

Lógica y Computabilidad

Santiago Figueira

Departamento de Computación, FCEyN, UBA

2do cuatrimestre 2020 - **A DISTANCIA**

Contenido - Computabilidad

1. Introducción, máquinas de Turing, funciones parciales, funciones Turing computables, ejemplos, p. 5
2. Funciones iniciales, composición, recursión, clases PRC, funciones primitivas recursivas , p. 21
3. Sumatorias y productorias, cuantificadores acotados, minimización acotada, codificación de pares y secuencias, p. 34
4. Lenguaje \mathcal{S} , estado, descripción instantánea, cómputo, funciones parciales computables, minimización no acotada, p. 50
5. Codificación de programas, Halting problem, diagonalización, tesis de Church, programa universal, step counter, snapshot, p. 71
6. Teorema de la forma normal, teorema del parámetro, teorema de la recursión y aplicaciones, teorema del punto fijo, p. 101
7. Conjuntos c.e., teorema de la enumeración, teorema de Rice y aplicaciones, p. 115
8. Opcional - Oráculos, reducibilidad de Turing, jerarquía aritmética, problema de Post, p. 131

Contenido - Lógica

Lógica proposicional

1. Lenguaje de lógica proposicional, semántica, tautología, consecuencia semántica, conjunto satisfacible, sistema axiomático SP , consecuencia sintáctica, p. 144
2. Teorema de la deducción, lema de Lindenbaum, completitud de SP , compacidad, p. 163

Lógica de primer orden

1. Lenguaje de lógica de primer orden, términos, fórmulas, variables libres y ligadas, interpretación, valuación, niveles de verdad, consecuencia semántica, p. 180
2. Lema de sustitución, sistema axiomático SQ , consecuencia sintáctica, teorema de la generalización, teorema de la generalización en constantes, p. 200
3. Completitud de SQ , compacidad, p. 219
4. Aplicaciones de compacidad, indecidibilidad de la lógica de primer orden, p. 235
5. Opcional - Teorema de incompletitud de Gödel, p. 251

Temas y bibliografía

- ▶ Computabilidad
 - ▶ Computability, Complexity and Languages, fundamentals of theoretical computer science. Martin Davis, Ron Sigal, Elaine Weyuker, Elsevier, 1994
- ▶ Lógica proposicional
 - ▶ Metalógica, introducción a la metateoría de la lógica clásica de primer orden. Geoffrey Hunter, Editorial Paraninfo, 1981.
- ▶ Lógica de primer orden
 - ▶ A mathematical introduction to logic. Herbert Enderton, Elsevier, 2006.

Computabilidad

Clase 1

Introducción, máquinas de Turing, funciones parciales,
funciones Turing computables, ejemplos

Orígenes

- ▶ fines del siglo XIX y principios del siglo XX: interés por los fundamentos de la matemática
- ▶ dos grandes búsquedas (Hilbert)
 1. completitud de la aritmética
 - ▶ se buscaba un sistema axiomático que capturara todas las verdades de la aritmética
 - ▶ Gödel (1931): cualquier sistema axiomático suficientemente poderoso es incompleto o inconsistente
 2. el problema de la decisión (*Entscheidungsproblem*)
 - ▶ se buscaba un procedimiento efectivo para decidir si cualquier fórmula de primer orden era válida o no
 - ▶ Turing (1936): no existe tal procedimiento efectivo



David Hilbert

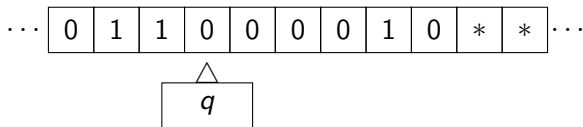


Kurt Gödel



Alan Turing

Máquinas de Turing



Se compone de :

- ▶ una **cinta**
 - ▶ dividida en 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
 - L y R son símbolos reservados (representarán acciones que puede realizar la cabeza)
- ▶ 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

Tabla de instrucciones

- ▶ Σ es el alfabeto. $L, R \notin \Sigma, * \in \Sigma$.
- ▶ Q es el conjunto finito de estados
- ▶ $A = \Sigma \cup \{L, R\}$ es el conjunto de acciones
 - ▶ un símbolo $s \in \Sigma$ se interpreta como “escribir s en la posición actual”
 - ▶ L se interpreta como “mover la cabeza una posición hacia la izquierda”
 - ▶ R se interpreta como “mover la cabeza una posición hacia la derecha”

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'

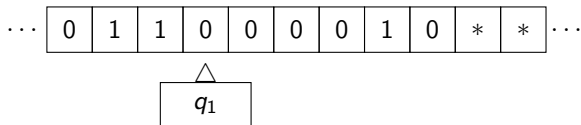
Ejemplo de ejecución de una instrucción

Supongamos un alfabeto $\Sigma = \{0, 1\}$.

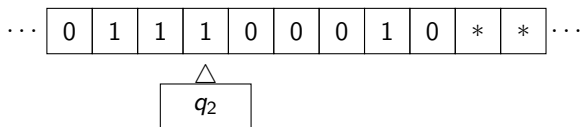
Una máquina con esta tabla de instrucciones:

$$\{ (q_1, 0, 1, q_2) , (q_2, 1, R, q_1) \}$$

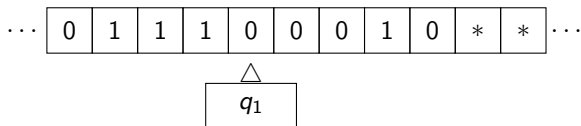
Si empieza en esta configuración



pasa a



y luego a



Definición de máquina de Turing

Una **máquina de Turing** es una tupla

$$(\Sigma, Q, T, q_0, q_f)$$

donde

- ▶ Σ (finito) es el conjunto **símbolos** ($L, R \notin \Sigma, * \in \Sigma$)
- ▶ Q (finito) es el conjunto de **estados**
 - ▶ tiene dos estados distinguidos:
 - ▶ $q_0 \in Q$ es el **estado inicial**
 - ▶ $q_f \in Q$ es el **estado final**
- ▶ $T \subseteq Q \times \Sigma \times \Sigma \cup \{L, R\} \times Q$ es la **tabla de instrucciones**
 - ▶ va a ser finita porque Σ y Q lo son

- ▶ cuando no hay restricciones sobre T decimos que \mathcal{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 \mathcal{M} es una máquina de Turing **determinística**

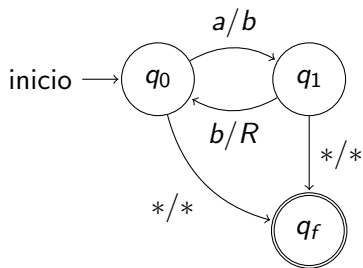
Ejemplo

Sea $\mathcal{M} = (\Sigma, Q, T, q_0, q_f)$ con

- ▶ $\Sigma = \{*, a, b\}$
- ▶ $Q = \{q_0, q_1, q_f\}$
- ▶ tabla de instrucciones

q_0	a	b	q_1
q_1	b	R	q_0
q_0	$*$	$*$	q_f
q_1	$*$	$*$	q_f

Visto como autómata



- ▶ si empieza en q_0

*	a	a	a	a	*

 termina en q_f

*	b	b	b	b	*
- ▶ si empieza en q_0

*	a	a	b	a	*

 termina en q_0

*	b	b	b	a	*
- ▶ si empieza en q_0

*	a	a	b	a	*

 termina en q_f

*	a	a	b	a	*

Representación de números y tuplas

Fijamos $\Sigma = \{*, 1\}$.

- ▶ representaremos a los **números** naturales en unario (con palotes).
- ▶ el número $x \in \mathbb{N}$ se representa como

$$\bar{x} = \underbrace{1 \dots 1}_{x+1}$$

- ▶ representamos a las **tuplas** (x_1, \dots, x_n) como lista de (representaciones de) los x_i separados por blanco
- ▶ la tupla (x_1, \dots, x_n) se representa como

$$*\bar{x}_1 * \bar{x}_2 * \dots * \bar{x}_n*$$

Por ejemplo,

- ▶ el número 0 se representa como 1
- ▶ el número 3 se representa como 1111
- ▶ la tupla (1, 2) se representa como *11 * 111*
- ▶ la tupla (0, 0, 1) se representa como *1 * 1 * 11*

Funciones parciales

Siempre vamos a trabajar con funciones $f : \mathbb{N}^n \rightarrow \mathbb{N}$.

Pero van a ser 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 . En este caso $f(x_1, \dots, x_n)$ es un número natural.
- ▶ notamos $f(x_1, \dots, x_n) \uparrow$ cuando f está indefinida para x_1, \dots, x_n

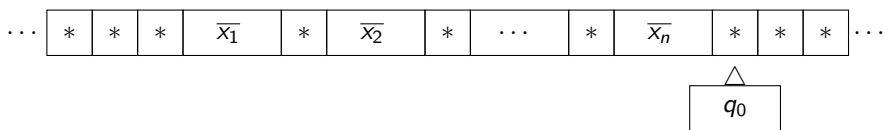
El conjunto de argumentos para los que f está definida se llama **dominio** de f , notado $\text{dom}(f)$.

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

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

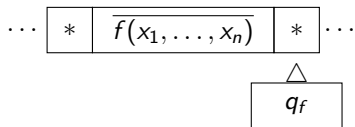
Cómputo de funciones parciales en máquinas de Turing

Una función parcial $f : \mathbb{N}^n \rightarrow \mathbb{N}$ es **Turing computable** si existe una máquina de Turing determinística $\mathcal{M} = (\Sigma, Q, T, q_0, q_f)$ con $\Sigma = \{*, 1\}$ tal que cuando empieza en la configuración inicial



(con los enteros x_i representados en unario y nada más en la entrada salvo la representación de la entrada):

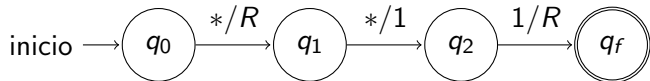
- ▶ si $f(x_1, \dots, x_n) \downarrow$ entonces siguiendo sus instrucciones en T llega a una configuración final de la forma



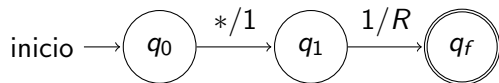
(quizá algo más en la cinta)

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

Cómputo de la función $f(x) = 0$



Cómputo de la función $f(x) = x + 1$



Cómputo de la función $f(x) = 2x$

Idea: por cada 1 que borro de la entrada, pongo 11 bien a la derecha.
Repito esto hasta que quede solo un 1 en la entrada. Ahí pongo un 1 más a la derecha.

Ejemplo: entrada = 2

1. * * * * * 1 1 1 * * * * * * * * *
2. * * * * * * 1 1 * 1 1 * * * * * *
3. * * * * * * * 1 * 1 1 1 1 * * * *
4. * * * * * * * 1 * 1 1 1 1 1 * * *

Invariante: a lo largo de cada iteración, la cinta está así:

$***\underbrace{1\dots 1}_n * \underbrace{1\dots\dots 1}_{2m}***$ para algún $n > 0, m \geq 0, n + m - 1 = \text{entrada}$

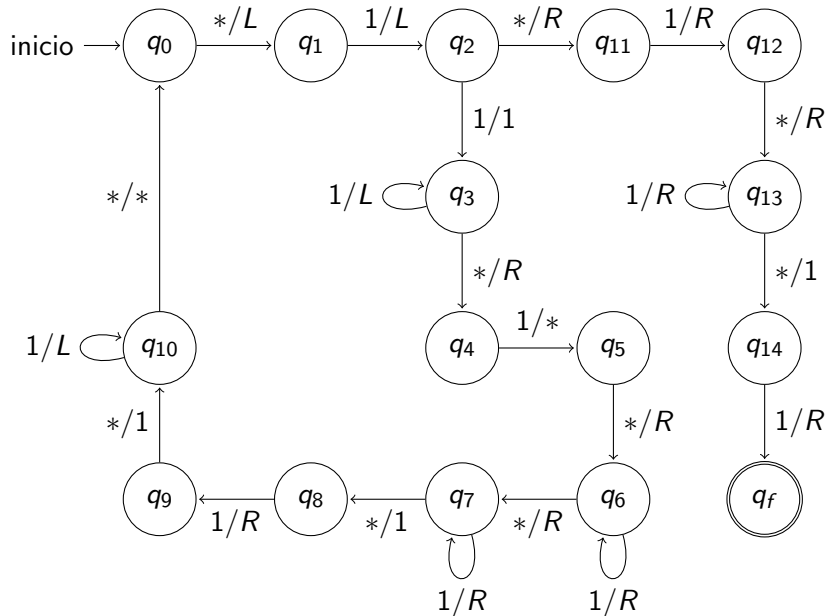
1. si $n = 1$ entonces pongo un 1 más a la derecha y termina en q_f con

$***1 * \underbrace{1\dots\dots 1}_{2m+1}***$

2. si $n > 1$ transformo la cinta en $***\underbrace{1\dots 1}_{n-1} * \underbrace{1\dots\dots 1}_{2(m+1)}***$ y vuelvo

al paso 1

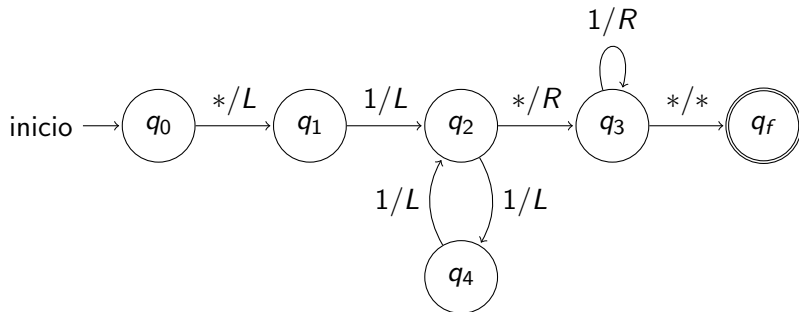
Cómputo de la función $f(x) = 2x$



Cómputo de una función parcial

Supongamos

$$f(x) = \begin{cases} x & \text{si } x \text{ es par} \\ \uparrow & \text{si no} \end{cases}$$



Poder de cómputo

Teorema

Sea $f : \mathbb{N}^m \rightarrow \mathbb{N}$ una función parcial. Son equivalentes:

1. f es computable en Java
2. f es computable en C
3. f es computable en Haskell
4. f es Turing computable

No es importante

- ▶ qué base usamos para representar a los números
 - ▶ usamos representación unaria ($\Sigma = \{*, 1\}$)
 - ▶ pero podríamos haber elegido la binaria ($\Sigma = \{*, 0, 1\}$)
 - ▶ o base 10 ($\Sigma = \{*, 0, 1, 2, \dots, 9\}$)
- ▶ si permitimos que al terminar la cinta tenga otras cosas escritas además de la salida o solo contenga la salida
- ▶ si usamos esta variante de arquitectura:
 - ▶ una cinta de entrada (solo de lectura)
 - ▶ una cinta de salida (solo de escritura)
 - ▶ una o varias cintas de trabajo, de lectura/escritura

¡Siempre computamos la misma clase de funciones!

Computabilidad

Clase 2

Funciones iniciales, composición, recursión, clases PRC,
funciones primitivas recursivas

Funciones iniciales

Otra manera de formalizar la idea de **función calculable de manera efectiva**:

- ▶ empezar por funciones muy simples, efectivas intuitivamente
- ▶ si mezclamos de alguna manera efectiva dos o más funciones que ya eran efectivas, entonces obtenemos una función calculable de manera efectiva

Definición

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 $i \in \{1, \dots, n\}$

Composición y recursión primitiva

Definición

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 **composición** si

$$h(x_1, \dots, x_n) = f(g_1(x_1, \dots, x_n), \dots, g_k(x_1, \dots, x_n))$$

Definición

$h : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ se obtiene a partir de $g : \mathbb{N}^{n+2} \rightarrow \mathbb{N}$ y $f : \mathbb{N}^n \rightarrow \mathbb{N}$ por **recursión primitiva** si

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

(En este contexto, una función 0-aria es una constante k . Si h es 1-aria y $t = 0$, entonces $h(t) = k = s^{(k)}(n(t))$.)

Clases PRC

Una clase \mathcal{C} de funciones totales es **PRC (primitive recursive closed)** si

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

Teorema

La clase de funciones totales Turing computables (o computables en C, o en Java) 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.

Demostración.

- (\Leftarrow) La clase de funciones p.r. es una clase PRC. Luego, si f pertenece a toda clase PRC, en particular f es p.r.
- (\Rightarrow) Sea f p.r. y sea \mathcal{C} una clase PRC. Como f es p.r., hay una lista

tal que f_1, f_2, \dots, f_n

- ▶ $f = f_n$
- ▶ f_i es inicial (luego está en \mathcal{C}) o se obtiene por composición o recursión primitiva a partir de funciones $f_j, j < i$ (luego también está en \mathcal{C}).

Entonces todas las funciones de la lista están en \mathcal{C} (por inducción). En particular, $f_n \in \mathcal{C}$.



¿Funciones totales Turing computables = funciones primitivas recursivas?

Entonces la clase de funciones p.r. es la clase PRC más chica.

Corolario

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

Demostración.

Sabemos que la clase de funciones totales Turing computables es PRC. Por el teorema anterior, si f es p.r., entonces f pertenece a la clase de funciones Turing computables. □

No toda función **parcial** Turing computable es p.r porque toda función p.r. es total. Pero...

¿toda función total Turing computable es p.r.?

Ejemplo de función p.r.

La función

$$\textit{suma}(x, y) = x + y$$

es p.r., porque

$$\textit{suma}(x, 0) = u_1^1(x)$$

$$\textit{suma}(x, y + 1) = g(\textit{suma}(x, y), x, y)$$

donde

$$g(x_1, x_2, x_3) = s(u_1^3(x_1, x_2, x_3))$$

Otras funciones primitivas recursivas

- ▶ $x \cdot y$
- ▶ $x!$
- ▶ x^y
- ▶ $x \dot{-} y = \begin{cases} x - y & \text{si } x \geq y \\ 0 & \text{si no} \end{cases}$
- ▶ $\alpha(x) = \begin{cases} 1 & \text{si } x = 0 \\ 0 & \text{si no} \end{cases}$
- ▶ y muchas más. ¿Todas las funciones totales Turing computables..?

Predicados primitivos recursivos

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

- ▶ 1 se interpreta como verdadero
- ▶ 0 se interpreta como falso

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

Por ejemplo, el predicado $x \leq y$ es p.r. porque se puede definir como

$$\alpha(x \dot{-} y)$$

Operadores lógicos

Teorema

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

Demostración.

- ▶ $\neg p$ se define como $\alpha(p)$
- ▶ $p \wedge q$ se define como $p \cdot q$
- ▶ $p \vee q$ se define como $\neg(\neg p \wedge \neg q)$



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

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

está en \mathcal{C} .

Demostración.

$$f(x_1, \dots, x_n) = g(x_1, \dots, x_n) \cdot p(x_1, \dots, x_n) + h(x_1, \dots, x_n) \cdot \alpha(p(x_1, \dots, x_n)) \quad \square$$

Definición por casos ($m + 1$)

Teorema

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

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

está en \mathcal{C} .

Recursión primitiva

- ▶ todavía no respondimos si p.r. = Turing computables totales
- ▶ no lo vamos a responder todavía

Observar que el esquema de recursión

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

es muy simple:

- ▶ 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}

Computabilidad

Clase 3

Sumatorias y productorias, cuantificadores acotados, minimización acotada, codificación de pares y secuencias

Sumatorias y productorias (desde 0)

Teorema

Sea \mathcal{C} una clase PRC. Si $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ está en \mathcal{C} entonces también están las funciones

$$g(y, x_1, \dots, x_n) = \sum_{t=0}^y f(t, x_1, \dots, x_n)$$

$$h(y, x_1, \dots, x_n) = \prod_{t=0}^y f(t, x_1, \dots, x_n)$$

Demostración.

$$g(0, x_1, \dots, x_n) = f(0, x_1, \dots, x_n)$$

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

Idem para h con \cdot en lugar de $+$. □

Observar que no importa la variable en la que se hace la recursión:

podemos definir $g'(x, t)$ como la clase pasada y luego

$$g(t, x) = g'(u_2^2(t, x), u_1^2(t, x)) = g'(x, t).$$

Sumatorias y productorias (desde 1)

Teorema

Sea \mathcal{C} una clase PRC. Si $f : \mathbb{N}^{n+1} \rightarrow \mathbb{N}$ está en \mathcal{C} entonces también están las funciones

$$g(y, x_1, \dots, x_n) = \sum_{t=1}^y f(t, x_1, \dots, x_n)$$

$$h(y, x_1, \dots, x_n) = \prod_{t=1}^y f(t, x_1, \dots, x_n)$$

(como siempre, sumatoria vacía = 0, productoria vacía = 1)

Demostración.

$$g(0, x_1, \dots, x_n) = 0$$

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

Idem para h con \cdot en lugar de $+$ y 1 en lugar de 0 en el caso base. □

Cuantificadores acotados

Sea $p : \mathbb{N}^{n+1} \rightarrow \{0, 1\}$ un predicado.

$(\forall t)_{\leq y} p(t, x_1, \dots, x_n)$ es verdadero sii

▶ $p(0, x_1, \dots, x_n)$ es verdadero y

⋮

▶ $p(y, x_1, \dots, x_n)$ es verdadero

$(\exists t)_{\leq y} p(t, x_1, \dots, x_n)$ es verdadero sii

▶ $p(0, x_1, \dots, x_n)$ es verdadero o

⋮

▶ $p(y, x_1, \dots, x_n)$ es verdadero

Lo mismo se puede definir con $< y$ en lugar de $\leq y$.

$$(\exists t)_{< y} p(t, x_1, \dots, x_n) \quad \text{y} \quad (\forall t)_{< y} p(t, x_1, \dots, x_n)$$

Cuantificadores acotados (con \leq)

Teorema

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

$$(\forall t)_{\leq y} p(t, x_1, \dots, x_n)$$

$$(\exists t)_{\leq y} p(t, x_1, \dots, x_n)$$

Demostración.

$$(\forall t)_{\leq y} p(t, x_1, \dots, x_n) \text{ sii } \prod_{t=0}^y p(t, x_1, \dots, x_n) = 1$$

$$(\exists t)_{\leq y} p(t, x_1, \dots, x_n) \text{ sii } \sum_{t=0}^y p(t, x_1, \dots, x_n) \neq 0$$

- ▶ la sumatoria y productoria están en \mathcal{C}
- ▶ la comparación por $=$ está en \mathcal{C}



Cuantificadores acotados (con $<$)

Teorema

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

$$(\forall t)_{<y} p(t, x_1, \dots, x_n)$$

$$(\exists t)_{<y} p(t, x_1, \dots, x_n)$$

Demostración.

$$(\forall t)_{<y} p(t, x_1, \dots, x_n) \text{ sii } (\forall t)_{\leq y} (t = y \vee p(t, x_1, \dots, x_n))$$

$$(\exists t)_{<y} p(t, x_1, \dots, x_n) \text{ sii } (\exists t)_{\leq y} (t \neq y \wedge p(t, x_1, \dots, x_n))$$



Más ejemplos de funciones primitivas recursivas

- ▶ $y|x$ sii y divide a x . Se define como

$$(\exists t)_{\leq x} y \cdot t = x$$

Notar que con esta definición $0|0$.

- ▶ $\text{primo}(x)$ sii x es primo.

Minimización acotada

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

$$g(y, x_1, \dots, x_n) = \sum_{u=0}^y \prod_{t=0}^u \alpha(p(t, x_1, \dots, x_n))$$

¿Qué hace g ?

- ▶ supongamos que existe un $t \leq y$ tal que $p(t, x_1, \dots, x_n)$ es verdadero

- ▶ sea t_0 el mínimo tal t

- ▶ $p(t, x_1, \dots, x_n) = 0$ para todo $t < t_0$

- ▶ $p(t_0, x_1, \dots, x_n) = 1$

- ▶ $\prod_{t=0}^u \alpha(p(t, x_1, \dots, x_n)) = \begin{cases} 1 & \text{si } u < t_0 \\ 0 & \text{si no} \end{cases}$

- ▶ $g(y, x_1, \dots, x_n) = \underbrace{1 + 1 + \dots + 1}_{t_0 \text{ veces}} + 0 + 0 + \dots + 0 = t_0$

- ▶ entonces $g(y, x_1, \dots, x_n)$ es el mínimo $t \leq y$ tal que $p(t, x_1, \dots, x_n)$ es verdadero

- ▶ si no existe tal t , $g(y, x_1, \dots, x_n) = y + 1$

Minimización acotada

Notamos

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

Teorema

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

$$\min_{t \leq y} p(t, x_1, \dots, x_n)$$

también está en \mathcal{C} .

Más ejemplos de funciones primitivas recursivas

- ▶ $x \text{ div } y$ es la división entera de x por y

$$\min_{t \leq x} ((t + 1) \cdot y > x)$$

Notar que con esta definición $0 \text{ div } 0$ es 0 .

- ▶ $x \text{ mód } y$ es el resto de dividir a x por y
- ▶ p_n es el n -ésimo primo ($n > 0$). Se define $p_0 = 0, p_1 = 2, p_2 = 3, p_3 = 5, \dots$

$$p_0 = 0$$
$$p_{n+1} = \min_{t \leq K(n)} (\text{primo}(t) \wedge t > p_n)$$

Necesitamos una cota $K(n)$ que sea buena, i.e.

- ▶ suficientemente grande y
- ▶ primitiva recursiva

$K(n) = p_n! + 1$ funciona (ver que $p_{n+1} \leq p_n! + 1$).

Codificación de pares

Definimos la función primitiva recursiva

$$\langle x, y \rangle = 2^x(2 \cdot y + 1) - 1$$

Notar que $2^x(2 \cdot y + 1) \neq 0$.

Proposición

Hay una única solución (x, y) a la ecuación $\langle x, y \rangle = z$.

Demostración.

- ▶ x es el máximo número tal que $2^x | (z + 1)$
- ▶ $y = ((z + 1)/2^x - 1)/2$



Observadores de pares

Los **observadores** del par $z = \langle x, y \rangle$ son

- ▶ $l(z) = x$
- ▶ $r(z) = y$

Proposición

Los observadores de pares son primitivas recursivas.

Demostración.

Como $x, y < z + 1$ tenemos que

- ▶ $l(z) = \min_{x \leq z} ((\exists y)_{\leq z} z = \langle x, y \rangle)$
- ▶ $r(z) = \min_{y \leq z} ((\exists x)_{\leq z} z = \langle x, y \rangle)$



Por ejemplo,

- ▶ $\langle 2, 5 \rangle = 2^2(2 \cdot 5 + 1) - 1 = 43$
- ▶ $l(43) = 2$
- ▶ $r(43) = 5$

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 ($i \geq 1$).

Por ejemplo el número de Gödel de la secuencia

$$1, 3, 3, 2, 2$$

es

$$[1, 3, 3, 2, 2] = 2^1 \cdot 3^3 \cdot 5^3 \cdot 7^2 \cdot 11^2 = 40020750.$$

Propiedades de la codificación de secuencias

Teorema

Si $[a_1, \dots, a_n] = [b_1, \dots, b_n]$ entonces $a_i = b_i$ para todo $i \in \{1, \dots, n\}$.

Demostración.

Por la factorización única en primos. □

Observar que

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

pero

$$[a_1, \dots, a_n] \neq [0, a_1, \dots, a_n]$$

Observadores de secuencias

Los **observadores** de la secuencia $x = [a_1, \dots, a_n]$ son

- ▶ $x[i] = a_i$
- ▶ $|x| = \text{longitud de } x$

Proposición

Los observadores de secuencias son primitivas recursivas.

Demostración.

- ▶ $x[i] = \text{mín}_{t \leq x} (\neg p_i^{t+1} |x)$
- ▶ $|x| = \text{mín}_{i \leq x} (x[i] \neq 0 \wedge (\forall j)_{\leq x} (j \leq i \vee x[j] = 0))$



Por ejemplo,

- ▶ $[1, 3, 3, 2, 2][2] = 3 = 40020750[2]$
- ▶ $[1, 3, 3, 2, 2][6] = 0 = 40020750[6]$
- ▶ $|[1, 3, 3, 2, 2]| = 5 = |40020750|$
- ▶ $|[1, 3, 3, 2, 2, 0]| = |[1, 3, 3, 2, 2, 0, 0]| = 5 = |40020750|$
- ▶ $x[0] = 0$ para todo x
- ▶ $0[i] = 0$ para todo i

En resumen: codificación y decodificación de pares y secuencias

Teorema (Codificación de pares)

- ▶ $l(\langle x, y \rangle) = x, r(\langle x, y \rangle) = y$
- ▶ $z = \langle l(z), r(z) \rangle$
- ▶ $l(z), r(z) \leq z$
- ▶ *la codificación y observadores de pares son p.r.*

Teorema (Codificación de secuencias)

- ▶ $[a_1, \dots, a_n][i] = \begin{cases} a_i & \text{si } 1 \leq i \leq n \\ 0 & \text{si no} \end{cases}$
- ▶ *si* $n \geq |x|$ *entonces* $[x[1], \dots, x[n]] = x$
- ▶ *la codificación y observadores de secuencias son p.r.*

Computabilidad

Clase 4

Lenguaje \mathcal{S} , estado, descripción instantánea, cómputo, funciones parciales computables, minimización no acotada

Lenguaje de programación \mathcal{S} (Davis/Sigal/Weyuker)

- ▶ resulta igual de poderoso que las máquinas de Turing, pero es más fácil de programar
- ▶ imperativo, muy simple
 - variables de entrada: X_1, X_2, \dots
 - única variable de salida: Y
 - variables temporales: Z_1, Z_2, \dots } empiezan inicializadas en 0
- ▶ las variables almacenan números naturales
- ▶ 3 instrucciones (cada una puede o no estar etiquetada):
 1. $V \leftarrow V + 1$
 - ▶ la variable V se incrementa en 1
 2. $V \leftarrow V - 1$
 - ▶ V se decrementa en 1 si antes era > 0 ; si no queda en 0
 - ▶ es el $V-1$ que ya vimos
 3. IF $V \neq 0$ GOTO A
 - ▶ condicional muy primitivo
 - ▶ A es una etiqueta que denota una instrucción del programa
 - ▶ si el valor de V es distinto de 0, la ejecución sigue con la primera instrucción que tenga etiqueta A
 - ▶ si el valor de V es 0, sigue con la próxima instrucción
- ▶ programa = sucesión finita de instrucciones

Ejemplo 1

Programa P

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

Ejecución para
entrada $X = 3$:

X	Y
3	0
2	1
1	2
0	3

- ▶ escribimos X por X_1 ; Z por Z_1
- ▶ P termina cuando $X = 0$ porque no hay siguiente instrucción
- ▶ P computa la función $f : \mathbb{N} \rightarrow \mathbb{N}$,

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

- ▶ siempre deja la variable X en 0

Ejemplo 2

```
[A]  IF  $X \neq 0$  GOTO B
       $Z \leftarrow Z + 1$ 
      IF  $Z \neq 0$  GOTO E
[B]   $X \leftarrow X - 1$ 
       $Y \leftarrow Y + 1$ 
       $Z \leftarrow Z + 1$ 
      IF  $Z \neq 0$  GOTO A
```

- ▶ computa la función $f : \mathbb{N} \rightarrow \mathbb{N}$, $f(x) = x$
- ▶ cuando intenta ir a E , termina
- ▶ en el ejemplo, Z solo sirve para un salto incondicional. En general GOTO L es equivalente a

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

donde V es una variable nueva (en el ejemplo es Z)

Macros

- ▶ \mathcal{S} no tiene salto incondicional
- ▶ pero podemos simularlo con GOTO L
- ▶ lo usamos, como si fuera parte del lenguaje, pero:
 - ▶ cada vez que aparece

GOTO L

en un programa P , lo reemplazamos con

$V \leftarrow V + 1$

IF $V \neq 0$ GOTO L

donde V tiene que ser una variable que no aparece en P .

Vamos a ver que se pueden simular muchas otras operaciones. Una vez que sepamos que se pueden escribir en el lenguaje \mathcal{S} , las usamos como si fueran propias (son **pseudoinstrucciones**).

- ▶ la forma abreviada se llama **macro**
- ▶ el segmento de programa que la macro abrevia se llama **expansión del macro**

Asignación de cero: $V \leftarrow 0$

En un programa P , la pseudoinstrucción $V \leftarrow 0$ se expande como

```
[L]   V ← V - 1  
      IF V ≠ 0 GOTO L
```

donde L es una etiqueta que no aparece en P

Asignación de variables: $Y \leftarrow X$

$Y \leftarrow 0$

[A] IF $X \neq 0$ GOTO B
GOTO C

[B] $X \leftarrow X - 1$

$Y \leftarrow Y + 1$

$Z \leftarrow Z + 1$

GOTO A

[C] IF $Z \neq 0$ GOTO D
GOTO E

[D] $Z \leftarrow Z - 1$

$X \leftarrow X + 1$

GOTO C

- ▶ el primer ciclo copia el valor de X en Y y en Z ; deja X en cero
- ▶ el segundo ciclo pone en X el valor que tenía originalmente y deja Z en cero
- ▶ se usa la macro GOTO A

- ▶ no debe expandirse como

$Z \leftarrow Z + 1$

IF $Z \neq 0$ GOTO A

- ▶ sino como

$Z_2 \leftarrow Z_2 + 1$

IF $Z_2 \neq 0$ GOTO A

Asignación de variables: $V \leftarrow V'$

$Y \leftarrow 0$

[A] IF $X \neq 0$ GOTO B
GOTO C

[B] $X \leftarrow X - 1$
 $Y \leftarrow Y + 1$
 $Z \leftarrow Z + 1$
GOTO A

[C] IF $Z \neq 0$ GOTO D
GOTO E

[D] $Z \leftarrow Z - 1$
 $X \leftarrow X + 1$
GOTO C

se puede usar para asignar a la variable V el contenido de la variable V' y dejar V' sin cambios dentro de un programa P cualquiera: $V \leftarrow V'$.

- ▶ cambiar Y por V
- ▶ cambiar X por V'
- ▶ cambiar Z por una variable temporal que no aparezca en P
- ▶ cambiar A, B, C, D por etiquetas que no aparezcan en P

Suma de dos variables

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

computa la función $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

$$f(x_1, x_2) = x_1 + x_2$$

Resta de dos variables

```
    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
```

computa la función $g : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

$$g(x_1, x_2) = \begin{cases} x_1 - x_2 & \text{si } x_1 \geq x_2 \\ \uparrow & \text{si no} \end{cases}$$

- ▶ g es una función **parcial**
- ▶ la indefinición se nota con \uparrow (en el metalenguaje)
- ▶ el comportamiento del programa que se indefine es la **no terminación**
 - ▶ no hay otra causa de indefinición

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

Por ejemplo, para P :

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

▶ son estados de P :

- ▶ $X = 3, Y = 1$
- ▶ $X = 3, Y = 1, Z = 0$
- ▶ $X = 3, Y = 1, Z = 8$
 - ▶ no hace falta que sea alcanzado

▶ no son estados de P :

- ▶ $X = 3$
- ▶ $X = 3, Z = 0$
- ▶ $X = 3, Y = 1, X = 0$

Descripción instantánea

Supongamos que el programa P tiene 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
 - 3.1 si σ tiene $V = 0$ entonces $j = i + 1$
 - 3.2 si σ tiene $V = m$ para $m \neq 0$ entonces
 - ▶ si existe en P una instrucción con etiqueta L entonces $j = \min\{k : k\text{-ésima instrucción de } P \text{ tiene etiqueta } L\}$
 - ▶ 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

$$X_1 = r_1 \quad , \quad X_2 = r_2 \quad , \quad \dots \quad , \quad X_m = r_m \quad , \quad Y = 0$$

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 .

- ▶ en particular, $\Psi_P^{(m)}(r_1, \dots, r_m)$ está definido (not. $\Psi_P^{(m)}(r_1, \dots, r_m) \downarrow$)
- ▶ 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 $\Psi_P^{(m)}(r_1, \dots, r_m)$ está indefinido (not. $\Psi_P^{(m)}(r_1, \dots, r_m) \uparrow$)

Funciones computables

Una función (parcial) $f : \mathbb{N}^m \rightarrow \mathbb{N}$ es **\mathcal{S} -parcial computable** (o simplemente **parcial computable**) si existe un programa P tal que

$$f(r_1, \dots, r_m) = \Psi_P^{(m)}(r_1, \dots, r_m)$$

para todo $(r_1, \dots, r_m) \in \mathbb{N}^m$.

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 **\mathcal{S} -computable** (o simplemente **computable**) si es parcial computable y total.

Notar que un mismo programa P puede servir para computar funciones de 1 variable, 2 variables, etc. Supongamos que en P aparece X_n y no aparece X_i para $i > n$

- ▶ si solo se especifican $m < n$ variables de entrada, X_{m+1}, \dots, X_n toman el valor 0
- ▶ si se especifican $m > n$ variables de entrada, P ignorará X_{n+1}, \dots, X_m

Minimización no acotada

Recordar la definición de **minimización acotada**:

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

Definimos la **minimización no acotada**

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

Minimización no acotada

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.

Demostración.

El siguiente programa computa $\min_t p(t, x_1, \dots, x_n)$:

```
[A]   IF  $p(Y, X_1, \dots, X_n) = 1$  GOTO E  
       $Y \leftarrow Y + 1$   
      GOTO A
```



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.

Demostración.

El siguiente programa computa h :

$$Z_1 \leftarrow g_1(X_1, \dots, X_n)$$

$$\vdots$$

$$Z_k \leftarrow g_k(X_1, \dots, X_n)$$

$$Y \leftarrow f(Z_1, \dots, Z_k)$$

Si f, g_1, \dots, g_k son totales entonces h es total. □

Clausura por recursión primitiva

Teorema

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

Demostración.

El siguiente programa computa h :

$Y \leftarrow k$ (es una macro, se puede hacer fácil)

[A] IF $X = 0$ GOTO E (otra macro, condición del IF por =)

$Y \leftarrow g(Z, Y)$

$Z \leftarrow Z + 1$

$X \leftarrow X - 1$

GOTO A

Si g es total entonces h es total.



Las funciones computables forman una clase PRC

Teorema

La clase de funciones computables es una clase PRC.

Demostración.

Ya vimos que la clase de funciones computables está cerrada por composición (p. 68) y recursión primitiva (p. 69). Veamos que las funciones iniciales son computables:

- ▶ $s(x) = x + 1$ se computa con el programa

$$Y \leftarrow X + 1$$

- ▶ $n(x) = 0$ se computa con el programa vacío
- ▶ $u_i^n(x_1, \dots, x_n) = x_i$ se computa con el programa

$$Y \leftarrow X_i$$



Corolario

Toda función primitiva recursiva es computable.

Computabilidad

Clase 5

Codificación de programas, Halting problem, diagonalización, tesis de Church, programa universal, step counter, snapshot

Tipos de datos en \mathcal{S}

- ▶ vimos que el único tipo de dato en \mathcal{S} son los naturales
- ▶ sin embargo podemos simular otros tipos. Por ejemplo,
 - ▶ **tipo bool**: lo representamos con el 1 (verdadero) y el 0 (falso)
 - ▶ **tipo par de números naturales**: la codificación y decodificación de pares son funciones primitivas recursivas
 - ▶ **tipo entero**: podría ser codificada con un par

$\langle \text{bool}, \text{número natural} \rangle$

- ▶ **tipo secuencias finitas de números naturales**: la codificación y decodificación de secuencias son funciones primitivas recursivas
- ▶ ahora vamos a ver como simular el **tipo programa en \mathcal{S}**

Codificación de programas en \mathcal{S}

Recordemos que las instrucciones de \mathcal{S} eran:

1. $V \leftarrow V + 1$
2. $V \leftarrow V - 1$
3. IF $V \neq 0$ GOTO L'

Por conveniencia vamos a agregar una cuarta instrucción

4. $V \leftarrow V$: no hace nada

Observar que toda instrucción

- ▶ puede o no estar etiquetada con L
- ▶ menciona exactamente una variable V
- ▶ el IF además menciona siempre una etiqueta L'

Codificación de variables y etiquetas de \mathcal{S}

Ordenamos las variables:

$$Y, X_1, Z_1, X_2, Z_2, X_3, Z_3, \dots$$

Ordenamos las etiquetas:

$$A, B, C, D, \dots, Z, AA, AB, AC, \dots, AZ, BA, BB, \dots, BZ, \dots$$

Escribimos $\#(V)$ para la posición que ocupa la variable V en la lista. Idem para $\#(L)$ con la etiqueta L

Por ejemplo,

- ▶ $\#(Y) = 1$
- ▶ $\#(X_2) = 4$
- ▶ $\#(A) = 1$
- ▶ $\#(C) = 3$

Codificación de instrucciones de \mathcal{S}

Codificamos a 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
 - 3.1 $V \leftarrow V$ entonces $b = 0$
 - 3.2 $V \leftarrow V + 1$ entonces $b = 1$
 - 3.3 $V \leftarrow V - 1$ entonces $b = 2$
 - 3.4 IF $V \neq 0$ GOTO L' entonces $b = \#(L') + 2$

Por ejemplo,

- ▶ $\#(X \leftarrow X + 1) = \langle 0, \langle 1, 1 \rangle \rangle = \langle 0, 5 \rangle = 10$
- ▶ $\#([A] \ X \leftarrow X + 1) = \langle 1, \langle 1, 1 \rangle \rangle = \langle 1, 5 \rangle = 21$
- ▶ $\#(\text{IF } X \neq 0 \text{ GOTO } A) = \langle 0, \langle 3, 1 \rangle \rangle = \langle 0, 23 \rangle = 46$
- ▶ $\#(Y \leftarrow Y) = \langle 0, \langle 0, 0 \rangle \rangle = \langle 0, 0 \rangle = 0$

Todo número x representa a una única instrucción I .

Codificación de programas en \mathcal{S}

Un programa P es una lista (finita) de instrucciones I_1, \dots, I_k

Codificamos al programa P con

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

Por ejemplo, para el programa P

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

tenemos

$$\#(P) = [\#(I_1), \#(I_2)] - 1 = [21, 46] - 1 = 2^{21} \cdot 3^{46} - 1$$

Ambigüedades

Dijimos que P

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

tiene número $[21, 46] - 1$. Pero

$$[21, 46] = [21, 46, 0]$$

¡Un mismo número podría representar a más de un programa!
Por suerte, el programa $[21, 46, 0]$ es

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

y es equivalente a P .

De todos modos, eliminamos esta ambigüedad estipulando que

la instrucción final de un programa no puede ser $Y \leftarrow Y$

Con esto, cada número representa a un **único** programa.

Hay funciones no computables

- ▶ hay una cantidad no numerable de funciones $\mathbb{N} \rightarrow \mathbb{N}$
- ▶ 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**
Pero ¿qué ejemplo concreto tenemos?

El problema de la detención (halting problem)

$\text{HALT}(x, y) : \mathbb{N}^2 \rightarrow \{0, 1\}$ es verdadero sii el programa con número y y entrada x no se indefine, i.e.

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

donde P es el único programa tal que $\#(P) = y$.

Ejemplo:

(pseudo)programa P

```
[A]   Y ← 2
      Y ← Y + 2
      IF (∃a)≤Y(∃b)≤Y [Primo(a) ∧ Primo(b) ∧ a + b = Y] GOTO A
```

Supongamos que $\#(P) = e$. ¿Cuánto vale $\text{HALT}(x, e)$?

$\text{HALT}(x, e) = 1$ sii $\Psi_P(x) \downarrow$ sii la conjetura de Goldbach es falsa

HALT no es computable

Teorema (Turing, 1936)

HALT *no es computable*.

Demostración.

Supongamos que HALT es computable.

Construimos el siguiente programa:

Programa Q

[A] IF HALT(X, X) = 1 GOTO A

Supongamos que $\#(Q) = e$. Entonces

$$\Psi_Q(x) = \begin{cases} \uparrow & \text{si HALT}(x, x) = 1 \\ 0 & \text{si no} \end{cases}$$

Entonces

$$\text{HALT}(x, e) = 1 \quad \text{sii} \quad \Psi_Q(x) \downarrow \quad \text{sii} \quad \text{HALT}(x, x) \neq 1$$

e está fijo pero x es variable. Llegamos a un absurdo con $x = e$.



Diagonalización

En general, sirve para definir una función distinta a muchas otras.

En el caso de $\text{HALT}(x, y)$,

- ▶ sea P_i el programa con número i
- ▶ supongo que $\text{HALT}(x, y)$ es computable
- ▶ defino una función f computable
- ▶ núcleo de la demostración: ver que $f \notin \{\Psi_{P_0}, \Psi_{P_1}, \Psi_{P_2}, \dots\}$
 - ▶ para esto, me aseguro que $f(x) \neq \Psi_{P_x}(x)$, en particular:

	$\Psi_{P_0}(0)$	$\Psi_{P_0}(1)$	$\Psi_{P_0}(2)$	\dots
$f(x) \downarrow$	$\Psi_{P_1}(0)$	$\Psi_{P_1}(1)$	$\Psi_{P_1}(2)$	\dots
sii $\Psi_{P_x}(x) \uparrow$	$\Psi_{P_2}(0)$	$\Psi_{P_2}(1)$	$\Psi_{P_2}(2)$	\dots

- ▶ ¡pero f era computable! Absurdo: tenía que estar en

$$\{\Psi_{P_0}, \Psi_{P_1}, \Psi_{P_2}, \dots\}.$$

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

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

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

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

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 que se está por ejecutar (en la simulación de P)
- ▶ S describe el estado de P en cada momento

Inicialización

```
// entrada =  $x_1, \dots, x_n, e$   
//  $\#(P) = e = [i_1, \dots, i_m] - 1$   
//  $Z \leftarrow X_{n+1} + 1$   
//  $Z = [i_1, \dots, i_m]$   
//  $S \leftarrow \prod_{j=1}^n (p_{2j})^{X_j}$   
//  $S = [0, X_1, 0, X_2, \dots, 0, X_n]$  es el estado inicial  
//  $K \leftarrow 1$   
// la primera instrucción de  $P$  a analizar es la 1
```

Ciclo principal

```
// S codifica el estado, K es el número de instrucción
// Z = [i_1, ..., i_m]
[C] IF K = |Z| + 1 ∨ K = 0 GOTO F
// si llegó al final, terminar (ya veremos K = 0)
// si no, sea Z[K] = i_K = ⟨a, ⟨b, c⟩⟩
    U ← r(Z[K])
// U = ⟨b, c⟩
    P ← p_{r(U)+1}
// la variable que aparece en i_K es la c + 1-ésima
// P es el primo para la variable que aparece en i_K
```

Ciclo principal (cont.)

```
// S codifica el estado, K es el número de instrucción
//  $Z = [i_1, \dots, i_m]$ ,  $i_K = \langle a, \langle b, c \rangle \rangle$ ,  $U = \langle b, c \rangle$ 
// P es el primo para la variable V que aparece en  $i_K$ 
// IF  $I(U) = 0$  GOTO N
// si se trata de una instrucción  $V \leftarrow V$  salta a N
// IF  $I(U) = 1$  GOTO S
// si se trata de una instrucción  $V \leftarrow V + 1$  salta a S
// si no, es de la forma  $V \leftarrow V - 1$  o IF  $V \neq 0$  GOTO L
// IF  $\neg(P|S)$  GOTO N
// si P no divide a S (i.e.  $V = 0$ ), salta a N
// IF  $I(U) = 2$  GOTO R
//  $V \neq 0$ . Si se trata de una instrucción  $V \leftarrow V - 1$ , salta a R
```


Caso IF $V \neq 0$ GOTO L y $V \neq 0$

```
//  $S$  codifica el estado,  $K$  es el número de instrucción
//  $Z = [i_1, \dots, i_m]$ ,  $i_K = \langle a, \langle b, c \rangle \rangle$ ,  $U = \langle b, c \rangle$ 
//  $P$  es el primo para la variable  $V$  que aparece en  $i_K$ 
//  $V \neq 0$  y se trata de la instrucción IF  $V \neq 0$  GOTO  $L$ 
//  $b \geq 2$ , por lo tanto  $L$  es la  $(b - 2)$ -ésima etiqueta
//  $K \leftarrow \min_{j \leq |Z|} (I(Z[j]) + 2 = I(U))$ 
//  $K$  pasa a ser la primera instrucción con etiqueta  $L$ 
// si no hay tal instrucción,  $K = 0$  (saldrá del ciclo)
// GOTO  $C$ 
// vuelve a la primera instrucción del ciclo principal
```

Caso R (Resta)

```
//  $S$  codifica el estado,  $K$  es el número de instrucción  
//  $Z = [i_1, \dots, i_m], i_K = \langle a, \langle b, c \rangle \rangle, U = \langle b, c \rangle$   
//  $P$  es el primo para la variable  $V$  que aparece en  $i_K$   
// se trata de  $V \leftarrow V - 1$  con  $V \neq 0$   
[R]  $S \leftarrow S \text{ div } P$   
      GOTO  $N$   
//  $S$ =nuevo estado de  $P$  (resta 1 a  $V$ ) y salta a  $N$ 
```

Caso S (Suma)

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

Caso N (Nada)

```
//  $S$  codifica el estado,  $K$  es el número de instrucción
//  $Z = [i_1, \dots, i_m], i_K = \langle a, \langle b, c \rangle \rangle, U = \langle b, c \rangle$ 
//  $P$  es el primo para la variable  $V$  que aparece en  $i_K$ 
// la instrucción no cambia el estado
[N]  $K \leftarrow K + 1$ 
    GOTO C
//  $S$  no cambia
//  $K$  pasa a la siguiente instrucción
// vuelve al ciclo principal
```

Devolución del resultado

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

Todo junto

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

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

$$K \leftarrow 1$$

[C] IF $K = |Z| + 1 \vee K = 0$ GOTO F

$$U \leftarrow r(Z[K])$$

$$P \leftarrow p_{r(U)+1}$$

IF $l(U) = 0$ GOTO N

IF $l(U) = 1$ GOTO S

IF $\neg(P|S)$ GOTO N

IF $l(U) = 2$ GOTO R

$$K \leftarrow \min_{i \leq |Z|} (l(Z[i]) + 2 = l(U))$$

GOTO C

[R] $S \leftarrow S \operatorname{div} P$
GOTO N

[S] $S \leftarrow S \cdot P$
GOTO N

[M] $K \leftarrow K + 1$
GOTO C

[F] $Y \leftarrow S[1]$

Notación

A veces escribimos

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

A veces omitimos el superíndice cuando $n = 1$

$$\Phi_e(x) = \Phi(x, e) = \Phi^{(1)}(x, e)$$

Snapshot

Definimos

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

La configuración instantánea se representa como

⟨número de instrucción, lista representando el estado⟩

Teorema

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

Una función computable que no es primitiva recursiva

- ▶ se pueden codificar los programas de \mathcal{S} con constructores y observadores p.r.
- ▶ se pueden codificar las definiciones de funciones p.r. con constructores y observadores p.r.
- ▶ existe $\Phi_e^{(n)}(x_1, \dots, x_n)$ parcial computable que simula al e -ésimo programa con entrada x_1, \dots, x_n
- ▶ existe $\tilde{\Phi}_e^{(n)}(x_1, \dots, x_n)$ computable que simula a la e -ésima función p.r. con entrada x_1, \dots, x_n .

Analicemos $g : \mathbb{N} \rightarrow \mathbb{N}$ definida como $g(x) = \tilde{\Phi}_x^{(1)}(x)$

- ▶ claramente g es computable
- ▶ supongamos que g es p.r.
 - ▶ entonces también es p.r. la función $f(x) = g(x) + 1 = \tilde{\Phi}_x^{(1)}(x) + 1$
 - ▶ existe un e tal que $\tilde{\Phi}_e = f$
 - ▶ tendríamos $\tilde{\Phi}_e(x) = f(x) = \tilde{\Phi}_x(x) + 1$
 - ▶ e está fijo pero x es variable
 - ▶ instanciando $x = e$, $\tilde{\Phi}_e(e) = f(e) = \tilde{\Phi}_e(e) + 1$. Absurdo.
- ▶ ¿por qué esto no funciona para parcial comp. en lugar de p.r.?

La función de Ackermann (1928)

$$A(x, y, z) = \begin{cases} y + z & \text{si } x = 0 \\ 0 & \text{si } x = 1 \text{ y } z = 0 \\ 1 & \text{si } x = 2 \text{ y } z = 0 \\ y & \text{si } x > 2 \text{ y } z = 0 \\ A(x - 1, y, A(x, y, z - 1)) & \text{si } x, z > 0 \end{cases}$$

- ▶ $A_0(y, z) = A(0, y, z) = y + z = y \underbrace{+ 1 + \dots + 1}_{z \text{ veces}}$
- ▶ $A_1(y, z) = A(1, y, z) = y \cdot z = \underbrace{y + \dots + y}_{z \text{ veces}}$
- ▶ $A_2(y, z) = A(2, y, z) = y \uparrow z = y^z = \underbrace{y \cdot \dots \cdot y}_{z \text{ veces}}$
- ▶ $A_3(y, z) = A(3, y, z) = y \uparrow\uparrow z = \underbrace{y^{y^{\dots^y}}}_{z \text{ veces}}$
- ▶ ...

Para cada i , $A_i : \mathbb{N}^2 \rightarrow \mathbb{N}$ es p.r. pero $A : \mathbb{N}^3 \rightarrow \mathbb{N}$ no es p.r.

Versión de Robinson & Peter (1948)

$$B(m, n) = \begin{cases} n + 1 & \text{si } m = 0 \\ B(m - 1, 1) & \text{si } m > 0 \text{ y } n = 0 \\ B(m - 1, B(m, n - 1)) & \text{si } m > 0 \text{ y } n > 0 \end{cases}$$

- ▶ $B_0(n) = B(0, n) = n + 1$
- ▶ $B_1(n) = B(1, n) = 2 + (n + 3) - 3$
- ▶ $B_2(n) = B(2, n) = 2 \cdot (n + 3) - 3$
- ▶ $B_3(n) = B(3, n) = 2 \uparrow (n + 3) - 3$
- ▶ $B_4(n) = B(4, n) = 2 \uparrow \uparrow (n + 3) - 3$
- ▶ ...

Para cada i , $B_i : \mathbb{N} \rightarrow \mathbb{N}$ es p.r. pero $B : \mathbb{N}^2 \rightarrow \mathbb{N}$ no es p.r.

- ▶ $B(4, 2) \simeq 2 \times 10^{19728}$

$B'(x) = B(x, x)$ crece más rápido que cualquier función p.r.

$$(\forall f \text{ p.r.})(\exists n)(\forall x > n) B'(x) > f(x)$$

Computabilidad

Clase 6

Teorema de la forma normal, teorema del parámetro, teorema de la recursión y aplicaciones, teorema del punto fijo

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

$$f(x_1, \dots, x_n) = l \left(\underset{z}{\text{mín}} R(x_1, \dots, x_n, z) \right)$$

Demostración.

Sea e el número de algún programa para $f(x_1, \dots, x_n)$.

Recordar que la configuración instantánea se representa como

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

El siguiente predicado $R(x_1, \dots, x_n, z)$ es el buscado:

$$l(z) = \underbrace{r \left(\underbrace{\text{SNAP}^{(n)}(x_1, \dots, x_n, e, r(z))}_{\text{estado final de } e \text{ con entrada } x_1, \dots, x_n} \right)}_{\text{valor de la variable } Y \text{ en ese estado final}} [1]$$

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**
(del tipo $\min_t q(x_1, \dots, x_n, t)$ donde siempre existe al menos un t tal que $q(x_1, \dots, x_n, t)$ es verdadero)

Eliminando variables de entrada

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

INSTRUCCIÓN 1	$\#(I_1)$	Computa la función $f : \mathbb{N}^2 \rightarrow \mathbb{N}$
\vdots		
INSTRUCCIÓN k	$\#(I_k)$	$f(x, y) = \Psi_P^{(2)}(x, y)$
		$\#(P) = [\#(I_1), \dots, \#(I_k)] - 1$

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

[A] $X_2 \leftarrow X_2 - 1$	109	Computa la función $f_0 : \mathbb{N} \rightarrow \mathbb{N}$
IF $X_2 \neq 0$ GOTO A	110	
INSTRUCCIÓN 1	$\#(I_1)$	$f_0(x) = \Psi_{P_0}^{(1)}(x)$
\vdots		
INSTRUCCIÓN k	$\#(I_k)$	$\#(P_0) = [109, 110, \#(I_1), \dots, \#(I_k)] - 1$

(Supongo que A no aparece como etiqueta en P ; si aparece elijo otro nombre de etiqueta)

Eliminando variables de entrada

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

[A] $X_2 \leftarrow X_2 - 1$	109
IF $X_2 \neq 0$ GOTO A	110
$X_2 \leftarrow X_2 + 1$	26
INSTRUCCIÓN 1	$\#(I_1)$
\vdots	
INSTRUCCIÓN k	$\#(I_k)$

Computa la función $f_1 : \mathbb{N} \rightarrow \mathbb{N}$

$$f_1(x) = \Psi_{P_1}^{(1)}(x)$$

$$\#(P_1) =$$

$$[109, 110, 26, \#(I_1), \dots, \#(I_k)] - 1$$

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

[A] $X_2 \leftarrow X_2 - 1$	109
IF $X_2 \neq 0$ GOTO A	110
$X_2 \leftarrow X_2 + 1$	26
$X_2 \leftarrow X_2 + 1$	26
INSTRUCCIÓN 1	$\#(I_1)$
\vdots	
INSTRUCCIÓN k	$\#(I_k)$

Computa la función $f_2 : \mathbb{N} \rightarrow \mathbb{N}$

$$f_2(x) = \Psi_{P_2}^{(1)}(x)$$

$$\#(P_2) =$$

$$[109, 110, 26, 26, \#(I_1), \dots, \#(I_k)] - 1$$

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 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

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

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, i.e. $\#(P)$

Teorema de la Recursión

Teorema

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

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

Demostración.

Sea S_n^1 la función del Teorema del Parámetro:

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

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

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

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

Teorema de la Recursión

Corolario

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

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

Demostración.

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

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

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

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



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 número 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 $g : \mathbb{N}^2 \rightarrow \mathbb{N}$, $g(z, x) = z$.

Aplicando el Teorema de la Recursión, existen infinitos e tal que

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



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 $g : \mathbb{N}^2 \rightarrow \mathbb{N}$, $g(z, x) = h(z)$.

Aplicando el Teorema de la Recursión, existen infinitos e tal que

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



Teorema del Punto Fijo

Teorema

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

Demostración.

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

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

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

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



Ejercicio

Probar que $f : \mathbb{N} \rightarrow \mathbb{N}$,

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

no es computable.

Supongamos f computable. Puedo definir el siguiente programa P :

[A] IF $f(X) = 1$ GOTO A

Tenemos

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

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

Computabilidad

Clase 7

Conjuntos c.e., teorema de la enumeración, teorema de Rice y aplicaciones

Conjuntos en teoría de la computabilidad

Cuando hablamos de un conjunto de naturales A pensamos siempre en la función característica de ese conjunto.

$$A(x) = \begin{cases} 1 & \text{si } x \in A \\ 0 & \text{si no} \end{cases}$$

Así, un conjunto puede ser:

- ▶ computable
- ▶ primitivo recursivo

Teorema

Sean A, B conjuntos de una clase PRC \mathcal{C} . Entonces $A \cup B$, $A \cap B$ y \overline{A} están en \mathcal{C} .

Conjuntos computablemente enumerables

Igual que con las funciones

- ▶ hay conjuntos computables, por ejemplo

$$\emptyset, \mathbb{N}, \{p : p \text{ es primo}\}$$

- ▶ hay conjuntos no computables, por ejemplo

$$\{\langle x, y \rangle : \text{HALT}(x, y)\}, \{\langle x, \langle y, z \rangle \rangle : \Phi_x(y) = z\}$$

Definición

Un conjunto A es **computablemente enumerable (c.e.)** cuando existe una función parcial computable $g : \mathbb{N} \rightarrow \mathbb{N}$ tal que

$$A = \{x : g(x) \downarrow\} = \text{dom } g$$

- ▶ podemos decidir algorítmicamente si un elemento **sí** pertenece a A , pero para elementos que **no** pertenecen a A , el algoritmo se indefin
- ▶ se llaman algoritmos de **semi-decisión**: resuelven una aproximación al problema de decidir la pertenencia de un elemento al conjunto A

Propiedades de los conjuntos c.e.

Un conjunto A es co-c.e. si \bar{A} es c.e.

Teorema

Si A es computable entonces A es c.e.

Demostración.

Sea P_A un programa para [la función característica de] A .
Consideremos el siguiente programa P :

[C] IF $P_A(X) = 0$ GOTO C

Tenemos

$$\Psi_P(x) = \begin{cases} 0 & \text{si } x \in A \\ \uparrow & \text{si no} \end{cases}$$

y por lo tanto

$$A = \{x : \Psi_P(x) \downarrow\}$$



Propiedades de los conjuntos c.e.

Teorema

Si A y B son c.e. entonces $A \cup B$ y $A \cap B$ también son c.e.

Demostración.

Sean $A = \{x : \Phi_p(x) \downarrow\}$, $B = \{x : \Phi_q(x) \downarrow\}$

$(A \cap B)$ El siguiente programa R tiene como dominio a $A \cap B$:

$Y \leftarrow \Phi_p(x)$

$Y \leftarrow \Phi_q(x)$

En efecto, $\Psi_R(x) \downarrow$ sii $\Phi_p(x) \downarrow$ y $\Phi_q(x) \downarrow$.

$(A \cup B)$ El siguiente programa R' tiene como dominio a $A \cup B$:

[C] IF STP⁽¹⁾(X, p, T) = 1 GOTO E

IF STP⁽¹⁾(X, q, T) = 1 GOTO E

$T \leftarrow T + 1$

GOTO C

En efecto, $\Psi_{R'}(x) \downarrow$ sii $\Phi_p(x) \downarrow$ o $\Phi_q(x) \downarrow$.

Propiedades de los conjuntos c.e.

Teorema

A es computable sii A y \bar{A} son c.e.

Demostración.

(\Rightarrow) si A es computable entonces \bar{A} es computable

(\Leftarrow) supongamos que A y \bar{A} son c.e.

$$A = \{x : \Phi_p(x) \downarrow\} \quad , \quad \bar{A} = \{x : \Phi_q(x) \downarrow\}$$

Consideremos P :

```
[C]   IF STP(1)(X, p, T) = 1 GOTO F
      IF STP(1)(X, q, T) = 1 GOTO E
      T ← T + 1
      GOTO C
[F]   Y ← 1
```

Para cada x , $x \in A$ o bien $x \in \bar{A}$. Entonces Ψ_P computa A . \square

Teorema de la enumeración

Definimos

$$W_n = \{x : \Phi_n(x) \downarrow\} = \text{dominio del } n\text{-ésimo programa}$$

Teorema

Un conjunto A es c.e. sii existe un n tal que $A = W_n$.

Existe una enumeración de todos los conjuntos c.e.

$$W_0, W_1, W_2, \dots$$

Problema de la detención (visto como conjunto)

Recordar que

$$W_n = \{x : \Phi_n(x) \downarrow\}$$

Definimos

$$K = \{n : n \in W_n\}$$

Observar que

$$n \in W_n \quad \text{sii} \quad \Phi_n(n) \downarrow \quad \text{sii} \quad \text{HALT}(n, n)$$

Teorema

K es c.e. pero no computable.

Demostración.

- ▶ la función $n \mapsto \Phi(n, n)$ es parcial computable, de modo que K es c.e.
- ▶ supongamos que K fuera computable. Entonces \overline{K} también lo sería. Luego existe un e tal que $\overline{K} = W_e$. Por lo tanto

$$e \in K \quad \text{sii} \quad e \in W_e \quad \text{sii} \quad e \in \overline{K}$$



Más propiedades de los conjuntos c.e.

Teorema

Si A es c.e., existe un predicado p.r. $R : \mathbb{N}^2 \rightarrow \mathbb{N}$ tal que

$$A = \{x : (\exists t) R(x, t)\}$$

Demostración.

Sea $A = W_e$. Es decir,

$$A = \{x : \Phi_e(x) \downarrow\}.$$

Entonces $x \in A$ cuando en algún tiempo t , el programa e con entrada x termina, i.e.

$$A = \{x : (\exists t) \underbrace{\text{STP}^{(1)}(x, e, t)}_{R(x, t)}\}$$



Más propiedades de los conjuntos c.e.

Teorema

Si $A \neq \emptyset$ es c.e., existe una función p.r. $f : \mathbb{N} \rightarrow \mathbb{N}$ tal que

$$A = \{f(0), f(1), f(2), \dots\}$$

Demostración.

Por el teorema anterior, existe P p.r. tal que

$$A = \{x : (\exists t) P(x, t)\}.$$

Sea $a \in A$ y definamos

$$f(u) = \begin{cases} l(u) & \text{si } P(l(u), r(u)) \\ a & \text{si no} \end{cases}$$

- ▶ $x \in A \Rightarrow$ existe t tal que $P(x, t) \Rightarrow f(\langle x, t \rangle) = x$
- ▶ sea x tal que $f(u) = x$ para algún u . Entonces $x = a$ o bien u es de la forma $u = \langle x, t \rangle$, con $P(x, t)$. Luego $x \in A$. □

Más propiedades de los conjuntos c.e.

Teorema

Si $f : \mathbb{N} \rightarrow \mathbb{N}$ es parcial computable, $A = \{f(x) : f(x) \downarrow\}$ es c.e.

Demostración.

Sea $\Phi_p = f$. Definamos el programa Q

```
[A]  IF STP(1)(Z, p, T) = 0 GOTO B
      IF  $\Phi_p(Z) = X$  GOTO E
[B]  Z  $\leftarrow$  Z + 1
      IF Z  $\leq$  T GOTO A
      T  $\leftarrow$  T + 1
      Z  $\leftarrow$  0
      GOTO A
```

Notar que $\Psi_Q(X) \downarrow$ si existen Z, T tal que

- ▶ $Z \leq T$
- ▶ STP⁽¹⁾(Z, p, T) es verdadero (i.e. el programa para f termina en T o menos pasos con entrada Z)
- ▶ $X = f(Z)$

$$\Psi_Q(x) = \begin{cases} 0 & \text{si } x \in A \\ \uparrow & \text{si no} \end{cases}$$

Luego A es c.e. □

Caracterizaciones de los conjuntos c.e.

Teorema

Si $A \neq \emptyset$, son equivalentes:

1. A es c.e.
2. A es el rango de una función primitiva recursiva
3. A es el rango de una función computable
4. A es el rango de una función parcial computable

Demostración.

(1 \Rightarrow 2) *Teorema de hoja 124*

(2 \Rightarrow 3) *Trivial*

(3 \Rightarrow 4) *Trivial*

(4 \Rightarrow 1) *Teorema de hoja 125*



Teorema de Rice

$A \subseteq \mathbb{N}$ es un conjunto **de índices** si existe una clase de funciones $\mathbb{N} \rightarrow \mathbb{N}$ parciales computables \mathcal{C} tal que $A = \{x : \Phi_x \in \mathcal{C}\}$

Teorema

Si A es un conjunto de índices tal que $\emptyset \neq A \neq \mathbb{N}$, A no es computable.

Demostración.

Supongamos \mathcal{C} tal que $A = \{x : \Phi_x \in \mathcal{C}\}$ computable. Sean $f \in \mathcal{C}$ y $g \notin \mathcal{C}$ funciones parciales computables.

Sea $h : \mathbb{N}^2 \rightarrow \mathbb{N}$ la siguiente función parcial computable:

$$h(t, x) = \begin{cases} g(x) & \text{si } t \in A \\ f(x) & \text{si no} \end{cases}$$

Por el Teorema de la Recursión, existe e tal que $\Phi_e(x) = h(e, x)$.

- ▶ $e \in A \Rightarrow \Phi_e = g \Rightarrow \Phi_e \notin \mathcal{C} \Rightarrow e \notin A$
- ▶ $e \notin A \Rightarrow \Phi_e = f \Rightarrow \Phi_e \in \mathcal{C} \Rightarrow e \in A$



Aplicaciones del Teorema de Rice

El teorema da una fuente de conjuntos no computables:

- ▶ $\{x : \Phi_x \text{ es total}\}$
- ▶ $\{x : \Phi_x \text{ es creciente}\}$
- ▶ $\{x : \Phi_x \text{ tiene dominio infinito}\}$
- ▶ $\{x : \Phi_x \text{ es primitiva recursiva}\}$

¡Todos son no computables porque todos son conjuntos de índices no triviales!

Ejemplos de conjuntos no c.e.

- ▶ $\overline{K} = \{x : \Phi_x(x) \uparrow\}$ no es c.e.
 - ▶ K es c.e. de modo que si \overline{K} lo fuera, K sería computable
- ▶ $Tot = \{x : \Phi_x \text{ es total}\}$ no es c.e.:
 - ▶ es una diagonalización simple. Supongamos que Tot es c.e.
 - ▶ existe f computable tal que $Tot = \{f(0), f(1), f(2), \dots\}$
 - ▶ entonces existe e tal que $\Phi_e(x) = \Phi_{f(x)}(x) + 1$
 - ▶ como Φ_e es total, $e \in Tot$. De modo que existe u tal que $f(u) = e$
 - ▶ $\Phi_{f(u)}(x) = \Phi_{f(x)}(x) + 1$. Absurdo para $x = u$.
- ▶ $\overline{Tot} = \{x : \Phi_x \text{ no es total}\}$ no es c.e.
 - ▶ parecido a lo que hicimos la clase pasada. Supongamos \overline{Tot} c.e.
 - ▶ existe d tal que $\overline{Tot} = \text{dom } \Phi_d$
 - ▶ definimos el siguiente programa P :

```
[C]   IF STP(1)(X, d, T) = 1 GOTO E
      T ← T + 1
      GOTO C
```

- ▶ sigue igual a lo que vimos la clase pasada

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

Conjuntos más difíciles que el halting problem

- ▶ $K = \{x : \Phi_x(x) \downarrow\}$ no es computable
 - ▶ pero K es c.e.
- ▶ $Tot = \{x : \Phi_x \text{ es total}\}$ no es computable
 - ▶ Tot no es c.e.
 - ▶ \overline{Tot} no es c.e.
- ▶ de alguna forma, Tot es más difícil que K
 - ▶ esto se formaliza dentro de la teoría
 - ▶ no hay tiempo para verlo en esta materia
 - ▶ pero lo estudiamos en **Teoría de la Computabilidad**
 - ▶ se suele dar los primeros cuatrimestres
 - ▶ da 3 puntos como optativa para la licenciatura y se cursa 1 vez por semana
 - ▶ <http://www.glyc.dc.uba.ar/teocomp/>

Computabilidad

Clase 8

Oráculos, reducibilidad de Turing, jerarquía aritmética,
problema de Post

Oráculos

El lenguaje \mathcal{S} se extiende:

- ▶ tiene entradas $X_1, \dots, X_n \in \mathbb{N}$ (como antes) y una entrada especial que se llama **oráculo**
- ▶ en el oráculo no se pone un número natural sino un **conjunto**

$$A \subseteq \mathbb{N}.$$

- ▶ un nuevo término para leer el oráculo seteado en A

$$ORACULO[i] = \begin{cases} 1 & \text{si } i \in A \\ 0 & \text{sino} \end{cases}$$

Todo se puede aritmetizar como antes. Existe un programa universal:

$$\Phi_e^A(x_1, \dots, x_n) = \begin{array}{l} \text{salida del } e\text{-ésimo programa} \\ \text{con entrada } x_1, \dots, x_n \text{ y oráculo } = A \end{array}$$

Tienen más poder de cómputo

Por ejemplo, hay un programa que calcula el halting problem

$$K = \{x : \Phi_x(x) \downarrow\}$$

Pero claro... con oráculo K .

Por ejemplo, hay un programa p tal que

- ▶ $\Phi_p^A = A$ para todo A
- ▶ $\Phi_p^A = \bar{A}$ para todo A
- ▶ $\Phi_p^K = \{x : \text{dom } \Phi_x = \emptyset\}$

Sin embargo, hay problemas que no se pueden resolver aun teniendo el oráculo K :

- ▶ $\{x : \Phi_x \text{ es total}\}$
- ▶ $\{x : \Phi_x^K(x) \downarrow\}$

Reducibilidad de Turing

Para $A, B \subseteq \mathbb{N}$, decimos que $A \leq_T B$ cuando se puede calcular el conjunto A con oráculo B , i.e. existe p tal que

$$\Phi_p^B = A$$

o sea, para todo $x \in \mathbb{N}$

$$\Phi_p^B(x) = A(x) = \begin{cases} 1 & \text{si } x \in A \\ 0 & \text{sino} \end{cases}$$

También se dice que A es B -computable

- ▶ $A \leq_T A$ para todo A
- ▶ $\bar{A} \leq_T A$ para todo A
- ▶ $\{x : \text{dom } \Phi_x = \emptyset\} \leq_T K$
- ▶ $\{x : \Phi_x \text{ es total}\} \not\leq_T K$
- ▶ $\{x : \Phi_x^K(x) \downarrow\} \not\leq_T K$

El salto

Para $A \subseteq \mathbb{N}$ se define el **salto de A** como

$$A' = \{x : \Phi_x^A(x) \downarrow\}$$

Por ejemplo,

- ▶ $\emptyset' = \{x : \Phi_x^\emptyset(x) \downarrow\} = \{x : \Phi_x(x) \downarrow\} = K$
- ▶ $\emptyset'' = \{x : \Phi_x^{\emptyset'}(x) \downarrow\}$
- ▶ $\emptyset''' = \{x : \Phi_x^{\emptyset''}(x) \downarrow\}$
- ▶ $\emptyset^{(n)} = \{x : \Phi_x^{(n-1)}(x) \downarrow\}$

Decimos $A <_T B$ cuando $A \leq_T B$ y $B \not\leq_T A$.

En general, para cualquier $A \subseteq \mathbb{N}$

$$A <_T A' <_T A'' <_T A''' \dots$$

Conjuntos $\leq_T \emptyset$

- ▶ A es **c.e.** sii
 - ▶ existe e tal que $A = \text{dom}\Phi_e^\emptyset$
 - ▶ existe P p.r. tal que $A(x) = 1$ sii $(\exists y) P(x, y)$
 - ▶ se los llama conjuntos Σ_1
- ▶ A es **co-c.e.** sii \bar{A} es c.e. sii
 - ▶ existe e tal que $\bar{A} = \text{dom}\Phi_e^\emptyset$
 - ▶ existe R p.r. tal que $A(x) = 1$ sii $(\forall y) R(x, y)$
 - ▶ se los llama conjuntos Π_1
- ▶ A es **computable** sii $A \leq_T \emptyset$ sii
 - ▶ A es c.e. y co-c.e.
 - ▶ existen P y R p.r. tal que $A(x) = 1$ sii $(\exists y) P(x, y)$ sii $(\forall y) R(x, y)$
 - ▶ se los llama conjuntos Δ_1

Conjuntos $\leq_T \emptyset'$

- ▶ A es \emptyset' -c.e. sii
 - ▶ existe e tal que $A = \text{dom}\Phi_e^{\emptyset'}$
 - ▶ existe P p.r. tal que $A(x) = 1$ sii $(\exists y)(\forall z) P(x, y, z)$
 - ▶ se los llama conjuntos Σ_2
- ▶ A es \emptyset' -co-c.e. sii \bar{A} es \emptyset' -c.e. sii
 - ▶ existe e tal que $\bar{A} = \text{dom}\Phi_e^{\emptyset'}$
 - ▶ existe R p.r. tal que $A(x) = 1$ sii $(\forall y)(\exists z) R(x, y, z)$
 - ▶ se los llama conjuntos Π_2
- ▶ A es \emptyset' -computable sii $A \leq_T \emptyset'$ sii
 - ▶ A es \emptyset' -c.e. y \emptyset' -co-c.e.
 - ▶ existen P y R p.r. tal que $A(x) = 1$ sii $(\exists y)(\forall z) P(x, y, z)$ sii $(\forall y)(\exists z) R(x, y, z)$
 - ▶ se los llama conjuntos Δ_2

Conjuntos $\leq_T \emptyset''$

- ▶ A es \emptyset'' -c.e. sii
 - ▶ existe e tal que $A = \text{dom}\Phi_e^{\emptyset''}$
 - ▶ existe P p.r. tal que
$$A(x) = 1 \quad \text{sii} \quad (\exists y)(\forall z)(\exists w) P(x, y, z, w)$$
 - ▶ se los llama conjuntos Σ_3
- ▶ A es \emptyset'' -co-c.e. sii \bar{A} es \emptyset'' -c.e. sii
 - ▶ existe e tal que $\bar{A} = \text{dom}\Phi_e^{\emptyset''}$
 - ▶ existe R p.r. tal que
$$A(x) = 1 \quad \text{sii} \quad (\forall y)(\exists z)(\forall w) R(x, y, z, w)$$
 - ▶ se los llama conjuntos Π_3
- ▶ sii A es \emptyset'' -computable sii $A \leq_T \emptyset''$
 - ▶ A es \emptyset'' -c.e. y \emptyset'' -co-c.e.
 - ▶ existen P y R p.r. tal que

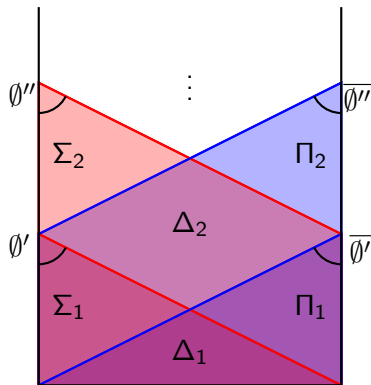
$$\begin{aligned} A(x) = 1 \quad \text{sii} \quad & (\exists y)(\forall z)(\exists w) P(x, y, z, w) \\ & \text{sii} \quad (\forall y)(\exists z)(\forall w) R(x, y, z, w) \end{aligned}$$

- ▶ se los llama conjuntos Δ_3

La jerarquía aritmética

- ▶ A es $\emptyset^{(n)}$ -c.e. sii
 - ▶ existe e tal que $A = \text{dom}\Phi_e^{\emptyset^{(n)}}$
 - ▶ existe P p.r. tal que
$$A(x) = 1 \quad \text{sii} \quad (\exists y)(\forall z)(\exists w) \dots P(x, y, z, w, \dots)$$
 con n alternancias de cuantificadores (empezando con \exists)
 - ▶ se los llama conjuntos Σ_{n+1}
- ▶ A es $\emptyset^{(n)}$ -co-c.e. sii \bar{A} es $\emptyset^{(n)}$ -c.e. sii
 - ▶ existe e tal que $\bar{A} = \text{dom}\Phi_e^{\emptyset^{(n)}}$
 - ▶ existe R p.r. tal que
$$A(x) = 1 \quad \text{sii} \quad (\forall y)(\exists z)(\forall w) \dots R(x, y, z, w, \dots)$$
 con n alternancias de cuantificadores (empezando con \forall)
 - ▶ se los llama conjuntos Π_{n+1}
- ▶ A es $\emptyset^{(n)}$ -computable sii $A \leq_T \emptyset^{(n)}$ sii
 - ▶ A es $\emptyset^{(n)}$ -c.e. y $\emptyset^{(n)}$ -co-c.e.
 - ▶ existen P y R p.r. tal que
$$A(x) = 1 \quad \text{sii} \quad (\exists y)(\forall z)(\exists w) \dots P(x, y, z, w, \dots)$$
$$\text{sii} \quad (\forall y)(\exists z)(\forall w) \dots R(x, y, z, w, \dots)$$
 con n alternancias de cuantificadores.
 - ▶ se los llama conjuntos Δ_{n+1}

La jerarquía aritmética



Problema de Post

Ejemplos de conjuntos c.e.

- ▶ $K = \{x : \Phi_x(x) \downarrow\} \equiv_T \emptyset'$
- ▶ $\emptyset \equiv_T \emptyset$
- ▶ $\{\langle x, y \rangle : \Phi_x(y) \downarrow\} \equiv_T \emptyset'$
- ▶ $\{x : x \text{ es primo}\} \equiv_T \emptyset$
- ▶ $\{\langle x, \langle y, z \rangle \rangle : \Phi_x(y) = z\} \equiv_T \emptyset'$
- ▶ $\mathbb{N} \equiv_T \emptyset$
- ▶ $\{x : \text{dom } \Phi_x \neq \emptyset\} \equiv_T \emptyset'$
- ▶ $\{x : \Phi_x(x) \text{ tarda más de 10 pasos en terminar}\} \equiv_T \emptyset$
- ▶ $\{x : 0 \in \text{dom } \Phi_x\} \equiv_T \emptyset'$

Decimos $A \equiv_T B$ cuando $A \leq_T B$ y $B \leq_T A$.

Problema de Post (1944): ¿Existe un A c.e. tal que $\emptyset <_T A <_T \emptyset'$?

Solución al problema de Post [Muchnik (1956), Friedberg (1957)]

Construir un conjunto A c.e. tal que

1. A es **bajo** (*low*): $A' \equiv_T \emptyset'$

esto garantiza

$$A <_T A' \equiv_T \emptyset'$$

2. A es **simple**: \bar{A} es infinito y \bar{A} no contiene ningún conjunto infinito c.e.

no puede ser $A \equiv_T \emptyset$. Si lo fuera, \bar{A} sería c.e. e infinito.

Materia optativa

Todo esto y mucho más en...

Teoría de la Computabilidad

- ▶ se suele dar los primeros cuatrimestres
- ▶ da 3 puntos como optativa para la licenciatura y se cursa 1 vez por semana
- ▶ <http://www.glyc.dc.uba.ar/teocomp/>

Lógica Proposicional

Clase 1

Lenguaje de lógica proposicional, semántica, tautología, consecuencia semántica, conjunto satisfacible, sistema axiomático SP , consecuencia sintáctica

El lenguaje \mathcal{P}

- ▶ **símbolos** $p \quad ' \quad \neg \quad \rightarrow \quad (\quad)$
 - ▶ p, p', p'', p''', \dots son **símbolos proposicionales**
- ▶ **fórmulas**
 1. todo símbolo proposicional es una fórmula
 2. si φ es una fórmula entonces $\neg\varphi$ es una fórmula
 3. si φ y ψ son fórmulas entonces $(\varphi \rightarrow \psi)$ es una fórmula
 4. nada más es una fórmula
- ▶ **convenciones**
 - ▶ escribimos q por p' r por p'' s por $p''' \dots$
 - ▶ escribimos $(\varphi \vee \psi)$ en lugar de $(\neg\varphi \rightarrow \psi)$
 - ▶ escribimos $(\varphi \wedge \psi)$ en lugar de $\neg(\varphi \rightarrow \neg\psi)$
 - ▶ escribimos φ en lugar de (φ) cuando convenga
- ▶ llamamos **PROP** al conjunto de todos los símbolos proposicionales
- ▶ llamamos **FORM** al conjunto de todas las fórmulas

Semántica

Una **interpretación** es una función

$$v : \text{PROP} \rightarrow \{0, 1\}$$

A v también se la llama **valuación**.

Definimos la noción de **verdad de una fórmula para una valuación**.

Si $\varphi \in \text{FORM}$ y v es una valuación, notamos

- ▶ $v \models \varphi$ si φ es verdadera para v
- ▶ $v \not\models \varphi$ si φ es falsa para v

La definición de \models es recursiva:

1. si $p \in \text{PROP}$, $v \models p$ sii $v(p) = 1$
2. $v \models \neg\psi$ sii $v \not\models \psi$
3. $v \models (\psi \rightarrow \rho)$ sii $v \not\models \psi$ o $v \models \rho$

Semántica

Observar que, por la convención,

5. $v \models (\psi \wedge \rho)$ sii $v \models \psi$ y $v \models \rho$

6. $v \models (\psi \vee \rho)$ sii $v \models \psi$ o $v \models \rho$

Por ejemplo, si $v(p) = 1$, $v(q) = 0$, $v(r) = 1$

▶ $v \models (p \rightarrow r)$

▶ $v \models (q \rightarrow r)$

▶ $v \not\models \neg p$

▶ $v \not\models (p \wedge q)$

Tautologías y método de decisión

Una fórmula φ es una **tautología** ($\models \varphi$) si φ es verdadera para toda interpretación, i.e. para toda valuación v , $v \models \varphi$.

Proposición

Sea $\varphi \in \text{FORM}$ y sean v y w son dos valuaciones tal que $v(p) = w(p)$ para toda variable proposicional que aparece en φ . Entonces $v \models \varphi$ sii $w \models \varphi$.

Existe un **método de decisión** para saber si φ es tautología o no:

- ▶ supongamos que φ tiene variables proposicionales p_1, \dots, p_n
- ▶ sea $\mathcal{P}(\{p_1, \dots, p_n\}) = \{V_1, \dots, V_{2^n}\}$
- ▶ para $i \in \{1, \dots, 2^n\}$ definimos $v_i(p) = \begin{cases} 1 & \text{si } p \in V_i \\ 0 & \text{si no} \end{cases}$
- ▶ φ es tautología sii

$$v_i \models \varphi \text{ para todo } i \in \{1, \dots, 2^n\}$$

Consecuencia semántica y conjunto satisfacible

Sea $\Gamma \subseteq \text{FORM}$ y $\varphi \in \text{FORM}$

φ es **consecuencia semántica** de Γ ($\Gamma \models \varphi$) si para toda interpretación v :

si $v \models \psi$ para toda $\psi \in \Gamma$, entonces $v \models \varphi$
lo notamos $v \models \Gamma$

Γ es **satisfacible** si existe una interpretación v tal que $v \models \psi$ para toda $\psi \in \Gamma$ (i.e. tal que $v \models \Gamma$)

Por ejemplo

- ▶ $\{q\} \models q$
- ▶ $\{q\} \models p \rightarrow q$
- ▶ $\{r, p \vee q\} \not\models p$
- ▶ $\{r, p \vee q\} \not\models s$
- ▶ \emptyset es satisfacible
- ▶ $\{p, q\}$ es satisfacible
- ▶ $\{\neg p, p \wedge q\}$ no es satisfacible
- ▶ $\{p, p \rightarrow q, \neg q\}$ no es satisfacible

Algunos resultados sobre \models

Proposición

1. $\emptyset \models \varphi$ sii $\models \varphi$ (i.e. φ es tautología)
2. si $\models \varphi$ entonces $\Gamma \models \varphi$
3. $\{\varphi\} \models \varphi$
4. si $\Gamma \subseteq \Delta$ y $\Gamma \models \varphi$ entonces $\Delta \models \varphi$
5. si $\Gamma \models \varphi$ y $\Gamma \models \varphi \rightarrow \psi$ entonces $\Gamma \models \psi$

Demostración de 5.

- ▶ sea v una interpretación tal que $v \models \Gamma$
- ▶ sabemos $v \models \varphi$
- ▶ sabemos $v \models \varphi \rightarrow \psi$
- ▶ concluimos $v \models \psi$



Mecanismo deductivo SP

- ▶ **axiomas.** Sean $\varphi, \psi, \rho \in \text{FORM}$

$$SP1 \quad \varphi \rightarrow (\psi \rightarrow \varphi)$$

$$SP2 \quad (\varphi \rightarrow (\psi \rightarrow \rho)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \rho))$$

$$SP3 \quad (\neg\varphi \rightarrow \neg\psi) \rightarrow (\psi \rightarrow \varphi)$$

- ▶ **regla de inferencia**

MP Sean $\varphi, \psi \in \text{FORM}$. ψ es una consecuencia inmediata de $\varphi \rightarrow \psi$ y φ

Una **demostración** de φ en SP es una cadena finita y no vacía

$$\varphi_1, \dots, \varphi_n$$

de fórmulas de P tal que $\varphi_n = \varphi$ y

- ▶ φ_i es un axioma o
- ▶ φ_i es una consecuencia inmediata de $\varphi_k, \varphi_l, k, l < i$

En este caso, decimos que φ es un **teorema** ($\vdash \varphi$)

Ejemplo: demostración de $p \rightarrow p$

Recordar

$$\text{SP1 } \varphi \rightarrow (\psi \rightarrow \varphi)$$

$$\text{SP2 } (\varphi \rightarrow (\psi \rightarrow \rho)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \rho))$$

$$\text{SP3 } (\neg\varphi \rightarrow \neg\psi) \rightarrow (\psi \rightarrow \varphi)$$

MP Sean $\varphi, \psi \in \text{FORM}$. ψ es una consecuencia inmediata de $\varphi \rightarrow \psi$ y φ

Demostración:

- | | | |
|----|---|----------|
| 1. | $p \rightarrow ((p \rightarrow p) \rightarrow p)$ | SP1 |
| 2. | $(p \rightarrow ((p \rightarrow p) \rightarrow p)) \rightarrow ((p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow p))$ | SP2 |
| 3. | $(p \rightarrow (p \rightarrow p)) \rightarrow (p \rightarrow p)$ | MP 1 y 2 |
| 4. | $p \rightarrow (p \rightarrow p)$ | SP1 |
| 5. | $p \rightarrow p$ | MP 3 y 4 |

Concluimos $\vdash p \rightarrow p$ (i.e. $p \rightarrow p$ es un teorema)

Consecuencia sintáctica

Sea $\Gamma \subseteq \text{FORM}$ y $\varphi \in \text{FORM}$

φ es una **consecuencia sintáctica** de Γ ($\Gamma \vdash \varphi$) si existe una cadena finita y no vacía

$$\varphi_1, \dots, \varphi_n$$

de fórmulas de P tal que $\varphi_n = \varphi$ y

- ▶ φ_i es un axioma o
- ▶ $\varphi_i \in \Gamma$ o
- ▶ φ_i es una consecuencia inmediata de $\varphi_k, \varphi_l, k, l < i$

Aquí, $\varphi_1, \dots, \varphi_n$ se llama **derivación** de φ a partir de Γ . Γ se llama **teoría**. Decimos que φ es un **teorema de la teoría** Γ .

Correctitud de SP

Teorema

Si $\Gamma \vdash \varphi$ entonces $\Gamma \models \varphi$ (i.e. si es teorema de la teoría Γ , es válido en toda interpretación de Γ).

Demostración.

Supongamos $\Gamma \vdash \varphi$. Es decir, existe una cadena finita y no vacía

$$\varphi_1, \dots, \varphi_n$$

de fórmulas de P tal que $\varphi_n = \varphi$ y

- ▶ φ_i es un axioma o
- ▶ $\varphi_i \in \Gamma$ o
- ▶ φ_i es una consecuencia inmediata de $\varphi_k, \varphi_l, k, l < i$

Demostramos que $\Gamma \models \varphi$ por inducción en n (la longitud de la demostración). Detalles a continuación.



Demostración de Correctitud de SP

Propiedad a demostrar:

$P(n) =$ “si $\varphi_1, \dots, \varphi_n = \varphi$ es una derivación de φ a partir de Γ entonces $v \models \Gamma \Rightarrow v \models \varphi$ ”

Demostramos que vale $P(n)$ por inducción en n .

- caso base.** Veamos que vale $P(1)$. Sup. v tal que $v \models \Gamma$. Queremos ver que $v \models \Gamma \Rightarrow v \models \varphi$. Hay 2 posibilidades
 - φ is axioma de SP : en este caso, $v \models \varphi$;
 - $\varphi \in \Gamma$: en este caso, también $v \models \varphi$.
- paso inductivo.** Sup. v tal que $v \models \Gamma$. Sup. que vale $P(m)$ para todo $m \leq n$. Queremos ver que vale $P(n+1)$. Sup. $\varphi_1, \dots, \varphi_n, \varphi_{n+1} = \varphi$ es una derivación de φ a partir de Γ . Hay 3 posibilidades
 - φ is axioma de SP : igual que en caso base;
 - $\varphi \in \Gamma$: igual que en caso base;
 - φ es consecuencia inmediata de φ_i y $\varphi_j = \varphi_i \rightarrow \varphi$ ($i, j \leq n$). Por HI ($P(i)$ y $P(j)$), sabemos $v \models \varphi_i$ y $v \models \varphi_i \rightarrow \varphi$. Entonces necesariamente $v \models \varphi$.

Ejemplos

- ▶ $\Gamma_1 = \{p\} \vdash p$
 1. p $p \in \Gamma_1$

- ▶ $\Gamma_2 = \{p\} \vdash \varphi \rightarrow p$
 1. p $p \in \Gamma_2$
 2. $p \rightarrow (\varphi \rightarrow p)$ SP1
 3. $\varphi \rightarrow p$ MP 1 y 2

- ▶ $\Gamma_3 = \{p\} \not\vdash q$

porque $\Gamma_3 \not\models q$ (considerar $v(p) = 1; v(q) = 0$)

- ▶ $\Gamma_4 = \{p, \neg p\} \vdash \varphi$
 1. $\neg p \rightarrow (\neg \varphi \rightarrow \neg p)$ SP1
 2. $\neg p$ $\neg p \in \Gamma_4$
 3. $\neg \varphi \rightarrow \neg p$ MP 1 y 2
 4. $(\neg \varphi \rightarrow \neg p) \rightarrow (p \rightarrow \varphi)$ SP3
 5. $p \rightarrow \varphi$ MP 3 y 4
 6. p $p \in \Gamma_4$
 7. φ MP 5 y 6

Conjuntos y sistemas consistentes

$\Gamma \subseteq \text{FORM}$ es **consistente** si no existe $\varphi \in \text{FORM}$ tal que

$$\Gamma \vdash \varphi \quad \text{y} \quad \Gamma \vdash \neg\varphi$$

Un sistema S es **consistente** si no existe $\varphi \in \text{FORM}$ tal que

$$\vdash_S \varphi \quad \text{y} \quad \vdash_S \neg\varphi$$

Teorema

El sistema SP es consistente.

Demostración.

- ▶ sea v cualquier valuación
- ▶ por correctitud, todo teorema de SP es verdadero para v

$$\vdash \varphi \Rightarrow v \models \varphi \Rightarrow v \not\models \neg\varphi \Rightarrow \not\vdash \neg\varphi$$

- ▶ luego no puede pasar que φ y $\neg\varphi$ sean teoremas



Algunos resultados sobre \vdash

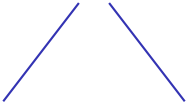
Proposición

1. $\emptyset \vdash \varphi$ *sii* $\vdash \varphi$ (i.e. φ es teorema)
2. *si* $\vdash \varphi$ *entonces* $\Gamma \vdash \varphi$
3. $\{\varphi\} \vdash \varphi$
4. *si* $\Gamma \subseteq \Delta$ *y* $\Gamma \vdash \varphi$ *entonces* $\Delta \vdash \varphi$
5. *si* $\Gamma \vdash \varphi$ *y* $\Gamma \vdash \varphi \rightarrow \psi$ *entonces* $\Gamma \vdash \psi$

Si reemplazamos \vdash por \models , obtenemos los mismos resultados (ver hoja 150)

Resumen

lenguaje P



semántica

tautología

(verdadera en toda interpretación)

consecuencia semántica \models

conjunto satisfacible

(existe modelo para todos sus elementos)

método deductivo

teorema

(tiene demostración en SP)

consecuencia sintáctica \vdash

conjunto consistente

(no permite probar φ y $\neg\varphi$)

Notas sobre computabilidad

Se pueden codificar las fórmulas de P con números naturales.

- ▶ a cada fórmula φ se le asigna un número $\#\varphi > 0$
- ▶ cada número positivo representa una única fórmula

Se puede decidir algorítmicamente si una fórmula es un axioma o no

- ▶ es computable la función

$$ax(x) = \begin{cases} 1 & \text{si la fórmula de número } x \text{ es un axioma de } SP \\ 0 & \text{si no} \end{cases}$$

Se puede decidir algorítmicamente si una fórmula es consecuencia inmediata de otras dos

- ▶ es computable la función

$$mp(x, y, z) = \begin{cases} 1 & \text{si la fórmula de número } z \text{ es consecuencia} \\ & \text{inmediata de las fórmulas de números } x \text{ e } y \\ 0 & \text{si no} \end{cases}$$

Notas sobre computabilidad

Las demostraciones son listas (finitas) de fórmulas.

- ▶ la demostración $\varphi_1 \dots \varphi_n$ se codifica como $[\#\varphi_1, \dots, \#\varphi_n]$

Se puede decidir algorítmicamente si una lista de fórmulas es una demostración válida o no

- ▶ es computable la función

$$dem(x) = \begin{cases} 1 & x \text{ es una demostración válida} \\ 0 & \text{si no} \end{cases}$$

- ▶ en efecto,

$$\begin{aligned} dem(x) &= (\forall k \in \{1, \dots, |x|\})[ax(x[k]) \vee cons(x, k)] \\ cons(x, k) &= (\exists i, j \in \{1, \dots, k-1\})[mp(x[i], x[j], x[k])] \end{aligned}$$

Notas sobre computabilidad

- ▶ considerar el siguiente programa P :

[A] IF $dem(D) = 1 \wedge D[|D|] = X$ GOTO E
 $D \leftarrow D + 1$
GOTO A

- ▶ P busca una demostración para la fórmula con número X
 - ▶ si la encuentra, se detiene
 - ▶ si no, se indefine
- ▶ $\vdash \varphi$ sii $\Psi_P(\# \varphi) \downarrow$, o equivalentemente

φ es teorema sii $\# \varphi \in \text{dom} \Psi_P$

- ▶ el conjunto de teoremas de SP es c.e.
- ▶ esto pasa en general para cualquier sistema axiomático
 - ▶ es decir, cualquier sistema de deducción con un conjunto computable de axiomas y reglas de inferencia computables tiene un conjunto de teoremas c.e.
- ▶ ¿será computable el conjunto de teoremas de SP ?

Lógica Proposicional

Clase 2

Teorema de la deducción, lema de Lindenbaum, completitud de SP , compacidad

Plan

- ▶ vimos que SP es correcto: $\Gamma \vdash \varphi \Rightarrow \Gamma \models \varphi$

Esto prueba

- ▶ Γ satisfacible $\Rightarrow \Gamma$ consistente

- ▶ ahora veremos que SP es completo: $\Gamma \vdash \varphi \Leftarrow \Gamma \models \varphi$

Para esto:

- ▶ Lema de Lindenbaum
- ▶ Γ satisfacible $\Leftarrow \Gamma$ consistente

- ▶ consecuencia: Teorema de Compacidad

El Teorema de la Deducción

Teorema

Si $\Gamma \cup \{\varphi\} \vdash \psi$ entonces $\Gamma \vdash \varphi \rightarrow \psi$

Demostración.

Por inducción en la longitud de la demostración de $\Gamma \cup \{\varphi\} \vdash \psi$.

Supongamos que

$$\varphi_1, \dots, \varphi_n$$

es una derivación de ψ ($= \varphi_n$) a partir de $\Gamma \cup \{\varphi\}$.

- ▶ caso base ($n = 1$)
- ▶ paso inductivo
 - ▶ **HI:** para toda derivación de ψ' a partir de $\Gamma \cup \{\varphi\}$ de longitud $< n$ tenemos $\Gamma \vdash \varphi \rightarrow \psi'$
 - ▶ probamos que para una demostración de longitud n de $\Gamma \cup \{\varphi\} \vdash \psi$ tenemos $\Gamma \vdash \varphi \rightarrow \psi$.



Demostración del Teorema de la Deducción (caso base)

Supongamos

- ▶ $\varphi_1, \dots, \varphi_n$ es una derivación de ψ a partir de $\Gamma \cup \{\varphi\}$
- ▶ $n = 1$ (i.e. la derivación es una sola fórmula $\varphi_1 = \psi$)

Queremos ver que $\Gamma \vdash \varphi \rightarrow \psi$. Hay 3 posibilidades:

1. ψ es un axioma de SP

$$\left. \begin{array}{ll} 1. & \psi \qquad \psi \text{ es axioma} \\ 2. & \psi \rightarrow (\varphi \rightarrow \psi) \quad \text{SP1} \\ 3. & \varphi \rightarrow \psi \quad \text{MP 1,2} \end{array} \right\} \vdash \varphi \rightarrow \psi$$

2. $\psi \in \Gamma$

$$\left. \begin{array}{ll} 1. & \psi \qquad \psi \in \Gamma \\ 2. & \psi \rightarrow (\varphi \rightarrow \psi) \quad \text{SP1} \\ 3. & \varphi \rightarrow \psi \quad \text{MP 1,2} \end{array} \right\} \Gamma \vdash \varphi \rightarrow \psi$$

3. $\psi = \varphi$

vimos que $\vdash p \rightarrow p$.

la misma demostración sirve para probar $\vdash \varphi \rightarrow \varphi$

Demostración del Teorema de la Deducción (paso inductivo)

Supongamos

$$\varphi_1, \dots, \varphi_n$$

es una derivación de ψ a partir de $\Gamma \cup \{\varphi\}$

HI: para toda derivación de ψ' a partir de $\Gamma \cup \{\varphi\}$ de longitud $< n$ tenemos $\Gamma \vdash \varphi \rightarrow \psi'$

Queremos ver que $\Gamma \vdash \varphi \rightarrow \psi$. Hay 4 posibilidades:

1. ψ es un axioma de SP: igual que en en caso base
2. $\psi \in \Gamma$: igual que en en caso base
3. $\psi = \varphi$: igual que en en caso base
4. ψ se infiere por MP de φ_i y φ_j ($i, j < n$)
 - ▶ sin pérdida de generalidad, $\varphi_j = \varphi_i \rightarrow \psi$
 - ▶ $\Gamma \cup \{\varphi\} \vdash \varphi_i$ y la derivación tiene longitud $< n$
 - ▶ por HI $\Gamma \vdash \varphi \rightarrow \varphi_i$
 - ▶ $\Gamma \cup \{\varphi\} \vdash \varphi_j$ y la derivación tiene longitud $< n$
 - ▶ por HI $\Gamma \vdash \varphi \rightarrow \varphi_j$, i.e. $\Gamma \vdash \varphi \rightarrow (\varphi_i \rightarrow \psi)$
 - ▶ sabemos (SP2) $\vdash ((\varphi \rightarrow (\varphi_i \rightarrow \psi)) \rightarrow ((\varphi \rightarrow \varphi_i) \rightarrow (\varphi \rightarrow \psi)))$
 - ▶ por MP 2 veces $\Gamma \vdash \varphi \rightarrow \psi$

Conjuntos consistentes

Proposición

1. $\Gamma \cup \{\neg\varphi\}$ es inconsistente sii $\Gamma \vdash \varphi$
2. $\Gamma \cup \{\varphi\}$ es inconsistente sii $\Gamma \vdash \neg\varphi$

Demostración de 1.

(\Leftarrow) $\Gamma \vdash \varphi \Rightarrow \left. \begin{array}{l} \Gamma \cup \{\neg\varphi\} \vdash \varphi \\ \text{trivialmente} \quad \Gamma \cup \{\neg\varphi\} \vdash \neg\varphi \end{array} \right\} \Gamma \cup \{\neg\varphi\} \text{ es inconsistente}$

(\Rightarrow) existe ψ tal que $\Gamma \cup \{\neg\varphi\} \vdash \psi$ y $\Gamma \cup \{\neg\varphi\} \vdash \neg\psi$
por el Teorema de la Deducción,

$$\Gamma \vdash \neg\varphi \rightarrow \psi \quad \text{y} \quad \Gamma \vdash \neg\varphi \rightarrow \neg\psi$$

se puede ver que (ejercicio)

$$\vdash (\neg\varphi \rightarrow \psi) \rightarrow ((\neg\varphi \rightarrow \neg\psi) \rightarrow \varphi)$$

por MP 2 veces tenemos $\Gamma \vdash \varphi$

Satisfacible \Rightarrow consistente

Teorema

Si $\Gamma \subseteq \text{FORM}$ es satisfacible entonces Γ es consistente.

Demostración.

- ▶ supongamos v tal que $v \models \Gamma$ pero Γ es inconsistente
- ▶ existe ψ tal que $\Gamma \vdash \psi$ y $\Gamma \vdash \neg\psi$
- ▶ por correctitud de *SP*, $\Gamma \models \psi$ y $\Gamma \models \neg\psi$
- ▶ $v \models \psi$ y $v \models \neg\psi$



Lema de Lindenbaum

$\Gamma \subseteq \text{FORM}$ es **maximal consistente (m.c.)** en SP si

- ▶ Γ es consistente y
- ▶ para toda fórmula φ
 - ▶ $\varphi \in \Gamma$ o
 - ▶ existe ψ tal que $\Gamma \cup \{\varphi\} \vdash \psi$ y $\Gamma \cup \{\varphi\} \vdash \neg\psi$

Lema

Si $\Gamma \subseteq \text{FORM}$ es consistente, existe Γ' m.c. tal que $\Gamma \subseteq \Gamma'$.

Demostración del Lema de Lindenbaum (obtener $\Gamma' \supseteq \Gamma$ m.c.)

Enumeramos todas las fórmulas: $\varphi_1, \varphi_2, \dots$. Definimos

- ▶ $\Gamma_0 = \Gamma$
- ▶ $\Gamma_{n+1} = \begin{cases} \Gamma_n \cup \{\varphi_{n+1}\} & \text{si } \Gamma_n \cup \{\varphi_{n+1}\} \text{ es consistente} \\ \Gamma_n & \text{si no} \end{cases}$
- ▶ $\Gamma' = \bigcup_{i \geq 0} \Gamma_i$

Tenemos

1. $\Gamma' \supseteq \Gamma$
2. cada Γ_i es consistente
3. Γ' es consistente
 - ▶ si no, existe ψ tal que $\Gamma' \vdash \psi$ y $\Gamma' \vdash \neg\psi$
 - ▶ en ambas derivaciones aparecen únicamente $\{\gamma_1, \dots, \gamma_k\} \subseteq \Gamma'$.
 - ▶ sea j suficientemente grande tal que $\{\gamma_1, \dots, \gamma_k\} \subseteq \Gamma_j$
 - ▶ entonces $\Gamma_j \vdash \psi$ y $\Gamma_j \vdash \neg\psi$; contradice 2
4. Γ' es maximal
 - ▶ supongamos $\varphi \notin \Gamma'$. Debe existir n tal que $\varphi_{n+1} = \varphi$
 - ▶ $\varphi_{n+1} \notin \Gamma_{n+1}$, entonces $\Gamma_n \cup \{\varphi_{n+1}\}$ es inconsistente
 - ▶ luego $\Gamma' \cup \{\varphi_{n+1}\}$ es inconsistente (pues $\Gamma' \supseteq \Gamma_n$)

Conjuntos maximales consistentes

Proposición

Si Γ' es m.c. entonces para toda $\varphi \in \text{FORM}$, o bien $\varphi \in \Gamma'$ o bien $\neg\varphi \in \Gamma'$.

Demostración.

- ▶ no puede ser que φ y $\neg\varphi$ estén en Γ' porque sería inconsistente
- ▶ supongamos que ninguna está. Como Γ' es maximal y por Proposición de la hoja 168,

$$\left. \begin{array}{l} \Gamma' \cup \{\varphi\} \text{ es inconsistente} \Rightarrow \Gamma' \vdash \neg\varphi \\ \Gamma' \cup \{\neg\varphi\} \text{ es inconsistente} \Rightarrow \Gamma' \vdash \varphi \end{array} \right\} \Gamma' \text{ inconsistente}$$

□

Proposición

Sea Γ' m.c. $\Gamma' \vdash \varphi$ sii $\varphi \in \Gamma'$.

Consistente \Rightarrow satisficible

Teorema

Si $\Gamma \subseteq \text{FORM}$ es consistente entonces Γ es satisficible.

Demostración.

Dado Γ consistente, construimos $\Gamma' \supseteq \Gamma$ m.c. (Lindenbaum)

Definimos la interpretación v tal que

$$v(p) = 1 \text{ sii } p \in \Gamma'$$

Veamos $v \models \varphi$ sii $\varphi \in \Gamma'$ por inducción en la **complejidad** de φ (i.e. cantidad de \neg o \rightarrow que aparecen en φ)

- ▶ caso base: $\varphi = p$. Trivial por definición de v .
- ▶ paso inductivo:

HI: $v \models \varphi$ sii $\varphi \in \Gamma'$ para toda φ de complejidad $< m$

Sea φ de complejidad m . Hay 2 casos:

1. $\varphi = \neg\psi$
2. $\varphi = \psi \rightarrow \rho$

Demostración de consistente \Rightarrow satisficible (caso $\varphi = \neg\psi$)

HI: $v \models \varphi$ sii $\varphi \in \Gamma'$ para toda φ de complejidad $< m$

$\varphi = \neg\psi$ tiene complejidad m .

Quiero probar que $v \models \varphi$ sii $\varphi \in \Gamma'$

$$(\Rightarrow) v \models \varphi \Rightarrow v \not\models \psi \stackrel{\text{HI}}{\Rightarrow} \psi \notin \Gamma' \Rightarrow \neg\psi \in \Gamma' \Rightarrow \varphi \in \Gamma'$$

$$(\Leftarrow) \varphi \in \Gamma' \Rightarrow \psi \notin \Gamma' \stackrel{\text{HI}}{\Rightarrow} v \not\models \psi \Rightarrow v \models \neg\psi \Rightarrow v \models \varphi$$

Demostración de consistente \Rightarrow satisficible (caso $\varphi = \psi \rightarrow \rho$)

HI: $v \models \varphi$ sii $\varphi \in \Gamma'$ para toda φ de complejidad $< m$

$\varphi = \psi \rightarrow \rho$ tiene complejidad m .

Quiero probar que $v \models \varphi$ sii $\varphi \in \Gamma'$

(\Rightarrow) $v \models \varphi \Rightarrow v \models (\psi \rightarrow \rho) \Rightarrow v \not\models \psi$ o $v \models \rho$

► $v \not\models \psi \stackrel{\text{HI}}{\Rightarrow} \psi \notin \Gamma' \Rightarrow \neg\psi \in \Gamma' \Rightarrow \Gamma' \vdash \neg\psi$

sabemos $\vdash \neg\psi \rightarrow (\psi \rightarrow \rho)$

por MP $\Gamma' \vdash \psi \rightarrow \rho$

entonces $\psi \rightarrow \rho \in \Gamma'$

► $v \models \rho \stackrel{\text{HI}}{\Rightarrow} \rho \in \Gamma' \Rightarrow \Gamma' \vdash \rho$

sabemos por SP1 que $\vdash \rho \rightarrow (\psi \rightarrow \rho)$

por MP $\Gamma' \vdash \psi \rightarrow \rho$

entonces $\psi \rightarrow \rho \in \Gamma'$

(ejercicio)

(\Leftarrow) $v \not\models \varphi \Rightarrow v \models \psi$ y $v \not\models \rho \stackrel{\text{HI}}{\Rightarrow} \psi \in \Gamma'$ y $\rho \notin \Gamma'$

$\psi \in \Gamma'$ y $\neg\rho \in \Gamma' \Rightarrow \Gamma' \vdash \psi$ y $\Gamma' \vdash \neg\rho$

sabemos $\vdash \psi \rightarrow (\neg\rho \rightarrow \neg(\psi \rightarrow \rho))$

aplicando MP 2 veces, $\Gamma' \vdash \neg(\psi \rightarrow \rho)$

por lo tanto $\neg(\psi \rightarrow \rho) \in \Gamma'$

entonces $\psi \rightarrow \rho \notin \Gamma'$

(ejercicio)

Teorema de Completitud (fuerte)

Probamos que

- ▶ Γ consistente sii Γ satisfacible (hojas 169 y 173)
- ▶ $\Gamma \cup \{\neg\varphi\}$ es inconsistente sii $\Gamma \vdash \varphi$ (hoja 168)

Teorema

Si $\Gamma \models \varphi$ entonces $\Gamma \vdash \varphi$.

Demostración.

- ▶ supongamos $\Gamma \models \varphi$
- ▶ $\Gamma \cup \{\neg\varphi\}$ es insatisfacible
- ▶ $\Gamma \cup \{\neg\varphi\}$ es inconsistente
- ▶ $\Gamma \vdash \varphi$



Consecuencias del Teorema de Completitud

Corolario

$\Gamma \vdash \varphi$ *sii* $\Gamma \models \varphi$

Corolario

$\vdash \varphi$ *sii* $\models \varphi$ (i.e. φ es un teorema de SP *sii* es tautología)

Teorema (Compacidad)

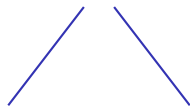
Sea $\Gamma \subseteq \text{FORM}$. Si todo subconjunto finito de Γ es satisfacible, entonces Γ es satisfacible.

Demostración.

- ▶ supongamos Γ insatisfacible
- ▶ Γ es inconsistente
- ▶ existe ψ tal que $\Gamma \vdash \psi$ y $\Gamma \vdash \neg\psi$
- ▶ se usan solo finitos axiomas de Γ
- ▶ existe $\Delta \subseteq \Gamma$ finito tal que $\Delta \vdash \psi$ y $\Delta \vdash \neg\psi$
- ▶ Δ es inconsistente
- ▶ Δ es insatisfacible

Resumen

lenguaje P



semántica

método deductivo

tautología

(verdadera en toda interpretación)



teorema

(tiene demostración en SP)

consecuencia semántica \models



consecuencia sintáctica \vdash

conjunto satisfacible

(existe modelo para todos sus elementos)



conjunto consistente

(no permite probar φ y $\neg\varphi$)

Notas sobre computabilidad

Habíamos visto que el conjunto de teoremas de SP es c.e.

Vemos que, de hecho, es computable:

método de decisión = tablas de verdad

$$\vdash \varphi \quad \text{sii} \quad \models \varphi$$

$\vdash \varphi$ sii en la tabla de verdad de φ solo hay 1s en la última columna

Lógica de Primer Orden

Clase 1

Lenguaje de lógica de primer orden, términos, fórmulas, variables libres y ligadas, interpretación, valuación, niveles de verdad, consecuencia semántica

Lenguajes de primer orden

- ▶ símbolos lógicos y auxiliares: $x \quad ' \quad \forall \quad \neg \quad \rightarrow \quad (\quad)$
 - ▶ x, x', x'', x''', \dots son **variables**
 - ▶ **VAR** es el conjunto de variables
 - ▶ \forall se llama **cuantificador universal**
- ▶ símbolos de cada lenguaje particular $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$, donde
 - ▶ \mathcal{C} es un conjunto de **símbolos de constantes** (puede ser $\mathcal{C} = \emptyset$)
 - ▶ \mathcal{F} es un conjunto de **símbolos de funciones** (puede ser $\mathcal{F} = \emptyset$)
 - ▶ \mathcal{P} es un conjunto de **símbolos de predicados** ($\mathcal{P} \neq \emptyset$)

Términos

Para un lenguaje fijo \mathcal{L} , definimos los **términos de \mathcal{L}** :

1. toda variable es un término
2. todo símbolo de constante de \mathcal{L} es un término
3. si f es un símbolo de función n -ádico de \mathcal{L} y t_1, \dots, t_n son términos de \mathcal{L} , entonces $f(t_1, \dots, t_n)$ es un término de \mathcal{L}
4. nada más es un término de \mathcal{L}

TERM(\mathcal{L}) es el conjunto de todos los términos del lenguaje \mathcal{L}

Un término es **cerrado** si no tiene variables.

Por ejemplo, para $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$, con $\mathcal{C} = \{c, d\}$, $\mathcal{F} = \{f\}$ y $\mathcal{P} = \{R\}$ (f de aridad 3, R binario) son términos:

$$c, d, x, f(c, d, x'), f(c, f(x''', x'', x''), x')$$

Fórmulas

Para un lenguaje fijo \mathcal{L} , definimos las **fórmulas de \mathcal{L}** :

1. si P es un símbolo de predicado n -ádico de \mathcal{L} y t_1, \dots, t_n son términos de \mathcal{L} , entonces $P(t_1, \dots, t_n)$ es una fórmula de \mathcal{L} (atómica)
2. si φ es una fórmula de \mathcal{L} entonces $\neg\varphi$ es una fórmula de \mathcal{L}
3. si φ y ψ son fórmulas de \mathcal{L} entonces $(\varphi \rightarrow \psi)$ es una fórmula de \mathcal{L}
4. si φ es una fórmula de \mathcal{L} y x una variable entonces $(\forall x)\varphi$ es una fórmula de \mathcal{L}
5. nada más es una fórmula de \mathcal{L}

FORM(\mathcal{L}) es el conjunto de todas las fórmulas del lenguaje \mathcal{L}

Por ejemplo, para $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$, con $\mathcal{C} = \{c, d\}$, $\mathcal{F} = \{f\}$ y $\mathcal{P} = \{R\}$ (f de aridad 3, R binario) son fórmulas:

$$R(d, x') \quad , \quad (\forall x') R(d, x'') \quad , \quad (\forall x'') R(f(x'', x', x'''), d)$$

Convenciones

- ▶ usamos x, y, z, \dots para variables
- ▶ usamos a, b, c, d, \dots para símbolos de constante
- ▶ usamos f, g, h, \dots para símbolos de función (la aridad siempre va a quedar clara del contexto)
- ▶ usamos P, Q, R, \dots para símbolos de predicado (la aridad siempre va a quedar clara del contexto)
- ▶ escribimos $(\exists x)\varphi$ en lugar de $\neg(\forall x)\neg\varphi$
- ▶ escribimos $(\varphi \vee \psi)$ en lugar de $(\neg\varphi \rightarrow \psi)$
- ▶ escribimos $(\varphi \wedge \psi)$ en lugar de $\neg(\varphi \rightarrow \neg\psi)$
- ▶ escribimos φ en lugar de (φ) cuando convenga

Variables libres y ligadas

- ▶ una **aparición** de una variable x en una fórmula está **ligada** si está dentro del alcance de un cuantificador. En caso contrario, dicha aparición está **libre**.
- ▶ una variable está **libre** en una fórmula si todas sus apariciones están libres.
- ▶ una variable está **ligada** en una fórmula si todas sus apariciones están ligadas.
- ▶ una fórmula es una **sentencia** si todas las variables son ligadas (es decir, no hay apariciones libres de variables)

Por ejemplo, (para un lenguaje con un símbolo de predicado binario P)

- ▶ en $P(x, y)$, x está libre
- ▶ en $(\forall y) P(x, y)$, x está libre
- ▶ en $(\forall x) P(x, y)$, x está ligada
- ▶ en $(\forall x)(\forall y) P(x, y)$, x está ligada
- ▶ en $P(x, y) \rightarrow (\forall x)(\forall y) P(x, y)$
 - ▶ la primera aparición de x está libre
 - ▶ la segunda aparición de x está ligada
 - ▶ entonces, x no está ni libre ni ligada

Interpretación de un lenguaje

Una \mathcal{L} -estructura \mathcal{A} de un lenguaje $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$ es

- ▶ un conjunto A no vacío, se lo llama **universo** o **dominio**
- ▶ las siguientes asignaciones:
 - ▶ para cada símbolo de constante $c \in \mathcal{C}$, un elemento fijo

$$c_{\mathcal{A}} \in A$$

- ▶ para cada símbolo de función n -aria $f \in \mathcal{F}$, una función

$$f_{\mathcal{A}} : A^n \rightarrow A$$

- ▶ para cada símbolo de predicado n -ario $P \in \mathcal{P}$, una relación

$$P_{\mathcal{A}} \subseteq A^n$$

Las funciones $f_{\mathcal{A}}$ y predicados $P_{\mathcal{A}}$ son siempre totales.

Ejemplos

Para $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$, con $\mathcal{C} = \{c, d\}$, $\mathcal{F} = \{f, g\}$ y $\mathcal{P} = \{P\}$
(f unaria, g binaria, P binario)

\mathcal{L} -estructura \mathcal{A}

- ▶ $A = \mathbb{Z}$
- ▶ $c_{\mathcal{A}} = 0$
- ▶ $d_{\mathcal{A}} = 1$
- ▶ $f_{\mathcal{A}}(x) = -x$
- ▶ $g_{\mathcal{A}}(x, y) = x + y$
- ▶ $P_{\mathcal{A}}(x, y)$ sii x divide a y

\mathcal{L} -estructura \mathcal{B}

- ▶ $B = \mathcal{P}(\mathbb{N})$
- ▶ $c_{\mathcal{B}} = \emptyset$
- ▶ $d_{\mathcal{B}} = \mathbb{N}$
- ▶ $f_{\mathcal{B}}(x) = \bar{x}$
- ▶ $g_{\mathcal{B}}(x, y) = x \cup y$
- ▶ $P_{\mathcal{B}}(x, y)$ sii $x \subseteq y$

No ejemplos

Para $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$, con $\mathcal{C} = \{c, d\}$, $\mathcal{F} = \{f, g\}$ y $\mathcal{P} = \{P\}$
(f unaria, g binaria, P binario)

\mathcal{L} -estructura \mathcal{M}

- ▶ $M = \mathbb{Z}$
- ▶ $c_{\mathcal{M}} = 0$
- ▶ $d_{\mathcal{M}} = 1$
- ▶ $f_{\mathcal{M}}(x) = 1/x$
- ▶ $g_{\mathcal{M}}(x, y) = x^y$
- ▶ $P_{\mathcal{M}}(x, y)$ sii x divide a y

en general

- ▶ $1/x \notin \mathbb{Z}$
- ▶ $x^y \notin \mathbb{Z}$

\mathcal{L} -estructura \mathcal{N}

- ▶ $N =$ funciones $\mathbb{R} \rightarrow \mathbb{R}$
- ▶ $c_{\mathcal{N}} =$ función identidad
- ▶ $d_{\mathcal{N}} =$ función 0
- ▶ $f_{\mathcal{N}}(x) =$ derivada de x
- ▶ $g_{\mathcal{N}}(x, y) = x \circ y$
- ▶ $P_{\mathcal{N}}(x, y)$ sii $x = y$

una función $\mathbb{R} \rightarrow \mathbb{R}$ puede no ser derivable

Valuaciones

Fijemos una \mathcal{L} -estructura \mathcal{A} con dominio A .

Una **valuación** para \mathcal{A} es una función $v : \text{VAR} \rightarrow A$

Extendemos v a $\tilde{v} : \text{TERM}(\mathcal{L}) \rightarrow A$, que interpreta un término t en una \mathcal{L} -estructura \mathcal{A} :

- ▶ si $t = x$ (variable) entonces $\tilde{v}(t) = v(x)$
- ▶ si $t = c$ (constante) entonces $\tilde{v}(t) = c_{\mathcal{A}}$
- ▶ si $t = f(t_1, \dots, t_n)$ (función) entonces

$$\tilde{v}(t) = f_{\mathcal{A}}(\tilde{v}(t_1), \dots, \tilde{v}(t_n))$$

Sea v una valuación de \mathcal{A} y sea $a \in A$. Definimos la valuación $v(x = a)$ de la siguiente manera

$$v(x = a)(y) = \begin{cases} v(y) & x \neq y \\ a & x = y \end{cases}$$

Escribimos v en vez de \tilde{v} .

Ejemplos

Para $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$, con $\mathcal{C} = \{c, d\}$, $\mathcal{F} = \{f, g\}$ y $\mathcal{P} = \{P\}$
(f unaria, g binaria, P binario)

\mathcal{L} -estructura \mathcal{A}

- ▶ $A = \mathbb{Z}$
- ▶ $c_{\mathcal{A}} = 0$
- ▶ $d_{\mathcal{A}} = 1$
- ▶ $f_{\mathcal{A}}(x) = -x$
- ▶ $g_{\mathcal{A}}(x, y) = x + y$
- ▶ $P_{\mathcal{A}}(x, y)$ sii x divide a y

Tenemos

- ▶ si $v(x) = 2$
 $\tilde{v}(g(x, f(d))) = 2 + (-1) = 1$
- ▶ para cualquier v
 $\tilde{v}(g(c, f(d))) = 0 + (-1) = -1$

\mathcal{L} -estructura \mathcal{B}

- ▶ $B = \mathcal{P}(\mathbb{N})$
- ▶ $c_{\mathcal{B}} = \emptyset$
- ▶ $d_{\mathcal{B}} = \mathbb{N}$
- ▶ $f_{\mathcal{B}}(x) = \bar{x}$
- ▶ $g_{\mathcal{B}}(x, y) = x \cup y$
- ▶ $P_{\mathcal{B}}(x, y)$ sii $x \subseteq y$

Tenemos

- ▶ si $v(x) = \{1, 2\}$
 $\tilde{v}(g(x, f(d))) = \{1, 2\} \cup \bar{\mathbb{N}} = \{1, 2\}$
- ▶ para cualquier v
 $\tilde{v}(g(c, f(d))) = \emptyset \cup \bar{\mathbb{N}} = \emptyset$

Interpretación de una fórmula

Sea \mathcal{A} una \mathcal{L} -estructura con dominio A y v una valuación de \mathcal{A} .
Definimos cuando φ es verdadera en \mathcal{A} bajo la valuación v
(notación: $\mathcal{A} \models \varphi[v]$)

1. φ es de la forma $P(t_1, \dots, t_n)$ (atómica)

$$\mathcal{A} \models P(t_1, \dots, t_n)[v] \quad \text{sii} \quad (\tilde{v}(t_1), \dots, \tilde{v}(t_n)) \in P_{\mathcal{A}}$$

2. φ es de la forma $\neg\psi$

$$\mathcal{A} \models \neg\psi[v] \quad \text{sii} \quad \text{no } \mathcal{A} \models \psi[v]$$

3. φ es de la forma $(\psi \rightarrow \rho)$

$$\mathcal{A} \models (\psi \rightarrow \rho)[v] \quad \text{sii} \quad \text{no } \mathcal{A} \models \psi[v] \text{ o } \mathcal{A} \models \rho[v]$$

4. φ es de la forma $(\forall x)\psi$

$$\mathcal{A} \models (\forall x)\psi[v] \quad \text{sii} \quad \text{para cualquier } a \in A, \mathcal{A} \models \psi[v(x = a)]$$

Ejemplos

Para $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$, con $\mathcal{C} = \{c, d\}$, $\mathcal{F} = \{f, g\}$ y $\mathcal{P} = \{P\}$
(f unaria, g binaria, P binario)

\mathcal{L} -estructura \mathcal{A}

- ▶ $A = \mathbb{Z}$
- ▶ $c_{\mathcal{A}} = 0$
- ▶ $d_{\mathcal{A}} = 1$
- ▶ $f_{\mathcal{A}}(x) = -x$
- ▶ $g_{\mathcal{A}}(x, y) = x + y$
- ▶ $P_{\mathcal{A}}(x, y)$ sii x divide a y

\mathcal{L} -estructura \mathcal{B}

- ▶ $B = \mathcal{P}(\mathbb{N})$
- ▶ $c_{\mathcal{B}} = \emptyset$
- ▶ $d_{\mathcal{B}} = \mathbb{N}$
- ▶ $f_{\mathcal{B}}(x) = \bar{x}$
- ▶ $g_{\mathcal{B}}(x, y) = x \cup y$
- ▶ $P_{\mathcal{B}}(x, y)$ sii $x \subseteq y$

Tenemos

- ▶ para $v(x) = 1$
 $\mathcal{A} \models P(x, d)[v]$
- ▶ para $v(x) = 0$
 $\mathcal{A} \not\models P(x, c)[v]$
- ▶ para cualquier v
 $\mathcal{A} \not\models (\forall y)P(y, g(y, d))[v]$

Tenemos

- ▶ para $v(x) = \emptyset$
 $\mathcal{B} \models P(x, d)[v]$
- ▶ para $v(x) = \{1, 2, 3\}$
 $\mathcal{B} \not\models P(x, c)[v]$
- ▶ para cualquier v
 $\mathcal{B} \models (\forall y)P(y, g(y, d))[v]$

Notación (\wedge , \vee , \exists)

Sea \mathcal{A} una \mathcal{L} -estructura y ν una valuación de \mathcal{A} . Se deduce:

5. φ es de la forma $(\psi \vee \rho)$

$$\mathcal{A} \models (\psi \vee \rho)[\nu] \quad \text{sii} \quad \mathcal{A} \models \psi[\nu] \quad \text{o} \quad \mathcal{A} \models \rho[\nu]$$

6. φ es de la forma $(\psi \wedge \rho)$

$$\mathcal{A} \models (\psi \wedge \rho)[\nu] \quad \text{sii} \quad \mathcal{A} \models \psi[\nu] \quad \text{y} \quad \mathcal{A} \models \rho[\nu]$$

7. φ es de la forma $(\exists x)\psi$

$$\mathcal{A} \models (\exists x)\psi[\nu] \quad \text{sii} \quad \text{hay un } a \in A \text{ tal que } \mathcal{A} \models \psi[\nu(x = a)]$$

3 niveles de verdad

Para un lenguaje \mathcal{L} fijo.

1. φ es **satisfacible** si existe una \mathcal{L} -estructura \mathcal{A} y una valuación v de \mathcal{A} tal que $\mathcal{A} \models \varphi[v]$
2. φ es **verdadera (o válida) en una \mathcal{L} -estructura \mathcal{A} ($\mathcal{A} \models \varphi$)** si $\mathcal{A} \models \varphi[v]$ para toda valuación v de \mathcal{A}
 - ▶ decimos que \mathcal{A} es un **modelo** de φ
3. φ es **universalmente válida ($\models \varphi$)** si $\mathcal{A} \models \varphi[v]$ para toda \mathcal{L} -estructura \mathcal{A} y toda valuación v de \mathcal{A}

Ejemplos

- ▶ $\mathcal{A} = \langle \mathbb{Z}; <, 0 \rangle$ con la interpretación usual
 - ▶ $\mathcal{A} \models (\forall x)(\exists y) x < y$
 - ▶ $\mathcal{A} \models (\exists y) x < y$
 - ▶ $\mathcal{A} \not\models x < y \rightarrow (\exists z) (x < z \wedge z < y)$
 - ▶ $\mathcal{A} \models (\exists x) x < 0$
- ▶ $\mathcal{B} = \langle \mathbb{N}; <, 0 \rangle$ con la interpretación usual
 - ▶ $\mathcal{B} \not\models x < y \rightarrow (\exists z) (x < z \wedge z < y)$
 - ▶ $\mathcal{B} \not\models (\exists x) x < 0$
- ▶ $\mathcal{C} = \langle \mathbb{Q}; <, 0 \rangle$ con la interpretación usual
 - ▶ $\mathcal{C} \models x < y \rightarrow (\exists z) (x < z \wedge z < y)$
 - ▶ $\mathcal{C} \models (\exists x) x < 0$
- ▶ $(\exists x)(\forall y) P(x, y)$ es satisfacible
 - ▶ $\mathcal{D} = \langle \{0\}; = \rangle$ con la interpretación usual
 - ▶ $\mathcal{E} = \langle \mathbb{N}; \leq \rangle$ con la interpretación usual
- ▶ $\models (\forall x) P(x) \rightarrow P(x)$ se entiende $((\forall x) P(x)) \rightarrow P(x)$
- ▶ $\not\models P(x) \rightarrow (\forall x) P(x)$
 - ▶ $\mathcal{F} = \langle \mathbb{N}; \text{par} \rangle$ con la interpretación usual, $v(x) = 0$

Algunos resultados sobre satisfacibilidad y validez

- ▶ si φ es una sentencia, $\mathcal{A} \models \varphi$ sii $\mathcal{A} \models \varphi[v]$
- ▶ φ es universalmente válida sii $\neg\varphi$ es insatisfacible
- ▶ preservación de validez del Modus Ponens:
 - ▶ $\mathcal{A} \models \varphi[v]$ y $\mathcal{A} \models (\varphi \rightarrow \psi)[v]$ entonces $\mathcal{A} \models \psi[v]$
 - ▶ $\mathcal{A} \models \varphi$ y $\mathcal{A} \models (\varphi \rightarrow \psi)$ entonces $\mathcal{A} \models \psi$
 - ▶ $\models \varphi$ y $\models (\varphi \rightarrow \psi)$ entonces $\models \psi$
- ▶ clausura universal
 - ▶ $\mathcal{A} \models \varphi$ sii $\mathcal{A} \models (\forall x)\varphi$
 - ▶ $\models \varphi$ sii $\models (\forall x)\varphi$

Consecuencia semántica

Sea $\Gamma \subseteq \text{FORM}(\mathcal{L})$ y $\varphi \in \text{FORM}(\mathcal{L})$

φ es **consecuencia semántica** de Γ ($\Gamma \models \varphi$) si para toda \mathcal{L} -estructura \mathcal{A} y toda valuación v de \mathcal{A} :

si $\mathcal{A} \models \Gamma[v]$ entonces $\mathcal{A} \models \varphi[v]$

Notación:

$$\mathcal{A} \models \Gamma[v]$$

significa que para toda $\psi \in \Gamma$,

$$\mathcal{A} \models \psi[v]$$

Ejemplos

$\mathcal{L} = \{P, Q\}$, con P y Q símbolos de predicado 1-arios

- ▶ $\Gamma_1 = \{ (\forall x)(P(x) \rightarrow Q(x)) \}$
 - ▶ $\Gamma_1 \not\models (\exists x)P(x)$
 - ▶ $\Gamma_1 \models (\exists x)P(x) \rightarrow (\exists x)Q(x)$
 - ▶ $\Gamma_1 \models (\forall x)P(x) \rightarrow (\forall x)Q(x)$
- ▶ $\Gamma_2 = \{ (\forall x)(P(x) \rightarrow Q(x)) , (\exists x)P(x) \}$
 - ▶ $\Gamma_2 \models (\exists x)Q(x)$
 - ▶ $\Gamma_2 \models (\exists x)(P(x) \wedge Q(x))$
 - ▶ $\Gamma_2 \not\models (\exists x)(\neg P(x) \wedge Q(x))$
- ▶ $\Gamma_3 = \{ (\forall x)(P(x) \rightarrow Q(x)) , (\exists x)(P(x) \wedge \neg Q(x)) \}$
 - ▶ $\Gamma_3 \models \varphi$ para cualquier φ

Lenguajes con igualdad

\mathcal{L} es un **lenguaje con igualdad** si tiene un símbolo proposicional binario especial (el $=$) que sólo se interpreta como la igualdad.

Fijemos un lenguaje \mathcal{L} con igualdad y con ningún otro símbolo. Buscamos $\varphi \in \text{FORM}(\mathcal{L})$ tal que $\{\mathcal{A} : \mathcal{A} \models \varphi\}$ sea la clase de modelos

- ▶ con exactamente 1 elemento

$$\varphi = (\exists x)(\forall y)x = y$$

- ▶ con exactamente 2 elementos

$$\varphi = (\exists x)(\exists y) \overbrace{(x \neq y)}^{\neg x=y} \wedge (\forall z)(z = x \vee z = y)$$

- ▶ con al menos 3 elementos

$$\varphi = (\exists x)(\exists y)(\exists z)(x \neq y \wedge x \neq z \wedge y \neq z)$$

- ▶ con infinitos elementos... con finitos elementos. ¿Se podrá?

Lógica de Primer Orden

Clase 2

Lema de sustitución, sistema axiomático SQ , consecuencia sintáctica, teorema de la generalización, teorema de la generalización en constantes

Reemplazo de variables libres por términos

Sea $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$ un lenguaje fijo.

Sea $\varphi \in \text{FORM}(\mathcal{L})$, $t \in \text{TERM}(\mathcal{L})$ y $x \in \text{VAR}$. $\varphi[x/t]$ es la fórmula obtenida a partir de φ sustituyendo todas las apariciones libres de la variable x por t

Por ejemplo, (para un lenguaje con símbolo de predicado binario P y símbolo de función unario f)

- ▶ $P(x, y)[x/f(z)] = P(f(z), y)$
- ▶ $P(x, y)[x/f(x)] = P(f(x), y)$
- ▶ $((\forall x)(\forall y) P(x, y))[x/f(z)] = (\forall x)(\forall y) P(x, y)$
- ▶ $(P(x, y) \rightarrow (\forall x)(\forall y) P(x, y))[x/f(z)] = P(f(z), y) \rightarrow (\forall x)(\forall y) P(x, y)$

Si $c \in \mathcal{C}$, $\varphi[c/t]$ es la fórmula obtenida a partir de φ sustituyendo todas las apariciones de la constante c por t

Variable reemplazable por un término

Sea $t \in \text{TERM}(\mathcal{L})$, $x \in \text{VAR}$, $\varphi \in \text{FORM}(\mathcal{L})$.

Decimos que x es **reemplazable** por t en φ cuando

1. t es un término cerrado (i.e. no tiene variables) o
2. t tiene variables pero ninguna de ellas queda atrapada por un cuantificador en el reemplazo $\varphi[x/t]$

Por ejemplo, (para un lenguaje con símbolo de predicado unario P y símbolo de función binaria f)

En

$$(\forall y)((\forall x)P(x) \rightarrow P(x))$$

- ▶ x es reemplazable por z : $(\forall y)((\forall x)P(x) \rightarrow P(z))$
- ▶ x es reemplazable por $f(x, z)$: $(\forall y)((\forall x)P(x) \rightarrow P(f(x, z)))$
- ▶ x no es reemplazable por $f(x, y)$: $(\forall \mathbf{y})((\forall x)P(x) \rightarrow P(f(x, \mathbf{y})))$

Lema de Sustitución

Lema

Si x es reemplazable por t en φ entonces

$$\mathcal{A} \models (\varphi[x/t])[v] \quad \text{sii} \quad \mathcal{A} \models \varphi[v(x = \tilde{v}(t))].$$

Demostración.

Por inducción en la complejidad de φ .

(a veces escribo v por \tilde{v})

- ▶ $\varphi = P(u)$ es atómica (u es término; el caso n -ario es similar).
 $\mathcal{A} \models P(u[x/t])[v]$ sii $\tilde{v}(u[x/t]) \in P_{\mathcal{A}}$
sii (lema auxiliar que dice $v(u[x/t]) = v(x = v(t))(u)$)
 $v[x = \tilde{v}(t)](u) \in P_{\mathcal{A}}$ sii $\mathcal{A} \models \varphi[v(x = \tilde{v}(t))]$.
- ▶ φ es de la forma $\neg\psi$ o de la forma $\psi \rightarrow \rho$. Directo.

(sigue \rightarrow)



Lema de Sustitución

Demostración (cont.)

- φ es de la forma $(\forall y) \psi$.

Sup. x no aparece libre en φ . Entonces v y $v[x = v(t)]$ coinciden en todas las variables que aparecen libres en φ . Además, $\varphi = \varphi[x/t]$. Inmediato.

Sup. x aparece libre en φ . Como x es reemplazable por t en φ , y no ocurre en t .
Luego para todo $d \in A$,

$$v(t) = v(y = d)(t). \quad (1)$$

Además, x es reemplazable por t en ψ . Como $x \neq y$,

$$\varphi[x/t] = ((\forall y) \psi)[x/t] = (\forall y) (\psi[x/t]).$$

Luego

$\mathcal{A} \models \varphi[x/t][v]$	sii (def.)	para todo $d \in A$, $\mathcal{A} \models \psi[x/t][v(y = d)]$
	sii (HI)	para todo $d \in A$, $\mathcal{A} \models \underbrace{\psi[v(y = d)(x = w(t))]}_w$
	sii (1)	para todo $d \in A$, $\mathcal{A} \models \psi[v(y = d)(x = v(t))]$
	sii ($x \neq y$)	para todo $d \in A$, $\mathcal{A} \models \psi[v(x = v(t))(y = d)]$
	sii (def.)	$\mathcal{A} \models \varphi[v(x = v(t))]$

Mecanismo deductivo SQ (para un lenguaje fijo \mathcal{L})

Para un lenguaje fijo \mathcal{L}

- ▶ **axiomas.** Sean $\varphi, \psi, \rho \in \text{FORM}(\mathcal{L})$, $x \in \text{VAR}$, $t \in \text{TERM}(\mathcal{L})$
 - SQ1 $\varphi \rightarrow (\psi \rightarrow \varphi)$
 - SQ2 $(\varphi \rightarrow (\psi \rightarrow \rho)) \rightarrow ((\varphi \rightarrow \psi) \rightarrow (\varphi \rightarrow \rho))$
 - SQ3 $(\neg\varphi \rightarrow \neg\psi) \rightarrow (\psi \rightarrow \varphi)$
 - SQ4 $(\forall x)\varphi \rightarrow \varphi[x/t]$ si x es reemplazable por t en φ
 - SQ5 $\varphi \rightarrow (\forall x)\varphi$ si x no aparece libre en φ
 - SQ6 $(\forall x)(\varphi \rightarrow \psi) \rightarrow ((\forall x)\varphi \rightarrow (\forall x)\psi)$
 - SQ7 si φ es un axioma entonces $(\forall x)\varphi$ también es un axioma
- ▶ **regla de inferencia**
 - MP Sean $\varphi, \psi \in \text{FORM}(\mathcal{L})$. ψ es una consecuencia inmediata de $\varphi \rightarrow \psi$ y φ

Consecuencia sintáctica, demostraciones, teoremas, teorías

Fijamos un lenguaje \mathcal{L} . Sea $\Gamma \subseteq \text{FORM}(\mathcal{L})$ y $\varphi \in \text{FORM}(\mathcal{L})$

1. una **demostración** de φ en SQ es una cadena finita y no vacía

$$\varphi_1, \dots, \varphi_n$$

de fórmulas de \mathcal{L} tal que $\varphi_n = \varphi$ y

- ▶ φ_i es un axioma o
- ▶ φ_i es una consecuencia inmediata de $\varphi_k, \varphi_l, k, l < i$

En este caso, decimos que φ es un **teorema** ($\vdash \varphi$).

2. φ es una **consecuencia sintáctica** de Γ ($\Gamma \vdash \varphi$) si existe una cadena finita y no vacía

$$\varphi_1, \dots, \varphi_n$$

de fórmulas de \mathcal{L} tal que $\varphi_n = \varphi$ y

- ▶ φ_i es un axioma o
- ▶ $\varphi_i \in \Gamma$ o
- ▶ φ_i es una consecuencia inmediata de $\varphi_k, \varphi_l, k, l < i$

Aquí, $\varphi_1, \dots, \varphi_n$ se llama **derivación** de φ a partir de Γ . Γ se llama **teoría**. Decimos que φ es un **teorema de la teoría** Γ .

Ejemplo $\Gamma = \{(\forall x)(\varphi[z/x])\} \vdash (\forall z)\varphi$ (x no aparece en φ)

- | | | |
|----|--|----------|
| 1. | $(\forall z)((\forall x)(\varphi[z/x]) \rightarrow \varphi)$ | SQ4+SQ7 |
| 2. | $(\forall z)((\forall x)(\varphi[z/x]) \rightarrow \varphi) \rightarrow ((\forall z)(\forall x)(\varphi[z/x]) \rightarrow (\forall z)\varphi)$ | SQ6 |
| 3. | $(\forall z)(\forall x)(\varphi[z/x]) \rightarrow (\forall z)\varphi$ | MP 1,2 |
| 4. | $(\forall x)(\varphi[z/x]) \rightarrow (\forall z)(\forall x)(\varphi[z/x])$ | SQ5 |
| 5. | $(\forall x)(\varphi[z/x])$ | Γ |
| 6. | $(\forall z)(\forall x)(\varphi[z/x])$ | MP 4,5 |
| 7. | $(\forall z)\varphi$ | MP 3,6 |

Observar

- ▶ paso 1: x es reemplazable por z en $\varphi[z/x]$
- ▶ paso 1: $\varphi[z/x][x/z] = \varphi$
- ▶ paso 4: z no aparece libre en $(\forall x)(\varphi[z/x])$

Correctitud y consistencia

Teorema (Correctitud)

El sistema SQ es correcto, i.e. si $\Gamma \vdash \varphi$ entonces $\Gamma \models \varphi$.

Teorema (Consistencia)

El sistema SQ es consistente, i.e. no existe $\varphi \in \text{FORM}(\mathcal{L})$ tal que

$$\vdash \varphi \quad \text{y} \quad \vdash \neg\varphi$$

Resultados similares a los de SP

Teorema (de la Deducción)

Si $\Gamma \cup \{\varphi\} \vdash \psi$ entonces $\Gamma \vdash \varphi \rightarrow \psi$

$\Gamma \subseteq \text{FORM}(\mathcal{L})$ es **consistente** si no existe $\varphi \in \text{FORM}(\mathcal{L})$ tal que

$$\Gamma \vdash \varphi \quad \text{y} \quad \Gamma \vdash \neg\varphi$$

Proposición

1. $\Gamma \cup \{\neg\varphi\}$ es inconsistente sii $\Gamma \vdash \varphi$
2. $\Gamma \cup \{\varphi\}$ es inconsistente sii $\Gamma \vdash \neg\varphi$

Teorema

Si Γ es satisficible, entonces Γ es consistente.

Teorema

Si Γ es inconsistente, entonces existe un subconjunto finito de Γ que es inconsistente.

Instancias de esquemas tautológicos

- ▶ sea $\varphi(p_1, \dots, p_n)$ una tautología de P con variables proposicionales p_1, \dots, p_n .
- ▶ sean ψ_1, \dots, ψ_n fórmulas cualesquiera de primer orden
- ▶ $\varphi(\psi_1, \dots, \psi_n)$ es una **instancia de un esquema tautológico** (reemplazar p_i por ψ_i en la fórmula original φ)

Proposición

Si φ es una instancia de un esquema tautológico entonces $\vdash \varphi$.

Por ejemplo, la fórmula de P

$$(p \wedge q) \rightarrow p$$

es tautología. Entonces

$$\vdash ((\forall x)R(x) \wedge (\exists y)Q(y)) \rightarrow (\forall x)R(x)$$

Variantes alfabéticas

Sea $\mathcal{L} = \{0, S\}$ con igualdad y $\varphi \in \text{FORM}(\mathcal{L})$ definida como

$$\varphi = x \neq 0 \rightarrow (\exists y)x = S(y)$$

En φ la variable x es reemplazable por z :

$$\varphi[x/z] = z \neq 0 \rightarrow (\exists y)z = S(y)$$

Sin embargo, la variable x no es reemplazable por y :

$$\varphi[x/y] = y \neq 0 \rightarrow (\exists y)y = S(y)$$

No habría habido problema si la fórmula original hubiese sido

$$\varphi' = x \neq 0 \rightarrow (\exists w)x = S(w)$$

φ' se llama **variante alfabética** de φ

Lema

Sea $\varphi \in \text{FORM}(\mathcal{L})$. Dados $x \in \text{VAR}$ y $t \in \text{TERM}(\mathcal{L})$ podemos encontrar φ' (variante alfabética de φ) tal que

- ▶ $\{\varphi\} \vdash \varphi'$ y $\{\varphi'\} \vdash \varphi$
- ▶ x es reemplazable por t en φ'

Teorema de Generalización (TG)

Teorema

Si $\Gamma \vdash \varphi$ y x no aparece libre en ninguna fórmula de Γ , entonces $\Gamma \vdash (\forall x)\varphi$

Observar que es necesario pedir que x no aparezca libre en ninguna fórmula de Γ :

- ▶ $\{R(x)\} \not\vdash (\forall x)R(x)$
- ▶ entonces $\{R(x)\} \not\vdash (\forall x)R(x)$ (por correctitud)

Demostración del teorema.

Planteo

$P(n) =$ para toda ψ , Γ y x tal que $\Gamma \vdash \psi$ y x no aparece libre en ninguna fórmula de Γ , si ψ_1, \dots, ψ_n es una derivación de ψ a partir de Γ entonces $\Gamma \vdash (\forall x)\psi$

Demostración por inducción en n (detalles a continuación). □

Demostración del TG (caso base)

$P(n) =$ para toda ψ , Γ y x tal que $\Gamma \vdash \psi$ y x **no aparece libre en ninguna fórmula de Γ** , si ψ_1, \dots, ψ_n es una derivación de ψ a partir de Γ entonces $\Gamma \vdash (\forall x)\psi$

Probamos $P(1)$:

- ▶ sea φ , Γ y x tal que x **no aparece libre en Γ**
- ▶ sea φ una derivación de φ a partir de Γ
- ▶ queremos ver que $\Gamma \vdash (\forall x)\varphi$

Hay 2 posibilidades:

1. φ es axioma de SQ $\stackrel{\text{SQ7}}{\Rightarrow} \vdash (\forall x)\varphi \Rightarrow \Gamma \vdash (\forall x)\varphi$
2. $\varphi \in \Gamma$ entonces
 - ▶ $\Gamma \vdash \varphi$
 - ▶ por hipótesis x no aparece libre en φ
 - ▶ por SQ5, $\vdash \varphi \rightarrow (\forall x)\varphi$
 - ▶ por MP, $\Gamma \vdash (\forall x)\varphi$

Demostración del TG (paso inductivo)

$P(n) =$ para toda ψ , Γ y x tal que $\Gamma \vdash \psi$ y x no aparece libre en ninguna fórmula de Γ , si ψ_1, \dots, ψ_n es una derivación de ψ a partir de Γ entonces $\Gamma \vdash (\forall x)\psi$

Probamos $P(n)$:

- ▶ sea φ , Γ y x tal que x no aparece libre en Γ
- ▶ sea $\varphi_1, \dots, \varphi_n$ una derivación de φ a partir de Γ
- ▶ queremos ver que $\Gamma \vdash (\forall x)\varphi$
- ▶ **HI:** vale $P(m)$ para todo $m < n$

Hay 3 posibilidades:

1 y 2. φ es axioma de SQ o $\varphi \in \Gamma$: igual que en caso base.

3. φ se obtiene por MP de φ_i y φ_j ($i, j < n$):

supongamos que $\varphi_j = \varphi_i \rightarrow \varphi$. Usamos **HI** 2 veces:

- ▶ como $i < n$, vale $P(i)$, en particular, $\Gamma \vdash (\forall x)\varphi_i$
- ▶ como $j < n$, vale $P(j)$, en particular, $\Gamma \vdash (\forall x)(\varphi_i \rightarrow \varphi)$

por SQ6, $\vdash (\forall x)(\varphi_i \rightarrow \varphi) \rightarrow ((\forall x)\varphi_i \rightarrow (\forall x)\varphi)$

usando MP 2 veces, $\Gamma \vdash (\forall x)\varphi$

Teorema de Generalización en Constantes (TGC)

Teorema

Supongamos que $\Gamma \vdash \varphi$ y c es un símbolo de constante que no aparece en Γ . Entonces existe una variable x que no aparece en φ tal que $\Gamma \vdash (\forall x)(\varphi[c/x])$. Más aun, hay una derivación de $(\forall x)(\varphi[c/x])$ a partir de Γ en donde c no aparece.

Idea de la demostración.

Sea $\varphi_1, \dots, \varphi_n$ una derivación de φ a partir de Γ .

Sea x la primera variable que no aparece en ninguna de las φ_i .

1. demostrar que $\varphi_1[c/x], \dots, \varphi_n[c/x]$
 - 1.1 es una derivación de $\varphi[c/x]$ a partir de Γ (por inducción en n)
 - 1.2 no contiene al símbolo de constante c
2. hay un $\Delta \subseteq \Gamma$ finito tal que $\Delta \vdash \varphi[c/x]$ con derivación que no usa c y tal que x no aparece libre en ninguna fórmula de Δ
 - 2.1 Δ es el conjunto de axiomas de Γ que se usan en $\varphi_1, \dots, \varphi_n$
3. por el TG, $\Delta \vdash (\forall x)(\varphi[c/x])$ con derivación que no usa c



Consecuencias del TGC

Corolario

Supongamos que $\Gamma \vdash \varphi[z/c]$ y c es un símbolo de constante que no aparece en Γ ni en φ . Entonces $\Gamma \vdash (\forall z)\varphi$. Más aun, hay una derivación de $(\forall z)\varphi$ a partir de Γ en donde c no aparece.

Demostración.

- ▶ por el TGC, existe x tal que
 - ▶ x no aparece en $\varphi[z/c]$
 - ▶ $\Gamma \vdash (\forall x)(\varphi[z/c][c/x])$
 - ▶ en esta última derivación no aparece c
- ▶ como c no aparece en φ , $\varphi[z/c][c/x] = \varphi[z/x]$
- ▶ entonces $\Gamma \vdash (\forall x)(\varphi[z/x])$
- ▶ sabemos $\vdash (\forall x)(\varphi[z/x]) \rightarrow (\forall z)\varphi$
(aplicar el Teorema de la Deducción a derivación de hoja 207)
- ▶ por MP concluimos $\Gamma \vdash (\forall z)\varphi$
- ▶ en esta última derivación no aparece c

Lenguajes con igualdad

Fijamos un lenguaje \mathcal{L} con igualdad.

Para los lenguajes con igualdad, se considera el sistema $SQ^=$ con los axiomas y regla de inferencia de SQ , sumando estos dos axiomas

- ▶ **axiomas adicionales para $SQ^=$** . Sean $\varphi, \psi \in \text{FORM}(\mathcal{L})$,
 $x, y \in \text{VAR}$

$$SQ^=1 \quad x = x$$

$$SQ^=2 \quad x = y \rightarrow (\varphi \rightarrow \psi) \text{ donde } \varphi \text{ es atómica y } \psi \text{ se obtiene de } \varphi \text{ reemplazando } x \text{ por } y \text{ en cero o más lugares}$$

Se puede probar que

- ▶ $SQ^=$ es consistente
- ▶ Si hay una derivación de φ en $SQ^=$ entonces φ es verdadera en toda \mathcal{L} -estructura en donde el $=$ se interpreta como la igualdad

Notas sobre computabilidad

Fijemos un lenguaje numerable \mathcal{L} . Se pueden codificar las fórmulas de $\text{FORM}(\mathcal{L})$ con números naturales.

Igual que para la lógica proposicional:

- ▶ es computable la función

$$\text{dem}_{\mathcal{L}}(x) = \begin{cases} 1 & x \text{ es una demostración válida en } SQ \text{ para } \mathcal{L} \\ 0 & \text{si no} \end{cases}$$

- ▶ considerar el siguiente programa P :

```
[A]   IF  $\text{dem}_{\mathcal{L}}(D) = 1 \wedge D[|D|] = X$  GOTO  $E$ 
       $D \leftarrow D + 1$ 
      GOTO  $A$ 
```

- ▶ φ es teorema de SQ para \mathcal{L} sii $\# \varphi \in \text{dom} \Psi_P$
- ▶ el conjunto de teoremas del sistema SQ para \mathcal{L} es c.e.
- ▶ ¿será el conjunto de teoremas de SQ para \mathcal{L} computable?

Lógica de Primer Orden

Clase 3

Compleitud de SQ , compacidad

Consistente \Rightarrow satisficible

Sea \mathcal{L} un lenguaje fijo. Sea $\Gamma \subseteq \text{FORM}(\mathcal{L})$ consistente. Queremos construir un modelo canónico \mathcal{B} y una valuación v de \mathcal{B} tal que:

$$\mathcal{B} \models \varphi[v] \text{ para toda } \varphi \in \Gamma$$

Demostración en 5 pasos:

- Paso 1.** expandir \mathcal{L} a \mathcal{L}' con **nuevas constantes**. $\mathcal{L}' = \mathcal{L} \cup \mathcal{C}$. En \mathcal{C} hay una cantidad infinita numerable de nuevas constantes (“nuevas” porque no aparecen en \mathcal{L})
- Paso 2.** agregar **testigos** a Γ . Trabajamos con $\Gamma \cup \Theta$, donde Θ es un conjunto de formulas especiales que usan las constantes nuevas de \mathcal{L}'
- Paso 3.** aplicar el **Lema de Lindenbaum** para $\Gamma \cup \Theta$. Obtener $\Delta \supseteq \Gamma \cup \Theta$ maximal consistente
- Paso 4.** construir el **modelo canónico** \mathcal{A} y valuación v (para el lenguaje \mathcal{L}') tal que $\mathcal{A} \models \varphi[v]$ sii $\varphi \in \Delta$
- Paso 5.** **restringir** \mathcal{A} y v al lenguaje original \mathcal{L} y obtener \mathcal{B}

Paso 1: expandir de \mathcal{L} a \mathcal{L}' con nuevas constantes

Teorema

Sea $\Gamma \subseteq \text{FORM}(\mathcal{L})$ consistente. Sea \mathcal{C} un conjunto de nuevas constantes que no aparecen en \mathcal{L} . Si $\mathcal{L}' = \mathcal{L} \cup \mathcal{C}$ entonces Γ es consistente en el lenguaje \mathcal{L}' .

Demostración.

- ▶ supongamos Γ inconsistente en el nuevo lenguaje \mathcal{L}' . Entonces existe $\varphi \in \text{FORM}(\mathcal{L}')$ tal que $\Gamma \vdash \varphi$ y $\Gamma \vdash \neg\varphi$
- ▶ cada una de estas derivaciones usa fórmulas en $\text{FORM}(\mathcal{L}')$, pero aparecen solo finitas constantes nuevas
- ▶ por el TGC, cada constante nueva utilizada (por hipótesis no aparece en Γ) puede reemplazarse por una variable nueva
- ▶ obtenemos una derivación de $\Gamma \vdash \varphi[c_1, \dots, c_n/x_1, \dots, x_n]$ y $\Gamma \vdash \neg\varphi[c_1, \dots, c_n/x_1, \dots, x_n]$ en el lenguaje original \mathcal{L} (c_i son nuevas constantes; x_i son nuevas variables)
- ▶ entonces Γ es inconsistente en el lenguaje \mathcal{L}

Paso 2: agregar testigos a Γ

Sean Γ y \mathcal{C} como en el paso 1. Sea

$$\langle \varphi_1, x_1 \rangle, \langle \varphi_2, x_2 \rangle, \dots$$

una enumeración de $\text{FORM}(\mathcal{L}') \times \text{VAR}$

Definimos

$$\theta_n = \neg(\forall x_n)\varphi_n \rightarrow \neg(\varphi_n[x_n/c_n])$$

donde c_n es la primera constante de \mathcal{C} que

- ▶ no aparece en φ_n y
- ▶ no aparece en $\theta_1, \dots, \theta_{n-1}$

Definimos

$$\Theta = \{\theta_1, \theta_2, \dots\}$$

Teorema

$\Gamma \cup \Theta \subseteq \text{FORM}(\mathcal{L}')$ es consistente.

Observar que Θ agrega **testigos** a Γ . Si ocurre $\neg(\forall x)\varphi$ entonces hay una constante c que atestigua que φ no vale para todo x , i.e. $\neg(\varphi[x/c])$

Demostración del paso 2

Supongamos Γ consistente. Recordemos que $\Theta = \{\theta_1, \theta_2, \dots\}$

- ▶ supongamos $\Gamma \cup \Theta$ inconsistente
- ▶ debe existir i tal que $\Gamma \cup \{\theta_1, \dots, \theta_{i+1}\}$ es inconsistente
- ▶ sea n el mínimo tal i
- ▶ observar que $\Gamma \cup \{\theta_1, \dots, \theta_n\}$ es consistente
- ▶ $\Gamma \cup \{\theta_1, \dots, \theta_n\} \vdash \underbrace{\neg(\neg(\forall x)\varphi \rightarrow \neg(\varphi[x/c]))}_{\theta_{n+1}}$

donde c no aparece en $\{\theta_1, \dots, \theta_n\}$ ni en φ

- ▶ las siguientes son instancias de esquemas tautológicos:
 - ▶ $\neg\theta_{n+1} \rightarrow \neg(\forall x)\varphi$
 - ▶ $\neg\theta_{n+1} \rightarrow (\varphi[x/c])$
- ▶ por lo tanto
 - ▶ $\vdash \neg\theta_{n+1} \rightarrow \neg(\forall x)\varphi \xRightarrow{\text{MP}} \Gamma \cup \{\theta_1, \dots, \theta_n\} \vdash \neg(\forall x)\varphi$
 - ▶ $\vdash \neg\theta_{n+1} \rightarrow (\varphi[x/c]) \xRightarrow{\text{MP}} \Gamma \cup \{\theta_1, \dots, \theta_n\} \vdash \varphi[x/c]$
- ▶ por el corolario del TGC, $\Gamma \cup \{\theta_1, \dots, \theta_n\} \vdash (\forall x)\varphi$
(notar que c no aparece en $\Gamma \cup \{\theta_1, \dots, \theta_n\}$ ni en φ)
- ▶ entonces $\Gamma \cup \{\theta_1, \dots, \theta_n\}$ es inconsistente

Paso 3: Lema de Lindenbaum para $\Gamma \cup \Theta$

Teorema

Sean Γ y Θ como en los pasos 1 y 2. Existe un conjunto $\Delta \supseteq \Gamma \cup \Theta$ tal que Δ es maximal consistente.

Demostración.

Igual que para el caso proposicional. □

Como en el caso proposicional, para toda $\varphi \in \text{FORM}(\mathcal{L}')$

- ▶ $\varphi \in \Delta$ o bien $\neg\varphi \in \Delta$
- ▶ $\varphi \in \Delta$ sii $\Delta \vdash \varphi$

Paso 4: construcción del modelo canónico \mathcal{A}

Definimos el modelo canónico \mathcal{A} :

- ▶ $A = \text{TERM}(\mathcal{L}')$
- ▶ para cada símbolo de función n -aria $f \in \mathcal{L}'$,

$$f_{\mathcal{A}}(\underbrace{t_1, \dots, t_n}_{\in A^n}) = f(t_1, \dots, t_n) \in A$$

- ▶ para cada símbolo de constante $c \in \mathcal{L}'$,

$$c_{\mathcal{A}} = c \in A$$

- ▶ para cada símbolo de predicado n -ario $P \in \mathcal{L}'$,

$$\underbrace{(t_1, \dots, t_n)}_{\in A^n} \in P^{\mathcal{A}} \quad \text{sii} \quad P(t_1, \dots, t_n) \in \Delta$$

Paso 4: definición de la valuación v

Definimos la valuación $v : \text{VAR} \rightarrow \underbrace{\text{TERM}(\mathcal{L}')}_A$ como

$$v(x) = x$$

Lema

Para todo $t \in \text{TERM}(\mathcal{L}')$, $\tilde{v}(t) = t$.

Demostración.

Por inducción en la complejidad de t (fácil). □

Teorema

Para toda $\varphi \in \text{FORM}(\mathcal{L}')$, $\mathcal{A} \models \varphi[v]$ sii $\varphi \in \Delta$.

Demostración.

Por inducción en la complejidad de φ (detalles a continuación). □

Paso 4: $\mathcal{A} \models \varphi[v]$ sii $\varphi \in \Delta$ (caso base)

Si φ es una fórmula atómica $P(t_1, \dots, t_n)$:

$$\begin{array}{lll} \mathcal{A} \models P(t_1, \dots, t_n)[v] & \text{sii} & (\tilde{v}(t_1), \dots, \tilde{v}(t_n)) \in P^{\mathcal{A}} \\ & \text{sii} & (t_1, \dots, t_n) \in P^{\mathcal{A}} \\ & \text{sii} & P(t_1, \dots, t_n) \in \Delta \end{array} \quad \begin{array}{l} \text{pues } \tilde{v}(t) = t \\ \text{por def. de } \mathcal{A} \end{array}$$

Paso 4: $\mathcal{A} \models \varphi[v]$ sii $\varphi \in \Delta$ (paso inductivo; $\varphi = \neg\psi$)

$\mathcal{A} \models \varphi[v]$	sii	$\mathcal{A} \not\models \psi[v]$	
	sii	$\psi \notin \Delta$	por HI
	sii	$\neg\psi \in \Delta$	por propiedad de Δ

Paso 4: $\mathcal{A} \models \varphi[v]$ sii $\varphi \in \Delta$ (paso inductivo; $\varphi = \psi \rightarrow \rho$)

$\mathcal{A} \models \varphi[v]$	sii	$\mathcal{A} \not\models \psi[v]$ o $\mathcal{A} \models \rho[v]$	
	sii	$\psi \notin \Delta$ o $\rho \in \Delta$	por HI
	sii	$\neg\psi \in \Delta$ o $\rho \in \Delta$	por propiedad de Δ
	\Rightarrow	$\Delta \vdash \psi \rightarrow \rho$	(ejercicio)
	\Rightarrow	$\psi \rightarrow \rho \in \Delta$	por propiedad de Δ

$\varphi \in \Delta$	\Rightarrow	$\psi \notin \Delta$ o $(\psi \in \Delta$ y $\Delta \vdash \rho)$	MP en 2do caso
	\Rightarrow	$\psi \notin \Delta$ o $(\psi \in \Delta$ y $\rho \in \Delta)$	por propiedad de Δ
	\Rightarrow	$\psi \notin \Delta$ o $\rho \in \Delta$	
	sii	$\mathcal{A} \not\models \psi[v]$ o $\mathcal{A} \models \rho[v]$	por HI
	sii	$\mathcal{A} \models \psi \rightarrow \rho[v]$	

Paso 4: $\mathcal{A} \models \varphi[v]$ sii $\varphi \in \Delta$ (paso inductivo (\Rightarrow)); $\varphi = (\forall x)\psi$

- ▶ supongamos $\mathcal{A} \models (\forall x)\psi[v]$
- ▶ para todo $t \in A$, $\mathcal{A} \models \psi[v(x = t)]$
- ▶ supongamos $\neg(\forall x)\psi \rightarrow \neg(\psi[x/c]) \in \Theta$
- ▶ en particular, $\mathcal{A} \models \psi[v(x = c)]$
- ▶ por definición de v , $\mathcal{A} \models \psi[v(x = \tilde{v}(c))]$
- ▶ por el Lema de Sustitución, $\mathcal{A} \models (\psi[x/c])[v]$
- ▶ por HI, $\psi[x/c] \in \Delta$
- ▶ por propiedad de Δ , $\neg(\psi[x/c]) \notin \Delta$
- ▶ veamos que $\neg(\forall x)\psi \notin \Delta$:
 - ▶ supongamos que $\neg(\forall x)\psi \in \Delta$
 - ▶ $\Delta \vdash \neg(\forall x)\psi$
 - ▶ como $\Delta \supseteq \Theta$, $\neg(\forall x)\psi \rightarrow \neg(\psi[x/c]) \in \Delta$
 - ▶ $\Delta \vdash \neg(\forall x)\psi \rightarrow \neg(\psi[x/c])$
 - ▶ por MP tenemos $\Delta \vdash \neg(\psi[x/c])$
 - ▶ por propiedad de Δ , $\neg(\psi[x/c]) \in \Delta$
- ▶ concluimos $(\forall x)\psi \in \Delta$

Paso 4: $\mathcal{A} \models \varphi[v]$ sii $\varphi \in \Delta$ (paso inductivo (\Leftarrow)) ; $\varphi = (\forall x)\psi$

- ▶ supongamos $\mathcal{A} \not\models \varphi[v]$
- ▶ existe $t \in A$, $\mathcal{A} \not\models \psi[v(x = t)]$
- ▶ sea ψ' una variante alfabética de ψ tal que x sea reemplazable por t en ψ'
- ▶ $\mathcal{A} \not\models \psi'[v(x = t)]$
- ▶ como $\tilde{v}(t) = t$, $\mathcal{A} \not\models \psi'[v(x = \tilde{v}(t))]$
- ▶ por el Lema de Sustitución $\mathcal{A} \not\models (\psi'[x/t])[v]$
- ▶ por HI, $\psi'[x/t] \notin \Delta$
- ▶ veamos que $(\forall x)\psi' \notin \Delta$:
 - ▶ supongamos que $(\forall x)\psi' \in \Delta$
 - ▶ $\Delta \vdash (\forall x)\psi'$
 - ▶ sabemos $\vdash (\forall x)\psi' \rightarrow \psi'[x/t]$ por SQ4
 - ▶ por MP concluimos $\Delta \vdash \psi'[x/t]$
 - ▶ por propiedad de Δ , $\psi'[x/t] \in \Delta$
- ▶ por equivalencia de variantes alfabéticas, $(\forall x)\psi \notin \Delta$

Paso 5: restringir \mathcal{A} y v al lenguaje original \mathcal{L}

Volvemos al lenguaje original \mathcal{L} .

Definimos \mathcal{B} como la restricción de \mathcal{A} a \mathcal{L} (i.e. ya no interpreto las nuevas constantes).

Del paso 4 sabemos que para toda $\varphi \in \text{FORM}(\mathcal{L}')$,

$$\mathcal{A} \models \varphi[v] \quad \text{sii} \quad \varphi \in \Delta.$$

Recordar que $\Delta \supseteq \Gamma$. Si $\varphi \in \Gamma \subseteq \text{FORM}(\mathcal{L})$ tenemos

$$\mathcal{A} \models \varphi[v] \quad \text{sii} \quad \mathcal{B} \models \varphi[v]$$

Luego, para Γ consistente, encontramos una \mathcal{L} -estructura \mathcal{B} tal que

$$\mathcal{B} \models \varphi[v] \quad \text{para toda } \varphi \in \Gamma$$

Concluimos que Γ es satisfacible.

Teorema de Löwenheim-Skolem

Corolario

Γ es consistente sii Γ es satisfacible

Teorema (sin igualdad)

Sea \mathcal{L} numerable y sin igualdad. Si $\Gamma \subseteq \text{FORM}(\mathcal{L})$ es satisfacible, es satisfacible en un modelo infinito numerable.

Demostración.

Es lo que acabamos de ver. Si \mathcal{L} es numerable, $A = \text{FORM}(\mathcal{L})$ es infinito numerable. □

Teorema (con igualdad)

Sea \mathcal{L} numerable y con igualdad. Si $\Gamma \subseteq \text{FORM}(\mathcal{L})$ es satisfacible, es satisfacible en un modelo finito o infinito numerable.

Se puede probar algo más fuerte

Teorema (ascendente)

Si \mathcal{L} es numerable y $\Gamma \subseteq \text{FORM}(\mathcal{L})$ tiene modelo infinito, tiene modelo de cualquier cardinalidad.

Completitud y Compacidad

Teorema (Completitud fuerte, Gödel)

Si $\Gamma \models \varphi$ entonces $\Gamma \vdash \varphi$.

Demostración.

Igual que para proposicional □

Corolario

$\Gamma \models \varphi$ *sii* $\Gamma \vdash \varphi$.

Teorema (Compacidad)

Sea $\Gamma \subseteq \text{FORM}(\mathcal{L})$. Si todo Δ finito, $\Delta \subseteq \Gamma$ es satisfacible, entonces Γ es satisfacible.

Demostración.

Igual que para proposicional □

Lógica de Primer Orden

Clase 4

Aplicaciones de compacidad, indecidibilidad de la lógica de primer orden

Aplicaciones de Compacidad - no expresividad

Teorema

Si Γ tiene modelos arbitrariamente grandes, tiene modelo infinito.

Demostración.

Definimos (en el lenguaje con solo la igualdad)

$$\varphi_2 = (\exists x)(\exists y)x \neq y$$

$$\varphi_3 = (\exists x)(\exists y)(\exists z)(x \neq y \wedge x \neq z \wedge y \neq z)$$

\vdots

$$\varphi_n = \text{“hay al menos } n \text{ elementos”}$$

- ▶ por hipótesis, todo subconjunto finito de $\Gamma \cup \{\varphi_i \mid i \geq 2\}$ tiene modelo
- ▶ por Compacidad, $\Gamma \cup \{\varphi_i \mid i \geq 2\}$ tiene algún modelo \mathcal{M}
- ▶ \mathcal{M} tiene que ser infinito □

Conclusión:

- ▶ \mathcal{A} es infinito sii $\mathcal{A} \models \{\varphi_i \mid i \geq 2\}$
- ▶ no existe Γ tal que \mathcal{A} es finito sii $\mathcal{A} \models \Gamma$

Aplicaciones de Compacidad - modelos no estándar

Consideremos un lenguaje $\mathcal{L} = \{0, S, <, +, \cdot\}$ con igualdad.

Consideremos la estructura $\mathcal{N} = \langle \mathbb{N}; 0, S, <, +, \cdot \rangle$ con la interpretación usual. Sea

$$\text{Teo}(\mathcal{N}) = \{\varphi \in \text{FORM}(\mathcal{L}) : \varphi \text{ es sentencia y } \mathcal{N} \models \varphi\}$$

Expandimos el lenguaje con una nueva constante c y definimos

$$\Gamma = \{0 < c, S(0) < c, S(S(0)) < c, S(S(S(0))) < c, \dots\}$$

- ▶ cada subconjunto finito de $\Gamma \cup \text{Teo}(\mathcal{N})$ tiene modelo
- ▶ por Compacidad, $\Gamma \cup \text{Teo}(\mathcal{N})$ tiene modelo
- ▶ por Löwenheim-Skolem $\Gamma \cup \text{Teo}(\mathcal{N})$ un modelo numerable

$$\mathcal{M} = \langle M; 0^M, S^M, <^M, +^M, \cdot^M, c^M \rangle$$

- ▶ sea \mathcal{M}' la restricción de \mathcal{M} al lenguaje original \mathcal{L}
- ▶ $\mathcal{N} \models \varphi$ sii $\mathcal{M}' \models \varphi$ para toda sentencia $\varphi \in \text{FORM}(\mathcal{L})$
 - ▶ $\mathcal{N} \models \varphi \Rightarrow \varphi \in \text{Teo}(\mathcal{N}) \Rightarrow \mathcal{M} \models \varphi \Rightarrow \mathcal{M}' \models \varphi$
 - ▶ $\mathcal{N} \not\models \varphi \Rightarrow \mathcal{N} \models \neg\varphi \Rightarrow \neg\varphi \in \text{Teo}(\mathcal{N}) \Rightarrow \mathcal{M} \models \neg\varphi \Rightarrow \mathcal{M}' \models \neg\varphi \Rightarrow \mathcal{M}' \not\models \varphi$
- ▶ \mathcal{N} y \mathcal{M}' no son isomorfos: c^M es inalcanzable en \mathcal{M}'

Repaso de Máquina Turing

Fijamos $\Sigma = \{1, *\}$.

Recordar que una **máquina de Turing** es una tupla

$$M = (\Sigma, Q, T, q_0, q_f)$$

donde

- ▶ Σ (finito) es el conjunto **símbolos** que puede escribir en la cinta
- ▶ Q (finito) es el conjunto de **estados**
 - ▶ tiene dos estados distinguidos:
 - ▶ $q_0 \in Q$ es el **estado inicial**
 - ▶ $q_f \in Q$ es el **estado final**
- ▶ $T \subseteq Q \times \Sigma \times \Sigma \cup \{L, R\} \times Q$ es la **tabla finita de instrucciones**

Modelo de cómputo de máquina de Turing

Recordar que la máquina

$$M = (\Sigma, Q, T, q_0, q_f)$$

con entrada $w \in \{1\}^+$ **termina** (notado $M(w) \downarrow$) sii partiendo de w en la cinta de entrada y la cabeza leyendo el primer caracter después de w ,

$$\dots * * 1 1 \dots 1 * * \dots$$

q_0

llega al estado q_f después de una cantidad finita de pasos.

No es computable determinar si una máquina de Turing termina o no.

Idea de la demostración de que Primer Orden es indecidible

- ▶ fijar un lenguaje adecuado \mathcal{L}
- ▶ dada una máquina M y $w \in \{1\}^+$, construir (uniformemente) una sentencia $\varphi_{M,w} \in \text{FORM}(\mathcal{L})$ tal que

$$M(w) \downarrow \quad \text{sii} \quad \vdash \varphi_{M,w}$$

- ▶ si el problema de determinar si vale $\vdash \psi$ o $\not\vdash \psi$ para $\psi \in \text{FORM}(\mathcal{L})$ fuese computable, en particular sería computable determinar si $\vdash \varphi_{M,w}$ o $\not\vdash \varphi_{M,w}$ para cualquier máquina M y entrada w .
- ▶ como esto último es no-computable, tampoco es computable determinar si vale $\vdash \psi$ o $\not\vdash \psi$ para cualquier $\psi \in \text{FORM}(\mathcal{L})$

Dados M y w , ¿quién es $\varphi_{M,w}$?

- ▶ una fórmula de \mathcal{L} que se construye **computablemente** a partir de M y w
- ▶ $\varphi_{M,w}$ describe el **comportamiento** de M con entrada w en una cierta interpretación \mathcal{A}
- ▶ $\varphi_{M,w}$ es una **fórmula-programa**

El lenguaje \mathcal{L}

- ▶ **símbolos de constante:**
 - ▶ uno solo: ϵ
- ▶ **símbolos de función:**
 - ▶ la función 1 unaria
 - ▶ la función * unaria
- ▶ **símbolos de relación:**
 - ▶ infinitos (tantas como necesitemos) símbolos de relaciones binarias
 - ▶ sea $E = \{q_0, q_f, p, q, r, \dots\}$ un conjunto infinito de estados que podemos llegar a usar en máquinas de Turing
 - ▶ cada máquina particular usará solo una cantidad **finita** de estados de E
 - ▶ los símbolos de relación son:

$$R_{q_0}, R_{q_f}, R_p, R_q, R_r, \dots$$

Notación de los términos de \mathcal{L}

- ▶ si t es un término de \mathcal{L} , $1(t)$ lo notamos $1t$
- ▶ si t es un término de \mathcal{L} , $*(t)$ lo notamos $*t$

Por ejemplo

- ▶ $1(x)$ lo notamos $1x$
- ▶ $1(1(\epsilon))$ lo notamos 11ϵ
- ▶ $1(1(*(1(y))))$ lo notamos $11 * 1y$

La interpretación \mathcal{A}

Dada una máquina

$$M = (\Sigma, Q, T, q_0, q_f)$$

y una entrada $w \in \{1\}^+$, fijamos una interpretación $\mathcal{A} = \mathcal{A}_{M,w}$

- ▶ **el universo:** $A = \{1, *\}^*$ = cadenas finitas sobre $\{1, *\}$
 - ▶ va a representar datos en la cinta de M
- ▶ $\epsilon_{\mathcal{A}}$ = cadena vacía
 - ▶ la cinta es infinita, pero infinitos $*$ se representan como ϵ
- ▶ las funciones

$$1_{\mathcal{A}} : A \rightarrow A \quad \text{y} \quad *_{\mathcal{A}} : A \rightarrow A$$

se interpretan así:

- ▶ $1_{\mathcal{A}}(x) = 1x$, o sea la cadena que empieza por 1 y sigue con x
- ▶ $*_{\mathcal{A}}(x) = *x$, o sea la cadena que empieza por $*$ y sigue con x
- ▶ para $q \in Q$, $(R_q)_{\mathcal{A}}(x, y)$ es verdadero sii la máquina M con entrada w llega a una configuración en la que:
 - ▶ el estado es q
 - ▶ en la cinta está escrito x en orden inverso y a continuación y
 - ▶ la cabeza de M apunta al primer caracter de y

Definición de la fórmula-programa $\varphi_{M,w}$

Dada una máquina

$$M = (\Sigma, Q, T, q_0, q_f)$$

y una entrada

$$w = \overbrace{1 \dots 1}^k$$

fijamos la interpretación $\mathcal{A} = \mathcal{A}_{M,w}$ que acabamos de ver.

- ▶ $\varphi_0 := R_{q_0}(\overbrace{1 \dots 1}^k \in, \in)$
 - ▶ dice: “el estado inicial es alcanzable”
 - ▶ $\mathcal{A} \models \varphi_0$

- ▶ $\varphi_f := (\exists x)(\exists y) R_{q_f}(x, y)$
 - ▶ dice: “el estado final es alcanzable”
 - ▶ $\mathcal{A} \models \varphi_f$ sii $M(w) \downarrow$

Definición de la fórmula-programa $\varphi_{M,w}$

Para cada instrucción $I \in T$:

- ▶ si I dice *si M está en el estado q y lee un 1, escribir b y pasar al estado r* , definir

$$\psi_I := (\forall x)(\forall y) (R_q(x, 1y) \rightarrow R_r(x, by))$$

- ▶ si I dice *si M está en el estado q y lee un $*$, escribir b y pasar al estado r* , definir

$$\psi_I := (\forall x)(\forall y) (R_q(x, *y) \rightarrow R_r(x, by)) \wedge (\forall x) (R_q(x, \epsilon) \rightarrow R_r(x, b\epsilon))$$

Definición de la fórmula-programa $\varphi_{M,w}$

- ▶ si I dice *si M está en el estado q y lee un 1 , moverse a la izquierda y pasar al estado r* , definir

$$\begin{aligned}\psi_I &:= (\forall x)(\forall y) (R_q(1x, 1y) \rightarrow R_r(x, 11y)) \wedge \\ &(\forall x)(\forall y) (R_q(*x, 1y) \rightarrow R_r(x, *1y)) \wedge \\ &(\forall y) (R_q(\epsilon, 1y) \rightarrow R_r(\epsilon, *1y))\end{aligned}$$

- ▶ si I dice *si M está en el estado q y lee un $*$ moverse a la izquierda y pasar al estado r* , definir

$$\begin{aligned}\psi_I &:= (\forall x)(\forall y) (R_q(1x, *y) \rightarrow R_r(x, 1 * y)) \wedge \\ &(\forall x)(\forall y) (R_q(*x, *y) \rightarrow R_r(x, **y)) \wedge \\ &(\forall y) (R_q(\epsilon, *y) \rightarrow R_r(\epsilon, **y)) \wedge \\ &(\forall x) (R_q(*x, \epsilon) \rightarrow R_r(x, \epsilon)) \wedge \\ &(\forall x) (R_q(1x, \epsilon) \rightarrow R_r(x, 1\epsilon)) \wedge \\ &(R_q(\epsilon, \epsilon) \rightarrow R_r(\epsilon, \epsilon))\end{aligned}$$

(Misma idea con *moverse a la derecha...*)

Definición de la fórmula-programa $\varphi_{M,w}$

Recordar que la máquina

$$M = (\Sigma, Q, T, q_0, q_f)$$

tiene siempre un conjunto **finito** de instrucciones T

Definimos

$$\varphi_{M,w} := (\varphi_0 \wedge \bigwedge_{I \in T} \psi_I) \rightarrow \varphi_f$$

Proposición

Si $\mathcal{A} \models \varphi_{M,w}$ sii $M(w) \downarrow$.

Demostración.

Sabemos que $\mathcal{A} \models \varphi_0$. Sabemos que $\mathcal{A} \models \varphi_f$ sii $M(w) \downarrow$. Es fácil ver que $\mathcal{A} \models \psi_I$ para cada $I \in T$.

Luego $\mathcal{A} \models \varphi_{M,w}$ sii $\mathcal{A} \models \varphi_f$ sii $M(w) \downarrow$. □

Entscheidungsproblem

Teorema

$\vdash \varphi_{M,w}$ sii $M(w) \downarrow$.

Demostración.

(\Rightarrow) Si $\vdash \varphi_{M,w}$ entonces $\models \varphi_{M,w}$, es decir, $\varphi_{M,w}$ es verdadera en toda interpretación. En particular, $\mathcal{A} \models \varphi_{M,w}$. Luego $M(w) \downarrow$.

(\Leftarrow) Idea. Si $M(w) \downarrow$ entonces existe un cómputo de $M(w)$:

$$(x_1, r_1, y_1) \rightsquigarrow (x_2, r_2, y_2) \rightsquigarrow \cdots \rightsquigarrow (x_n, r_n, y_n)$$

con $x_i, y_i \in \{1, *\}^*$, $r_i \in Q$, $x_1 = w$, $r_1 = q_0$, $y_1 = \epsilon$, $r_n = q_f$.

Cada (x_i, r_i, y_i) representa una **configuración** del cómputo $M(w)$:

- ▶ el estado es r_i
- ▶ la cinta contiene $\cdots *** [x_i] [y_i] *** \cdots$
- ▶ la cabeza está apuntando al primer caracter de y_i

Cada paso de la ejecución coincide con una sustitución de una de las fórmulas ψ_I .

- ▶ cómputo de $M(w) =$ demostración de $\varphi_{M,w}$
- ▶ fórmula $\varphi_{M,w} =$ programa de M

Programa = Demostración

Recordemos que

$$\varphi_{M,w} := (\varphi_0 \wedge \bigwedge_{I \in T} \psi_I) \rightarrow \varphi_f$$

Supongamos que M con entrada 111 da el siguiente cómputo:

$$(111, q_0, *) \xrightarrow{(q_0, *, L, q_1) \in T} (11, q_1, 1) \xrightarrow{(q_1, 1, L, q_f) \in T} (1, q_f, 11)$$

Veamos que $\{\varphi_0\} \cup \{\psi_I \mid I \in T\} \vdash \varphi_f$. Esto prueba que $\vdash \varphi_{M,w}$.

- ▶ Recordemos que

$$\psi_{(q_0, *, L, q_1)} := \dots \wedge (\forall x) (R_{q_0}(1x, \epsilon) \rightarrow R_{q_1}(x, 1\epsilon)) \wedge \dots$$

$$\text{Por SQ4, } \{\varphi_0\} \cup \{\psi_I \mid I \in T\} \vdash (R_{q_0}(111\epsilon, \epsilon) \rightarrow R_{q_1}(11\epsilon, 1\epsilon))$$

- ▶ Recordemos que

$$\psi_{(q_1, 1, L, q_f)} := (\forall x)(\forall y) (R_{q_1}(1x, 1y) \rightarrow R_{q_f}(x, 11y)) \wedge \dots$$

$$\text{Por SQ4, } \{\varphi_0\} \cup \{\psi_I \mid I \in T\} \vdash (R_{q_1}(11\epsilon, 1\epsilon) \rightarrow R_{q_f}(1\epsilon, 11\epsilon))$$

- ▶ Recordemos que $\varphi_0 := R_{q_0}(111\epsilon, \epsilon)$
- ▶ Por MP, concluimos $\{\varphi_0\} \cup \{\psi_I \mid I \in T\} \vdash R_{q_f}(1\epsilon, 11\epsilon)$
- ▶ De esto, se puede concluir $\{\varphi_0\} \cup \{\psi_I \mid I \in T\} \vdash \underbrace{(\exists x)(\exists y) R_{q_f}(x, y)}_{\varphi_f}$

Entscheidungsproblem

Teorema (Turing, 1936)

Sea \mathcal{L} el lenguaje descrito y sea $\psi \in \text{FORM}(\mathcal{L})$. El problema de decidir si $\vdash \psi$ o $\not\vdash \psi$ no es computable.

Demostración.

Supongamos que hay un programa que dada $\psi \in \text{FORM}(\mathcal{L})$ devuelve verdadero si $\vdash \psi$.

Dada M y w , habría un procedimiento para decidir si $M(w) \downarrow$ o $M(w) \uparrow$:

1. construir $\varphi_{M,w}$ (esto se hace computablemente)
2. si $\vdash \varphi_{M,w}$ entonces $M(w) \downarrow$; si no $M(w) \uparrow$



Lógica de Primer Orden

Clase 5 - Opcional

Teorema de incompletitud de Gödel

Aritmética de Peano

Lenguaje $\mathcal{L} = \{0, S, +, \cdot\}$ con igualdad.

- ▶ axiomas (para $x, y \in \text{VAR}$, $\varphi \in \text{FORM}(\mathcal{L})$ con variable libre x)

S1. $0 \neq S(x)$

S2. $S(x) = S(y) \rightarrow x = y$

S3. $x + 0 = x$

S4. $x + S(y) = S(x + y)$

S5. $x \cdot 0 = 0$

S6. $x \cdot S(y) = (x \cdot y) + x$

S7. $(\varphi[x/0] \wedge (\forall x)(\varphi \rightarrow \varphi[x/S(x)])) \rightarrow (\forall x)\varphi$

- ▶ definimos la teoría S :

$$S = \{\psi : \psi \text{ es instanciación de alguno de estos 7 esquemas}\}$$

Sea $\mathcal{N} = \langle \mathbb{N}; 0, S, +, \cdot \rangle$ el modelo estándar de los naturales.

- ▶ querríamos capturar todas las verdades de \mathcal{N} con los teoremas de S (en el sistema $SQ^=$)

- ▶ querríamos $\mathcal{N} \models \varphi$ sii $S \vdash \varphi$

- ▶ se puede ver que $\mathcal{N} \models \varphi \iff S \vdash \varphi$

Notación (solo para esta clase)

- ▶ notamos $\varphi(x_1, \dots, x_n)$ a una fórmula que tiene variables libres x_1, \dots, x_n
- ▶ sea $\varphi(x_1, \dots, x_n)$ y sean t_1, \dots, t_n términos

$\varphi(t_1, \dots, t_n)$ representa $\varphi[x_1, \dots, x_n/t_1, \dots, t_n]$

- ▶ notación para los numerales:

$$\bar{1} = S(0)$$

$$\bar{2} = S(S(0))$$

\vdots

$$\bar{n} = \underbrace{S(\dots S(0)\dots)}_{n \text{ veces}}$$

Por ejemplo,

- ▶ $\varphi(x) = (\exists y)y + \bar{2} = x$
- ▶ $\varphi(\bar{3}) = (\exists y)y + \bar{2} = \bar{3}$

Aritmetización de fórmulas

()	¬	→	∀	=	0	S	+	·	x ₁	x ₂	x ₃	...
1	2	3	4	5	6	7	8	9	10	11	12	13	...

Por ejemplo,

(∀	x ₁)	¬	S	(x ₁)	=	x ₁	
2	3	5	7	11	13	17	19	23	29	31	
1	5	11	2	3	8	1	11	2	6	11	
Π	2 ¹	3 ⁵	5 ¹¹	7 ²	11 ³	13 ⁸	17 ¹	19 ¹¹	23 ²	29 ⁶	31 ¹¹

- ▶ toda fórmula de \mathcal{L} se representa con un número natural (se llama **número de Gödel** de la fórmula)
- ▶ de la misma manera, toda demostración en S se representa con un número natural (se llama número de Gödel de la demostración)

Resultados previos

Teorema

Los siguientes predicados son primitivos recursivos

- ▶ $var(x)$: x es el número de Gödel de una expresión que consiste de una variable
- ▶ $term(x)$: x es el número de Gödel de una expresión que consiste de un término
- ▶ $form(x)$: x es el número de Gödel de una fórmula de $FORM(\mathcal{L})$
- ▶ $axSQ_i(x)$: x es el número de Gödel de una instancia del i -ésimo axioma de $SQ^=$
- ▶ $axS_i(x)$: x es el número de Gödel de una instancia del i -ésimo axioma de S
- ▶ $MP(x, y, z)$: z es el número de Gödel de una expresión que resulta de MP de las expresiones con número de Gödel x e y
- ▶ $dem(x) = x$ es el número de Gödel de una demostración de S

Funciones expresables en S

Una relación $R \subseteq \mathbb{N}^n$ es **expresable en S** si existe una fórmula φ con (únicas) variables libres x_1, \dots, x_n tal que para todo $k_1, \dots, k_n \in \mathbb{N}$:

- ▶ si $R(k_1, \dots, k_n)$ es verdadero en \mathcal{N} entonces

$$S \vdash \varphi(\overline{k_1}, \dots, \overline{k_n})$$

- ▶ si $R(k_1, \dots, k_n)$ es falso en \mathcal{N} entonces

$$S \vdash \neg\varphi(\overline{k_1}, \dots, \overline{k_n})$$

Teorema

Toda relación computable es expresable en S .

Consistencia y ω -consistencia

Sea $\Gamma \subseteq \text{FORM}(\mathcal{L})$

Γ es **consistente** si no existe $\varphi \in \text{FORM}(\mathcal{L})$ tal que

$$\Gamma \vdash \varphi \quad \text{y} \quad \Gamma \vdash \neg\varphi$$

Γ es **ω -consistente** cuando

si $\Gamma \vdash \varphi(\bar{n})$ para todo n , entonces $\Gamma \not\vdash (\exists x)\neg\varphi(x)$

Proposición

Si Γ es ω -consistente entonces Γ es consistente.

Una teoría es Γ **completa** si para toda sentencia φ , $\Gamma \vdash \varphi$ o $\Gamma \vdash \neg\varphi$

La fórmula de Gödel

$W(e, y)$: e es el número de Gödel de una fórmula ψ
con una única variable libre x_1 y además
 y es el número de Gödel de una demostración en S de $\psi(\bar{e})$

El predicado $W : \mathbb{N}^2 \rightarrow \{0, 1\}$ es primitivo recursivo, luego es expresable en S por una fórmula $\mathcal{W}(x_1, x_2)$. Consideremos

$$\begin{aligned}\varphi(x_1) &= (\forall x_2) \neg \mathcal{W}(x_1, x_2) \\ &= \text{"la fórmula con número de Gödel } x_1 \text{ instanciada en } \bar{x}_1 \\ &\quad \text{no es demostrable en } S\end{aligned}$$

Sea m el número de Gödel de $\varphi(x_1)$. Consideremos

$$\begin{aligned}\varphi(\bar{m}) &= (\forall x_2) \neg \mathcal{W}(\bar{m}, x_2) \\ &= \text{"la fórmula con número de Gödel } m \text{ instanciada en } \bar{m} \\ &\quad \text{no es demostrable en } S\end{aligned}$$
$$\begin{aligned}&= \text{"}\varphi(\bar{m}) \text{ no es demostrable en } S\text{"} \\ &= \text{"yo no soy demostrable en } S\end{aligned}$$

Teorema de incompletitud de Gödel (1931)

Recordemos que m es el número de Gödel de

$$\varphi(x_1) = (\forall x_2) \neg \mathcal{W}(x_1, x_2)$$

Teorema

1. si S es consistente, $S \not\vdash \varphi(\bar{m})$
2. si S es ω -consistente, $S \not\vdash \neg\varphi(\bar{m})$ } si S es ω -consistente, es incompleto

Demostración.

1. Sup. $S \vdash (\forall x_2) \neg \mathcal{W}(\bar{m}, x_2)$
- ▶ sea k el número de Gödel de alguna demostración en S
 - ▶ $\mathcal{W}(m, k)$ es verdadero
 - ▶ $S \vdash \mathcal{W}(\bar{m}, \bar{k})$
 - ▶ como $S \vdash (\forall x_2) \neg \mathcal{W}(\bar{m}, x_2)$ por SQ4, $S \vdash \neg \mathcal{W}(\bar{m}, \bar{k})$
 - ▶ S es inconsistente

2. Sup. $S \vdash \neg(\forall x_2) \neg \mathcal{W}(\bar{m}, x_2)$
- ▶ como S es consistente, $S \not\vdash (\forall x_2) \neg \mathcal{W}(\bar{m}, x_2)$
 - ▶ $\mathcal{W}(m, k)$ es falso para todo k
 - ▶ $S \vdash \neg \mathcal{W}(\bar{m}, \bar{k})$ para todo k
 - ▶ como S es ω -consistente, $S \not\vdash (\exists x_2) \neg \neg \mathcal{W}(\bar{m}, x_2)$
 - ▶ $S \not\vdash \neg(\forall x_2) \neg \mathcal{W}(\bar{m}, x_2)$
 - ▶ $S \not\vdash \neg\varphi(\bar{m})$

Decimos que $\varphi(\bar{m})$ es independiente



Fórmulas verdaderas en \mathcal{N} pero no demostrables en S

Recordemos que m es el número de Gödel de

$$\varphi(x_1) = (\forall x_2)\neg\mathcal{W}(x_1, x_2)$$

de modo que

$$\begin{aligned}\varphi(\bar{m}) &= (\forall x_2)\neg\mathcal{W}(\bar{m}, x_2) \\ &= \text{"}\varphi(\bar{m}) \text{ no es demostrable en } S\text{"}\end{aligned}$$

- ▶ si $\varphi(\bar{m})$ fuese falsa en \mathcal{N} (i.e. $\mathcal{N} \not\models \varphi(\bar{m})$), $\varphi(\bar{m})$ sería demostrable en S , pero acabamos de ver que esto no es así
- ▶ entonces $\varphi(\bar{m})$ es **verdadera** en \mathcal{N} , pero **no demostrable** en S :

$$\mathcal{N} \models \varphi(\bar{m}) \quad \text{y} \quad S \not\vdash \varphi(\bar{m})$$

- ▶ esto no contradice el teorema de completitud:

$$\underbrace{S \not\models \varphi(\bar{m})} \quad \text{sii} \quad S \not\vdash \varphi(\bar{m})$$

hay un modelo de S
en donde $\varphi(\bar{m})$ es falsa

Teorema de Gödel-Rosser (1936)

Teorema

Si S es consistente, es incompleta.

Una teoría Γ es **recursivamente axiomatizable** si existe una teoría Γ' tal que “ $\ulcorner x \in \Gamma' \urcorner$ ” es computable y $\Gamma \vdash \varphi$ sii $\Gamma' \vdash \varphi$

Corolario

Cualquier teoría recursivamente axiomatizable que extiende a S es incompleta.

Teorías completas para \mathcal{N}

Sin embargo, es posible dar teorías completas que capturen todas las verdades de \mathcal{N}

Para $\mathcal{L} = \{0, S, +, \cdot\}$ con igualdad, existe $\Gamma \subseteq \text{FORM}(\mathcal{L})$ tal que $\Gamma \vdash \varphi$ sii $\mathcal{N} \models \varphi$:

$$\blacktriangleright \Gamma = \{\psi : \mathcal{N} \models \psi\}$$

Γ es completa, pero no es **decidible**.

Aritmética de Robinson (1950)

Lenguaje $\mathcal{L} = \{0, S, +, \cdot\}$.

► axiomas (para $x, y \in \text{VAR}$)

R1. $S(x) = S(y) \rightarrow x = y$

R2. $0 \neq S(x)$

R3. $x \neq 0 \rightarrow (\exists y)x = S(y)$

R4. $x + 0 = x$

R5. $x + S(y) = S(x + y)$

R6. $x \cdot 0 = 0$

R7. $x \cdot S(y) = (x \cdot y) + x$

► definimos la teoría R :

$$R = \{\psi : \psi \text{ es instanciación de alguno de estos 7 esquemas}\}$$

► R es una sub-teoría de S

► R no tiene axioma de inducción

► R es incompleta

Aritmética de Presburger (1929)

Lenguaje $\mathcal{L} = \{0, S, +\}$ con igualdad. Sin \cdot .

- ▶ axiomas (como S pero sin S5 ni S6):

S1. $0 \neq S(x)$

S2. $S(x) = S(y) \rightarrow x = y$

S3. $x + 0 = x$

S4. $x + S(y) = S(x + y)$

S7. $(\varphi[x/0] \wedge (\forall x)(\varphi \rightarrow \varphi[x/S(x)])) \rightarrow (\forall x)\varphi$

- ▶ definimos la teoría P :

$$P = \{ \psi : \psi \text{ es instanciación de alguno de estos 5 esquemas} \}$$

- ▶ P es completa
- ▶ P es decidable

Cuerpos real cerrados - el sistema RCF

Lenguaje $\mathcal{L} = \{0, +, -, *\}$ con igualdad

- ▶ $(\forall x, y, z) (x + y) + z = x + (y + z)$
- ▶ $(\forall x) x + 0 = x$
- ▶ $(\forall x) x - x = 0$
- ▶ $(\forall x, y) x + y = y + x$
- ▶ $(\forall x, y, z) (xy)z = x(yz)$
- ▶ $(\forall x) x1 = x$
- ▶ $(\forall x) 1x = x$
- ▶ $(\forall x, y, z) x(y + z) = xy + xz$
- ▶ $(\forall x, y, z) (x + y)z = xz + yz$
- ▶ $(\forall x, y) xy = yx$
- ▶ $0 \neq 1$
- ▶ $(\forall x)(x \neq 0 \rightarrow (\exists y) xy = 1)$
- ▶ $(\forall x_1 \dots x_n) x_1^2 + \dots + x_n^2 \neq -1$ para todo $n > 0$
- ▶ $(\forall x)(\exists y) (x = y^2 \vee -x = y^2)$
- ▶ $(\forall x_1 \dots x_n)(\exists y) y^n + x_1 y^{n-1} + \dots + x_{n-1} y + x_n = 0$ para todo n impar

- ▶ RCF es una teoría completa.
- ▶ Si \mathcal{R} son los reales estándar, entonces $\mathcal{R} \models \varphi$ sii $\text{RCF} \vdash \varphi$ para toda sentencia φ
- ▶ RCF es decidible (decidir si $\text{RCF} \vdash \varphi$ o $\text{RCF} \not\vdash \varphi$) pero con complejidad muy alta: peor que cualquier torre $2^{2^{\dots^n}}$, donde n es la longitud de la fórmula φ