Tema 1
Un Modelo de Funciones Computables

Antonio J. Tomeu[1]

[1]Departamento de Ingeniería Informática
Universidad de Cádiz

Contenidos

Definición de Modelo

Definición (Modelo)

Un modelo es una representación simplificada de una realidad física, química, biológica, computacional, etc. El modelo se construye mediante un proceso de abstracción que elimina características consideradas irrelevantes de la realidad modelada, e incorpora aquellas consideradas relevantes.

Ejemplo (Ejemplos)

- La ecuación de campo gravitatorio de Einstein.
- Un mapa del mundo.
- Una clase en un lenguaje orientado a objetos.
- El diagrama de una molécula.

Modelo Físico: Ecuación de Campo de Einstein

- Modelo Gravitatorio.
- Ecuación Tensorial.
- La curvatura depende de la concentración de masa-energía.

$$G_{\mu\nu}+\Lambda g_{\mu\nu}=\frac{8\pi G}{c^4}T_{\mu\nu}$$

Modelo Físico: Mapa del Mundo

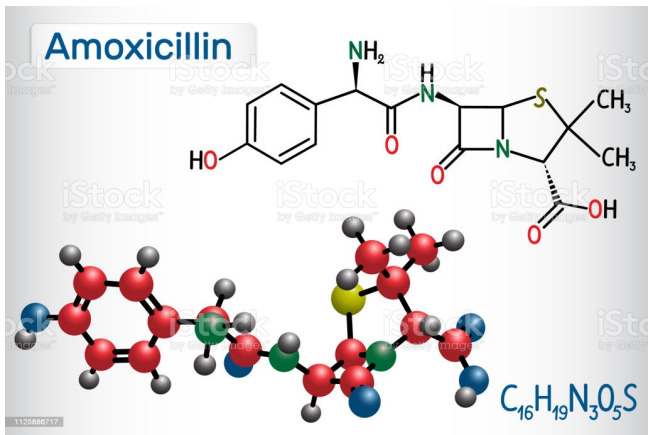
- Representación Simplificada del Relieve Terrestre.
- Elimina detalles muy pequeños o poco importantes.
- La proyección altera las proporciones.

Modelo Computacional: Una clase

- Representación de una cuenta corriente en un lenguaje de programación.
- Atributos almacenan información.
- Métodos constructores, destructores, observadores y modificadores.
- Sobradamente conocido.

Modelo Químico: Diagrama de una Molécula

- Representación de una realidad química: antibiótico de amplio espectro.
- Modela átomos y enlaces.
- No modela la intensidad de los enlaces.



Modelo de Computación

Definición (Modelo de Computación)

Un modelo de computación es una representación simplificada de una realidad computacional que puede ser hardware, software, o de otra naturaleza de interés, y que tiene como objetivo establecer de manera matemáticamente rigurosa el concepto de «computabilidad».

Ejemplo (Ejemplos)

- El cálculo λ de Church.
- La teoría de funciones recursivas de Kleene.
- Las máquinas de Turing.
- El modelo de Post.
- Las máquinas URM de Sheperdon Sturgis...
- ... y muchos más.

Propósitos de un Modelo de Computación

- Tradicionalmente, los modelos de computación se construían como una teoría (matemática) para dar respuesta a preguntas como las siguientes:
  - ¿Qué es computable (o algoritmizable)
  - ¿Existen elementos (problemas) no computables?
  - ¿Por qué las computadoras son programables?
- Actualmente, el concepto de modelo de computación es más amplio, y también más flexible, y sirve para englobar a cualquier herramienta formal que sirva para obtener nuevas teorías, conocimiento, desarrollos, técnicas, etc. en cualquier rama relacionada con la informática.
- En este curso, seguiremos en enfoque clásico, estudiando qué funciones numérica son computables (en informática, todo puede reducirse a una función numérica), sin perder de vista el enfoque actual, que no obstante se aborda y amplía en otras materias de la especialidad: Teoría de Autómatas, Teoría de la Complejidad, etc.

Preliminares: Funciones numéricas

Definición (Función numérica)

Una función numérica es una aplicación

$$f : \mathbb{N}^k \to \mathbb{N}, \text{ para } k \in \mathbb{N} \text{ y } k \geq 1$$

Ejemplo (Factorial)

$$fac : \mathbb{N} \to \mathbb{N}$$
$$fac(n) = n \cdot (n-1) \cdot \ldots \cdot 1$$

Ejemplo (Suma)

$$g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$$
$$g(x, y) = x + y$$

El Modelo L

Definición (Memoria)

La estructura de memoria de L es un conjunto de variables de la forma:

- De entrada: $V_E = \{X_1, X_2, X_3, ...\}$ con $X_i \in \mathbb{N}$, para todo i
- Locales: $V_L = \{Z_1, Z_2, Z_3, ...\}$ con $Z_j \in \mathbb{N}$, para todo j
- De salida: $Y \in \mathbb{N}$

Definición (Etiquetas)

L dispone de las siguientes etiquetas

$$Lab = \{A_1, B_1, C_1, D_1, E_1, A_2, B_2, C_2, D_2, E_2, ...\}$$

El lenguaje L

Definición (Instrucciones)

Instrucción	Significado
`V` $\leftarrow$ `V+1`	Incrementa a V la unidad.
`V` $\leftarrow$ `V-1`	Decrementa a V la unidad. Si $V = 0$ su valor no cambia.
`V` $\leftarrow$ `V`	Intrucción "dummy". No hace nada.
`IF V` $\neq$ `0 GOTO L`	Si $V \neq 0$, salta a la instrucción etiquetada con $[L]$. Si $V = 0$, continúa a la siguiente instrucción. Si no hay instrucción etiquetada con $[L]$, el programa acaba.

El lenguaje L

Definición (Valores iniciales de las variables)

- Entrada: $X_1 = x_1, X_2 = x_2, X_3 = x_3, ..., X_k = x_k$. El resto toman valor 0.
- Locales: $Z_j = 0, \forall j \in \mathbb{N}$
- Salida: $Y = 0$

Ejemplos de Instancias de L

Ejemplo (Función pseudoidentidad)

```
[A] X ← X-1
    Y ← Y + 1
    IF X ≠ 0 GOTO A
```

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

$$f(x) = \begin{cases} x, & \text{si } x \neq 0 \\ 1, & \text{si } x = 0 \end{cases}$$

Ejemplos de Instancias de L

Ejemplo (Función identidad)

```
[A] IF X ≠ 0 GOTO B
    Z ← Z + 1
    IF Z ≠ 0 GOTO E
[B] X ← X - 1
    Y ← Y + 1
    Z ← Z + 1
    IF Z ≠ 0 GOTO A
```

$$g : \mathbb{N} \to \mathbb{N}$$

$$g(x) = x, \forall x \in \mathbb{N}$$

Ejemplo (Macro de bifurcación incondicional)

```
GOTO E ↔  Z ← Z + 1
          IF Z ≠ 0 GOTO E
```

Ejemplos de Instancias de L

Ejemplo (Macro de asignación)

Se define la macro $V' \leftarrow V$ como sigue:

```
[A]  IF X ≠ 0 GOTO B
     GOTO C
[B]  X ← X - 1
     Y ← Y + 1
     Z ← Z + 1
     GOTO A
[C]  IF Z ≠ 0 GOTO D
     GOTO E
[D]  Z ← Z - 1
     X ← X + 1
     GOTO C
```

Esta macro se ayuda de la variable local Z para no destruir X.

Ejemplos de Instancias de L

Ejemplo (Función Suma)

Se define la macro $\mathtt{V} \leftarrow \mathtt{V}' + \mathtt{V}''$ como sigue:

$$
\begin{array}{ll}
 & \mathtt{Y} \leftarrow \mathtt{X}_1 \\
 & \mathtt{Z} \leftarrow \mathtt{X}_2 \\
\mathtt{[B]} & \mathtt{IF\ Z} \neq \mathtt{0\ GOTO\ A} \\
 & \mathtt{GOTO\ E} \\
\mathtt{[A]} & \mathtt{Z} \leftarrow \mathtt{Z - 1} \\
 & \mathtt{Y} \leftarrow \mathtt{Y + 1} \\
 & \mathtt{GOTO\ B}
\end{array}
$$

Ejemplos de Instancias de L

Ejemplo (Función Resta Parcial)

Se define la macro $V \leftarrow V' \text{ -}V''$ como sigue:

```
      Y ← X1
      Z ← X2
[C]   IF Z ≠ 0 GOTO A
      GOTO E
[A]   IF Y ≠ 0 GOTO B
      GOTO A
[B]   Y ← Y - 1
      Z ← Z - 1
      GOTO C
```

Ejemplos de Instancias de L

Ejemplo (Función Producto)

Se define la macro $V \leftarrow V' \cdot V''$ como sigue:

$$
\begin{array}{ll}
 & Z_2 \leftarrow X_2 \\
[B] & \texttt{IF } Z_2 \neq 0 \texttt{ GOTO A} \\
 & \texttt{GOTO E} \\
[A] & Z_2 \leftarrow Z_2 - 1 \\
 & Z_1 \leftarrow X_1 + Y \\
 & Y \leftarrow Z_1 \\
 & \texttt{GOTO B}
\end{array}
$$

Modelo semántico de L

Definición (Modélo semántico)

Un modelo semántico atribuye significado a los L-programas.

Definición (Tipos de semántica)

De menor a mayor nivel de abstracción:

- Operacional: Trata de estudiar el comportamiento de una rutina estudiando directamente su código.
- Denotacional: Trata de estudiar el comportamiento de una rutina a nivel funcional. Se trata como una caja negra, analizando la salida en base a la entrada.
- Axiomática: Trata de estudiar el comportamiento de una rutina mediante la aplicación de reglas lógicas.

Tanto la semántica operacional como la denotacional son usadas a nivel práctico, mientras la semántica axiomática es usada únicamente a nivel teórico.

Modelo semántico de L

Definición (Estado)

Un estado es una lista de ecuaciones de la forma $V = m$, donde V es una variable y $m \in \mathbb{N}$.

Ejemplo

- $< X_1 = 1, X_2 = 1, Z_1 = 0, Z_2 = 0, Y = 0 >$
- $< X_1 = 250, X_2 = 154, Z_1 = 3, Z_2 = 6, Y = 10000 >$

Modelo semántico de L

Definición (Configuración)

Dado un L-programa P, una configuración es un par de la forma (i,σ) donde $i \in \mathbb{N}$, $(1 \leq i \leq long(P)+1)$ y σ es un estado de P.

Ejemplo

- $(1, < X_1 = 1, X_2 = 1, Z_1 = 0, Z_2 = 0, Y = 0 >)$
- $(7, < X_1 = 250, X_2 = 154, Z_1 = 3, Z_2 = 6, Y = 10000 >)$

Modelo semántico de L

Definición (Sentencias de L)

- `V` $\leftarrow$ `V + 1`
- `V` $\leftarrow$ `V - 1`
- `V` $\leftarrow$ `V`
- `IF V` $\neq$ `0 GOTO L`

Definición (Instrucciones de L)

Una instrucción de un L-programa P es una sentencia de la forma `I` o `[L] I`, donde `I` es una sentencia con una variable concreta tomada en $V_E \cup V_L \cup \{Y\}$ y $L \in Lab$.

Modelo semántico de L

Definición (Configuración sucesora)

Sea P un L-programa y sean (i, σ) y (j, τ) dos configuraciones de P. Se dice que (j, τ) es sucesora de (i, σ) si:

- Si la i-ésima instrucción de P es de la forma V $\leftarrow$ V, entonces $j = i + 1$ y $\tau = \sigma$.
- Si la i-ésima instrucción de P es de la forma V $\leftarrow$ V + 1, entonces $j = i + 1$ y τ se obtiene de σ cambiando la ecuación $V = m$ por $V = m + 1$.
- Si la i-ésima instrucción de P es de la forma V $\leftarrow$ V - 1, entonces $j = i + 1$ y τ se obtiene de σ cambiando la ecuación $V = m$ por $V = m - 1$. Si $m = 0$, no hay cambios.

Modelo semántico de L

Definición (Configuración sucesora (cont.))

- Si la i-ésima instrucción es de la forma `IF V` $\neq$ `0 GOTO L`, entonces
  - Si $V \neq 0$, entonces $\tau = \sigma$ y j:
    - Si no hay instrucciones etiquetadas con $[L]$, entonces $j = long(P) + 1$.
    - Si hay una única instrucción etiquetada con $[L]$ en la k-ésima posición de P, entonces $j = k$.
    - Si hay varias instrucciones etiquetadas con $[L]$ en las posiciones $k_1, k_2, ..., k_n$ del programa, $j = min\{k_1, k_2, ..., k_n\}$.
  - Si $V = 0$, entonces $\tau = \sigma$ y $j = i + 1$.
- Notación: $(i, \sigma) \sim (j, \tau)$: (j, τ) es sucesora de (i, σ).

Definición (Computación)

Dado un L-programa P, una computación es una sucesión finita de configuraciones $S_1, S_2, ..., S_k$, donde $S_i \sim S_{i+1}$ para $i = 1, 2, ..., k - 1$ y S_k es final.

Computación: Ejemplo

Ejemplo

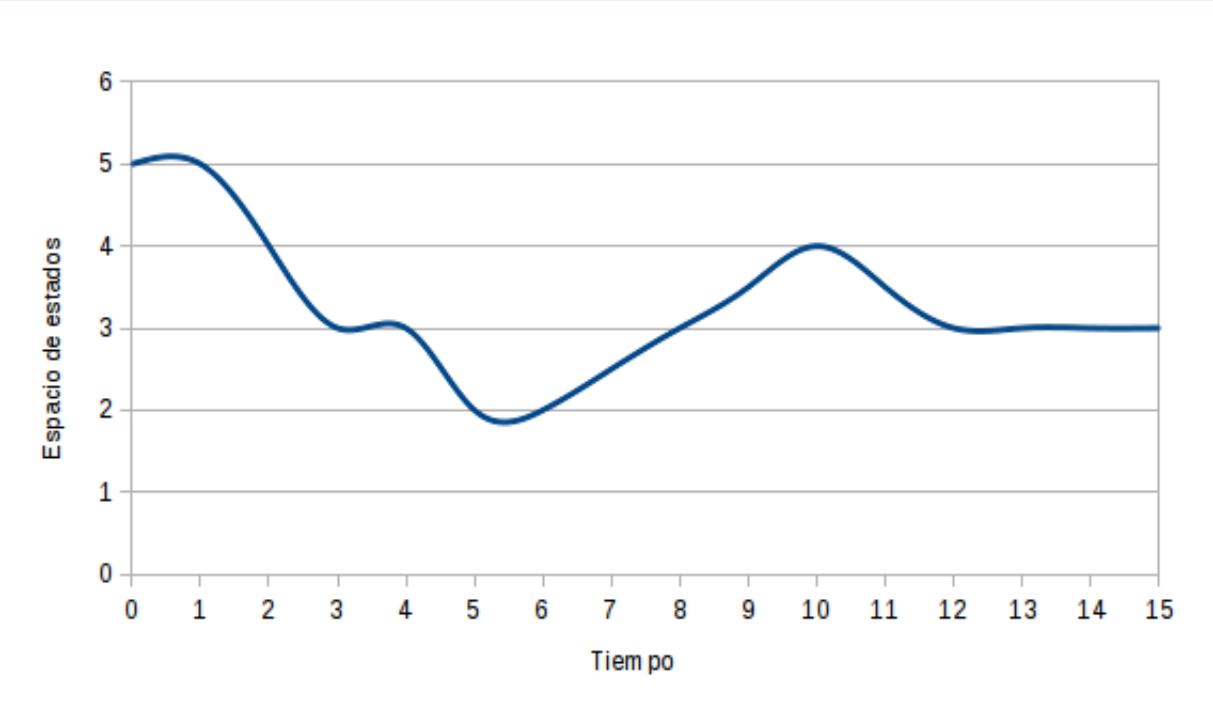
$$
\begin{array}{ll}
[A] & \textit{IF } X \neq 0 \textit{ GOTO } B \\
 & Z \longrightarrow Z + 1 \\
 & \textit{IF } Z \neq 0 \textit{ GOTO } E \\
[B] & X \longrightarrow X - 1 \\
 & Y \longrightarrow Y + 1 \\
 & \textit{IF } X \neq 0 \textit{ GOTO } A
\end{array}
$$

$$
\begin{aligned}
s_1 &= (1, \{X = 3,\ Y = 0,\ Z = 0\}) \\
s_2 &= (4, \{X = 3,\ Y = 0,\ Z = 0\}) \\
s_3 &= (5, \{X = 2,\ Y = 0,\ Z = 0\}) \\
s_4 &= (6, \{X = 2,\ Y = 1,\ Z = 0\}) \\
s_5 &= (1, \{X = 2,\ Y = 1,\ Z = 0\}) \\
s_6 &= (4, \{X = 2,\ Y = 1,\ Z = 0\}) \\
s_7 &= (5, \{X = 1,\ Y = 1,\ Z = 0\}) \\
s_8 &= (6, \{X = 1,\ Y = 2,\ Z = 0\}) \\
s_9 &= (1, \{X = 1,\ Y = 2,\ Z = 0\}) \\
s_{10} &= (4, \{X = 1,\ Y = 2,\ Z = 0\}) \\
s_{11} &= (5, \{X = 0,\ Y = 2,\ Z = 0\}) \\
s_{12} &= (6, \{X = 0,\ Y = 3,\ Z = 0\}) \\
s_{13} &= (7, \{X = 0,\ Y = 3,\ Z = 0\})
\end{aligned}
$$

Modelo semántico de L

Ejemplo

Una computación puede ser representada como una trayectoria en el espacio de estados, donde los valores tomados en el eje y indican la configuración adoptada en cada instante de tiempo.



Funciones L-computables

Definición (Función L-computable)

Sea un L-programa P. Sean $X_1, X_2, ..., X_k$ sus variables de entrada. Se construye la siguiente configuración inicial:

$$(1, < X_1 = r_1, X_2 = r_2, ..., X_k = r_k, Z_1 = 0, ..., Z_m = 0, Y = 0 >)$$

donde $r_i \in \mathbb{N}$ para $1 \leq i \leq k$. Se comienza la ejecución y entonces:

- Existe una computación $S_1, S_2, ..., S_k$ de P, donde S_k es terminal. Diremos que

$$Y = \varphi_P^{(k)}(X_1, X_2, ..., X_k)$$

  donde $X_1, X_2, ..., X_k$ son valores concretos y corresponden a $r_1, r_2, ..., r_k$.
- No existe tal computación. Diremos que

$$\varphi_P^{(k)}(X_1, X_2, ..., X_k) = \uparrow$$

Funciones L-computables

Definición (Función parcialmente computable)

Una función $f : \mathbb{N}^k \to \mathbb{N}$ es parcialmente computable si existe un L-programa P, tal que:

$$f(x_1, x_2, ..., x_k) = \varphi_P^{(k)}(x_1, x_2, ..., x_k)$$

Ejemplo

La resta parcial es parcialmente computable.

Funciones L-computables

Definición (Función computable)

Una función $f : \mathbb{N}^k \to \mathbb{N}$ es totalmente computable si existe un L-programa P, tal que:

$$f(x_1, x_2, ..., x_k) = \varphi_P^{(k)}(x_1, x_2, ..., x_k), \forall (x_1, x_2, ..., x_k) \in \mathbb{N}^k$$

Ejemplo

La suma y el producto son funciones totalmente computables.

Macros

Sea P un L-programa cualquiera, y sea $f : \mathbb{N}^k \to \mathbb{N}$ una función tal que

$$f(x_1, x_2, ..., x_k) = \varphi_P^k(x_1, x_2, ..., x_k)$$

Nuestro objetivo es disponer de macros de la forma $w \leftarrow f(V_1, V_2, ..., V_k)$. Escribimos $P = P(Y, X_1, ..., X_n, Z_1, ..., Z_k; E, A_1, ..., A_l)$ de forma que podemos representar programas obtenidos de P reemplazando las variables y etiquetas por otras. En particular, escribiremos $Q_m = P(Z_m, Z_{m+1}, ..., Z_{m+n}, Z_{m+n+1}, ..., Z_{m+n+k}; E_m, A_{m+1}, ..., A_{m+l})$ para cualquier m. Habrá que preparar las variables para un estado inicial.

Macros

- Preparación de las variables para la expansión de una macro

$$
\begin{aligned}
& Z_m \leftarrow 0 \\
& Z_{m+1} \leftarrow V_1 \\
& Z_{m+2} \leftarrow V_2 \\
& \vdots \\
& Z_{m+n} \leftarrow V_n \\
& Z_{m+n+1} \leftarrow 0 \\
& Z_{m+n+2} \leftarrow 0 \\
& \vdots \\
& Z_{m+n+k} \leftarrow 0 \\
& Q_m \\
[E_m]\ & W \leftarrow Z_m
\end{aligned}
$$

Macros

- Macros Basadas en Predicados

$$\texttt{IF P}(\texttt{V}_1,\texttt{V}_2,\ldots,\texttt{V}_k)\ \texttt{GOTO L} \leftrightarrow \begin{array}{l}\texttt{W} \leftarrow \texttt{P}(\texttt{V}_1,\texttt{V}_2,\ldots,\texttt{V}_k)\\ \texttt{IF W} \neq \texttt{0 GOTO L}\end{array}$$

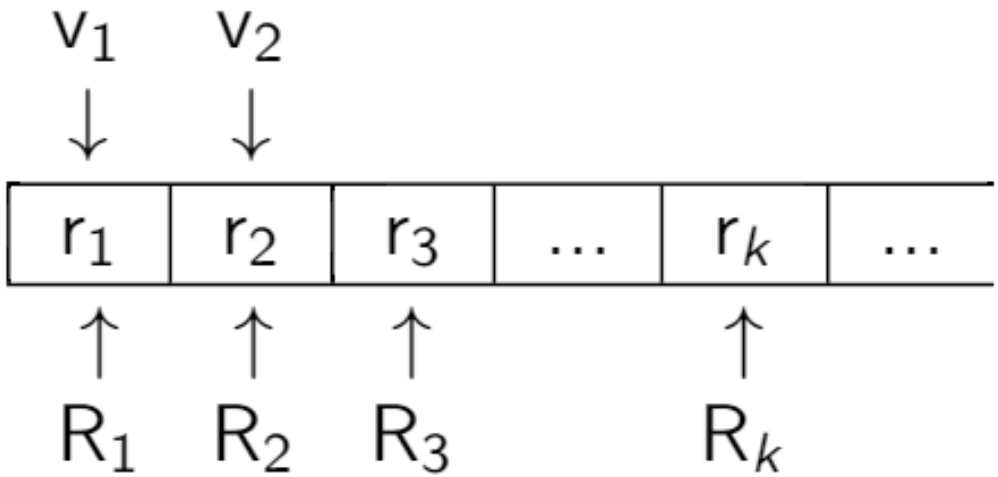
El Modelo URM

El modelo URM (Unlimited Register Machine) representa una abstracción de una máquina de registros ilimitados.

Definición (Memoria)

La memoria de URM se compone de una serie de celdas llamadas registros que pueden almacenar cualquier número natural. Un URM-programa puede hacer uso de un número finito de registros. Los registros serán referenciados con las letras $R_1, R_2, ..., R_k$, y sus valores correspondientes pueden ser referenciados con la letra minúscula correspondiente: $r_1, r_2, ..., r_k$.
Al inicio de un URM-programa, los valores de entrada $v_1, v_2, ..., v_k$ son almacenados en los registros $R_1, R_2, ..., R_k$, respectivamente, y el resto de registros son inicializados con valor 0.
Al finalizar la ejecución del programa, el valor de salida es almacenado en el registro R_1.

El Modelo URM



Definición (Instrucciones de URM)

Instrucción	Significado
`Z(n)`	Reemplaza el valor del registro R_n por 0.
`S(n)`	Incrementa el registro R_n en la unidad.
`T(m,n)`	Copia el valor del registro R_m en el registro R_n.
`J(m,n,i)`	Si los registros R_m y R_n son iguales, entonces se salta a la i-ésima instrucción. Si su valor es distinto, se continúa a la siguiente instrucción. Si no existe la i-ésima instrucción, el programa termina.

Ejemplo (Suma)

```
J(2,3,0)
S(1)
S(3)
J(1,1,1)
```

Ejemplo (Resta Parcial)

```
J(1,2,5)
S(2)
S(3)
J(1,1,1)
T(3,1)
```

Ejemplo (Producto)

```
J(1,3,0)
J(2,4,10)
Z(5)
J(1,5,8)
S(3)
S(5)
J(1,1,4)
S(4)
J(1,1,2)
T(3,1)
```

Definición (Estado)

Un estado es una lista de ecuaciones de la forma $R_i = m$, donde R_i es un registro, $i \in \mathbb{Z}$ y $m \in \mathbb{N}$.

Definición (Configuración)

Dado un URM-programa P, una configuración es un par de la forma (i,σ) donde $i \in \mathbb{N}$, $(1 \leq i \leq long(P) + 1)$ y σ es un estado de P.

Definición (Sentencias de URM)

- `Z(n)`
- `S(n)`
- `T(m,n)`
- `J(m,n,i)`

Definición (Instrucciones de URM)

Una instrucción de un URM-programa P es una sentencia donde cada uno de los registros a los que se hace referencia pertenecen a $\mathbb{N}$.

Definición (Configuración sucesora)

Sea P un URM-programa y sean (i, σ) y (j, τ) dos configuraciones de P. Se dice que (j, τ) es sucesora de (i, σ) si:

- Si la i-ésima instrucción de P es de la forma `Z(n)`, entonces $j = i + 1$ y τ se obtiene de σ cambiando la ecuación $R_n = v$ por $R_n = 0$.
- Si la i-ésima instrucción de P es de la forma `S(n)`, entonces $j = i + 1$ y τ se obtiene de σ cambiando la ecuación $R_n = v$ por $R_n = v + 1$.
- Si la i-ésima instrucción de P es de la forma `T(m,n)`, entonces $j = i + 1$ y τ se obtiene de σ cambiando la ecuación $R_n = v$ por $R_n = w$, siendo w el valor del registro R_m.

Definición (Configuración sucesora (cont.))

- Si la i-ésima instrucción es de la forma `J(m,n,k)`, entonces
  - Si $R_m \neq R_n$, entonces $\tau = \sigma$ y $j = i + 1$.
  - Si $R_m = R_n$, entonces $\tau = \sigma$ y:
    - Si existe una k-ésima instrucción, entonces $j = k$.
    - Si $k = 0$ o $k > long(P)$, entonces el programa termina.

Definición (Computación)

Dado un URM-programa P, una computación es una sucesión finita de configuraciones $S_1, S_2, ..., S_k$, donde $S_i \sim S_{i+1}$ para $i = 1, 2, ..., k - 1$ y S_k es final.

Definición (Función URM-computable)

Sea un URM-programa P. Se construye la siguiente configuración inicial:

$$(1, < R_1 = r_1, R_2 = r_2, ..., R_k = r_k >)$$

donde $r_i \in \mathbb{N}$ para $1 \leq i \leq k$. Se comienza la ejecución y entonces:

- Existe una computación $S_1, S_2, ..., S_k$ de P, donde S_k es terminal. Diremos que

$$R_1 = \mu_P^{(k)}(X_1, X_2, ..., X_k)$$

donde $X_1, X_2, ..., X_k$ son valores concretos y corresponden a $r_1, r_2, ..., r_k$.

Definición (Función URM-computable)

- No existe tal computación. Diremos que

$$\mu_P^{(k)}(X_1, X_2, ..., X_k) = \uparrow$$

donde $X_1, X_2, ..., X_k$ son valores concretos y corresponden a $r_1, r_2, ..., r_k$.

Definición (Función parcialmente URM- computable)

Una función $f : \mathbb{N}^k \to \mathbb{N}$ es parcialmente computable si existe un URM-programa P, tal que:

$$f(x_1, x_2, ..., x_k) = \mu_P^{(k)}(x_1, x_2, ..., x_k)$$

Definición (Función URM-computable)

Una función $f : \mathbb{N}^k \to \mathbb{N}$ es totalmente computable si existe un URM-programa P, tal que:

$$f(x_1, x_2, ..., x_k) = \mu_P^{(k)}(x_1, x_2, ..., x_k), \forall (x_1, x_2, ..., x_k) \in \mathbb{N}^k$$

El Modelo While-Loop

El modelo While-Loop es capaz de computar exactamente las funciones recursivas primitivas, cuya definición daremos más adelante, y que puede ver como un subconjunto de las funciones totalmente computables. que son Los programas escritos en este lenguaje suelen llamarse *While* − *Loop*-programas.

Definición (Memoria)

La estructura de memoria de L es similar a la vista anteriormente en el modelo L. Se define como un conjunto de variables de la forma:

- De entrada: $V_E = \{X_1, X_2, X_3, ...\}$ con $X_i \in \mathbb{N}$, para todo i
- Locales: $V_L = \{Z_1, Z_2, Z_3, ...\}$ con $Z_j \in \mathbb{N}$, para todo j
- De salida: $Y \in \mathbb{N}$

El Modelo While-Loop

Definición (Instrucciones de While-Loop)

Instrucción	Significado
`V ← 0`	Asigna a `V` el valor 0.
`V ← V+1`	Incrementa `V` en la unidad.
`V ← V'`	Copia en `V` el valor de `V'`.
`LOOP V`	Determina el comienzo de un bucle.
`END`	Determina el final de un bucle.
`skip`	Termina la ejecución.

El Modelo While-Loop

- Las instrucciones `LOOP V` y `END` siempre deben aparecer por pares.
- Su funcionamiento es el siguiente: en el momento en el que la ejecución pasa por una instrucción `LOOP V`, las instrucciones comprendidas entre ésta y la instrucción `END` correspondiente serán ejecutadas tantas veces como indique el valor de `V` en ese momento, sin importar si se modifica su valor en el interior del bloque `LOOP - END`.
- Este comportamiento implica que todas las funciones modeladas bajo este modelo son totalmente computables.

El Modelo While-Loop

Ejemplo (Suma)

```
Y ← X1
LOOP X2
Y ← Y+1
END
```

Ejemplo (Producto)

```
LOOP X1
LOOP X2
Y ← Y + 1
END
END
```

El Modelo While-Loop

Definición (Valores iniciales de las variables)

- Entrada: $X_1 = x_1, X_2 = x_2, X_3 = x_3, ..., X_k = x_k$. El resto toman valor 0.
- Locales: $Z_j = 0, \forall j \in \mathbb{N}$
- Salida: $Y = 0$

Definición (Estado)

Un estado es una lista de ecuaciones de la forma $V = m$, donde V es una variable y $m \in \mathbb{N}$.

Definición (Configuración)

Dado un *loop*-programa P, una configuración es un par de la forma (i,σ) donde $i \in \mathbb{N}$, $(1 \leq i \leq long(P) + 1)$ y σ es un estado de P.

El Modelo While-Loop

Definición (Sentencias de W)

- `V` $\leftarrow$ `0`
- `V` $\leftarrow$ `V + 1`
- `V` $\leftarrow$ `V'`
- `LOOP V...END`
- `skip`

Definición (Instrucciones de L)

Una instrucción de un *loop*-programa P es una sentencia del lenguaje con una variable concreta tomada en $V_E \cup V_L \cup \{Y\}$.

Definición (Configuración sucesora)

Sea P un *loop*-programa y sean (i, σ) y (j, τ) dos configuraciones de P. Se dice que (j, τ) es sucesora de (i, σ) si:

- Si la i-ésima instrucción de P es de la forma `V` $\leftarrow$ `0`, entonces $j = i + 1$ y τ se obtiene de σ cambiando la ecuación $V = m$ por $V = 0$.
- Si la i-ésima instrucción de P es de la forma `V` $\leftarrow$ `V + 1`, entonces $j = i + 1$ y τ se obtiene de σ cambiando la ecuación $V = m$ por $V = m + 1$.
- Si la i-ésima instrucción de P es de la forma `V` $\leftarrow$ `V'`, entonces $j = i + 1$ y τ se obtiene de σ cambiando la ecuación $V = m$ por $V = V'$.

Definición (Configuración sucesora (cont.))

- Si la i-ésima instrucción es de la forma `LOOP V`, entonces $j = i + 1$ y $\tau = \sigma$. Las instrucciones comprendidas entre ésta y su correspondiente instrucción `END` serán ejecutadas tantas veces como indique el valor de `V`.
- Si la i-ésima instrucción es de la forma `END`, entonces $\tau = \sigma$ y $j = k$, donde k es el número de la instrucción `LOOP V` emparejada con esta instrucción. Si el bloque se instrucciones ya se ha ejecutado V veces, entonces $j = i + 1$
- Si la i-ésima instrucción es de la forma `skip`, entonces $\tau = \sigma$ y $j = long(P) + 1$.

El Modelo While-Loop

Definición (Computación)

Dado un *loop*-programa P, una computación es una sucesión finita de configuraciones $S_1, S_2, ..., S_k$, donde $S_i \sim S_{i+1}$ para $i = 1, 2, ..., k-1$ y S_k es final.

Teorema

En el modelo While-Loop, todas las trayectorias (computaciones) en el espacio de estado son siempre finitas, y alcanzan una configuración terminal. Por tanto, todo input tiene un output asociado.

Definición (Función *While* – *Loop*-computable)

Sea un *loop*-programa P. Sean $X_1, X_2, ..., X_k$ sus variables de entrada. Se construye la siguiente configuración inicial:

$$(1, < X_1 = r_1, X_2 = r_2, ..., X_k = r_k, Z_1 = 0, ..., Z_m = 0, Y = 0 >)$$

donde $r_i \in \mathbb{N}$ para $1 \leq i \leq k$. Se comienza la ejecución y entonces:

- Existe una computación $S_1, S_2, ..., S_k$ de P, donde S_k es terminal. Diremos que

$$Y = \delta_P^{(k)}(X_1, X_2, ..., X_k)$$

donde $X_1, X_2, ..., X_k$ son valores concretos y corresponden a $r_1, r_2, ..., r_k$.

Funciones *While* – *Loop*-computables

Definición (Función computable)

Una función $f : \mathbb{N}^k \to \mathbb{N}$ es While-Loop-computable si existe un W-programa P, tal que:

$$f(x_1, x_2, ..., x_k) = \varphi_P^{(k)}(x_1, x_2, ..., x_k), \forall (x_1, x_2, ..., x_k) \in \mathbb{N}^k$$

Ejemplo

La suma y el producto son funciones totalmente computables.

Equivalencia de Modelos

- Estudiadas tres definiciones de computabilidad (L-computabilidad, URM-computabilidad y While-Loop-computabilidad), la pregunta que se plantea es obvia: ¿Qué funciones calculan estos modelos? O dicho de otra forma ¿Tienen todos la misma potencia computacional?
- En general, la potencia potencial de un modelo viene establecida por la «cantidad» de funciones que podemos calcular con él.
- La respuesta a estas preguntas exige estudiar si toda función computable con un modelo lo es también con los demás, utilizando la aproximación de transformaciones polinomiales entres modelos para efectuar estas demostraciones de equivalencia.
- En nuestro caso, nos limitaremos a decir que los modelos L y URM tiene la misma potencia computacional y pueden calcular exactamente el mismo universo de funciones.

Equivalencia de Modelos

- Sin embargo, While-Loop es menos potente, y el conjunto de funciones que puede calcular es más «pequeño». While-Loop únicamente puede calcular funciones computables en sentido total, porque en él, todos los programas paran en cualquier entrada.
- Las funciones calculada por While-Loop pueden calcularse con L y con URM. El recíproco por el contrario, no es cierto.

Referencias

- Davis, Sigal Weyuker: Computability, Complexity and Languages, second edition, Academic Press, 1994.
- Cutland: Computability, An Introduction To Recursive Function Theory. Cambridge University Press, 1980.
- Zucker Pretorius: Introduction To Computability Theory. South African Computer Journal, Number 9, 1993.