Tema 4: Programas universales y forma normal

Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

Ciencias de la Computación
(4º curso, Grado en Matemáticas)
Curso 2021–2022

Programas como datos

- Queremos manejar, de *manera efectiva*, los programas como objetos.
- Para hacerlo *codificamos* los programas mediante números naturales.
- Bajo este tipo de identificación (que permite interpretar los programas como datos de entrada/salida), problemas acerca de la programación se transforman en problemas susceptibles de ser abordados en `GOTO`.
- Utilizaremos las funciones p.r. antes definidas.

Primera etapa: Codificación de las instrucciones

- La codificación de las instrucciones

$$\mathtt{I} \mapsto \#(\mathtt{I})$$

- Información que debemos almacenar para codificar una instrucción...
  - La etiqueta, si está etiquetada.
  - La variable que ocurre en la instrucción.
  - El formato de la instrucción.
- En primer lugar identificamos los símbolos con determinados números naturales:
  - **Variables:** Las enumeramos siguiendo la ordenación

$$\mathtt{V}_1 = \mathtt{Y}, V_2 = \mathtt{X}_1, \mathtt{V}_3 = \mathtt{Z}_1, \mathtt{V}_4 = \mathtt{X}_2, \mathtt{V}_5 = \mathtt{Z}_2, \ldots$$

    es decir, $\#(\mathtt{X}_k) = 2k,\ \#(\mathtt{Y}) = 1,\ \#(\mathtt{Z}_k) = 2k+1$
  - **Etiquetas:** La enumeración será

$$\mathtt{A}_1, \mathtt{B}_1, \mathtt{C}_1, \mathtt{D}_1, \mathtt{E}_1, \mathtt{A}_2, \ldots$$

    - Es decir, $\#(\mathtt{A}_i) = 5(i-1)+1,\ \#(\mathtt{B}_i) = 5(i-1)+2,$ $\#(\mathtt{C}_i) = 5(i-1)+3,\ \#(\mathtt{D}_i) = 5(i-1)+4, \#(\mathtt{E}_i) = 5i$

Códigos de instrucciones

- **Instrucciones:** Sea `I` una instrucción `GOTO` (con etiqueta o no). Entonces el código de `I` se define como

$$\#(I) = \langle a, \langle b, c \rangle \rangle$$

donde

- $\begin{cases} a = 0 & \text{si } \texttt{I} \text{ no tiene etiqueta} \\ a = \#(\texttt{L}) & \text{si } \texttt{I} \text{ tiene como etiqueta } \texttt{L} \end{cases}$
- $c = \#(\texttt{V}) - 1$, donde `V` es la variable que ocurre en `I`.
- b identifica el formato de la instrucción:

Formato	Valor de b
$\texttt{V} \leftarrow \texttt{V}$	0
$\texttt{V} \leftarrow \texttt{V} + 1$	1
$\texttt{V} \leftarrow \texttt{V} - 1$	2
$\texttt{IF V} \neq 0 \texttt{ GOTO L}$	$\#(\texttt{L}) + 2$

Ejemplos:

- $\#(\texttt{[A] X} \leftarrow \texttt{X+1}) = \langle \#(\texttt{A}), \langle 1, \#(\texttt{X}) \dot{-} 1 \rangle \rangle = \langle 1, \langle 1, 1 \rangle \rangle =$
  $= \langle 1, 2^1(2 \cdot 1 + 1) \dot{-} 1 \rangle = \langle 1, 5 \rangle = 2^1 \cdot 11 - 1 = 21$
- Si $\texttt{I} \equiv \texttt{[A] IF Y} \neq 0 \texttt{ GOTO A}$, entonces
  $\#(\texttt{I}) = \langle \#(\texttt{A}), \langle \#(\texttt{L}) + 2, \#(\texttt{Y}) - 1 \rangle \rangle = \langle 1, \langle 3, 0 \rangle \rangle =$
  $\langle 1, 2^3 - 1 \rangle = \langle 1, 7 \rangle = 2 \cdot 15 - 1 = 29$
- ¿Qué instrucción $\texttt{I}$ verifica $\#(\texttt{I}) = 0$?

- ¿Qué instrucción tiene código 24?

Propiedades

Proposición: Todo número natural codifica una única instrucción, es decir

$$(\forall q \in \mathbb{N})(\exists! \mathtt{I})[\#(\mathtt{I}) = q]$$

Como $\langle .,. \rangle : \mathbb{N}^2 \to \mathbb{N}$ es biyectiva, existen $q_1, q_2 \in \mathbb{N}$ únicos tales que

$$\langle q_1, q_2 \rangle = q \quad \text{(nótese que } q_1 = l(q) \text{ y } q_2 = r(q))$$

- Si $q_1 = 0$, entonces la instrucción no lleva etiqueta.
- Si $q_1 \neq 0$, lleva como etiqueta $\mathtt{L}$ si $\#(\mathtt{L}) = q_1$.
- Existen q_3, q_4 únicos tales que $\langle q_3, q_4 \rangle = q_2$
  ($q_3 = l(q_2) = l(r(q_1))$ y $q_4 = r(q_2) = r(r(q_2)))$.
  La variable de la instrucción es $\mathtt{V}_i$, donde

$$\#(\mathtt{V}_i) = i = q_4 + 1 = r(q_2) = r(r(q)) + 1$$

y la instrucción es

q_3	Instrucción
0	$\mathtt{V}_i \leftarrow \mathtt{V}_i$
1	$\mathtt{V}_i \leftarrow \mathtt{V}_i + 1$
2	$\mathtt{V}_i \leftarrow \mathtt{V}_i - 1$
> 2	$\mathtt{IF}\ \mathtt{V}_i \neq 0\ \mathtt{GOTO}\ \mathtt{L}_j, \quad \#(\mathtt{L}_j) = q_3 - 2$

Codificando programas

- Utilizamos la codificación de Gödel para codificar el programa $\mathtt{p} \equiv \mathtt{I}_1, \ldots, \mathtt{I}_n$ como

$$\#(\mathtt{p}) = [\#(\mathtt{I}_1), \ldots, \#(\mathtt{I}_n)] - 1$$

- **Ejemplo:** Calculemos el código del programa p que calcula la función vacía:

$$\begin{array}{ll} [\mathtt{A}] & \mathtt{X} \leftarrow \mathtt{X} + 1 \\ & \mathtt{IF\ X} \neq 0 \mathtt{\ GOTO\ A} \end{array}$$

El código de $\mathtt{I}_1$, ya ha sido calculado, es $\#(\mathtt{I}_1) = 21$, y el código de $\mathtt{I}_2$ es

$$\langle 0, \langle \#(\mathtt{A}) + 2, \#(X) - 1 \rangle \rangle = \langle 0, \langle 3, 1 \rangle \rangle = 46$$

por tanto

$$\#(\mathtt{p}) = [21, 46] - 1 = 2^{21} \cdot 3^{46} - 1$$

Observaciones

- Con esta codificación, resulta que

$$\#(\mathtt{Y} \leftarrow \mathtt{Y}) = \langle 0, \langle 0, 0\rangle\rangle = 0$$

y como

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

entonces, si $\mathtt{p} \equiv \mathtt{I}_1, \ldots, \mathtt{I}_n$, se verifica

$$[\#(\mathtt{I}_1), \ldots, \#(\mathtt{I}_n), \#(\mathtt{Y} \leftarrow \mathtt{Y}), \ldots, \#(\mathtt{Y} \leftarrow \mathtt{Y})] = [\#(I_1), \ldots, \#(I_n)]$$

Por tanto, si admitiésemos que los programas acabando en $\mathtt{Y} \leftarrow \mathtt{Y}$, no sería inyectiva.

- **Teorema:** Para cada $r \in \mathbb{N}$ existe un <u>único</u> programa $\mathtt{p}$ tal que $\#(\mathtt{p}) = r$

Ejemplo

Veamos qué programa $\mathtt{p}$ codifica 35, es decir,

$$\#(\mathtt{p}) = [\#(\mathtt{I}_1), \ldots, \#(\mathtt{I}_n)] - 1 = 35$$

Si $[\mathtt{I}_1, \ldots \mathtt{I}_n] = 36 = 2^2 \cdot 3^2$, entonces $n = 2$, y $\#(I_1) = \#(\mathtt{I}_2)$ (luego $\mathtt{I}_1 = \mathtt{I}_2$). Como

$$2 = \langle 0, \langle 1, 0 \rangle \rangle$$

entonces $\mathtt{I}_1 = \mathtt{Y} \leftarrow \mathtt{Y} + 1$. Por tanto el programa es

$$\mathtt{Y} \leftarrow \mathtt{Y} + 1$$
$$\mathtt{Y} \leftarrow \mathtt{Y} + 1$$

Planteamiento del problema

- **Cuestión:** ¿Es computable la función

$$(\vec{x}, \#(\mathtt{p})) \mapsto [\![\mathtt{p}]\!]^{(n)}(\vec{x})$$

  definida sobre $\mathbb{N}^{n+1}$?

- La respuesta es **sí**: construiremos un programa que la calcula, un **programa universal**.

  Es como una *computadora*, pues sus datos de entrada son un programa $\mathtt{p}$ (su código) y datos de entrada para dicho programa; y su ejecución consiste en

  ejecutar el programa de entrada $\mathtt{p}$ *sobre los datos de entrada*

Programa universal. Consecuencias

Teorema. Para cada $n \geq 1$, existe un programa U_n tal que para todo $\vec{x} \in \mathbb{N}^n$ y todo programa GOTO, $\mathtt{p}$, se tiene

$$[\![\mathtt{U_n}]\!]^{(n+1)}(\vec{x}, \#(\mathtt{p})) = [\![\mathtt{p}]\!]^{(n)}(\vec{x})$$

- La idea que encierra la construcción del programa es la de *intérprete* de un lenguaje en otro. En este caso, en el mismo lenguaje.
- Como consecuencia obtendremos que

$$\mathtt{GCOMP} = \mathcal{P}$$

- Formalización del concepto de *método efectivo sobre programas*, definido a través de la codificación, y que nos permitirá describir distintas técnicas de *transformación* y *obtención automática* de programas,

Descripción del programa universal

Esencialmente, el programa universal es un **intérprete** de GOTO, que simula la ejecución del programa P sobre la tupla $\vec{x}$ que recibe como entrada. Para describir la computación de P sobre $\vec{x}$ codificaremos numéricamente las descripciones instantáneas.

Codificación de descripciones instantáneas.

- **Estados**: Dado el estado $\sigma = \{V_1 = r_1, V_2 = r_2, ..., V_n = r_n\}$, con $\#(V_i) = i$, definimos el código de σ así:

$$\#(\sigma) = [r_1, ..., r_n]$$

- **Descripciones instantáneas**: Sea (i, σ) una d.i. Definimos su código como:

$$\#(i, \sigma) = \langle i, \#(\sigma) \rangle$$

El programa universal U_n

$$
\begin{array}{ll}
 & \left.\begin{array}{l} Z \leftarrow X_{n+1} + 1 \\ S \leftarrow \prod_{i=1}^{n} (p_{2i})^{X_i} \\ K \leftarrow 1 \end{array}\right\} (1) \\
[C] & \left.\begin{array}{l} IF\ K = Long(Z) + 1\ GOTO\ F \end{array}\right\} (2) \\
 & \left.\begin{array}{l} U \leftarrow r((Z)_k) \\ P \leftarrow p_{r(U)+1} \\ IF\ l(U) = 0\ GOTO\ N \\ IF\ l(U) = 1\ GOTO\ A \\ IF\ P \nmid S\ GOTO\ N \\ IF\ l(U) = 2\ GOTO\ M \\ K \leftarrow (\mu i)_{\leq Long(Z)}[l((Z)_i) + 2 = l(U)] \\ GOTO\ C \end{array}\right\} (3) \\
[M] & \left.\begin{array}{ll} S \leftarrow qt(S, P) \\ GOTO\ N \\ [A] & S \leftarrow S \cdot P \\ [N] & K \leftarrow K + 1 \\ & GOTO\ C \\ [F] & Y \leftarrow (S)_1 \end{array}\right\} (4)
\end{array}
$$

Descripción de $\mathtt{U}_n$

- **BLOQUE 1**: Iniciación del programa U_n
  - En Z guardamos la información acerca del programa P.
  - En S el estado actual
  - En K la siguiente instrucción que debe ejecutarse.

  (Si $K = k$ y $S = s$, el par (k, s) almacena una d.i.)
- **BLOQUE 2**: Test de salida.
  U_n para cuando lleguemos a una d.i. de P terminal.
  Como la longitud de P es $|P| = Long(Z)$, la instrucción de salida será:

$$IF\ K = Long(Z) + 1\ GOTO\ F$$

  donde se incluye que una instrucción condicional remita a una etiqueta que no exista.
- **BLOQUE 3**: Obtención del tipo de instrucción que debe ejecutarse.
- **BLOQUE 4**: Obtención de la nueva d.i.

El predicado $STEP^{(n)}$ y la función $di^{(n)}$

Con ligeras modificaciones en U_n podemos obtener

- un predicado, $\mathbf{STEP}^{(n)}$, que controla la parada de un programa dado tras un número de pasos
- una función, $\mathbf{di}^{(n)}$, que devuelve, la descripción instantánea alcanzada tras un número dado de pasos.

Proposición. Para cada $n \geq 1$ el predicado:

$\mathbf{STEP}^{(n)}(\vec{x}, y, t) \equiv$ El programa de código y, sobre $\vec{x}$, **para** en, a lo sumo, t pasos.

es *GOTO*–computable.

Proposición. Para cada $n \geq 1$ la función:

$$\mathbf{di}^{(n)}(\vec{x}, y, k) = \begin{cases} \#(s_{j+1}) & \text{si el programa de código } y \text{ para en} \\ & j \leq k \text{ pasos de la computación del} \\ & \text{mismo sobre el dato } \vec{x} \\ \#(s_{k+1}) & \text{e.c.o.c. de dicha computación.} \end{cases}$$

es *GOTO*–computable (siendo $s_1, s_2, \ldots$ la computación del programa de código y sobre $\vec{x}$).

Programa que calcula el predicado $STEP^{(n)}$

$$
\begin{array}{ll}
 & Z \longleftarrow X_{n+1} + 1 \\
 & S \longleftarrow \prod_{i=1}^{n} (p_{2i})^{X_i} \\
 & K \longleftarrow 1 \\
[C] & \textbf{IF Q} > \textbf{X}_{\textbf{n+2}} \textbf{ GOTO E} \\
 & \textbf{Q} \longleftarrow \textbf{Q} + \textbf{1} \\
 & IF\ K = Long(Z) + 1\ GOTO\ F \\
 & U \longleftarrow r((Z)_k) \\
 & P \longleftarrow p_{r(U)+1} \\
 & IF\ l(U) = 0\ GOTO\ N \\
 & IF\ l(U) = 1\ GOTO\ A \\
 & IF\ P \nmid S\ GOTO\ N \\
 & IF\ l(U) = 2\ GOTO\ M \\
 & K \longleftarrow (\mu i)_{\leq Long(Z)}[l((Z)_i) + 2 = l(U)] \\
 & GOTO\ C \\
[M] & S \longleftarrow qt(S, P) \\
 & GOTO\ N \\
[A] & S \longleftarrow S \cdot P \\
[N] & K \longleftarrow K + 1 \\
 & GOTO\ C \\
[F] & \textbf{Y} \longleftarrow \textbf{1}
\end{array}
$$

(**Recuento**) (4) *rutinas*

Nota: Obsérvese que, en el Programa Universal, se ha añadido un bloque de recuento y se ha modificado convenientemente la última instrucción.

Programa que calcula la función $di^{(n)}$

$$
\begin{array}{ll}
 & Z \longleftarrow X_{n+1} + 1 \\
 & S \longleftarrow \prod_{i=1}^{n} (p_{2i})^{X_i} \\
 & K \longleftarrow 1 \\
[C] & \textbf{IF Q} \geq \textbf{X}_{\textbf{n+2}} \textbf{ GOTO F} \\
 & \textbf{Q} \longleftarrow \textbf{Q} + \textbf{1} \\
 & IF\ K = Long(Z) + 1\ GOTO\ F \\
 & U \longleftarrow r((Z)_k) \\
 & P \longleftarrow p_{r(U)+1} \\
 & IF\ l(U) = 0\ GOTO\ N \\
 & IF\ l(U) = 1\ GOTO\ A \\
 & IF\ P \nmid S\ GOTO\ N \\
 & IF\ l(U) = 2\ GOTO\ M \\
 & K \longleftarrow (\mu i)_{\leq Long(Z)}[l((Z)_i) + 2 = l(U)] \\
 & GOTO\ C \\
[M] & S \longleftarrow qt(S, P) \\
 & GOTO\ N \\
[A] & S \longleftarrow S \cdot P \\
[N] & K \longleftarrow K + 1 \\
 & GOTO\ C \\
[F] & \textbf{Y} \longleftarrow \langle \textbf{K}, \textbf{S} \rangle
\end{array}
$$

(**Recuento**) (4) *rutinas*

Nota: Obsérvese que, de nuevo en el Programa Universal, se ha añadido un bloque de recuento y se ha modificado convenientemente la última instrucción.

Descripción $\mathcal{PR}$ de la computaciones

Codifiquemos cada d.i. $\#(s) = (i, \sigma)$, mediante $y = \langle i, \#(\sigma)\rangle$.

Proposición. Para cada $n \geq 1$ el predicado $STEP^{(n)}(x_1, ..., x_n, y, t)$ y la función $di^{(n)}$ son p.r.
La proposición es consecuencia del siguiente resultado:

Lema. Las siguientes funciones son primitivas recursivas:

- La función $suc : \mathbb{N}^2 \to \mathbb{N}$ dada por

$suc(x, y) =$
la d.i. sucesora de x en la computación del programa y.

- La función de iniciación, $inic^{(n)} : \mathbb{N}^n \to \mathbb{N}$ dada por: $inic^{(n)}(x_1, ..., x_n) = \langle 1, \prod_{j=1}^{n}(p_{2j})^{x_i}\rangle$
- La función $di^{(n)} : \mathbb{N}^{n+2} \to \mathbb{N}$ dada por:

$$\begin{cases} di^{(n)}(x_1, ..., x_n, y, 0) & = inic^{(n)}(x_1, ...x_n) \\ di^{(n)}(x_1, ..., x_n, y, k+1) & = suc(di^{(n)}(x_1, ..., x_n, y, k), y) \end{cases}$$

Teorema de la forma normal de Kleene

Teorema de la forma normal. Para cada $n \geq 1$, existen

- Un predicado $(n+2)$-ario, primitivo recursivo, $\mathcal{T}_n$, y
- Una función L primitiva recursiva.

tales que, para cada función, f, n-aria y G-computable existe un $e \in \mathbb{N}$ verificando:

1. $f(\vec{x}) \downarrow \Leftrightarrow \exists z \mathcal{T}_n(\vec{x}, e, z)$
2. $f(\vec{x}) = L((\mu z)\mathcal{T}_n(\vec{x}, e, z))$

Demostración: Dada $f \in GCOMP^{(n)}$, existe $P \in GOTO_P$, $[\![P]\!]^{(n)} = f$. Sea $e = \#(P)$. Entonces

$$f(\vec{x}) \downarrow \Leftrightarrow [\![P]\!]^{(n)}(\vec{x}) \downarrow \Leftrightarrow \exists t\, STEP^{(n)}(\vec{x}, e, t)$$

Definamos el predicado (n+2)-ario, $\mathcal{T}_n(\vec{x}, e, z)$ como:

$$STEP^{(n)}(\vec{x}, e, r(z)) \wedge (r(di^{(n)}(\vec{x}, e, r(z))))_1 = l(z)$$

donde $z = \langle y, t \rangle$ es la codificación del valor almacenado en Y $(y = l(z))$ y el número de pasos $(t = r(z))$ de la computación de P para la entrada $\vec{x}$.

Enumeración efectiva

- Para $n \geq 1$ y $e \in \mathbb{N}$,
  la función recursiva n-aria de índice e es la función $\varphi_e^{(n)} : \mathbb{N}^n - \longrightarrow \mathbb{N}$, dada por

$$\varphi_e^{(n)}(\vec{x}) = \mathcal{U}_n(\vec{x}, e) = L(\mu z \mathcal{T}_n(\vec{x}, e, z))$$

  Obsérvese que:
  - $\varphi_{\#(P)}^{(n)} = [\![P]\!]^{(n)}$.

Proposición. Sea $n \geq 1$. La sucesión $\{\varphi_e^{(n)} : e \in \mathbb{N}\}$ es una enumeración efectiva de $\mathcal{P}^{(n)}$, es decir:

1. Para todo $e \in \mathbb{N}$, $\varphi_e^{(n)} \in \mathcal{P}^{(n)}$
2. Si $f \in \mathcal{P}^{(n)}$, entonces existe $e \in \mathbb{N}$ tal que $f = \varphi_e^{(n)}(\vec{x})$.
3. Existe $F_n \in \mathcal{P}^{(n+1)}$ tal que para cada $\vec{x} \in \mathbb{N}^n, e \in \mathbb{N}$,

$$F_n(\vec{x}, e) = \varphi_e^{(n)}(\vec{x})$$

Función universal

Teorema de la función Universal.

Existe una función $\mathcal{U} \in GCOMP^{(2)}$ tal que para cada $n \geq 1$ y cada $f \in GCOMP^{(n)}$, existe $e \in \mathbb{N}$:

$$\forall \vec{x} \in \mathbb{N}^n (f(\vec{x}) = \mathcal{U}([x_1, ..., x_n], e))$$

- $\mathcal{U}$ es la función que calcula el programa universal U, diseñado como U_n pero cambiando el bloque de iniciación, $S \leftarrow \prod_{i=1}^{n} p_{2i}^{x_i}$ por

$$S \leftarrow \prod_{i=1}^{Long(x)} p_{2i}^{(x)_i}$$

La tesis de Church–Turing

- **Teorema**. Toda función GOTO-computable es recursiva.
- **Corolario**. La clase de las funciones GOTO–computables coincide con la clase de las funciones recursivas.

Estos resultados (junto con muchos otros de naturaleza similar) apoyan la validez de la siguiente hipótesis:

Tesis de Church–Turing. La clase de las funciones numéricas que pueden calcularse mediante algún algoritmo es precisamente la clase de las funciones recursivas (o GOTO–computables).