, \ldots, x_k), y)\big)$

  Ahora bien:

  $\Pi_1^{(k+1)}(x_1, \ldots, x_k, y) = x_1, \ldots, \Pi_k^{(k+1)}(x_1, \ldots, x_k, y) = x_k, \Pi_{k+1}^{(k+1)}(x_1, \ldots, x_k, y) = y$

  Por tanto: $\theta = \mathcal{C}\big(P_=; \mathcal{C}(f; \Pi_1^{(k+1)}, \ldots, \Pi_k^{(k+1)}), \Pi_{k+1}^{(k+1)}\big)$.

  Puesto que $P_=, f, \Pi_1^{(k+1)}, \ldots, \Pi_k^{(k+1)}, \Pi_{k+1}^{(k+1)}$ son GOTO-computables, concluimos que el predicado θ también es GOTO-computable.

* Análogamente se prueba que: $\theta' = \mathcal{C}\big(P_\leq; \mathcal{C}(f; \Pi_1^{(k+1)}, \ldots, \Pi_k^{(k+1)}), \Pi_{k+1}^{(k+1)}\big)$ y que $\theta'' = \mathcal{C}\big(P_<; \mathcal{C}(f; \Pi_1^{(k+1)}, \ldots, \Pi_k^{(k+1)}), \Pi_{k+1}^{(k+1)}\big)$.

  Por tanto, los predicados θ' y θ'' son GOTO-computables.

■

Conjuntos GOTO–computables.

Recuérdese que la función característica, C_A, de un **conjunto** $A \subseteq \mathbb{N}^k$ se define como sigue: $C_A(\vec{x}) = 1$ si $\vec{x} \in A$ y $C_A(\vec{x}) = 0$ si $\vec{x} \in \mathbb{N}^k - A$.

Por tanto, la función característica, C_A, de un **conjunto** $A \subseteq \mathbb{N}^k$ es un **predicado** de aridad k sobre $\mathbb{N}$ (el **predicado** de **pertenencia** al conjunto A).

Un **conjunto** $A \subseteq \mathbb{N}^k$ es **GOTO–computable sii** su función característica, C_A, es GOTO–computable.

Ejemplos:

* Los conjuntos $\emptyset$ y $\mathbb{N}^k$, para cada $k \geq 1$, son GOTO–computables.
* Si $A, B \subseteq \mathbb{N}^k$ son conjuntos GOTO-computables, entonces los conjuntos $\mathbb{N}^k - A$; $A \cap B$; $A \cup B$ también son GOTO-computables.

  En efecto: Basta tener presente que:

  - $C_{\mathbb{N}^k - A} = \neg C_A$.
  - $C_{A \cap B} = C_A \wedge C_B$.
  - $C_{A \cup B} = C_A \vee C_B$.

La gráfica de una función

Sea f una función de aridad $k \geq 1$ sobre $\mathbb{N}$. La **gráfica** de f es el conjunto $G(f)$ definido como sigue: $G(f) = \{(\vec{x}, y) \in \mathbb{N}^{k+1} : \ f(\vec{x}) = y\}$.

Proposición: Si f es una **función total** GOTO–computable de $\mathbb{N}^k$ en $\mathbb{N}$, entonces la **gráfica** de f es un **conjunto GOTO-computable**.

Demostración: Basta tener presente que la función característica de $G(f)$ es el predicado $(k+1)$-ario sobre $\mathbb{N}$ siguiente: $\theta(\vec{x}, y) \equiv \big(f(\vec{x}) = y\big)$.

■

Definición por casos para funciones totales

Teorema. Sean $q \geq 2$ y $f_1, \ldots, f_q$ funciones **totales** GOTO-computables de $\mathbb{N}^k$ en $\mathbb{N}$, con $k \geq 1$. Sea $\{A_1, \ldots, A_q\}$ una partición de $\mathbb{N}^k$ formada por conjuntos GOTO-computables. Sea $g : \mathbb{N}^k \longrightarrow \mathbb{N}$ la función total definida por casos, como sigue:

$$g(\vec{x}) = \begin{cases} f_1(\vec{x}) & si & \vec{x} \in A_1 \\ \vdots & \vdots & \vdots \\ f_q(\vec{x}) & si & \vec{x} \in A_q \end{cases}$$

Entonces la función g es GOTO-computable.

Demostración: Basta tener presente que la función g (*definida por casos*) puede ser descrita como sigue: $g = f_1 \cdot C_{A_1} + \ldots + f_q \cdot C_{A_q}$.

Es decir, g es una suma de productos de funciones GOTO-computables.

En consecuencia, g también será una función GOTO-computable.

■

Nota: El anterior enunciado se puede expresar con predicados GOTO-computables que sean exhaustivos y excluyentes $C_{A_1}, \ldots, C_{A_q}$.

GOTO-computabilidad de otras funciones relevantes en $\mathbb{N}$

1. Las funciones constantes en $\mathbb{N}$.

La función constante igual a b $\in \mathbb{N}$ de aridad $k \geq 1$ en $\mathbb{N}$, es la aplicación $C_b^{(k)}$ de $\mathbb{N}^k$ en $\mathbb{N}$ definida como sigue: $C_b^{(k)}(x_1, \ldots, x_k) = b$, para cada $(x_1, \ldots, x_k) \in \mathbb{N}^k$.

Consideremos el programa GOTO que consta, exactamente, de b instrucciones $Y \longleftarrow Y + 1$.

Entonces, la función de aridad k calculada por dicho programa es, precisamente, la aplicación $C_b^{(k)}$.

Por tanto, la función constante $C_b^{(k)}$ es GOTO-computable, para cada $k \geq 1$ y cada $b \in \mathbb{N}$.

2. La función predecesor en $\mathbb{N}$.

La función predecesor en $\mathbb{N}$ es la aplicación *Pred* de $\mathbb{N}$ en $\mathbb{N}$ definida como sigue:

(1) $Pred(0) = 0$.

(2) $Pred(a+1) = a$, para cada $a \in \mathbb{N}$.

Veamos que la función *Pred* se puede describir por recursión primitiva, a partir de una constante $c \in \mathbb{N}$ y una cierta función h. En efecto: si $Pred = \mathcal{R}(c, h)$, entonces:

(3) $Pred(0) = c$.

(4) $Pred(a+1) = h(a, Pred(a))$, para cada $a \in \mathbb{N}$.

Comparando las relaciones (3) y (4) con (1) y (2) se deduce

* $c = 0$.

* $h(a, Pred(a)) = a$, para cada $a \in \mathbb{N}$.

Considerando la constante $c = 0$ y la función $h(a, b) = a$ se deduce que $Pred = \mathcal{R}(c, h)$.

Teniendo presente que $h = \Pi_1^{(2)}$ es una función GOTO-computables, se concluye que la función *Pred* también es GOTO-computable.

3. La función diferencia reducida en $\mathbb{N}$.

La función diferencia reducida en $\mathbb{N}$ es la aplicación $\overset{\bullet}{-}$ de $\mathbb{N}^2$ en $\mathbb{N}$ definida así:

$\overset{\bullet}{-}(a,b) = 0$ **si** $a \leq b$ y $\overset{\bullet}{-}(a,b) = a - b$ **si** $a > b$, para cada $a, b \in \mathbb{N}$.

Notaremos $\overset{\bullet}{-}(x,y) = x \overset{\bullet}{-} y$

La función diferencia reducida en $\mathbb{N}$ verifica las condiciones siguientes:

(1) $\overset{\bullet}{-}(a,0) = a$, para cada $a \in \mathbb{N}$.

(2) $\overset{\bullet}{-}(a,b+1) = \mathrm{Pred}(\overset{\bullet}{-}(a,b))$, para cada $a, b \in \mathbb{N}$.

La expresión (2) se prueba distinguiendo casos, según sea $a \leq b+1$ (entonces ambos miembros valen 0), o $a > b+1$ (entonces ambos miembros valen $a-(b+1)$).

Veamos que la función $\overset{\bullet}{-}$ se puede describir por recursión primitiva, a partir de ciertas funciones g y h. En efecto: si $\overset{\bullet}{-} = \mathcal{R}(g,h)$, entonces se verificaría:

(3) $\overset{\bullet}{-}(a,0) = g(a)$, para cada $a \in \mathbb{N}$.

(4) $\overset{\bullet}{-}(a,b+1) = h(a,b,\overset{\bullet}{-}(a,b))$, para cada $a, b \in \mathbb{N}$.

Comparando las relaciones (3) y (4) con (1) y (2) se deduce

* $g(a) = a$, para cada $a \in \mathbb{N}$.
* $h(a,b,\overset{\bullet}{-}(a,b)) = \mathrm{Pred}(\overset{\bullet}{-}(a,b))$, para cada $a, b \in \mathbb{N}$.

Considerando $g = Id_{\mathbb{N}}$ y $h(a,b,c) = Pred(c)$ se deduce que $\overset{\bullet}{-} = \mathcal{R}(g,h)$.

Teniendo presente que g y h son funciones GOTO-computables, se concluye que la función diferencia reducida $\overset{\bullet}{-}$ también es GOTO-computable.

4. La función mínimo en $\mathbb{N}$.

La función mínimo en $\mathbb{N}$ es la aplicación *min* de $\mathbb{N}^2$ en $\mathbb{N}$ definida como sigue: para cada $a, b \in \mathbb{N}$,

$$min(a,b) = \begin{cases} a & si\ a \leq b \\ b & si\ a > b \end{cases}$$

La **función total** *min* se puede considerar **definida por casos** a partir de:

- Las funciones $f_1(a,b) = a$, para cada $a \in \mathbb{N}$ y $f_2(a,b) = b$, para cada $b \in \mathbb{N}$.
- Los predicados $\theta_1(a,b) \equiv a \leq b$, para cada $a, b \in \mathbb{N}$ y $\theta_2(a,b) \equiv b < a$, para cada $a, b \in \mathbb{N}$.

Teniendo presente que las funciones totales f_1, f_2 y los predicados θ_1, θ_2 son GOTO-computables, se deduce que la función total *min*, definida por casos, a partir de esas funciones y esos predicados, será también GOTO-computable.

5. La función máximo en $\mathbb{N}$.

La función máximo en $\mathbb{N}$ es la aplicación *max* de $\mathbb{N}^2$ en $\mathbb{N}$ definida como sigue: para cada $a, b \in \mathbb{N}$,

$$max(a,b) = \begin{cases} b & si\ a \leq b \\ a & si\ a > b \end{cases}$$

La **función total** *max* se puede considerar **definida por casos** a partir de:

- Las funciones $f_1(a,b) = b$, para cada $b \in \mathbb{N}$ y $f_2(a,b) = a$, para cada $a \in \mathbb{N}$.
- Los predicados $\theta_1(a,b) \equiv a \leq b$, para cada $a, b \in \mathbb{N}$ y $\theta_2(a,b) \equiv b < a$, para cada $a, b \in \mathbb{N}$.

Teniendo presente que las funciones totales f_1, f_2 y los predicados θ_1, θ_2 son GOTO-computables, se deduce que la función total *max*, definida por casos, a partir de esas funciones y esos predicados, será también GOTO-computable.

6. La función distancia (o valor absoluto) en $\mathbb{N}$.

La función distancia (o valor absoluto) en $\mathbb{N}$ es la aplicación d de $\mathbb{N}^2$ en $\mathbb{N}$ definida como sigue: para cada $a, b \in \mathbb{N}$,

$$d(a,b) = \begin{cases} b - a & si\ a \leq b \\ a - b & si\ a > b \end{cases}$$

La **función total** d se puede considerar **definida por casos** a partir de:

- Las funciones $f_1(a,b) = b \stackrel{\bullet}{-} a$ y $f_2(a,b) = a \stackrel{\bullet}{-} b$, para cada $a, b \in \mathbb{N}$.
- Los predicados $\theta_1(a,b) \equiv a \leq b$, para cada $a, b \in \mathbb{N}$ y $\theta_2(a,b) \equiv b < a$, para cada $a, b \in \mathbb{N}$.

Teniendo presente que las funciones totales f_1, f_2 y los predicados θ_1, θ_2 son GOTO-computables, se deduce que la función total d, definida por casos, a partir de esas funciones y esos predicados, será también GOTO-computable.

Nota: También se podría establecer la GOTO-computabilidad de la función distancia d observando que se verifica la relación: $d(a,b) = (a \stackrel{\bullet}{-} b) + (b \stackrel{\bullet}{-} a)$

7. La función exponencial en $\mathbb{N}$.

La función exponencial en $\mathbb{N}$ es la aplicación *exp* de $\mathbb{N}^2$ en $\mathbb{N}$ definida como sigue: para cada $a, b \in \mathbb{N}$,

$$exp(a,b) = \begin{cases} 1 & si\ a = 0 \wedge b = 0 \\ a^b & si\ a \neq 0 \vee b \neq 0 \end{cases}$$

De la definición anterior resulta que se verifican las siguientes propiedades:

(1) $exp(a, 0) = 1$.

(2) $exp(a, b+1) = a \cdot exp(a, b)$, para cada $a, b \in \mathbb{N}$.

Veamos que la función *exp* se puede describir por recursión primitiva, a partir ciertas funciones g y h. En efecto: si $exp = \mathcal{R}(g, h)$, entonces:

(3) $exp(a, 0) = g(a)$, para cada $a \in \mathbb{N}$.

(4) $exp(a, b+1) = h(a, b, exp(a, b))$, para cada $a, b \in \mathbb{N}$.

Comparando las relaciones (3) y (4) con (1) y (2) se deduce

* $g(a) = 1$, para cada $a \in \mathbb{N}$.

* $h(a, b, exp(a)) = a \cdot exp(a, b)$, para cada $a, b \in \mathbb{N}$.

Considerando la función constante $g = C_1^{(1)}$ y la función $h(a, b, c) = a \cdot c$ se deduce que $exp = \mathcal{R}(g, h)$.

Teniendo presente que las funciones g y h son GOTO-computables, se concluye que la función *exp* también es GOTO-computable.

8. La función cociente en $\mathbb{N}$.

La función cociente en $\mathbb{N}$ es la aplicación qt de $\mathbb{N}^2$ en $\mathbb{N}$ definida como sigue:

$$\mathrm{qt}(x, y) = \mu t \leq x\ (x < (t + 1) \cdot y), \text{ para cada } x, y \in \mathbb{N}$$

Es decir, la función cociente está descrita como una **minimización acotada** en la que intervienen las siguientes funciones:

* El predicado "estrictamente menor" sobre $\mathbb{N}$.
* La función producto.
* La función suma.

Por tanto, la **función cociente** es GOTO-computable.

9. La función resto en $\mathbb{N}$.

La función resto en $\mathbb{N}$ es la aplicación rm de $\mathbb{N}^2$ en $\mathbb{N}$ definida como sigue:

$$\mathrm{rm}(x, y) = x \stackrel{\bullet}{-} (y \cdot \mathrm{qt}(x, y)), \text{ para cada } x, y \in \mathbb{N}$$

Resulta que la función resto está descrita como una **composición** en la que intervienen las siguientes funciones:

* La función diferencia reducida $\stackrel{\bullet}{-}$.
* La función producto.
* La función cociente, qt.

Por tanto, la **función resto** es GOTO-computable.

Cuantificación acotada de predicados

Proposición. Sea θ un predicado de aridad $(k+1)$ y GOTO–computable. Los siguientes predicados son GOTO-computables:

$$\theta_1(\vec{x}, y) \equiv \exists z \leq y\, \theta(\vec{x}, z); \qquad \theta_2(\vec{x}, y) \equiv \forall z \leq y\, \theta(\vec{x}, z)$$

- Basta tener presente que

$$\theta_1(\vec{x}, y) = sg\big((y+1) \dot{-} \mu t \leq y(\theta(\vec{x}, t))\big)$$

$$\theta_2(\vec{x}, y) = \overline{sg}\big((y+1) \dot{-} \mu t \leq y(\neg\theta(\vec{x}, t))\big)$$

Ejemplos interesantes:

1. El **predicado de divisibilidad**, $x \,|\, y$, es GOTO–computable:

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

2. El **predicado ser primo** es GOTO–computable:

$$primo(x) \equiv (x > 1) \wedge \forall t \leq x((t|x) \rightarrow (t = 1 \vee t = x))$$

La sucesión de números primos

Consideremos la siguiente función $p : \mathbb{N} \to \mathbb{N}$ definida así:

$$p(n) = \begin{cases} 0 & \text{si} \quad n = 0 \\ \text{el } n\text{–ésimo primo} & \text{si} \quad n \geq 1 \end{cases}$$

Notaremos $p(n) = p_n$, para cada $n \geq 1$.

Proposición. La función $p(n)$ es GOTO–computable.

Para demostrar este resultado se suele usar el siguiente lema de Euclides:

- **Lema**. Para cada $n \in \mathbb{N}$ se tiene que $p(n+1) \leq 1 + p(n)!$

A partir de aquí, basta describir $p(n)$ por **recursión primitiva** como sigue:

$$\begin{cases} p(0) & = & 0 \\ p(n+1) & = & \mu y \leq \left(1 + p(n)!\right) \left[primo(y) \wedge y > p(n)\right] \end{cases}$$

Codificaciones

Definición: Un conjunto A es **codificable** a partir de un conjunto B si existen funciones inyectivas y GOTO–computables,

$$f : A \to B \quad \text{y} \quad g : B \dashrightarrow A$$

tales que $g(f(x)) = x$, para cada $x \in A$.

- f es la función **codificadora**.
- g es la función **decodificadora**.

Teorema: Si $\Sigma = \{s_1, \ldots, s_n\}$ es un alfabeto entonces Σ^* es codificable a partir de $\{0,1\}^*$ mediante una función GOTO–computable, f, para la que existe $c \in \mathbb{N}$ verificando $|f(\gamma)| = c \cdot |\gamma|$, para cada $\gamma \in \Sigma^*$.

Codificación de sucesiones

Definición (**Codificación de sucesiones finitas**). Una **codificación de $\mathbb{N}^p$** a partir de $\mathbb{N}$, es una función total $f : \mathbb{N}^p \longrightarrow \mathbb{N}$ que verifica:

* f es GOTO–computable e inyectiva.
* Para cada $i, 1 \leq i \leq p$, la función total, $g_i : \mathbb{N} \longrightarrow \mathbb{N}$ definida por

$$g_i(x) = \begin{cases} a_i & si\ x \in rang(f) \wedge x = f(a_1, ..., a_p) \\ 0 & e.c.o.c. \end{cases}$$

es GOTO–computable.

Definición (**Codificación de sucesiones de longitud arbitraria**).
Si $\mathbb{N}' = \{\varepsilon\} \cup \mathbb{N} \cup \mathbb{N}^2 \cup ... \cup \mathbb{N}^k \cup ...$ (ε es la sucesión vacía), diremos que una función total $f : \mathbb{N}' \longrightarrow \mathbb{N}$, **codifica $\mathbb{N}'$ a partir de $\mathbb{N}$**, si $f(\epsilon) = 0$ y, además, para cada $p \geq 1$ la restricción de f a $\mathbb{N}^p$ es una codificación de $\mathbb{N}^p$ a partir de $\mathbb{N}$.

La función par (codificación de pares ordenados)

La **función par** es la función total binaria $\langle \cdot\,, \cdot \rangle : \mathbb{N}^2 \longrightarrow \mathbb{N}$ definida por

$$\langle x, y \rangle = 2^x \cdot (2y + 1) \stackrel{\bullet}{-} 1$$

Propiedades:

- La **función par** es GOTO-computable y biyectiva.
- Para todo $x, y \in \mathbb{N}$ se verifica que $x \leq \langle x, y \rangle$, $y \leq \langle x, y \rangle$.
- Existen funciones totales $l, r : \mathbb{N} \longrightarrow \mathbb{N}$ tales que si $z = \langle x, y \rangle$, entonces

$$l(z) = x; \quad r(z) = y; \qquad z = \langle l(z), r(z) \rangle$$

  l y r son **funciones decodificadoras** de la función par.
- Las funciones l y r son GOTO-computables y totales.

Nota: (**codificación de ternas ordenadas**)
A partir de la función par se puede codificar $\mathbb{N}^3$ a partir de $\mathbb{N}$ como sigue:

$$(x, y, z) \longrightarrow \langle x, \langle y, z \rangle \rangle$$

En general, se puede codificar $\mathbb{N}^k$ a partir de $\mathbb{N}$, para cada $k \geq 3$, de "manera natural".

$$(x_1, x_2, x_3, x_4) \longrightarrow \langle \langle x_1, x_2 \rangle, \langle x_3, x_4 \rangle \rangle$$

La codificación de Gödel (codificación de sucesiones finitas)

Sea $\{p_n : n \geq 1\}$ es la sucesión de números primos.

La **codificación de Gödel** es la función total de $\mathbb{N}'$ en $\mathbb{N}$ definida como sigue:

$$\begin{cases} [\varepsilon] & = 1 \\ [a_1, ..., a_n] & = p_1^{a_1} \bullet \ldots \bullet p_n^{a_n} \end{cases}$$

Diremos que $[a_1, ..., a_n]$ es el número de Gödel de la sucesión finita $a_1, ..., a_n$.

Propiedades:

- La función $[\cdots]$ codifica $\mathbb{N}'$ a partir de $\mathbb{N}$.
- Para cada i, $1 \leq i \leq n$, se verifica que $a_i \leq [a_1, ..., a_n]$.
- La función $[\cdots]$ **no** es inyectiva.
- rang($[\cdots]$)$= \mathbb{N} - \{0\}$.
- Cada número natural $t \geq 1$ codifica **infinitas** sucesiones, ya que

$$t = [a_1, ..., a_n] = [a_1, ..., a_n, 0] = [a_1, ..., a_n, 0, 0] = \ldots$$

- En general se tiene que $[a_1, ..., a_n] \neq [0, a_1, ..., a_n]$.

Función proyección versus función componente

Para cada n, k tales que $1 \leq k \leq n$, la **función proyección** k-ésima de aridad n, en el conjunto de los números naturales, se representa por $\Pi_k^{(n)}$.

La función proyección $\Pi_k^{(n)}$ asocia a cada tupla $(a_1, \ldots, a_n)$ de números naturales, su componente k-ésima a_k (siendo $1 \leq k \leq n$).

Ahora bien, toda n-tupla $(a_1, \ldots, a_n)$, $n \geq 1$, de números naturales tiene asociada un número de Gödel $[a_1, \ldots, a_n]$.

Se trata de definir una función (la **función componente**) que asigne a cada número natural que sea el número de Gödel de una n-tupla $(a_1, \ldots, a_n)$, la componente k-ésima de la misma $(1 \leq k \leq n)$.

Es decir, vamos a definir qué son las **componentes** de un **número natural**.

La función componente

Para cada número natural $x \geq 2$ existen unos únicos números naturales $a_1, \ldots, a_n$ verificando que $a_n > 0$ y $x = p_1^{a_1} \cdot \cdots \cdot p_n^{a_n}$ (es decir, $x = [a_1, ..., a_n]$).

Vamos a definir el concepto de **componentes** de un número de Gödel. De tal manera que si $x \geq 2$, entonces la componente k–ésima de $x = [a_1, ..., a_n]$ sea, precisamente, a_k (para $1 \leq k \leq n$).

En el caso $x \geq 2$ **¿qué propiedad caracteriza al número a_k en relación con $x = [a_1, ..., a_n]$?**

Teniendo presente que $x = [a_1, ..., a_n] = p_1^{a_1} \cdot \cdots \cdot p_k^{a_k} \cdot \cdots \cdot p_n^{a_n}$ se deduce que:

* El número $p_k^{a_k}$ divide al número x.
* El número $p_k^{1+a_k}$ **no divide** a x.

Por tanto, a_k es el menor número t para el que p_k^{t+1} **no divide** a x. Además, ese número t debe ser menor o igual que x.

La función componente

La **función componente** se define como sigue:

$$(x)_k = \begin{cases} 0 & \text{si } x = 0, 1 \vee k = 0 \\ \mu t \leq x \, (\neg (p_k^{t+1} | x)) & \text{si } x \geq 2 \wedge k \geq 1 \end{cases}$$

De la definición anterior resulta que:

* Sea $x = p_1^{a_1} \bullet \ldots \bullet p_n^{a_n} = [a_1, ..., a_n]$ tal que $a_n > 0$ (es decir, $x \geq 2$). Entonces

$$(x)_k = \begin{cases} a_k & \text{si } \; 1 \leq k \leq n \\ 0 & \text{si } \; i = 0 \vee k > n \end{cases}$$

  Diremos que $(x)_k$ es la componente k-ésima del número natural x.

* La función componente es GOTO–computable y total.

La función longitud

Para cada número natural $x \geq 2$ existen unos únicos números naturales $a_1, \ldots, a_n$ verificando que $a_n > 0$ y $x = p_1^{a_1} \cdot \cdots \cdot p_n^{a_n}$ (es decir, $x = [a_1, ..., a_n]$).

Vamos a definir el concepto de **longitud** del número de Gödel $[a_1, ..., a_n]$. De tal manera que si $x \geq 2$ y $x = [a_1, ..., a_n]$, entonces la longitud de x sea n.

En el caso que estamos considerando ($x \geq 2$) ¿qué propiedad caracteriza al número n en relación con $x = [a_1, ..., a_n]$?

Teniendo presente que $x = [a_1, ..., a_n] = p_1^{a_1} \cdot \cdots \cdot p_n^{a_n}$ se deduce que:

* $(x)_n \neq 0$.
* Para cada k $(k \leq x \wedge k > n)$ se tiene que $(x)_k = 0$.

Por tanto, n es el menor número t que verifica:

* $(x)_t \neq 0$.
* $\forall k\ (k \leq x \wedge k > t \rightarrow (x)_k = 0)$.

La función longitud

La **función longitud** se define como sigue:

$$Long(x) = \begin{cases} 0 & \text{si } x = 0, 1 \\ \mu t \leq x\ ((x)_t \neq 0 \wedge \forall k \leq x\ (k > t \rightarrow (x)_k = 0)) & \text{si } x \geq 2 \end{cases}$$

De la definición anterior resulta que:

* Sea $x = p_1^{a_1} \bullet \ldots \bullet p_n^{a_n} = [a_1, ..., a_n]$ tal que $a_n > 0$ (es decir, $x \geq 2$). Entonces $Long(x) = n$.

* La función longitud es GOTO–computable y total.