

Resumen Computabilidad

Maquinas de Turing

Componentes

- **Cinta**
 - Celdas, infinita en ambas direcciones, cada celda contiene un símbolo de un alfabeto dado Σ .
 - $*$ $\in \Sigma$
 - $L, R \notin \Sigma$
 - $*$ representa el blanco en una celda
- **Una cabeza**
 - Lee y escribe un símbolo a la vez
 - Se mueve una posición a la izquierda o una posición a la derecha
- **Una tabla finita de instrucciones**
 - Dice qué hacer en cada paso
 - Σ es el alfabeto, Q es el conjunto finito de estados, $A = \Sigma \cup \{L, R\}$ es el conjunto de acciones
 - Una **tabla de instrucciones** T es un subconjunto (finito) de $Q \times \Sigma \times A \times Q$
 - La tupla $(q, s, a, q') \in T$ se interpreta como
Si la máquina está en el estado q leyendo en la cinta el símbolo s , entonces realiza la acción a y pasa al estado q'
 - $q_0 \in Q$ es el estado inicial, $q_f \in Q$ es el estado final
 - cuando no hay restricciones sobre T decimos que M es una **máquina de Turing no determinística**
 - cuando no hay dos instrucciones en T que empiezan con las mismas primeras dos coordenadas, decimos que M es una **máquina de Turing determinística**

Funciones parciales

Una función parcial f es una función que puede estar indefinida para algunos (tal vez ninguno; tal vez todos) sus argumentos.

- notamos $f(x_1, \dots, x_n) \downarrow$ cuando f está definida para x_1, \dots, x_n .
- notamos $f(x_1, \dots, x_n) \uparrow$ cuando f está indefinida para x_1, \dots, x_n

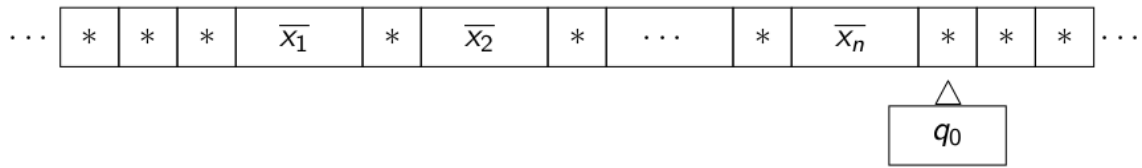
El conjunto de argumentos para los que f está definida se llama dominio de f , notado $\text{dom}(f)$.

$$\text{dom}(f) = \{(x_1, \dots, x_n) : f(x_1, \dots, x_n) \downarrow\}$$

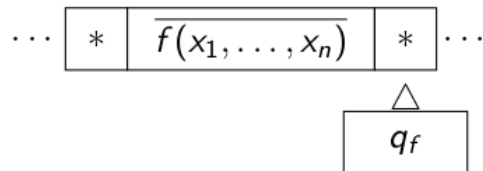
f es **total** si $\text{dom}(f) = \mathbb{N}^n$

C  mputo de funciones parciales en m  quinas de Turing

Una funci  n parcial $f : \mathbb{N}^n \rightarrow \mathbb{N}$ es Turing computable si existe una m  quina de Turing determin  stica $M = (\Sigma, Q, T, q_0, q_f)$ con $\Sigma = \{*, 1\}$ tal que cuando empieza en la configuraci  n inicial



- si $f(x_1, \dots, x_n) \downarrow$ entonces siguiendo sus instrucciones en T llega a una configuraci  n final de la forma



- si $f(x_1, \dots, x_n) \uparrow$ entonces nunca termina en el estado q_f .

Poder de computo

Teorema: Sea $f : \mathbb{N}^n \rightarrow \mathbb{N}$ una funci  n parcial. Son equivalentes

f es computable en Java, C, Haskell, Turing

Funciones computables

Otra manera de formalizar la idea de funci  n calculable de manera efectiva:

- empezar por funciones muy simples, efectivas intuitivamente
- si mezclamos de alguna manera efectiva dos o m  s funciones que ya eran efectivas, entonces obtenemos una funci  n calculable de manera efectiva

Las siguientes funciones se llaman iniciales:

- $s(x) = x + 1$
- $n(x) = 0$
- proyecciones: $u_i^n(x_1, \dots, x_n) = x_i$ para $i \in \{1, \dots, n\}$

Composici  n y recursi  n primitiva

Sea $f : \mathbb{N}^k \rightarrow \mathbb{N}$ y $g_1, \dots, g_k : \mathbb{N}^n \rightarrow \mathbb{N}$. $h : \mathbb{N}^n \rightarrow \mathbb{N}$ se obtiene a partir de f y g_1, \dots, g_k por composici  n si

- $h(x_1, \dots, x_n, 0) = f(x_1, \dots, x_n)$
- $h(x_1, \dots, x_n, t+1) = g(h(x_1, \dots, x_n, t), x_1, \dots, x_n, t)$

Clases PRC

Una clase **C** de funciones totales es **PRC** (primitive recursive closed) si

1. las funciones iniciales est  n en **C**

2. si una función f se obtiene a partir de otras pertenecientes a \mathbf{C} por medio de composición o recursión primitiva, entonces f también está en \mathbf{C}

Teorema: La clase de funciones totales Turing computables es una clase PRC.

Funciones primitivas recursivas

Una función es **primitiva recursiva (p.r.)** si se puede obtener a partir de las funciones iniciales por un número finito de aplicaciones de composición y recursión primitiva.

Teorema: Una función es p.r. sii pertenece a toda clase PRC.

Corolario: Toda función p.r. es total y Turing computable.

Predicados primitivos recursivos

Los **predicados** son funciones que toman valores en $\{0, 1\}$.

Los **predicados p.r.** son aquellos representados por funciones p.r. en $\{0, 1\}$.

Operadores lógicos

Teorema: Sea \mathbf{C} una clase PRC. Si p y q son predicados en \mathbf{C} entonces $\neg p$, $p \wedge q$ y $p \vee q$ están en \mathbf{C} .

Corolario: Si p y q son predicados p.r., entonces también lo son los predicados $\neg p$, $p \vee q$ y $p \wedge q$

Corolario: Si p y q son predicados totales Turing computables entonces también lo son los predicados $\neg p$, $p \vee q$ y $p \wedge q$

Definición por casos (2)

Teorema: Sea \mathbf{C} una clase PRC. Sean $h, g: \mathbb{N}^n \rightarrow \mathbb{N}$ funciones en \mathbf{C} y sea $p: \mathbb{N}^n \rightarrow \{0, 1\}$ un predicado en \mathbf{C} .

La siguiente función está en \mathbf{C}

$$f(x_1, \dots, x_n) = \begin{cases} g(x_1, \dots, x_n) & \text{si } p(x_1, \dots, x_n) \\ h(x_1, \dots, x_n) & \text{si no} \end{cases}$$

Definición por casos (m+1)

Teorema: Sea \mathbf{C} una clase PRC. Sean $g_1, \dots, g_m, h: \mathbb{N}^n \rightarrow \mathbb{N}$ funciones en \mathbf{C} y sean $p_1, \dots, p_m: \mathbb{N}^n \rightarrow \{0, 1\}$ predicados mutuamente excluyentes en \mathbf{C} .

La siguiente función está en \mathbf{C}

$$f(x_1, \dots, x_n) = \begin{cases} g_1(x_1, \dots, x_n) & \text{si } p_1(x_1, \dots, x_n) \\ \vdots & \\ g_m(x_1, \dots, x_n) & \text{si } p_m(x_1, \dots, x_n) \\ h(x_1, \dots, x_n) & \text{si no} \end{cases}$$

Recursión primitiva

$$\begin{aligned} h(x_1, \dots, x_n, 0) &= f(x_1, \dots, x_n) \\ h(x_1, \dots, x_n, t+1) &= g(h(x_1, \dots, x_n, t), x_1, \dots, x_n, t) \end{aligned}$$

- la recursión siempre se hace en el último parámetro
- la función variante de $h(x_1, \dots, x_n, x_{n+1})$ es x_{n+1}

Sumatorias y productorias (desde 0 y 1)

Teorema: Sea C una clase PRC. Si $f: \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ esta en C entonces tambien estan las funciones

$$\begin{aligned} g(y, x_1, \dots, x_n) &= \sum_{t=0}^y f(t, x_1, \dots, x_n) & g(y, x_1, \dots, x_n) &= \sum_{t=1}^y f(t, x_1, \dots, x_n) \\ h(y, x_1, \dots, x_n) &= \prod_{t=0}^y f(t, x_1, \dots, x_n) & h(y, x_1, \dots, x_n) &= \prod_{t=1}^y f(t, x_1, \dots, x_n) \end{aligned}$$

Cuantificadores acotados

Teorema: Sea $p: \mathbb{N}^{n+1} \rightarrow \{0,1\}$ un predicado perteneciente a una clase PRC C . Los siguientes predicados también están en C :

$$\begin{aligned} (\forall t)_{\leq y} p(t, x_1, \dots, x_n) \\ (\exists t)_{\leq y} p(t, x_1, \dots, x_n) \\ (\forall t)_{< y} p(t, x_1, \dots, x_n) \\ (\exists t)_{< y} p(t, x_1, \dots, x_n) \end{aligned}$$

Demostración.

$$\begin{aligned} (\forall t)_{\leq y} p(t, x_1, \dots, x_n) &\text{ sii } \prod_{t=0}^y p(t, x_1, \dots, x_n) = 1 \\ (\exists t)_{\leq y} p(t, x_1, \dots, x_n) &\text{ sii } \sum_{t=0}^y p(t, x_1, \dots, x_n) \neq 0 \end{aligned}$$

- la sumatoria y productoria están en C
- la comparación por $=$ está en C

Demostración.

$$\begin{aligned} (\forall t)_{< y} p(t, x_1, \dots, x_n) &\text{ sii } (\forall t)_{\leq y} (t = y \vee p(t, x_1, \dots, x_n)) \\ (\exists t)_{< y} p(t, x_1, \dots, x_n) &\text{ sii } (\exists t)_{\leq y} (t \neq y \wedge p(t, x_1, \dots, x_n)) \end{aligned}$$

Minimizacion acotada

Sea $p: \mathbb{N}^{n+1} \rightarrow \{0,1\}$ un predicado de una clase PRC C .

$$g(y, x_1, \dots, x_n) = \sum_{u=0}^y \prod_{t=0}^u \alpha(p(t, x_1, \dots, x_n)) \quad \min_{t \leq y} p(t, x_1, \dots, x_n) = \begin{cases} \text{mínimo } t \leq y \text{ tal que} \\ p(t, x_1, \dots, x_n) \text{ es verdadero} & \text{si existe tal } t \\ 0 & \text{si no} \end{cases}$$

Teorema: Sea $p: \mathbb{N}^{n+1} \rightarrow \{0,1\}$ un predicado de una clase PRC C . La funcion $\min_{t \leq y} p(t, x_1, \dots, x_n)$ tmb esta en C

Codificacion de pares

Definimos la función primitiva recursiva $\langle x, y \rangle = 2^x(2 \cdot y + 1) - 1$ (solo tiene una solucion (x,y) que es $= z$)

Observadores de pares

Los observadores del par $z = \langle x, y \rangle$ son $l(z)=x$, $r(z)=y$ y son p.r.

Codificación de secuencias

El número de Gödel de la secuencia a_1, \dots, a_n es el número $[a_1, \dots, a_n] = \prod_{i=1}^n p_i^{a_i}$ donde p_i es el i -ésimo primo.

Teorema: Si $[a_1, \dots, a_n] = [b_1, \dots, b_n]$ entonces $a_i = b_i \forall i$

Observadores: (Son p.r.)

- $x[i] = a_i$
- $|x| = \text{long}(x)$

Lenguaje de programación S

Variables: empiezan inicializadas en 0 y almacenan números naturales.

- *Variables de entrada:* X_1, X_2, \dots
- *Variable de salida:* Y
- *Variables temporales:* Z_1, Z_2, \dots

Instrucciones:

- $V \leftarrow V + 1$
- $V \leftarrow V - 1$ (si V es 0, queda en 0)
- $IF V \neq 0 GOTO A$

Por conveniencia agregamos una más que no hace nada:

- $V \leftarrow V$

Macros: Se pueden simular muchas operaciones a partir de las instrucciones.

Una vez que sepamos que se pueden escribir en el lenguaje S, las usamos como si fueran propias (son **pseudoinstrucciones**).

Estados: Un estado de un programa P es una lista de ecuaciones de la forma $V = m$ (donde V es una variable y m es un número) tal que

- hay una ecuación para cada variable que se usa en P
- no hay dos ecuaciones para la misma variable

Descripción instantánea

Con un programa P de longitud n . Para un estado σ de P y un $i \in \{1, \dots, n+1\}$,

- el par (i, σ) es una **descripción instantánea** de P
- (i, σ) se llama **terminal** si $i = n+1$

Para un (i, σ) no terminal, podemos definir su **sucesor** (j, τ) como:

1. si la i -ésima instrucción de P es $V \leftarrow V + 1$.
 - $j = i + 1$
 - τ es σ , salvo que $V = m$ se reemplaza por $V = m + 1$
2. si la i -ésima instrucción de P es $V \leftarrow V - 1$.

- $j = i + 1$
 - τ es σ , salvo que $V = m$ se reemplaza por $V = \max\{m - 1, 0\}$
3. si la i -ésima instrucción de P es $IF V \neq 0 GOTO L$
- τ es idéntico a σ
 - si σ tiene $V = 0$ entonces $j = i + 1$
 - si σ tiene $V = m$ para $m \neq 0$ entonces
 1. si existe en P una instrucción con etiqueta L entonces $j = \min\{k : k\text{-ésima instrucción de } P \text{ tiene etiqueta } L\}$
 2. si no $j = n + 1$

Cómputos

Un cómputo de un programa P a partir de una descripción instantánea d_1 es una lista d_1, d_2, \dots, d_k de descripciones instantáneas de P tal que

- d_{i+1} es sucesor de d_i para $i \in \{1, 2, \dots, k - 1\}$
- d_k es terminal

Estados y descripciones iniciales

Sea P un programa y sean r_1, \dots, r_m números dados.

- el estado inicial de P para r_1, \dots, r_m es el estado σ_1 , que tiene $X_1 = r_1, \dots, X_m = r_m, Y = 0$ junto con $V = 0$ para cada variable V que aparezca en P y no sea X_1, \dots, X_m, Y
- la descripción inicial de P para r_1, \dots, r_m es $(1, \sigma_1)$

Cómputos a partir del estado inicial

Sea P un programa y sean r_1, \dots, r_m números dados, σ_1 el estado inicial

Dos casos

- hay un cómputo de P d_1, \dots, d_k tal que $d_1 = (1, \sigma_1)$
 Notamos $\psi_P^{(m)}(r_1, \dots, r_m)$ al valor de Y en d_k (esta definido)
- no hay tal cómputo, i.e. existe una secuencia infinita d_1, d_2, d_3, \dots
 donde $d_1 = (1, \sigma_1)$, d_{i+1} es sucesor de d_i
 Decimos que $\psi_P^{(m)}(r_1, \dots, r_m)$ esta indefinido

Funciones computables

Una función (parcial) $f: \mathbb{N}^m \rightarrow \mathbb{N}$ es S-parcial computable (o simplemente parcial computable) si existe un programa P tal que $f(r_1, \dots, r_m) = \psi_P^{(m)}(r_1, \dots, r_m) \forall (r_1, \dots, r_m)$

La **igualdad** (del meta-lenguaje) es verdadera si

- los dos lados están definidos y tienen el mismo valor o
- los dos lados están indefinidos

La función f es **S-computable** (o simplemente computable) si es parcial computable y total.

- *Minimización no acotada*

$$\min_t p(t, x_1, \dots, x_n) = \begin{cases} \text{mínimo } t \text{ tal que} \\ p(t, x_1, \dots, x_n) \text{ es verdadero} & \text{si existe tal } t \\ \uparrow & \text{si no} \end{cases}$$

Teorema: Si $p: \mathbb{N}^{n+1} \rightarrow \{0,1\}$ es un predicado computable entonces $\min_t p(t, x_1, \dots, x_n)$ es parcial computable

- *Clausura por composición*

Teorema: Si h se obtiene a partir de las funciones (parciales) computables f, g_1, \dots, g_k por composición entonces h es (parcial) computable.

- *Clausura por recursión primitiva*

Teorema: Si h se obtiene a partir de g por recursión primitiva y g es computable entonces h es computable.

Teorema: Las funciones computables forman una clase PRC.

Codificación de programas en S

S solo tiene el tipo de datos naturales, pero podemos simular otros tipos a partir de estos.

Codificación de variables y etiquetas

Ordenamos las variables: $Y, X_1, Z_1, X_2, Z_2, \dots$

Y las etiquetas: $A, B, C, \dots, Z, AA, AB, \dots$

Escribimos $\#(W)$ para la posición que ocupa la variable/etiqueta W en la lista

Codificación de instrucciones

Codificamos la instrucción I con $\#(I) = \langle a, \langle b, c \rangle \rangle$ donde

1. si I tiene etiqueta L , entonces $a = \#(L)$; si no $a = 0$
2. si la variable mencionada en I es V entonces $c = \#(V) - 1$
3. si la instrucción I es
 1. $V \leftarrow V$ entonces $b = 0$
 2. $V \leftarrow V + 1$ entonces $b = 1$
 3. $V \leftarrow V - 1$ entonces $b = 2$
 4. IF $V \neq 0$ GOTO L' entonces $b = \#(L') + 2$

Codificación de programas

Un programa P es una lista (finita) de instrucciones $\#(P) = [\#(I_1), \dots, \#(I_k)] - 1$

Ambigüedades

La instrucción final de un programa no puede ser $Y \leftarrow Y$. Con esto, cada número representa a un único programa.

Funciones no computables

Teorema: El conjunto de las funciones (totales) $\mathbb{N} \rightarrow \mathbb{N}$ no es numerable.

\Rightarrow o sea, hay más funciones $\mathbb{N} \rightarrow \mathbb{N}$ que números naturales

- hay tantos programas como números naturales
- hay tantas funciones computables como números naturales
- tiene que haber funciones $\mathbb{N} \rightarrow \mathbb{N}$ no computables

Halting problem

Verdadero sii el programa con número y y entrada x no se indefine:

$$\text{HALT}(x, y) = \begin{cases} 1 & \text{si } \Psi_P^{(1)}(x) \downarrow \\ 0 & \text{si no} \end{cases}$$

Donde P es el unico programa tal que $\#(P) = y$

Tesis de Church

Tesis de Church: Todos los algoritmos para computar en los naturales se pueden programar en S .

Entonces, el problema de la detención dice no hay algoritmo para decidir la verdad o falsedad de $\text{HALT}(x, y)$

Universalidad

Para cada $n > 0$ definimos

$\Phi^{(n)}(x_1, \dots, x_n, e)$ = salida del programa e con entrada x_1, \dots, x_n = $\Psi^{(n)}(x_1, \dots, x_n, e)$ donde $\#(P) = e$

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).

Notación: $\Phi^{(n)}(x_1, \dots, x_n) = \Phi^{(n)}(x_1, \dots, x_n, e)$ (también se puede omitir el superíndice si $n = 1$)

Step counter

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

sii hay un cómputo del programa e de longitud $\leq t + 1$, comenzando con la entrada x_1, \dots, x_n

Teorema: Para cada $n > 0$, el predicado $STP^{(n)}(x_1, \dots, x_n, e, t)$ es p.r.

Snapshot

$SNAP^{(n)}(x_1, \dots, x_n, e, t)$ = representación de la configuración instantánea del programa e con entrada x_1, \dots, x_n en el paso t

Se representa como $\langle \#instruccion, lista representando estado \rangle$

Teorema: Para cada $n > 0$, la función $SNAP^{(n)}(x_1, \dots, x_n, e, t)$ es p.r.

Teorema de la forma normal

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

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

Otra caracterización de funciones computables

Teorema: Una función es parcial computable si se puede obtener a partir de las funciones iniciales por un número finito de aplicaciones de

- composición,
- recursión primitiva y
- minimización

Teorema: Una función es computable si se puede obtener a partir de las funciones iniciales por un número finito de aplicaciones de

- composición,
- recursión primitiva y
- minimización propia

Teorema del parámetro

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

La transformación $(x_2, \#(P)) \rightarrow \#(P_{x_2})$ es p.r., es decir, existe una función $S: \mathbb{N}^2 \rightarrow \mathbb{N}$ p.r. tal que dado x_2 e $y = \#(P)$ calcula $\#(P_{x_2})$:

$$S(x_2, y) = \left(2^{109} \cdot 3^{110} \cdot \prod_{j=1}^{x_2} p_{j+2}^{26} \cdot \prod_{j=1}^{|y+1|} p_{j+x_2+2}^{(y+1)[j]} \right) - 1$$

Teorema: Hay una función p.r. $S: \mathbb{N}^2 \rightarrow \mathbb{N}$ 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 p.r. inyectiva $S_m^n: \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ tal que

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

Teorema de la Recursión

Teorema: Si $g: \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ es parcial computable, existe un e tal que

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

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

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

Quines

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

Proposición: Hay infinitos e tal que $\Phi_e(x) = e$ y $\Phi_e(x) = h(e)$

Teorema del punto fijo

Teorema: Si $f: \mathbb{N} \rightarrow \mathbb{N}$ es computable, existe un e tal que $\Phi_{f(e)}(x) = \Phi(e)$.