Clase práctica

Qué no podemos computar

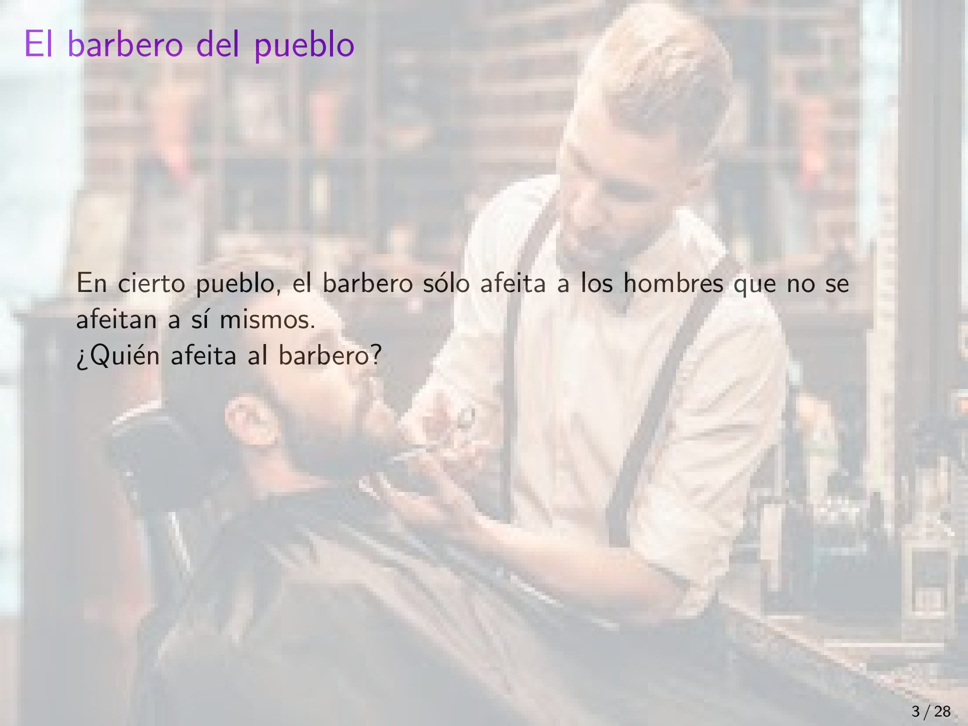
Halting problem y la reducción como estrategia de demostración de no computabilidad

DC - FCEyN - UBA

1er cuatrimestre 2025

Veamos algunos problemas del folklore lógico...

El barbero del pueblo



En cierto pueblo, el barbero sólo afeita a los hombres que no se afeitan a sí mismos.
¿Quién afeita al barbero?

Paradoja

El barbero afeita a un hombre si y sólo si afeita el hombre que no se afeita a sí mismo.

Caso 1: Si el barbero se afeita a sí mismo, como sólo afecta a quien no se afeite a sí mismo, entonces no se afeita a sí mismo ¡Absurdo!

Caso 2: Si el barbero no se afeita a sí mismo, como sólo afecta a quien no se afeite a sí mismo, entonces se afeita a sí mismo ¡Absurdo!

Paradoja de Russell: conjuntos no autocontenidos

Considerar el siguiente conjunto:

$$X = \{x \mid x \notin x\}$$

Surge la pregunta ¿$X \in X$?

¿$X \in X$?

Caso 1: $X \in X \Leftrightarrow_{(\text{definición de } X)} X \notin X$ ¡Absurdo!

Caso 2: $X \notin X \Leftrightarrow_{(\text{definición de } X)} \neg(X \notin X) \Leftrightarrow X \in X$ ¡Absurdo!

Recordando la clase pasada ... Herramientas para demostrar computabilidad

Formas de demostrar que una función f es computable:

1) Argumentar que f es total y mostrar un programa (escrito en lenguaje $\mathcal{S}++$) que la compute.
   Atenti a que no se cuelgue.
2) Expresar f como minimización acotada, o existencial acotado, o para todo acotado, de un predicado computable P.
   Atenti a **acotado** y a que P **sea computable**.

Halting Problem o el problema de terminación

$$HALT(x, y) = \begin{cases} 1 & \text{si el programa } x \text{ evaluado en } y \text{ termina} \\ 0 & \text{si el programa } x \text{ evaluado en } y \text{ se cuelga} \end{cases}$$

Hablemos con propiedad

Recordemos definiciones:

- Dado un programa P, para cada $N \in \mathbb{N}$ se define $\Psi_P^{(n)} : \mathbb{N}^n \to \mathbb{N}$ como la **función computada por P** con n entradas.
- Definimos entonces $\Phi_e^{(n)} : \mathbb{N}^n \to \mathbb{N}$ como la función computada por el programa de número e.

Si P es un programa de número e, entonces:

$$\Psi_P^{(n)} = \Phi_e^{(n)}$$

Halting Problem o el problema de terminación

$$HALT(x, y) = \begin{cases} 1 & si\ \Phi_x(y) \downarrow \\ 0 & \text{en otro caso} \end{cases}$$

Halting Problem o el problema de terminación

¿Es computable $HALT(x, y)$?

No computabilidad de Halting Problem, idea de demo

```
function D(program X):
if HALT(X, X) then
    LoopForever()
else
    Halt()
end if
```

Si *HALT* es computable, también lo es *D*.

¿Qué pasa con $D(D)$?

¿Qué pasa con $D(D)$?

1) Si $HALT(D, D)$, entonces entra en la primera guarda del if y D se cuelga ¡Absurdo!
2) Si $\neg HALT(D, D)$, entonces entra en la cláusula else, y termina ¡Absurdo!

Ahora escribamos la demo formalmente

Demostramos por absurdo. Asumimos que la función $HALT(x, y)$ es computable y llegamos a una contradicción.
Sea P un programa que computa $HALT(x, y)$ (existe tal programa porque asumimos $HALT(x, y)$ computable).
Definimos D el siguiente programa:

P
if $Y == 1$ **then**
  $Y_2 + +$
  while $Y_2 > 0$ **do**
    $Y_2 + +$
  end while
else
  $Y = 1$
end if

¿Qué función computa D?

¿Qué función computa D?

Dado e un número natural,

- $\Psi_D(e)\uparrow$ sii $\Phi_e(e)\downarrow$
- $\Psi_D(e)\downarrow$ sii $\Phi_e(e)\uparrow$

Sea e' la codificación de D, resulta que

$$\Psi_D(e') = \Phi_{e'}(e')\uparrow \text{ sii } \Phi_{e'}(e')\downarrow$$

¡Absurdo!

Ejemplo Yapa

Dada $f(x)$:

$$f(x) = \begin{cases} 1 & \textit{si } \Phi_x(x)\downarrow \\ 0 & \text{en otro caso} \end{cases} = HALT(x, x)$$

Demostrar que $f(x)$ no es computable.

Notar que casi misma demostración que usamos para $HALT(x, y)$ para ver que nos sirve para demostrar que $f(x)$ no es computable.

Demostramos por absurdo.
Asumimos que la función $f(x) = HALT(x, x)$ es computable y llegamos a una contradicción.
Sea P un programa que computa $HALT(x, x)$ (existe tal programa porque asumimos $HALT(x, x)$ computable).
Definimos D el siguiente programa:

P
if $Y == 1$ **then**
    $Y_2 + +$
    while $Y_2 > 0$ **do**
        $Y_2 + +$
    end while
else
    $Y = 1$
end if

Y llegamos a un absurdo de la misma forma que en la demostración anterior.

¿Cómo demuestro no computabilidad?

- ¿Cómo demuestro que otras funciones no son computables?
- Creo en RRR (Reducir, Reutilizar y Reciclar)
  ¿Cómo lo uso en este contexto?
- Reciclemos, reusemos el hecho de que sabemos que $HALT(x, y)$ no es computable y veamos cómo podemos aprovecharlo para demostrar que otras funciones no son computables.

Decidir si un programa siempre termina

Dada $f(x)$:

$$f(x) = \begin{cases} 1 & \text{si } \forall y \ \Phi_x(y)\downarrow \\ 0 & \text{en otro caso} \end{cases}$$

Demostrar que $f(x)$ no es computable.

Decidir si un programa siempre termina

Aprovechemos que sabemos que $HALT(x, x)$ no es computable.
Asumamos que $f(x)$ es computable y sea P un programa que la computa.
Fijamos e un número natural.
Definimos el programa D:

$\Phi_e(e)$
$Y = 1$

¿Qué computa D?
¡$\Psi_e(e)$ es una función constante!

$$\Psi_e(e) = \begin{cases} 1 & \text{si } \Phi_e(e)\downarrow \\ \uparrow & \text{en otro caso} \end{cases}$$

Decidir si un programa siempre termina

¿Qué pasa si evaluamos f en el número de programa de D?

$$f(\#(D)) = 1 \text{ sii } HALT(e, e)$$

Para cada e, definimos D_e como arriba. Entonces:

$$f(\#(D_e)) = f(e) = 1 \text{ sii } HALT(e, e)$$

Como asumimos f computable, resulta también $HALT(x, x)$
¡Absurdo!

Decidir si un programa alguna vez termina

Dada $f(x)$:

$$f(x) = \begin{cases} 1 & \text{si } \exists y \; \Phi_x(y)\downarrow \\ 0 & \text{en otro caso} \end{cases}$$

Demostrar que $f(x)$ no es computable.

Decidir si un programa alguna vez termina

Podemos resolver de manera similar.
Definamos un programa que tenga el mismo comportamiento para todas sus entradas.
Fijamos e un número natural.
Definimos el programa D:

$\Phi_e(e)$
$Y = 1$

¿Qué computa D?
¡$\Psi_e(e)$ es una función constante!

$$\Psi_e(e) = \begin{cases} 1 & \text{si } \Phi_e(e)\downarrow \\ \uparrow & \text{en otro caso} \end{cases}$$

Decidir si un programa alguna vez termina

¿Qué pasa si evaluamos f en el número de programa de D?

$$f(\#(D)) = 1 \text{ sii } HALT(e, e)$$

Para cada e, definimos D_e como arriba. Entonces:

$$f(\#(D_e)) = f(e) = 1 \text{ sii } HALT(e, e)$$

Como asumimos f computable, resulta también $HALT(x, x)$
¡Absurdo!

Computabilidad con cuantificadores

Sea $f : \mathbb{N}^2 \to \mathbb{N}$ tal que

$$f(x, y) = (\exists t)_{\leq y}(HALT(x + 1, t) \wedge t > x)$$

a) Decidir si la función es computable y demostrar.
b) Analizar qué pasa si se cambia el existencial por un para todo o por un mínimo, usando la misma cota.

parte a)

Como primera observación, notar que t debe cumplir $t \leq y < x$.
Entonces tomando $x = y + 1$, debe valer $t = y$.

$$f(x, x+1) =$$

$$(\exists t)_{\leq x+1}(HALT(x+1, t) \wedge t > x) =$$

$$HALT(x+1, x+1)$$

Si asumimos que f es computable, resulta que también lo es $HALT(x+1, x+1)$, ya que la transformación de parámetros que tenemos que hacer para ir de f a $HALT$ es computable (proyectamos en una de las variables de entrada y sumamos 1, composición finita de funciones computables). ¡Absurdo!
Resulta entonces que f no es computable.

parte b)

Si reemplazamos el existencial por un mínimo la función sigue sin ser computable. Podemos reducir f a $HALT$.
Tomamos

$$f'(x) = \begin{cases} x+1 & \text{si HALT(x+1,x+1)} \\ x+2 & \text{si no} \end{cases}$$

Como antes, sólo puede ser $t = x + 1$.
Resulta $1 - (f'(x) - (x+1)) = HALT(x+1, x+1)$.

parte b)

Si reemplazamos el $\exists$ por $\forall$, la función resultante sí resulta computable, ya que en particular no vale para $t = 0$ (entonces $f(x, y)$ resulta la función constante 0, que es computable).