AUTÓMATAS Y LENGUAJES FORMALES
OTROS MODELOS DE COMPUTABILIDAD

Francisco Hernández Quiroz

Departamento de Matemáticas
Facultad de Ciencias, UNAM
E–mail: fhq@ciencias.unam.mx
Página Web: www.matematicas.unam.mx/fhq

Posgrado en Ciencia e Ingeniería de la Computación

Definiciones de computabilidad

- Turing propuso sus máquinas como una definición formal de la noción intuitiva de "calculabilidad efectiva".
- Él mismo aplicó su definición a dos problemas: la determinación de qué es un número computable y la demostración de irresolubilidad del *Entscheidungsproblem*.
- Otros matemáticos contemporáneos propusieron otras formas de definir la calculabilidad efectiva: el cálculo lambda, las gramáticas formales, etc.
- Es posible demostrar que, una vez que se acepta una noción de equivalencia adecuada, todos estas definiciones son equivalentes, en el sentido de que nos permiten calcular la misma clase de funciones.

Tesis de Church–Turing

- La equivalencia entre modelos de computación llevó a plantear la siguiente afirmación:

  Toda función es calculable efectivamente si y sólo si es calculable por una máquina de Turing.

- La afirmación se conoce como la tesis de Church–Turing.
- Como dice que una noción intuitiva (calculabilidad efectiva) es capturada por una noción formal, no se puede "demostrar".
- Pero es una hipótesis de trabajo conveniente y los resultados de computabilidad vistos antes pueden formularse de manera más general si se acepta la validez de la tesis.
- Es posible expresarla no como una afirmación dogmática, sino como axiomas de los que se derivan los mismos resultados.

Computabilidad–Turing

- Para poder comparar los modelos que estudiaremos adoptaremos una convención similar a la que vimos antes.
- Sea $M \in TM$, con Σ y Γ como alfabetos de entrada y de la cinta, respectivamente. Entonces, podemos definir una función $f_M : \Sigma^* \to \Gamma^*$ como

$$\{(\alpha, \beta) \mid (s_M, \alpha, 0) \Rightarrow_M^* (t, \beta, n)\}.$$

- Obsérvese que f_M no tiene por qué ser total.
- Por otro lado, diremos que la función $\sigma : \Sigma^* \to \Delta^*$ es:
  (a) *Turing–computable* sii existe una $M \in TM$ tal que $\sigma = f_M$;
  (b) *computable* sii existe una $M \in TM$ *total* tal que $\sigma = f_M$.
- La convención se puede adaptar para referirse a números naturales y no a cadenas. Por ejemplo, la cadena α se puede referir al número natural que corresponde a su lugar en el orden lexicográfico.

Funciones recursivas μ

Las funciones recursivas–μ son otro modelo de computabilidad. La terminología adoptada presupone la *tesis de Church–Turing*.
Estas funciones son el mínimo conjunto cerrado bajo las operaciones de composición, recursión primitiva, minimización acotada y que contiene a las siguientes funciones básicas (donde $\bar{x} = x_1, \ldots, x_n$):

- *Cero*. La constante $c : \mathbb{N}^0 \to \mathbb{N}$, con $c = 0$ es computable.
- *Sucesor*. La función $s : \mathbb{N} \to \mathbb{N}$, con $s(x) = x + 1$, es computable.
- *Proyecciones*. Las funciones $\pi^n_k : \mathbb{N}^n \to \mathbb{N}$ y $\pi^n_k(x_1, \ldots, x_n) = x_k$, con $1 \leq k \leq n$, son computables.
- De ahora en adelante usaremos la siguiente abreviatura

$$\bar{x} \equiv_{def} x_1, \ldots, x_n.$$

Funciones nuevas I

Las operaciones para crear nuevas funciones son:

- *Composición*. Si las funciones

$$f : \mathbb{N}^k \to \mathbb{N}$$

y

$$g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N}$$

son computables entonces también lo es la función

$$f \circ (g_1, \ldots, g_k) : \mathbb{N}^n \to \mathbb{N},$$

definida por

$$f(g_1(\bar{x}), \ldots, g_k(\bar{x})).$$

Funciones nuevas II

- *Recursión primitiva*. Si

$$h_1, \ldots, h_k : \mathbb{N}^n \to \mathbb{N}$$

y

$$g_1, \ldots, g_k : \mathbb{N}^{n+k+1} \to \mathbb{N}$$

son computables, entonces también lo son las funciones

$$f_1, \ldots, f_k : \mathbb{N}^{n+1} \to \mathbb{N},$$

definidas a continuación

$$\begin{aligned} f_i(0, \bar{x}) &= h_i(\bar{x}) \\ f_i(s(n), \bar{x}) &= g_i(n, \bar{x}, f_1(n, \bar{x}), \ldots, f_k(n, \bar{x})). \end{aligned}$$

Funciones nuevas III

- *Minimización no acotada*. Sea

$$g : \mathbb{N}^{n+1} \to \mathbb{N}$$

una función computable. Entonces, también es computable la función

$$f : \mathbb{N}^n \to \mathbb{N},$$

definida por

$$\begin{aligned} f(\bar{x}) &= \text{el mínimo valor } m \text{ tal que:} \\ &\quad g(m, \bar{x}) = 0 \text{ y } \forall n \leq m.\, g(n, \bar{x}) \text{ está definida} \\ &= \text{indefinida, en caso de que no exista tal } m \end{aligned}$$

El valor de $f(\bar{x})$ se denota con la expresión

$$\mu y.\, g(y, \bar{x}) = 0.$$

Ejemplos

La función + suele definirse por medio de las siguientes ecuaciones

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

En la notación de las funciones μ tendríamos:

$$\begin{aligned} h &= \pi_1^1 \\ g &= s \circ \pi_3^3 \\ f(0, x) &= \pi_1^1(x) \\ f(s(y), x) &= s(\pi_3^3(y, x, f(y, x))) \end{aligned}$$

Cálculo lambda I

- Las funciones recursivas también son expresables en el cálculo λ.
- El conjunto de términos del cálculo λ se denotará por Λ.
- La sintaxis del cálculo lambda es al siguiente

$$\Lambda \to V \mid (\lambda x.\Lambda) \mid (\Lambda\Lambda)$$

  donde V denota una variable: $w, x, y, z, w_1, \ldots$
- La segunda cláusula en la definición anterior se conoce como *abstracción* (se usa para definir funciones) y la tercera como *aplicación* (se usa para aplicarlas).

Cálculo lambda II

- La siguiente abreviatura nos ayudará a definir funciones con más de un parámetro:

$$\lambda x_1 x_2 \ldots x_n.\, M \equiv_{def} (\lambda x_1.(\lambda x_2.(\ldots(\lambda x_n.\, M)\ldots)$$

- La regla básica es la reducción β

$$((\lambda x.\, M)N) \to_\beta M_{[x/N]}$$

- Por ejemplo, la siguiente es una función que duplica su argumento aplicada al término z y su resultado al aplicar la reducción β

$$((\lambda x.\, xx)z) \to_\beta (xx)_{[x:=z]} = zz$$

Funciones recursivas en el cálculo lambda I

Sean M y $N \in \Lambda$. Entonces:

$$\begin{aligned} V &\equiv_{def} \lambda xy.\, x \\ F &\equiv_{def} \lambda xy.\, y \\ [M, N] &\equiv_{def} \lambda z.\, zMN \\ \bar{0} &\equiv_{def} \lambda x.\, x \\ \overline{n+1} &\equiv_{def} [F, \bar{n}] \\ S &\equiv_{def} \lambda x.\, [F, x] \\ P &\equiv_{def} \lambda x.\, xF \\ C &\equiv_{def} \lambda x.\, \bar{0} \\ Cero &\equiv_{def} \lambda x.\, xV \\ \pi_i^n &\equiv_{def} \lambda x_1, \ldots, x_n.\, x_i \\ Y &\equiv_{def} \lambda f.(\lambda x.\, f(xx))(\lambda x.\, f(xx)) \end{aligned}$$

Funciones recursivas en el cálculo lambda II

- Obviamente, $\bar{0}$ y $\overline{n+1}$ definen los números naturales.
- S y P son las funciones predecesor y sucesor.
- V y F son los valores de verdad.
- *Cero* es una función que da V con $\bar{0}$ y F en caso contrario.
- π_i^n es, por supuesto, la proyección i–ésima con n argumentos.
- El término Y será útil más adelante para definir la recursión.

Funciones recursivas en el cálculo lambda III

Sean $f : \mathbb{N}^k \to \mathbb{N}$ y $g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N}$ funciones definidas por los términos F y $G_1, \ldots G_k \in \Lambda$, respectivamente. La composición $f \circ (g_1, \ldots, g_k) : \mathbb{N}^n \to \mathbb{N}$ es el término

$$\lambda x_1 \ldots x_n.\, F(G_1 x_1 \ldots x_n) \ldots (G_k x_1 \ldots x_n).$$

Es posible definir las funciones recursivas primitivas en el cálculo λ. Por simplicidad, nos limitaremos a la definición de una sola función recursiva. Supongamos que $h : \mathbb{N}^n \to \mathbb{N}$ y $g : \mathbb{N}^{n+2} \to \mathbb{N}$ son funciones computables definidas por los términos H, $G \in \Lambda$.

La función recursiva primitiva $f : \mathbb{N}^{n+1} \to \mathbb{N}$ definida a partir de h y g se expresa por medio del término λ:

$$Y \lambda f.\, \lambda y x_1 \ldots x_n.\, (Cero\ y)(H x_1 \ldots x_n)(G(P\ y) x_1 \ldots x_n (f(P\ y) x_1 \ldots x_n)).$$

Ejemplo. Función recursiva primitiva en el cálculo λ

Supongamos que ahora queremos definir la función suma presentada como ejemplo en las funciones recursivas μ en términos de Λ. Entonces:

$$\begin{aligned} H &\equiv_{def} \lambda x_1.\, x_1 \\ G &\equiv_{def} \lambda z, x_1, x_2.\, S((\lambda x_1, x_2, x_3.\, x_3) z x_1 x_2) \\ \text{SUMA} &\equiv_{def} Y \lambda f.\, \lambda y x.\, (Cero\ y)(H x)(G(P\ y) x (f(P\ y) x)) \end{aligned}$$

Minimización no acotada en el cálculo λ

Sea $g : \mathbb{N}^{n+1} \to \mathbb{N}$ una función computable. Como se vio antes, es posible definir la función $f : \mathbb{N}^n \to \mathbb{N}$

$$f(\bar{x}) = \mu y.\, g(y, \bar{x}) = 0.$$

Usando una notación informal, que *no* es parte de las funciones recursivas μ, la función anterior se podría expresar así:

$$\begin{aligned} f'(x_0, \bar{x}) &\equiv_{def} \textbf{if } g(x_0, \bar{x}) = 0 \textbf{ then } x_0 \\ &\qquad \textbf{else } f'(s(x_0), \bar{x}) \\ f(\bar{x}) &\equiv_{def} f'(0, \bar{x}). \end{aligned}$$

Si el término $G \in \Lambda$ calcula la función g, definimos el término F como el equivalente a la función f:

$$\begin{aligned} F' &\equiv_{def} Y \lambda f.\, \lambda x_0 x_1 \ldots x_n.\, (Cero\ (G\ x_0 x_1 \ldots x_n)) x_0 (f(S\ x_0) x_1 \ldots x_n) \\ F &\equiv_{def} \lambda x_1 \ldots x_n.\, F'\ \bar{0}\ x_1 \ldots x_n. \end{aligned}$$

Lenguaje de programación IMP

Se trata de un CFL. Empezaremos con las expresiones aritméticas EA:

$$A \to n \mid X \mid (A + A) \mid (A - A) \mid (A \times A)$$

donde $n \in \mathbb{Z}$ y $X \in \text{Loc}$.
Tenemos ahora las expresiones booleanas EB:

$$B \to V \mid F \mid (A = A) \mid (A < A) \mid \neg B \mid (B \wedge B) \mid (B \vee B)$$

donde $A \in EA$. Finalmente, los comandos del lenguaje se definen así:

$$C \to \textbf{skip} \mid X := A \mid (C\ ;\ C) \mid (\textbf{if } B \textbf{ then } C \textbf{ else } C) \mid (\textbf{while } B \textbf{ do } C).$$

donde $A \in EA$ y $B \in EB$.

Semántica operacional estructural I

- La forma más común de definir el significado de las construcciones de un lenguaje es por medio de explicaciones en una lengua natural (inglés, generalmente).
- Sin embargo, el lenguaje natural se presta a las ambigüedades y éstas, a los errores o a las divergencias entre las diferentes implementaciones de un mismo lenguaje.
- Para evitar estos problemas, aquí se usarán *reglas de semántica operacional estructural* o SOE, para abreviar.
- Una regla de inferencia tiene la forma siguiente:

$$\frac{p_1, \ldots, p_n}{q}$$

donde $p_1, \ldots, p_n$ son las premisas y q es la conclusión.

Semántica operacional estructural II

- Las reglas de SOE son un tipo particular de reglas de inferencia en las que las premisas y la conclusión son *transiciones*.
- Una *transición* tiene la forma

$$\alpha \to \beta$$

- La forma concreta de las expresiones α y β, así como las transiciones aceptables, se definirán más adelante.

¿Qué es la programación imperativa?

- IMP es un lenguaje imperativo típico: el comando básico es la asignación de valor a las localidades de la memoria de la computadora.
- Para definir el significado de una asignación se parte de la existencia de una *máquina virtual* que posee una memoria con localidades con nombre:

$$\text{Loc} = \{X, Y, Z, X_0, \ldots\}$$

- Las localidades pueden tomar valores de $\mathbb{Z}$.

Estados de la memoria

- Un *estado de la memoria* es una función $\sigma : \text{Loc} \to \mathbb{Z}$.
- La expresión
$$\sigma(X)$$
nos dice qué valor contiene la localidad X en el estado σ.
- La alteración del valor de una sola localidad de la memoria cuando se encuentra en el estado σ se representa con la función
$$\sigma_{[X:=m]},$$
donde X es la localidad modificada y m es el nuevo valor.
- Formalmente:
$$\sigma_{[X:=m]}(Y) = \begin{cases} \sigma(Y) & \text{si } X \neq Y \\ m & \text{en caso contrario.} \end{cases}$$
- El conjunto de estados es Σ.

SOE para IMP. Expresiones aritméticas

En el caso de las expresiones aritméticas, las transiciones $\alpha \to \beta$ tiene la forma
$$\langle a, \sigma\rangle \to n$$
donde $a \in EA$, $\sigma \in \Sigma$ y $n \in \mathbb{Z}$.
Diremos que "*evaluar* operacionalmente la expresión a en el estado σ da como resultado el valor n".
Dicha evaluación se realiza de acuerdo con las siguientes reglas:

$$\langle n, \sigma\rangle \to n \qquad \langle X, \sigma\rangle \to \sigma(X)$$

$$\frac{\langle a_0, \sigma\rangle \to n_0 \quad \langle a_1, \sigma\rangle \to n_1 \quad n_0 \; op_{\mathbb{Z}} \; n_1 = n}{\langle a_0 \; \mathbf{op} \; a_1, \sigma\rangle \to n}$$

En esta regla **op** puede ser +, − o × y $op_{\mathbb{Z}}$, $+_{\mathbb{Z}}$, $-_{\mathbb{Z}}$ o $\times_{\mathbb{Z}}$ (ambos operadores deben coincidir).

SOE para IMP. Expresiones booleanas I

En las expresiones booleanas, una transición es $\langle b, \sigma\rangle \to T$, con $b \in EB$, $\sigma \in \Sigma$ y $T \in \{V, F\}$. Las reglas operacionales son:

$$\langle V, \sigma\rangle \to V \qquad \langle F, \sigma\rangle \to F$$

$$\frac{\langle a_0, \sigma\rangle \to n \quad \langle a_1, \sigma\rangle \to m}{\langle a_0 = a_1, \sigma\rangle \to V} \quad sii \quad n \text{ y } m \text{ son iguales}$$

$$\frac{\langle a_0, \sigma\rangle \to n \quad \langle a_1, \sigma\rangle \to m}{\langle a_0 = a_1, \sigma\rangle \to F} \quad sii \quad n \text{ y } m \text{ no son iguales}$$

$$\frac{\langle a_0, \sigma\rangle \to n \quad \langle a_1, \sigma\rangle \to m}{\langle a_0 < a_1, \sigma\rangle \to V} \quad sii \quad n \text{ es menor que } m$$

$$\frac{\langle a_0, \sigma\rangle \to n \quad \langle a_1, \sigma\rangle \to m}{\langle a_0 < a_1, \sigma\rangle \to F} \quad sii \quad n \text{ no es menor que } m$$

SOE para IMP. Expresiones booleanas II

$$\frac{\langle b, \sigma\rangle \to V}{\langle \neg b, \sigma\rangle \to F} \qquad \frac{\langle b, \sigma\rangle \to F}{\langle \neg b, \sigma\rangle \to V}$$

$$\frac{\langle b_0, \sigma\rangle \to T_0 \quad \langle b_1, \sigma\rangle \to T_1}{\langle b_0 \wedge b_1, \sigma\rangle \to T} \quad \begin{array}{l} \text{con } T = V \text{ si } T_0, T_1 = V \\ \text{y } T = F \text{ en caso contrario} \end{array}$$

$$\frac{\langle b_0, \sigma\rangle \to T_0 \quad \langle b_1, \sigma\rangle \to T_1}{\langle b_0 \wedge b_1, \sigma\rangle \to T} \quad \begin{array}{l} \text{con } T = V \text{ si } T_0 = V \text{ o } T_1 = V \\ \text{y } T = F \text{ en caso contrario} \end{array}$$

SOE para IMP. Comandos I

Las reglas de transición de los comandos son así:

$$\langle c, \sigma \rangle \to \sigma',$$

donde $c \in$ IMP y $\sigma, \sigma' \in \Sigma$.
El valor de σ' se infiere de este modo:

Comandos atómicos

$$\langle \textbf{skip}, \sigma \rangle \to \sigma \qquad \frac{\langle a, \sigma \rangle \to m}{\langle X := a, \sigma \rangle \to \sigma_{[X:=m]}}$$

Composición secuencial

$$\frac{\langle c_0, \sigma \rangle \to \sigma'' \quad \langle c_1, \sigma'' \rangle \to \sigma'}{\langle c_0 \,;\, c_1, \sigma \rangle \to \sigma'}$$

SOE para IMP. Comandos II

Condicionales

$$\frac{\langle b, \sigma \rangle \to V \quad \langle c_0, \sigma \rangle \to \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma \rangle \to \sigma'} \qquad \frac{\langle b, \sigma \rangle \to F \quad \langle c_1, \sigma \rangle \to \sigma'}{\langle \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1, \sigma \rangle \to \sigma'}$$

Ciclos

$$\frac{\langle b, \sigma \rangle \to F}{\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \to \sigma}$$

$$\frac{\langle b, \sigma \rangle \to V \quad \langle c, \sigma \rangle \to \sigma'' \quad \langle \textbf{while } b \textbf{ do } c, \sigma'' \rangle \to \sigma'}{\langle \textbf{while } b \textbf{ do } c, \sigma \rangle \to \sigma'}$$

Funciones recursivas y programas while I

Las funciones μ también son definibles por medio de programas de IMP.
Sea

$$f : \mathbb{N}^n \to \mathbb{N}$$

una función recursiva. El objeto es definir un programa P con al menos las localidades

$$X_0, \ldots, X_n$$

tal que si $\langle P, \sigma \rangle \to \sigma'$ entonces

$$\sigma'(X_0) = f(\sigma(X_1), \ldots, \sigma(X_n)).$$

Funciones recursivas y programas while II

Lo haremos por inducción en la funciones recursivas.

$$\begin{aligned} P_0 &\equiv_{def} X_0 := 0 \\ P_S &\equiv_{def} X_0 := X_1 + 1; \\ P_{\pi_i^n} &\equiv_{def} X_0 := X_i; \end{aligned}$$

Sean P_f y $P_{g_1}, \ldots, P_{g_k}$ los programas correspondientes a las funciones $f : \mathbb{N}^k \to \mathbb{N}$ y $g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N}$. Entonces, el programa correspondiente a $h = f \circ (g_1, \ldots, g_k)$ es

$$\begin{aligned} P_h \equiv_{def} \; & Y_1 := X_1; \ldots; Y_n := X_n; P_{g_1}; Z_1 := X_0; \\ & X_1 := Y_1; \ldots; X_n := Y_n; P_{g_2}; Z_2 := X_0; \\ & \ldots \\ & X_1 := Y_1; \ldots; X_n := Y_n; P_{g_k}; Z_k := X_0; \\ & X_1 := Z_1; \ldots X_k := Z_k; P_f \end{aligned}$$

Funciones recursivas y programas while III

Veamos ahora un caso restringido para la recursión primitiva. Supongamos que tenemos las funciones

$$h : \mathbb{N} \to \mathbb{N} \qquad \text{y} \qquad g : \mathbb{N}^3 \to \mathbb{N}$$

y a partir de ellas definimos la función $f : \mathbb{N}^2 \to \mathbb{N}$:

$$\begin{aligned} f(0, x_1) &= h(x_1) \\ f(s(n), x_1) &= g(n, x_1, f(n, x_1)) \end{aligned}$$

Queremos un programa P_f tal que si $\langle P, \sigma \rangle \to \sigma'$ entonces

$$\sigma'(X_0) = f(\sigma(Z), \sigma(X_1)).$$

Funciones recursivas y programas while IV

Por hipótesis inductiva, suponemos la existencia de dos programas

$$\begin{aligned} \sigma'(X_0) &= h(\sigma(X_1)) && \text{donde } \langle P_h, \sigma \rangle \to \sigma' \\ \sigma'(X_0) &= g(\sigma(Z), \sigma(X_1), \sigma(X_2)) && \text{donde } \langle P_g, \sigma \rangle \to \sigma' \end{aligned}$$

Entonces P_f es el programa

$W := Z; Y_1 := X_1;$ *Guardamos los argumentos originales de f*
$Y_0 := 0;$ *Usaremos a Y_0 como contador*
$P_h;$ *Ejecutamos el programa del caso base*
while $Y_0 < W$ **do**
  $Z := Y_0;$
  $X_1 := Y_1;$
  $X_2 := X_0;$
  $P_g;$
  $Y_0 := Y_0 + 1$

Funciones recursivas y programas while V

Por ejemplo, si tenemos la función recursiva

$$\begin{aligned} \text{suma}(0, x_1) &= x_1 \\ \text{suma}(s(n), x_1) &= s(\text{suma}(n, x_1)) \end{aligned}$$

aquí la función base $h : \mathbb{N} \to \mathbb{N}$ se define como la proyección $\pi^1_1(x_1) = x_1$ y la función $g : \mathbb{N}^3 \to \mathbb{N}$ es $s \circ \pi^3_3(n, x_1, x_2) = s(x_2)$. Los programas P_h y P_g tales que si $\langle P_h, \sigma \rangle \to \sigma'$ y $\langle P_g, \sigma_1 \rangle \to \sigma_1'$

$$\begin{aligned} \sigma'(X_0) &= \pi^1_1(\sigma(X_1)) \\ \sigma_1'(X_0) &= s \circ \pi^3_3(\sigma(Z), \sigma(X_1), \sigma(X_2)) \end{aligned}$$

son:

$$\begin{aligned} P_h &= X_0 := X_1 \\ P_g &= X_0 := X_2 + 1 \end{aligned}$$

Funciones recursivas y programas while VI

y entonces, el programa P_{suma} tal que si $\langle P_{suma}, \sigma \rangle \to \sigma'$ entonces

$$\sigma'(X_0) = \text{suma}(\sigma(Z), \sigma(X_1))$$

queda así:

$W := Z; Y_1 := X_1;$
$Y_0 := 0;$
$P_h;$
while $Y_0 < W$ **do**
  $Z := Y_0;$
  $X_1 := Y_1;$
  $X_2 := X_0;$
  $P_g;$
  $Y_0 := Y_0 + 1$

Máquinas de registros

Las máquinas de registros ilimitados (URM) son una abstracción del procesador central de una computadora.
La memoria es como la del lenguaje IMP, pero las localidades se llaman registros:

$$\text{Reg} = \{R_1, R_2 \ldots\}$$

Los programas para URM se basan en unas cuantas instrucciones:

Nombre	Sintaxis	Significado
Cero	$Z(n)$	$R_n := 0$
Sucesor	$S(n)$	$R_n := R_n + 1$
Transferencia	$T(n, m)$	$R_n := R_m$
Salto	$J(n, m, k)$	si $R_n = R_m$ entonces ve a la instrucción k–ésima; si no, continúa con la siguiente

Los registros guardan números naturales y los estados posibles son

$$\Sigma = \{\sigma : \text{Reg} \to \mathbb{N}\}$$

Este programa calcula la suma de los números en R_1 y R_2. La tabla presenta el estado de la memoria después de cada instrucción:

			R_1	R_2	R_3
		1.1	1	2	0
		2.1	2	2	0
1	$J(2, 3, 5)$	3.1	2	2	1
2	$S(1)$	4.1	2	2	1
3	$S(3)$	1.2	2	2	1
4	$J(1, 1, 1)$	2.2	3	2	1
5		3.2	3	2	2
		4.2	3	2	2
		1.3	3	2	2
		5.1	3	2	2

Funciones recursivas y URM

Las funciones recursivas μ pueden calcularse por medio de URM de una manera muy similar a los programas en IMP:

- Las funciones cero, sucesor y proyecciones tienen una implementación muy simple.
- Para la composición de funciones hay que adoptar convenciones para guardar resultados parciales de funciones y presentar parámetros de manera similar a como se hizo con los programas IMP.
- Además, hay que reenumerar las direcciones referidas en la operación de salto, para evitar interferir en el código de otros programas.
- Finalmente, la recursión y la minimalización se manejan como se hizo con los ciclos **while** en IMP.