Lenguajes Formales, Autómatas y Computabilidad

Teórica:
teoremas fundamentales de computabilidad

Primer Cuatrimestre 2025

Bibliografía

Introduction to Automata Theory, Languages and Computation, J. Hopcroft, R. Motwani, J. Ullman, Second Edition, Addison Wesley, 2001. Capítulo 8 y 9.

Computability, Complexity, and Languages: Fundamentals of Theoretical ... By Martin Davis, Ron Sigal, Elaine J. Weyuker, Second Ediiton, Morgan Kaufmann, 1994

Tesis de Church

Hay muchos modelos de cómputo.
Está probado que tienen el mismo poder que $\mathcal{S}$

- C
- Java
- Haskell
- máquinas de Turing
- ...

Tesis de Church. Todos los algoritmos para computar en los naturales se pueden programar en $\mathcal{S}$.

Entonces, el problema de la detención dice

no hay algoritmo para decidir la verdad o falsedad de HALT(x, y)

Universalidad

Para cada $n > 0$ definimos

$$\begin{aligned} \Phi^{(n)}(x_1, \ldots, x_n, e) &= \text{salida del programa } e \text{ con entrada } x_1, \ldots, x_n \\ &= \Psi_P^{(n)}(x_1, \ldots, x_n) \qquad \text{donde } \#(P) = e \end{aligned}$$

Teorema

Para cada $n > 0$ la función $\Phi^{(n)}$ es parcial computable.

Observar que el programa para $\Phi^{(n)}$ es un intérprete de programas. Se trata de un programa que interpreta programas (representados por números).

Para demostrar el teorema, construimos el programa U_n que computa $\Phi^{(n)}$.

Idea de U_n

U_n es un programa que computa

$$\begin{array}{rcl}\Phi^{(n)}(x_1,\dots,x_n,e) & = & \text{salida del programa } e \text{ con entrada } x_1,\dots,x_n \\ & = & \Psi_P^{(n)}(x_1,\dots,x_n) \qquad \text{donde } \#(P)=e\end{array}$$

U_n necesita

- averiguar quién es P (decodifica e)
- llevar cuenta de los estados de P en cada paso
    - parte del estado inicial de P cuando la entrada es $x_1,\dots,x_n$
    - codifica los estados con listas
    - por ejemplo $Y=0, X_1=2, X_2=1$ lo codifica como $[0,2,0,1]=63$

En el código de U_n

- K indica el número de instrucción de P que se está por ejecutar
- S describe el estado de P en cada momento

Inicialización

// entrada $= x_1, \ldots, x_n, e$

// $\#(P) = e = [i_1, \ldots, i_m] - 1$

$Z \leftarrow X_{n+1} + 1$

// $Z = [i_1, \ldots, i_m]$

$$S \leftarrow \prod_{j=1}^{n} (p_{2j})^{X_j}$$

// $S = [0, X_1, 0, X_2, \ldots, 0, X_n]$ es el estado inicial

$K \leftarrow 1$

// la primera instrucción de P a analizar es la 1

Caso $V++$

// S codifica el estado, K es el número de instrucción
// $Z = [i_1, \ldots, i_m], i_K = \langle a, b \rangle + 1,$
// R es el primo para la variable V que aparece en i_K
// se trata de $V++$
$S \leftarrow S \cdot R$
// $S =$ nuevo estado de P (suma 1 a V)
$K \leftarrow K + 1$

Caso $V--$

// S codifica el estado, K es el número de instrucción

// $Z = [i_1, \ldots, i_m], i_K = \langle a, b \rangle + 1$

// R es el primo para la variable V que aparece en i_K

// se trata de $V--$ con $V \neq 0$

$S \leftarrow S$ div R

// S=nuevo estado de P (resta 1 a V)

$K \leftarrow K + 1$

Caso **while**

Sea $g(x, n)$ es el código de programa x adaptado para recibir una secuencia de $2n + 1$ variables y dar una secuencia de $2n + 1$ variables.

// S codifica el estado, K es el número de instrucción
// $Z = [i_1, \ldots, i_m], i_K = \langle a, b \rangle + 1$
// se trata de **while**
while $(S[\ell(i_K)] \neq 0)$ **do** $\{$
  $S \leftarrow \Phi(S, g(r(i_K), n))$
// S=nuevo estado de P dentro del ciclo
$\}$
// S es el nuevo estado de P despues del ciclo
$K \leftarrow K + 1$

Caso **pass**

$//$ S codifica el estado, K es el número de instrucción
$//$ $Z = [i_1, \ldots, i_m], i_K = 0$
$//$ la instrucción no cambia el estado
$K \leftarrow K + 1$

Devolución del resultado

// S codifica el estado final de P
// sale del ciclo principal
$Y \leftarrow S[1]$
// $Y =$ el valor que toma la variable Y de P al terminar

$\Phi(x_1, \ldots, x_n, e) = \Psi_P(x_1, \ldots x_n)$, donde $e = \#(P)$

$Z \leftarrow X_{n+1} + 1$

$S \leftarrow \prod_{i=1}^{n} (primo_{2i})^{X_i}$

$K \leftarrow 1$

while $(0 \leq K \leq |Z|)$ **do** $\{$

  $I \leftarrow Z[K]$

  if $(I \neq 0)\{$

    if $(r(I) = 0)\{S \leftarrow S\ primo(\ell(I))\}$ suma

    if $(r(I) = 1)\{S \leftarrow \max(1, S/primo(\ell(I)))\ \}$ resta

    if $(r(I) \geq 2)\{$

      while $(S[\ell(I)] \neq 0)$ **do** $\{S \leftarrow \Phi(S, g(r(I), n))\}\}$

$g(x, n)$ es el código de programa x adaptado para recibir una secuencia de 2n+1 variables y dar una secuencia de 2n+1 variables

  $K++$

$\}$

$Y \leftarrow S[1]$

Step Counter

Definimos

$\mathsf{STP}^{(n)}(x_1, \ldots, x_n, e, t)$ sii el programa e termina en t o menos pasos con entrada $x_1, \ldots, x_n$

sii hay un cómputo del programa e de a lo sumo $t+1$ configuraciones, comenzando con la entrada $x_1, \ldots, x_n$

Teorema

Para cada $n > 0$, el predicado $\mathsf{STP}^{(n)}(x_1, \ldots, x_n, e, t)$ *es total computable.*

Snapshot

Definimos

$$\mathsf{SNAP}^{(n)}(x_1, \ldots, x_n, e, t) \quad = \quad \begin{array}{l}\text{representación de la configuración} \\ \text{instantánea del programa } e \\ \text{con entrada } x_1, \ldots, x_n \text{ en el paso } t\end{array}$$

La configuración instantánea se representa como

$$\langle \text{número de instrucción}, \text{lista representando el estado} \rangle$$

Teorema

Para cada $n > 0$, la función $\mathsf{SNAP}^{(n)}(x_1, \ldots, x_n, e, t)$ *es total comptable*

Ahora demostraremos que toda función total o parcial computable puede expresarse mediante la función universal y la minimización no acotada.

Teorema de la Forma Normal

Teorema

Sea $f : \mathbb{N}^n \to \mathbb{N}$ una función parcial computable. Entonces existe un predicado total computable $R : \mathbb{N}^{n+1} \to \mathbb{N}$ tal que

$$f(x_1, \ldots, x_n) = l\left(\min_z R(x_1, \ldots, x_n, z)\right)$$

Demostración.

Sea e el número de algún programa para $f(x_1, \ldots, x_n)$.
El predicado $R(x_1, \ldots, x_n, z)$ es

$$\big(z = (y,t)\big) \wedge \mathsf{STP}^{(n)}(x_1, \ldots, x_n, e, t) \wedge \Big(y = \underbrace{\underbrace{r\left(\mathsf{SNAP}^{(n)}(x_1, \ldots, x_n, e, t)\right)}_{\text{estado final de } e \text{ con entrada } x_1, \ldots, x_n}[1]}_{\text{valor de la variable } Y \text{ en ese estado final}}\Big)$$

Recordar que la configuración instantánea se representa como
$\langle$número de instrucción, lista representando el estado$\rangle$

$\square$

Hacia Teorema del Parametro: Eliminando variables de entrada

Consideremos un programa P que usa la entrada X_1 y X_2:

INSTRUCCIÓN 1 $\#(I_1)$

$\vdots$

INSTRUCCIÓN k $\#(I_k)$

Computa la función $f : \mathbb{N}^2 \to \mathbb{N}$

$$f(x, y) = \Psi_P^{(2)}(x, y)$$

$$\#(P) = [\#(I_1), \ldots, \#(I_k)] - 1$$

Busco número de programa P_0 para $f_0 : \mathbb{N} \to \mathbb{N}, f_0(x) = f(x, 0)$

while $(X_2 \neq 0)$ **do** $\{ X_2 - - \}$ α

INSTRUCCIóN 1 $\#(I_1)$

$\vdots$

INSTRUCCIóN k $\#(I_k)$

Computa la función $f_0 : \mathbb{N} \to \mathbb{N}$

$$f_0(x) = \Psi_{P_0}^{(1)}(x)$$

$$\#(P_0) = [\alpha, \#(I_1), \ldots, \#(I_k)] - 1$$

Hacia Teorema del Parametro: Eliminando variables de entrada

Busco número de programa P_1 para $f_1 : \mathbb{N} \to \mathbb{N}, f_1(x) = f(x, 1)$

while $(X_2 \neq 0)$ **do** $\{ X_2 - - \}$	α	Computa la función $f_1 : \mathbb{N} \to \mathbb{N}$
$X_2 + +$	β	
NSTRUCCIÓN 1	$\#(I_1)$	$f_1(x) = \Psi^{(1)}_{P_1}(x)$
$\vdots$		$\#(P_1) =$
INSTRUCCIÓN k	$\#(I_k)$	$[\alpha, \beta, \#(I_1), \ldots, \#(I_k)] - 1$

Busco número de programa P_2 para $f_2 : \mathbb{N} \to \mathbb{N}, f_2(x) = f(x, 2)$

while $(X_2 \neq 0)$ **do** $\{ X_2 - - \}$	α	Computa la función $f_2 : \mathbb{N} \to \mathbb{N}$
$X_2 + +$	β	
$X_2 + +$	β	$f_2(x) = \Psi^{(1)}_{P_2}(x)$
INSTRUCCIÓN 1	$\#(I_1)$	
$\vdots$		$\#(P_2) =$
INSTRUCCIÓN k	$\#(I_k)$	$[\alpha, \beta, \beta, \#(I_1), \ldots, \#(I_k)] - 1$

Teorema del Parámetro

Hay un programa P_{x_2} para la función $f_{x_2}(x_1) = f(x_1, x_2)$

La transformación $(x_2, \#(P)) \mapsto \#(P_{x_2})$ es total computable es decir, existe una función $S : \mathbb{N}^2 \to \mathbb{N}$ total computable tal que dado x_2 e $y = \#(P)$ calcula $\#(P_{x_2})$:

$$S(x_2, y) = \left(2^\alpha \cdot \prod_{j=1}^{x_2} p_{j+1}^{\beta} \cdot \prod_{j=1}^{|y+1|} {p_{j+x_2+1}}^{(y+1)[j]} \right) - 1$$

Teorema

Hay una función tatl computable $S : \mathbb{N}^2 \to \mathbb{N}$ tal que

$$\Phi_y^{(2)}(x_1, x_2) = \Phi_{S(x_2,y)}^{(1)}(x_1).$$

Teorema

Para cada $n, m > 0$ hay una función total computable inyectiva $S_m^n : \mathbb{N}^{n+1} \to \mathbb{N}$ tal que

$$\Phi_y^{(n+m)}(x_1, \ldots, x_m, u_1, \ldots, u_n) = \Phi_{S_m^n(u_1,\ldots,u_n,y)}^{(m)}(x_1, \ldots, x_m)$$

Programas autoreferentes

- en la demostración del Halting Problem construimos un programa P que, cuando se ejecuta con su mismo número de programa (i.e. $\#(P)$), evidencia una contradicción
- en general, los programas pueden dar por supuesto que conocen su mismo número de programa
- pero si un programa P conoce su número de programa, podría, por ejemplo, devolver su mismo número, es decir $\#(P)$

Hacia el Teorema de la Recursión

Permite definir funciones recursivamente sin caer en contradicciones.
Para toda $g : \mathbb{N}^{n+1} \to \mathbb{N}$ es parcial computable existe un e tal que

$$\Phi_e^{(n)}(x_1, \ldots, x_n) = g(e, x_1, \ldots, x_n)$$

Ejemplo Definicón recursiva de $\mathsf{fact}(x)$

$$\mathsf{fact}(x) = \begin{cases} 1 & \text{si } x = 0, \\ x \cdot \mathsf{fact}(x-1) & \text{si } x > 0. \end{cases}$$

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

$$g(i, x) = \begin{cases} 1 & \text{si } x = 0, \\ x \cdot \Phi_i^{(1)}(x-1) & \text{si } x > 0. \end{cases}$$

Por el Teorema de la Recursión, para toda $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ existe e tal que $\Phi_e^{(1)}(x) = g(e, x)$, es decir:

$$\Phi_e^{(1)}(x) = g(e, x) = \begin{cases} 1 & \text{si } x = 0, \\ x \cdot \Phi_e^{(1)}(x-1) & \text{si } x > 0. \end{cases}$$

Por lo tanto, $\Phi_e^{(1)}$ es precisamente la función fact.

Teorema de la Recursión

Teorema

Para toda $g : \mathbb{N}^{n+1} \to \mathbb{N}$ *parcial computable existe un* e *tal que*

$$\Phi_e^{(n)}(x_1, \ldots, x_n) = g(e, x_1, \ldots, x_n).$$

Demostración.

Sea S_n^1 la función del Teorema del Parámetro:

$$\Phi_y^{(n+1)}(x_1, \ldots, x_n, u) = \Phi_{S_n^1(u,y)}^{(n)}(x_1, \ldots, x_n).$$

La función $(x_1, \ldots, x_n, v) \mapsto g(S_n^1(v,v), x_1, \ldots, x_n)$ es parcial computable, de modo que existe d tal que

$$\begin{aligned} g(S_n^1(v,v), x_1, \ldots, x_n) &= \Phi_d^{(n+1)}(x_1, \ldots, x_n, v) \\ &= \Phi_{S_n^1(v,d)}^{(n)}(x_1, \ldots, x_n) \end{aligned}$$

d está fijo; v es variable. Elegimos $v = d$ y $e = S_n^1(d,d)$. □

Teorema de la Recursión

Corolario

Si $g : \mathbb{N}^{n+1} \to \mathbb{N}$ *es parcial computable, existen infinitos* e *tal que*

$$\Phi_e^{(n)}(x_1, \ldots, x_n) = g(e, x_1, \ldots, x_n)$$

Demostración.

En la demostración del teorema anterior, existen infinitos d tal que

$$\Phi_d^{(n+1)} = g(S_n^1(v, v), x_1, \ldots, x_n).$$

$v \mapsto S_n^1(v, v)$ es inyectiva de modo que existen infinitos

$$e = S_n^1(d, d).$$

$\square$

Quines

Un quine es un programa que cuando se ejecuta, devuelve como salida el mismo programa.

Por ejemplo:

```
char*f="char*f=%c%s%c;main()
{printf(f,34,f,34,10);}%c";
main(){printf(f,34,f,34,10);}
```

Quines

¿Existe e tal que $\Phi_e(x) = e$?

Sí, el programa vacío tiene numero 0 y computa la función constante 0, i.e. $\Phi_0(x) = 0$.

Proposición

Hay infinitos e tal que $\Phi_e(x) = e$.

Demostración.

Considerar la función $g : \mathbb{N}^2 \to \mathbb{N}, g(z, x) = z$.
Aplicando el Teorema de la Recursión, existen infinitos e tal que

$$\Phi_e(x) = g(e, x) = e.$$

$\square$

Quines

No hay nada especial con que la salida del programa sea su propio número en el resultado anterior. Funciona para cualquier h parcial computable.

¿Existe e tal que $\Phi_e(x) = h(e)$?

Proposición

Hay infinitos e tal que $\Phi_e(x) = h(e)$.

Demostración.

Considerar la función $g : \mathbb{N}^2 \to \mathbb{N}, g(z, x) = h(z)$.
Aplicando el Teorema de la Recursión, existen infinitos e tal que

$$\Phi_e(x) = g(e, x) = h(e).$$

$\square$

Teorema del Punto Fijo

Teorema

Si $f : \mathbb{N} \to \mathbb{N}$ *es computable, existe un* e *tal que* $\Phi_{f(e)} = \Phi_e$*.*

Demostración.

Considerar la función $g : \mathbb{N}^2 \to \mathbb{N}$,

$$g(z, x) = \Phi_{f(z)}(x).$$

Aplicando el Teorema de la Recursión, existe un e tal que para todo x,

$$\Phi_e(x) = g(e, x) = \Phi_{f(e)}(x)$$

$\square$

Aplicación del Torema de la Recursión

Probar que la función total $f : \mathbb{N} \to \mathbb{N}$ no es computable:

$$f(x) = \begin{cases} 1 & \Phi_x \text{ es total} \\ 0 & \text{si no} \end{cases}$$

Supongamos f computable. Puedo definir el siguiente programa P:

while $(f(X) = 1)$ **do** $\{$**pass** $\}$

Por lo tanto

$$\Psi_P^{(2)}(x, y) = g(x, y) = \begin{cases} \uparrow & \Phi_x \text{ es total} \\ 0 & \text{si no} \end{cases}$$

es parcial computable. Por el Teorema de la Recursión, sea e tal que $\Phi_e(y) = g(e, y)$.

- Φ_e es total $\Rightarrow g(e, y) \uparrow$ para todo $y \Rightarrow \Phi_e(y) \uparrow$ para todo $y \Rightarrow \Phi_e$ no es total
- Φ_e no es total $\Rightarrow g(e, y) = 0$ para todo $y \Rightarrow \Phi_e(y) = 0$ para todo $y \Rightarrow \Phi_e$ es total