Lógica y Computabilidad

Gianfranco Zambonni

21 de enero de 2020

Índice

Parte I
Computabilidad

1. Máquina de Turing

Una máquina de Turing está compuesta por:

- Una **cinta infinita** dividida en celdas que contienen un símbolo de un alfabeto Σ. En esta materia, Σ siempre contiene al símbolo $*$, que representa el blanco, y nunca contiene a L ni a R, que son las etiquetas usadas para indicar al cabezal hacia que lado debe moverse.
- El **cabezal** lee un símbolo y, dependiendo del estado de la máquina, puede escribir uno nuevo o moverse una posición a la derecha o una a la izquierda. Cuando completa una acción, cambia el estado de la máquina.
- Una **tabla finita de instrucciones** que, dado un estado y el símbolo que lee el cabezal, indica que acción debe ser tomada y cual es el estado al que se tiene que pasar.

1.1. Tabla de instrucciones

Cada instrucción es una tupla $(q, s, a, q') \in Q \times \Sigma \times (\Sigma \cup \{L, R\}) \times Q$ tal que:

- $q, q' \in Q$ son estados de la máquina de turing. q es el estado necesario para ejecutar la instrucción y q' el estado en el que queda la máquina después de ejecutarla.
- $s \in \Sigma$ es el símbolo que se tiene que leer cuando la máquina está en q para que la instrucción sea ejecutada.
- $a \in \Sigma \cup \{L, R\}$ es la acción a realizar. Puede ser escribir un símbolo del álfabeto Σ o mover el cabezal hacia alguno de los lados.

Entonces, la tupla (q, s, a, q') 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'*".

1.2. Definición mátematica

Una máquina de Turing $\mathcal{M}$ es una tupla (Σ, Q, T, q_0, q_f) donde:

- Σ es un conjunto finito de símbolos ($L, R \in \Sigma$ y $* \in \Sigma$)
- Q es un conjunto finito de **estados**, de los cuales dos son:
  - el **estado inicial** q_0
  - y el **estado final** q_f.
- $T \subseteq Q \times \Sigma \times \Sigma$ es la **tabla de instrucciones**.

Hay dos tipos de máquinas de turing:

- **Deterministicas:** Es cuando no hay dos instrucciones que tengan el mismo estado inicial y necesiten leer el mismo símbolo, es decir:

$$\nexists(q_1, s_1, a_1, q_1'), (q_2, s_2, a_2, q_2') \in T \text{ tal que } q_1 = q_2 \ \wedge \ s_1 = s_2$$

- **No deterministicas:** Cuando no es deterministica, osea que cabe la posibilidad que se ejecuten más de una instrucción en un paso y la máquina este en dos o más estados simultáneamente.

$$\exists(q_1, s_1, a_1, q_1'), (q_2, s_2, a_2, q_2') \in T \text{ tal que } q_1 = q_2 \ \wedge \ s_1 = s_2$$

1.3. Representación de números y tuplas

1.3.1. Números naturales

Sea $\sigma = \{*, 1\}$, representaremos a los números naturales en unario (como si usasemos palitos). La representación $\bar{x}$ de $x \in \mathbb{N}$, consta de $x + 1$ palitos.

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

Ejemplo: El 0 es "1", el 1 es "11", el 2 es "111", etc.

1.3.2. Tuplas

Las tuplas $(x_1, \ldots, x_n)$ las representamos como una lista de (representaciones de) x_i separados por un blanco $(*)$.

$$*\bar{x_1} * \bar{x_2} * \cdots * \bar{x_n} *$$

Ejemplo: La tupla (1,2) sería $*11 * 111*$

1.4. Funciones parciales

Sea $f : \mathbb{N}^n \to \mathbb{N}$, f es una función parcial si está definida para algunos (tal vez ninguno; tal vez todos) sus argumentos.

Notación:

- $f(x_1, \ldots, x_n) \downarrow$: f está definida para la tupla $(x_1, \ldots, x_n)$ y, en este caso, $f(x_1, \ldots, x_n)$ es un natural.
- $f(x_1, \ldots, x_n) \uparrow$: f se indefine para la tupla $(x_1, \ldots, x_n)$.

Dominio: Conjunto de argumentos para los que f está definida y se nota $\text{dom}(f)$.

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

Función Total: $f : \mathbb{N}^n \to \mathbb{N}$ es total si está definida para todos sus posibles argumentos:

$$\text{dom} f = \mathbb{N}^n$$

1.4.1. Cómputo de funciones parciales en máquinas de Turing

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

- Si $f(x_1, \ldots, x_n) \downarrow$ entonces, siguiendo sus instrucciones en T, llega al estado q_f,
- Si $f(x_1, \ldots, x_n) \uparrow$ nunca termina en el estado q_f

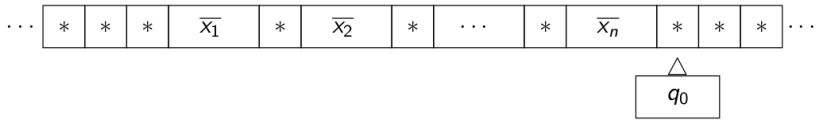

Figura 1: Estado inicial de la máquina de turring

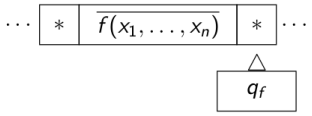

Figura 2: Estado final de la máquina de turing

1.4.2. Poder de cómputo

Sea $f : \mathbb{N}^m \to \mathbb{N}$ 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.

2. Funciones primitivas recursivas y clases PRC

2.1. Funciones iniciales

Función calculable de manera efectiva: Son funciones que podemos escribir combinando, de alguna manera, funciones más simples que sabemos que ya sabemos que son efectivas.

La idea es que, dado un **conjunto inicial** de funciones, podamos combinarlas de ciertas formas para conseguir nuevas funciones que permitan realizar cálculos más complejos.

Funciones iniciales:

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

Composición: Sea $f : \mathbb{N}^k \to \mathbb{N}$ y $g_1, \ldots, g_k : \mathbb{N}^n \to \mathbb{N}$, entonces $h : \mathbb{N}^n \to \mathbb{N}$ se obtiene a partir de composición de f y $g_1, \ldots, g_k$ por composición si:

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

En este contexto, una constante k puede ser definida como:

$$h(t) = s^{(k)}(n(t))$$

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

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

2.2. Clases Primitivas Recursivas Cerradas (PRC)

Función primitiva recursiva: Función que se puede obtener a partir de las funciones iniciales por un número finito de aplicaciones de composición y recursión primitiva.

Clase PRC: Una clase $\mathcal{C}$ de funciones totales es **PRC** si

- Las funciones iniciales están en $\mathcal{C}$.
- 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 $\mathcal{C}$.

Corolario 2.1. *La clase de funciones primitivas recursivas es una clase PRC.*

Teorema 2.1. *La clase de funciones totales Turing computables es una clase PRC.*

Teorema 2.2. *Una función es p.r. si y solo si pertenece a toda clase PRC.*

DEMOSTRACIÓN

⇐) Si una función f pertenece a todas las clases PRC, en particular, pertenece a la clase de funciones primitiva recursivas. Por lo tanto f es primitiva recursiva.

⇒) Sea f una función primitiva recursiva y sea $\mathcal{C}$ una clase PRC. Como f es p.r, hay una lista $f_1, \ldots, f_n$ tal que

- $f = f_n$ y
- f_i:
  - es una función incial (por lo tanto es p.r y está en $\mathcal{C}$) ó
  - o se obtiene por composición o recursión primitiva a partir de funciones f_j con $j < i$ (luego tambien está en $\mathcal{C}$)

Osea que $f = f_n$ o es una función inicial (y por lo tanto está en $\mathcal{C}$) ó se obtiene por composición o recursión primitiva de algunas de las anteriores (que están en $\mathcal{C}$ y por lo tanto f también lo está). Luego $f \in \mathcal{C}$

□

Corolario 2.2. *La clase de funciones primitivas recursivas es la clase PRC más chica.*

Corolario 2.3. *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 pertence a todas las clases PRC, en particular a la clase de funciones totales Turing computables. □

2.3. Funciones primitivas recursivas básicas

- Todas las constantes k están en todas las clases PRC.

  $$k = k(t) = s^{(k)}(n(t))$$

- $suma(x, y) = x + y$

  $$\begin{array}{lcl} suma(x, 0) & = & u_1^1(x) \\ suma(x, y+1) & = & g(suma(x, y), x, y) \\ & & \text{donde } g(x_1, x_2, x_3) = s(u_1^3(x_1, x_2, x_3)) \end{array}$$

- $x \cdot y$

$$\begin{aligned} x \cdot 0 &= 0 = n(x) \\ x \cdot (y+1) &= g(x \cdot y, x, y) = x + (x \cdot y) \end{aligned}$$

- x^y

$$\begin{aligned} x^0 &= 1 \\ x^{y+1} &= g(x^y, x, y) = x \cdot x^y \end{aligned}$$

- $factorial(x) = x!$

$$\begin{aligned} 0! &= 1 = s(n(x)) \\ (x+1)! &= g(x!, x) = (x+1) \cdot x! \end{aligned}$$

- $x \dot{-} y = \begin{cases} x - y & \text{si } x \geq y \\ 0 & \text{si no} \end{cases}$

$$\begin{aligned} x \dot{-} 0 &= x \\ x \dot{-} (y+1) &= g(x \dot{-} y, x, y) = (x \dot{-} y) \dot{-} 1 \end{aligned}$$

donde $x \dot{-} 1$ es:

$$\begin{aligned} 0 \dot{-} 1 &= 0 \\ (x+1) \dot{-} 1 &= g(x \dot{-} 1, x) = u_2^2(x \dot{-} 1, x) = x \end{aligned}$$

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

2.3.1. 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 primitivos recursivos son aquellos representados por funciones primitivas recursivas en $\{0, 1\}$.

2.3.2. Operadores lógicos

Teorema 2.3. *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

- $\neg p = \alpha(p)$
- $p \wedge q = p \cdot q$
- $p \vee q = \neg(\neg p \wedge \neg q)$

□

Corolario 2.4. *Si p y q son predicados primitivos recursivos también lo son los predicados $\neg p$, $p \vee q$ y $p \wedge q$.*

Corolario 2.5. *Si p y q son predicados totales Turing computables también lo son los predicados $\neg p$, $p \vee q$ y $p \wedge q$*

2.3.3. Definición por casos

Teorema 2.4. *Sea $\mathcal{C}$ una clase PRC. Sean $g_1, \ldots, g_m, h : \mathbb{N}^n \to \mathbb{N}$ funciones en $\mathcal{C}$ y sean $p_1, \ldots, p_m : \mathbb{N}^n \to \{0, 1\}$ predicados mutuamente excluyentes en $\mathcal{C}$. La función:*

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

está en $\mathcal{C}$

DEMOSTRACIÓN

$$\begin{aligned} f(x_1,\ldots,x_n) =& g_1(x_1,\ldots,x_n)\cdot p_1(x_1,\ldots,x_n) + \ldots \\ &+ g_m(x_1,\ldots,x_n)\cdot p_m(x_1,\ldots,x_n) \\ &+ h(x_1,\ldots,x_n)\cdot(\neg p_1(x_1,\ldots,x_n)\wedge\cdots\wedge\neg p_m(x_1,\ldots,x_n)) \end{aligned}$$

Si algún p_i es verdadero, valdrá 1 y el resto 0 porque todos los predicados son mutuamente excluyentes. Al evaluar f, obtendremos

$$f(x_1,\ldots,x_n) = g_i(x_1,\ldots,x_n) * p_i(x_1,\ldots,x_n) = g_i(x_1,\ldots,x_n)\cdot 1 = g_i(x_1,\ldots,x_n)$$

Si todos los predicados son falsos, entonces todas las g_i se multiplican por cero y, además vale la condición *si no* que es la conjuncion de sus negaciones. Y h es multiplicado por 1. □

2.3.4. Sumatorias, productorias

Teorema 2.5. *Sea* $\mathcal{C}$ *una clase PRC. Si* $f:\mathbb{N}^{n+1}\to\mathbb{N}$ *está en* $\mathcal{C}$ *entonces también están las funciones:*

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

DEMOSTRACIÓN

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

Para h hay que hacer lo mismo pero multiplicando en vez de sumar. □

Teorema 2.6. *Sea $\mathcal{C}$ una clase PRC. Si $f : \mathbb{N}^{n+1} \to \mathbb{N}$ está en $\mathcal{C}$ entonces también están las funciones:*

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

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

La demostración es la misma que la anterios, pero los resultados de los casos bases valen 0 y 1, respectivamente.

2.3.5. Cuantificadores acotados

Sean $p : \mathbb{N}^{n+1} \to \{0, 1\}$ un predicado:

$(\forall t)_{\leq y} p(t, x_1, \ldots, x_n)$ es verdadero si y solo si:

- $p(0, x_1, \ldots, x_n)$ es verdadero **y**

  $\vdots$

- $p(y, x_1, \ldots, x_n)$ es verdadero

$(\exists t)_{\leq y} p(t, x_1, \ldots, x_n)$ es verdadero si y solo si:

- $p(0, x_1, \ldots, x_n)$ es verdadero **o**

  $\vdots$

- $p(y, x_1, \ldots, x_n)$ es verdadero

De la misma manera se pueden definir el existencial y el para todo con $< y$ en vez de $\leq y$.

Teorema 2.7. *Sean $p : \mathbb{N}^{n+1} \to \{0, 1\}$ un predicado perteneciente a una clase PRC $\mathcal{C}$. Entonces los siguientes predicados también están en $\mathcal{C}$:*

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

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

DEMOSTRACIÓN

$(\forall t)_{\leq y} p(t, x_1, \ldots, x_n)$ si y solo si $\prod_{t=1}^{y} f(t, x_1, \ldots, x_n)) = 1$

$(\exists t)_{\leq y} p(t, x_1, \ldots, x_n)$ si y solo si $\sum_{t=1}^{y} f(t, x_1, \ldots, x_n)) \neq 0$

Definimos los cuantificadores en función de la sumatoria, productoria y la comparación de igualdad que están en $\mathcal{C}$ (Ver Seccion 2.3.4). Entonces el para todo y el existencial acotados por $\leq$ pertenece a $\mathcal{C}$. □

Teorema 2.8. *Sean $p : \mathbb{N}^{n+1} \to \{0,1\}$ un predicado perteneciente a una clase PRC $\mathcal{C}$. Entonces los siguientes predicados también están en $\mathcal{C}$:*

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

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

DEMOSTRACIÓN

$(\forall t)_{<y} p(t, x_1, \ldots, x_n)$ si y solo si $(\forall t)_{\leq y}(t = y \vee p(t, x_1, \ldots, x_n)$

$(\exists t)_{<y} p(t, x_1, \ldots, x_n)$ si y solo si $(\forall t)_{\leq y}(t \neq y \wedge p(t, x_1, \ldots, x_n)$

□

2.3.6. Minimización acotada

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

Teorema 2.9. *Sean $p : \mathbb{N}^{n+1} \to \{0,1\}$ un predicado de una clase PRC $\mathcal{C}$. Entonces la función $\min_{t \leq y} p(t, x_1, \ldots, x_n)$ también están en $\mathcal{C}$*

DEMOSTRACIÓN

$$\min_{t \leq y} p(t, x_1, \ldots, x_n) = \sum_{u=0}^{y} \prod_{t=0}^{u} \alpha(p(t, x_1, \ldots, x_n))$$

Veamos que esto funciona. Supongamos que t_0 es el mínimo valor que cumple p, entonces:

(1) $p(t_0, x_1, \ldots, x_n) = 1$

(2) $p(t, x_1, \ldots, x_n) = 0$ para todo $t < t_0$

Podemos dividir la sumotaria en dos partes:

$$\underbrace{\sum_{u=0}^{t_0-1} \prod_{t=0}^{u} \alpha(p(t, x_1, \ldots, x_n))}_{(3)} + \underbrace{\sum_{u=t_0}^{y} \prod_{t=0}^{u} \alpha(p(t, x_1, \ldots, x_n))}_{(4)}$$

(3) Son los primeros t_0 términos que corresponden a $u \in [0, t_0 - 1]$. Por (2) sabemos que $p(t, x_1, \ldots, x_n) = 0$ para todo $0 \leq t \leq u < t_0$. Entonces:

$$\sum_{u=0}^{t_0-1} \prod_{t=0}^{u} \alpha(p(t, x_1, \ldots, x_n)) = \sum_{u=0}^{t_0-1} \prod_{t=0}^{u} \alpha(0) = \sum_{u=0}^{t_0-1} \prod_{t=0}^{u} 1 = \sum_{u=0}^{t_0-1} 1 = t_0$$

(4) Resta ver que todos los términos que corresponden a $u \in [t_0, y]$ dan 0 (si diese algo más grande, el resultado final superaría a t_0).

Para esto solo hay que notar que todas la productorias restantes van desde $0 \leq t \leq u$ y que $u \geq t_0$, por lo que en todas se encuentra el valor $\alpha(p(t_0, x_1, \ldots, x_n)) = 0$ (t_0 cumple p). Luego cada una de ellas da cero .

□

2.3.7. Codificación de tuplas

Definimos la función primitiva recursiva

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

Proposición 2.1. *Hay una única solución (x, y) a la ecuación $\langle x, y\rangle = z$.*

- *x es el máximo número tal que $2^x|(z+1)$*
- $y = (\frac{z+1}{2^x} - 1)/2$

Observadores: Sea $z = \langle x, y\rangle$:

- $l(z) = x$
- $r(z) = y$

Proposición 2.2. *Los observadores de pares son primitivas recursivas*

DEMOSTRACIÓN

Como x e y son menores a $z+1$ tenemos que:

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

□

2.3.8. Codificación de secuencias

El **número de Gödel** de la secuencia $a_1, \ldots, a_n$ es el número:

$$[a_1, \ldots, a_n] = \prod_{i=1}^{n} p_i^{a_i}$$

donde p_i es el $i-$ésimo primo ($i \geq 1$)

Teorema 2.10. *Si $a_1, \ldots, a_n = b_1, \ldots, b_n$ entonces $a_i = b_i$ para todo $i \in \{1, \ldots, n\}$.*

Observación: La demostración del teorema se basa en que la factorización en primos de cualquier número es única.

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

Observadores: Sea $x = [a_1, \ldots, a_n]$:

- $x[i] = a_i$
- $|x|$ = longitud de x

Proposición 2.3. *Los observadores de secuencias son primitivas recursivas*

DEMOSTRACIÓN

- $x[i] = \min_{t \leq x} \left(\neg \left(p_i^{t+1} | x\right)\right)$

  Es el mínimo t tal que p_i^{t+1} no divide a x. La función p_i que nos devuelve el i-ésimo primo es primitiva recursiva y el predicado de divisibilidad son primitivos recursivos (Ver apéndice A)

- $|x| = \min_{i \leq x} \left(x[i] \neq 0 \wedge (\forall j)_{\leq x} \left(j \leq i \vee x[j] = 0\right)\right)$

□

3. Funciones $\mathcal{S}$-Computables

3.1. Sintaxis del lenguaje $\mathcal{S}$

En la materia usamos el lenguaje $\mathcal{S}$.

- Usaremos letras como variables cuyos valores serán números enteros no negativos. En particular:
  - Las letras $X_1, X_2, \ldots$ serán las **variables de entradas** de nuestro programa.
  - La letra Y, **variable de salida**. Siempre empieza inicializada en cero.
  - Las letras $Z_1, Z_2, \ldots$, las **variables locales**. Siempre empiezan inicializadas en cero.
- Un programa en $\mathcal{S}$ consistirá de una secuencia fínita de instrucciones. Tendremos de tres tipos:

Instrucción	Interpretación
$V \leftarrow V + 1$	Incrementa en uno el valor de la variable V.
$V \leftarrow V - 1$	Decrementa en uno el valor de la variable V. Si V es cero, no hace nada
IF $V \neq 0$ GOTO L	Si el valor de V no es cero, entonces ejecuta la instrucción con etiqueta L, sino sigue con la siguiente instrucción de la secuencia.

Un programa termina cuando se ejecuta la última instrucción o cuando se realiza un salto condicional a una etiqueta que no pertenece al programa

Macros: Cuando programemos habrá secuencias de intrucciones que usaremos frecuentemente. A éstas, les podremos asignar una abreviación que podremos usar como pseudo instrucciónes.

Cuando utilizemos macros, vamos a asumir que las etiquetas y variables usadas dentro de ellas se remplazan automáticamente por etiquetas y variables que no esté usando nuestro programa

Macro GOTO L : Nos permitirá realizar un salto sin la necesidad de una condición.

GOTO L:
$Z \leftarrow Z + 1$
IF $Z \neq 0$ GOTO L

Programa $Y \leftarrow X$: Asignar a Y el valor de X.

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

Si renombramos a X e Y por V_1 y V, respectivamente, conseguimos la macro asociada a este programa.

3.1.1. Estado

Estado de un programa P: Es una lista de equaciones de la forma $V = m$ donde V es una variable y m un número. La lista, incluye una de estas ecuaciones por cada variable que usa (no hay dos ecuaciones para la misma variable).

Programa P:

$[A]$ $X \leftarrow X - 1$
$Y \leftarrow Y + 1$
IF $X \neq 0$ GOTO A

Algunos estados de P:

- $X = 3, Y = 1$
- $X = 3, Y = 1, Z = 0$
- $X = 3, Y = 1, Z = 8$ (si bien no es alcanzable, es un estado válido)

3.1.2. Descripción instantánea

Descripción instantánea (snapshot): Sea P un programa de longitud n. Dado un estado σ de P y un $i \in 1, \ldots, n+1$, el par (i, σ) es una descripción instantanea de P e indica el valor de sus variables antes de ejecutar la i-ésima instrucción.

Descripción terminal: Si $i = n + 1$, entonces la descripción se llama **terminal**.

Para una descripción (i, σ), no terminal podemos definir su **sucesor** (j, τ) de la siguiente manera:

Si la i-ésima instrucción es:

- $V \leftarrow V + 1$ entonces $j = i + 1$ y τ es σ remplazando $V = m$ por $V = m + 1$
- $V \leftarrow V - 1$ entonces $j = i + 1$ y τ es σ remplazando $V = m$ por $V = \text{máx}\{m - 1, 0\}$
- IF $V \neq 0$ GOTO L entonces $\tau = \sigma$ y con j pueden suceder dos cosas:

- Si σ contiene la fórmula $V = 0$ entonces $j = i + 1$
- Si no:
  - Si existe una instrucción en P con etiqueta L, entonces

$$j = \text{mín}\{k : k\text{-ésima instrucción de } P \text{ tiene etiqueta } L\}$$

  - Si no, $j = n + 1$

Notar que podemos tenes más de una instrucción con la misma etiqueta. Sin embargo, nuestra definición de descripción instantanea, interpreta que un condicional siempre se refiere a la primera instrucción que la usa.

3.2. Funciones parciales computables

3.2.1. Cómputo

Estado inicial: Sea P un programa del lenguaje $\mathcal{S}$ y $r_1, \ldots, r_m$ números dados. Formamos el estado inicial σ_1 de P con las ecuaciones $X_1 = r_1,\ X_2 = r_2, \ldots, X_m = r_m, Y = 0$ y $V = 0$ para cada variable V que use el programa y no sean las ya mencionadas. Y $(1, \sigma_1)$ es la **descripción inicial**.

En las definiciones que usaremos, un mismo programa P puede servir para computar funciones de una variable, dos variables, etc. Esto depende de la cantidad de variables de entradas que especifiquemos. Si especificamos menos de las que deberíamos, el resto toma valor 0. Si especificamos más, entonces el programa simplemente las ignorará.

Computo de un programa P: Dado un programa P y una descripción inicial $d_1 = (1, \sigma)$, un cómputo es una secuencia finita de descripciones $d_1, \ldots, d_k$ tal que d_k es una descripción terminal. Si esta secuencia existe, notamos $\Psi_P^{(m)}(r_1, \ldots, r_m)$ al valor de Y en d_k.

En los casos en que la secuencia de instrucciones sea infinita (nunca lleguemos a un estado terminal si partimos desde d_1), diremos que $\Psi_P^{(m)}(r_1, \ldots, r_m)$ está indefinido.

$$\Psi_P^{(m)}(r_1, \ldots, r_m) = \begin{cases} \text{el valor de Y} & \text{si existe un cómputo de } P \text{ desde } (1, \sigma_1) \\ \uparrow & \text{si no} \end{cases}$$

3.2.2. Función parcial computable

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

$$f(r_1, \ldots, r_m) = \Psi_P^{(m)}(r_1, \ldots, r_m) \text{ para todo } (r_1, \ldots, r_m) \in \mathbb{N}^m$$

Esto quiere decir, que para $(r_1, \ldots, r_m)$, ambos lados están definidos y tienen el mismo valor o ambos lados están indefinidos.

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

3.2.3. Minimización no acotada

Definimos la minimización no acotada como:

$$\min_t p(t, x_1, \ldots, x_n) = \begin{cases} \text{mínimo } t \text{ tal que } p(t, x_1, \ldots, x_n) = 1 & \text{si existe tal } t \\ \uparrow & \text{si no} \end{cases}$$

Teorema 3.1. *Si $p : \mathbb{N}^{n+1} \to \{0, 1\}$ es un predicado computable entonces $\min_t p(t, x_1, \ldots, x_n)$ es parcial computable*

DEMOSTRACIÓN

Si mostramos un programa que computa $\min_t p(t, x_1, \ldots, x_n)$, entonces queda probado el teorema. Ese programa es:

$$\begin{array}{ll} [A] & {}^*\text{IF } p(Y, X_1, \ldots, X_n) \text{ GOTO } E \\ & Y \leftarrow Y + 1 \\ & \text{GOTO } A \end{array}$$

*Ver definición en el apéndice B.5 □

3.2.4. Clausura por composición y recursión primitiva

Teorema 3.2. *Si $h : \mathbb{N}^n \to \mathbb{N}$ se obtiene a partir de las funciones (parciales) computables $f, g_1, \ldots, g_k$ por composición, osea tiene la siguiente forma:*

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

entonces h es (parcial) computable.

DEMOSTRACIÓN

Se puede ver facilmente que el programa mostrado a continuación computa h. En el apéndice B.4, se define la macro que nos permite asignar el resultado de una función a una variable.

$$\begin{array}{c} Z_1 \leftarrow g_1(X_1, \ldots, X_n) \\ \vdots \\ Z_k \leftarrow g_k(X_1, \ldots, X_n) \\ Y \leftarrow f(Z_1, \ldots, Z_k) \end{array}$$

□

Proposición 3.1. *Si $f, g_1, \ldots, g_k$ son totales entonces h es total.*

Teorema 3.3. *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. La macro que asigna una constante y el programa que calcula X = 0 se pueden ver en el apéndice B.

$$
\begin{array}{ll}
 & Y \leftarrow f(X_1, \ldots, X_n) \\
[A] & \text{IF } X_{n+1} = 0 \text{ GOTO } E \\
 & Y \leftarrow g(Z, Y, X_1, \ldots, X_n) \\
 & Z \leftarrow Z + 1 \\
 & X_{n+1} \leftarrow X_{n+1} - 1 \\
 & \text{GOTO } A
\end{array}
$$

□

Proposición 3.2. *Si g es total entonces h es total.*

Teorema 3.4. *La clase de funciones computables es una clase PRC*

DEMOSTRACIÓN

Por los teorema 3.2 y 3.3, la clase de funciones computables está cerrada por composición y recursión primitiva. Osea que solo nos falta ver que las funciones iniciales son computables:

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

$$
\begin{array}{l}
Y \leftarrow X \\
Y \leftarrow Y + 1
\end{array}
$$

- $n(x) = 0$ se computa con el programa vacio.
- $u_i^n(x_1, \ldots, x_n)$ se computa con el programa:

$$Y \leftarrow X_i$$

□

Corolario 3.1. *Toda función primitiva recursiva es computable.*

3.3. Codificación de programas en $\mathcal{S}$

Vamos a asociar cada programa P el lenguaje S con un número $\#(P)$ de tal manera que podamos recuparlo solo conociendo su número. Para esto:

- Agregamos la instrucción $V \leftarrow V$ a $\mathcal{S}$ que no hace nada.
- Le damos un órden a las variables: $Y\ X_1\ Z_1\ X_2\ Z_2\ X_3\ Z_3 \ldots$
- Y hacemos los mismo para las etiquetas: $A_1\ B_1\ C_1\ D_1\ E_1\ A_2\ B_2\ C_2\ D_2\ E_2\ A_3$

Escribimos $\#(V)$ y $\#(L)$ para indicar la posición en la lista de una variable o etiquetas dada. Por ejemplo: $\#(X_2) = 4$, $\#(E_1) = 5$

Ahora, sea I una instrucción etiquetada (o no) del lenguaje S, entonces la podemos representar con una tupla $\#(I) = \langle a, \langle b, c\rangle\rangle$ donde:

1. Si I tiene etiqueta L, entonces $a = \#(L)$. Si no $a = 0$.
2. Si I es una instrucción del tipo:
   - $V \leftarrow V$ entonces $b = 0$
   - $V \leftarrow V + 1$ entonces $b = 1$
   - $V \leftarrow V - 1$ entonces $b = 2$
   - IF $V \neq 0$ GOTO L entonces $b = \#(L) + 2$
3. $c = \#(V) - 1$ siendo V la variable usada en esa instrucción.

Finalmente, sea P un programa que consiste de las instrucciones $I_1, \ldots, I_k$, lo codificamos como una secuencia de instrucciones:

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

Recordemos que tanto las tuplas como las listas pueden ser representadas como números naturales y sus observadores son primitivos recursivos (Ver Secciones 2.3.7 y 2.3.8).

Observemos que, por como habíamos definido las secuencias, pasaba que $[a_1, \ldots, a_k] = [a_1, \ldots, a_k, 0] = [a_1, \ldots, a_k, 0, 0] = [a_1, \ldots, a_k, 0, 0, \ldots]$. Ahora, $0 = \langle 0, \langle 0, 0\rangle\rangle = Y \leftarrow Y$ por lo que cada 0 agrega esta instrucción a un programa.

Si bien los programas $[a_1, \ldots, a_k]$, $[a_1, \ldots, a_k, 0]$ y $[a_1, \ldots, a_k, 0, 0, \ldots]$ son equivalentes, removemos esta ambigüedad pidiendo que ningún programa $\mathcal{S}$ puede terminar con la instrucción $Y \leftarrow Y$. Con esto, cada número representa a un único programa.

3.4. Teorema de Cantor

Teorema 3.5. *El conjunto de las funciones (totales)* $\mathbb{N} \to \mathbb{N}$ *no es numerable.*

DEMOSTRACIÓN

Supongamos que el conjunto mencionado es numerable, entonces puedo enumerar todas las funciones de la siguiente forma: $f_0, f_1, f_2, \ldots$.
Sea $g : \mathbb{N} \to \mathbb{N}$ tal que $g(x) = f_x(x) + 1$. Como g es una función de naturales en naturales, debería estar en la enumeración de funciones que hicimos más arriba. Por lo que $g = f_k$ para algún k.
Sin embargo, $g(k) = f_k(k) + 1 \neq f_k(k)$ para todo k. Luego, g no puede ser igual a ninguna de las funciones enumeradas. Absurdo ($f_0, f_1, \ldots$ era una enumeración de **todas** las funciones $\mathbb{N} \to \mathbb{N}$) □

3.5. El problema de la detención (Halting Problem)

Dado un programa P tal que $\#(P) = y$, entonces HALT(x,y) es verdadero si $\Psi_P^{(1)}(x)$ está definido (osea si el programa termina y devuelve un resultado) y falso si $\Psi_P^{(1)}(x)$ está indefinido.

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

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

Teorema 3.6. *HALT(x,y) no es un predicado computable.*

DEMOSTRACIÓN

Supongamos que HALT es computable, entonces podríamos construir el siguiente programa P:

$$[A] \quad \text{IF HALT}(X,X) \text{ GOTO } A$$

Es claro, que el cómputo de P es:

$$\Psi_P^{(1)}(x) = \begin{cases} \uparrow & \text{si HALT}(x, x) \\ 0 & \text{si } \neg \text{ HALT}(x, x) \end{cases}$$

Supongamos, ahora, que $\#(P) = e$. Entonces, para algún x, usando la definición de HALT nos queda que:

$$\text{HALT}(x, e) \iff \Psi_P^{(1)}(x) \downarrow \iff \Psi_P^{(1)}(x) = 0 \iff \neg\text{HALT}(x, x)$$

Si hacemos que $x = e$, entonces tenemos que $\text{HALT}(e, e) \iff \neg\text{HALT}(e, e)$. Pero esto es una contradicción. □

Con esto podemos concluir que no existe un algoritmo que nos permita computar $\text{HALT}(x, y)$.

Tesis de Church: Todos las funciones computables sobre naturales, se pueden programar en $\mathcal{S}$.

Diagonalización Dado un conjunto de funciones, la idea es generar una función que debería pertenecer al conjunto pero es distintas a todas las funciones de ese conjunto y, por lo tanto, llegar a una contradicción.

En el teorema de cantor (Teorema 3.5), usamos este método para concluir que había mas funciónes $\mathbb{N} \to \mathbb{N}$ que números naturales. Para esto, dimos un orden a la clase de funciones de este tipo y ordenamos sus valores de la siguiente forma:

$$\begin{array}{cccc} f_0(0) & f_0(1) & f_0(0) & \cdots \\ f_1(0) & f_1(1) & f_1(0) & \cdots \\ f_2(0) & f_2(1) & f_2(0) & \cdots \end{array}$$

Cuando definimos la nueva función que debería pertenecer a la enumeración propuesta, lo hacemos de tal manera que use los valores de la diagonal de esta matriz y sea distinto a todos ellos ($g(x) = f_x(x) + 1$).

Como $g(x)$ usa todas las funciones enumeradas y es distintas a todas ellas, se llega a un absurdo.

3.6. Universalidad

Definimos la función $\Phi^{(n)}(x_1, \ldots, x_n, e)$ como la salida del programa e con entrada $x_1, \ldots, x_n$, es decir $\Phi^{(n)}(x_1, \ldots, x_n, e) = \Psi_P^{(n)}(x_1, \ldots, x_n)$ con P tal que $\#(P) = e$.

Teorema 3.7. *Para cada $n > 0$ la función $\Phi^{(n)}$ es parcial computable.*

DEMOSTRACIÓN

Para demostrar esto vamos a construir el programa U_n que computa $\Phi^{(n)}$.

	$Z \leftarrow X_{n+1} + 1$	Z es la lista de instucciones del programa.
	$S \leftarrow \prod_{i=1}^{n} (p_{2i})^{x_i}$	S es la lista $[0, X_1, 0, X_2, 0, \ldots]$ que codifica su estado inicial (según el orden de variables que habíamos definido en la sección 3.3).
	$K \leftarrow 1$	K es la próxima instrucción a ejecutar.
$[C]$	IF $K = \lvert Z\rvert + 1 \vee K = 0$ GOTO F	El IF se fija si todavía hay instrucciones para ejecutar. Si no las hay, salta a la instrucción con etiqueta F. Si hay, se sigue en la siguiente instrucción.
	$U \leftarrow r(Z[K])$	U guarda el tipo y la variable de la K-ésima instrucción.
	$P \leftarrow p_{r(U)+1}$	P es el primo asociado a la variable de esta instrucción.
	IF $l(U) = 0$ GOTO N	Si el tipo $l(U)$ de la instrucción es $V \leftarrow V$, entonces vamos a N. Si es una suma vamos a S.
	IF $l(U) = 1$ GOTO S	
	IF $\neg(P\vert S)$ GOTO N	La condición $\neg(P\vert S)$ comprueba si la variable mencionada por U está en cero. Si lo está, vamos a N, sino nos fijamos que la instrucción sea, efectivamente, una resta. Si lo es, vamos a R, sino seguimos en la próxima instrucción.
	IF $l(U) = 2$ GOTO R	
	$K \leftarrow \min_{i \le \lvert Z\rvert} (l(Z[i]) + 2 = l(U))$	
	GOTO C	
$[R]$	$S \leftarrow S$ div P	
	GOTO N	
$[S]$	$S \leftarrow S \cdot P$	
	GOTO N	
$[N]$	$K \leftarrow K + 1$	
	GOTO C	
$[F]$	$Y \leftarrow S[1]$	

Si llegamos a la línea del mínimo, quiere decir que la instrucción era un salto condicional y, además, la variable era distinta de cero por lo quu hay que realizar el salto condicional. Aquí se computa la posición de la instrucción que se debe ejecutar a continuación. Y luego se vuelve a comenzar el ciclo.

En las etiqueta R y S se produce la resta y suma, respectivamente, de la variable correspondiente y luego se salta a N.
En N se incrementa K y se vuelve a comenzar el ciclo.
Finalmente, en F se asigna a Y el valor $S[1]$. □

A veces notaremos

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

. Y omitiremos el supra-índice cuando $n = 1$

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

3.6.1. Step counter

Definimos el predicado STP de la siguiente manera:

$\text{STP}^{(n)}(x_1, \ldots, x_n, e, t) \iff$ el programa e con entrada termina en t o menos pasos con la entrada $x_1, \ldots, x_n \iff$ hay un cómputo del programa e de longitud $\leq t + 1$, comenzando con la entrada $x_1, \ldots, x_n$

Teorema 3.8. *Para cada $n > 0$, el predicado $STP^{(n)}(x_1, \ldots, x_n, e, t)$ es primitivo recursivo.*

3.6.2. Snap

La función SNAP no devuelve la configuración instantánea del programa e con entrada $x_1, \ldots, x_n$ en el paso t.

$\text{SNAP}^{(n)}(x_1, \ldots, x_n, e, t) = \langle$ número de instrucción, lista representando el estado $\rangle$

Teorema 3.9. *Para cada $n > 0$, el predicado $SNAP^{(n)}(x_1, \ldots, x_n, e, t)$ es primitiva recursiva.*

3.6.3. Teorema de la forma normal

Teorema 3.10. *Sea $f : \mathbb{N}^N \to \mathbb{N}$ una función parcial computable. Entonces existe un predicado primitivo recursivo $R : \mathbb{N}^{n+1} \to \mathbb{N}$ tal que $f(x_1, \ldots, x_n) = l(\text{mín}_z\, R(x_1, \ldots, x_n, z))$.*

DEMOSTRACIÓN

Sea e el número de programa que computa $f(x_1, \ldots, x_n)$. Vamos a probar que es verdadero usando el predicado:

$$R(x_1, \ldots, x_n, z) = \text{STP}^{(n)}(x_1, \ldots, x_n, e, r(z)) \wedge (l(z) = r(\text{SNAP}^{(n)}(x_1, \ldots, x_n, e, r(z)))[1]$$

Primero, supongamos que $f(x_1, \ldots, x_n) \downarrow$. Entonces $\Phi_e^{(n)}(x_1, \ldots, x_n)$ termina en una cantidad $r(z)$ fínita de pasos. y $\text{STP}^{(n)}(x_1, \ldots, x_n, e, r(z))$ es verdadero. Además, $r(\text{SNAP}^{(n)}(x_1, \ldots, x_n, e, r(z)))$ representa el estado final del programa por lo que tiene almacenado el valor final de la variable Y en su primer posición.

De esta forma, caracterizamos z como la tupla que tiene, a la izquierda, el valor $f(x_1, \ldots, x_n)$ y, a la derecha, la cantidad de pasos que hizo el programa e para computar f con esa entrada. Por otro lado, si $f(x_1, \ldots, x_n) \uparrow$, entonces $\text{STP}^{(n)}(x_1, \ldots, x_n, e, r(z))$ nunca es verdadero por lo que la función mín se indefine (ya que nunca encuentra un $r(z)$ válido). □

A partir del teorema anterior, se derivan los siguientes:

Teorema 3.11. *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 3.12. *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** (es decir, siempre existe al menos un valor t que hace verdadera la propiedad que se busca minimizar).*

3.6.4. Teorema del parámetro

Teorema 3.13. *Para cada $n, m > 0$, hay una función primitiva recursiva inyectiva $S_n^m : \mathbb{N}^{n+1} \to \mathbb{N}$ tal que:*

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

Supongamos que dado un número de un programa con número y fijamos las variables $u_1, \ldots, u_n$. El teorema del parámetro nos dice que existe un programa con número $S_m^n(u_1, \ldots, u_n, y)(x_1, \ldots, x_m)$ que computa la función que toma como parámetros las m variables restantes. Y, además, ese número se puede obtener de manera computable.

DEMOSTRACIÓN

Se hace por inducción, en la cantidad de parámetros fijados.

Caso $n = 1$: Necesitamos mostrar que $S_m^1(u, y)$ es una función primitiva recursiva tal que $\Phi^{(m+1)}(x_1, \ldots, x_m, u, y) = \Phi^{(m)}(x_1, \ldots, x_m, S_m^1(u, y))$
Sea P el programa tal que $\#(P) = y$, entonces el programa Q tal que $\#(Q) = S_m^1(u, y)$ puede ser el program que primero asigna a X_{m+1} el valor u y luego ejecuta P.
En el ápendice B.2 se ve como asignar un valor cualquiera a una variable ejecuntando la instrucción $X_{m+1} \leftarrow X_{m+1} + 1$ la cantidad necesaria de veces.
El número de esta instrucción es: $\langle 0, \langle 1, 2m + 1\rangle\rangle = 16m + 10$. Por lo que hay que agregar u este número a la lista de instrucciones P, entonces el número de Q se podría computar de la siguiente forma:

$$S_m^1(u, y) = \left[\left(\prod_{i=1}^{u} p_i\right)^{16m+10} \prod_{j=1}^{|y+1|} p_{u+j}^{(y+1)_j}\right] \dot{-} 1$$

Lo que hace esta fución asignar a los primeros u primos, la instrucción mencionada y mover a $p_{u+1}, p_{u+2}, \ldots$ las instrucciones de P. Además, S_m^1 es primitiva recursiva (es nuestra codificación de listas).

Caso inductivo (n = k + 1): Asumimos que el resultado vale para $n = k$. Primero aplicamos el mismo resultado que para el caso base y obtenemos que:

$$\Phi_y^{(m+k+1)}(x_1, \ldots, x_m, u_1, \ldots, u_{k+1}) = \Phi^{(m+k)}_{S^1_{m+k}(u_{k+1},y)}(x_1, \ldots, x_m, u_1, \ldots, u_k)$$

Nuestra hipotesis inductiva es que para cualquier programa existe una función primitiva recursiva S_m^k que nos permite fijar los últimos k parámetros de la entrada. En particular, existe para el programa $S^1_{m+k}(u_{k+1}, y)$, entonces:

$$\Phi^{(m+k)}_{S^1_{m+k}(u_{k+1},y)}(x_1, \ldots, x_m, u_1, \ldots, u_k) = \Phi^{(m+k)}_{S^k_m(u_1,\ldots,u_k,S^1_{m+k}(u_{k+1},y))}(x_1, \ldots, x_m)$$

Luego podemos tomar la función primitiva recursiva

$$S_m^{k+1}(u_1, \ldots, u_{k+1}, y) = S_m^k(u_1, \ldots, u_k, S^1_{m+k}(u_{k+1}, y))$$

□

3.6.5. Teorema de la recursión

En la demostración del Halting Problem (Teorema 3.6) construimos un programa P que, cuando se ejecuta con su mismo número de programa, evidencia una contradicción. El fénomeno de un programa actuando sobre su propia descripción se conoce como *auto-referencia*. El teorema de la recursión nos presenta una técnica para obtener programas *auto-referentes*.

Teorema 3.14. *Si $g : \mathbb{N}^{n+1} \to \mathbb{N}$ es parcial computable, existe un e tal que $\Phi_e^{(n)} = g(e, x_1, \ldots, x_n)$*

DEMOSTRACIÓN

Sea S_n^1 la función del Teorema del parámetro:

$$\Phi_y^{(n+1)}(x_1, \ldots, x_n, u) = \phi^{(n)}_{S^1_n(u,y)}(x_1, \ldots, x_n)$$

Consideremos la función parcial computable

$$h(x_1, \ldots, x_n, v) = g(S_n^1(v, v), x_1, \ldots, x_n)$$

y sea z el número de un programa que computa h, entonces or el teorema del parámetro tenemos que $\Phi_z^{(n+1)}(x_1, \ldots, x_n, v) = \Phi^{(n)}_{S^1_n(v,z)}(x_1, \ldots, x_n)$.
Si fijamos $v = z$, entonces

$$\Phi^{(n)}_{S^1_n(v,z)}(x_1,\ldots,x_n) = \Phi^{(n)}_{S^1_n(z,z)}(x_1,\ldots,x_n) = g(S^1_n(z,z),x_1,\ldots,x_n)$$

Finalmente, podemos remplazar $e = S^1_n(z,z)$ en la formula anterior:

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

□

Corolario 3.2. *Si $g : \mathbb{N}^{n+1} \to \mathbb{N}$ es parcial computable, existen **infinitos** e tal que*

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

DEMOSTRACIÓN

Si, dada una función (parcialmente) computable, hay infinitos programas que la pueden computar. En particular, vale para la función h de la demostración anterior. Por ejemplo, como z es el número de un programa que computa h y le agregamos la instrucciónes $Y \leftarrow Y+1$ y $Y \leftarrow Y-1$ al principio, entonces obtenemos un programa equivalente con distinto número que también la computa. Podemos agregar este par de instrucciones la cantidad de veces que queramos y, cada vez, obtendremos un número diferente. □

3.6.6. Teorema del punto fijo

Un **quine** es un programa que cuando se ejecuta, devuelve como salida el mismo programa. Osea si P es un quine tal que $\#(P) = e$, entonces $\Phi_e(x) = e$.

Proposición 3.3. *Hay infinitos e tal que $\Phi_e(x) = e$*

DEMOSTRACIÓN

La demostración sale directamente de aplicar el teorema de la recursión sobre $g : \mathbb{N}^2 \to \mathbb{N}$, $g(z,x) = z$. Por el teorema de la recursión sabemos que existen infinitos programas con número e tal que $\Phi_e(x) = g(e,x) = e$. □

Proposición 3.4. *Sea h una función parcial computable, hay infinitos e tal que $\Phi_e(x) = h(e)$*

DEMOSTRACIÓN

Si consideramos $g : \mathbb{N}^2 \to \mathbb{N}$, $g(z,x) = h(z)$. Por el teorema de la recursión sabemos que existen infinitos programas con número e tal que $\Phi_e(x) = g(e,x) = h(e)$. □

Teorema 3.15. *Si $f : \mathbb{N} \to \mathbb{N}$ es computable, existe un e tal que $\Phi_{f(e)} = \Phi(e)$*

DEMOSTRACIÓN

Considerar la función $g : \mathbb{N}^2 \to \mathbb{N}$, $g(z, x) = \Phi_{f(z)}(x)$
Aplicando el Teorema de la Recursión, existe un e tal que para todo x:

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

□

4. Conjuntos computables enumerables

Función caractersitica: Dado un conjunto A, su función característica es un predicado que nos indica si un elemento pertenece o no al mismo, osea es de la forma:

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

Conjunto computable: Un conjunto A es computable si su función característica lo es.

Teorema 4.1. *Sean A, B conjuntos de una clase PRC $\mathcal{C}$. Entonces $A \cup B$, $A \cap B$ y $\bar{A}$ están en $\mathcal{C}$*

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

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

Osea A es el conjunto de valores que hacen que g termine. Esto quiere decir que podremos decidir algoritmicamente (corriendo el programa que computa g) si un elemento pertenece a A (si g termina entonces pertenece). Sin embargo, como g no está definida para los elementos que no pertenecen, si le pasamos uno de estos valores, el programa nunca terminará. Los algoritmos que computan g se llaman algoritmos de **semi-decisión**.

Conjunto co-c.e.: A es co-c.e. si $\bar{A}$ es c.e.

Teorema 4.2. *Si A es computable entonces A es c.e.*

DEMOSTRACIÓN

Como A es computable, su función carácterisitica f es computable. Sea P_A el programa que computa f. Y sea Q el siguiente programa:

$$[C] \quad \text{IF } P_A(X) = 0 \text{ GOTO } C$$

Entonces, el computo de Q es:

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

y por lo tanto, podemos escribir $A = \{x \ : \Psi_Q(x) \downarrow\}$ □

Teorema 4.3. *Si A y B son c.e. entonces $A \cup B$ y $A \cap B$ tambien son c.e.*

DEMOSTRACIÓN

Como A y B son c.e, existen dos funciones f y g tal que:

$$f(x) \downarrow \text{ si y solo si } x \in A \quad \text{y} \quad g(x) \downarrow \text{ si y solo si } x \in B$$

Sean P y Q los programas de semi-decisión que computan f y g y números p y q, respectivamente, vamos a armar programas que computen $A \cap B$ y $A \cup B$:

$\boldsymbol{A \cap B}$: Definimos el programa R:

$$Y \leftarrow \Phi_p(x)$$
$$Y \leftarrow \Phi_q(x)$$

R ejecuta los programas P y Q para el valor x, si alguno se indefine entonces R se indefine. Tenemos que $\Psi_R(x) \downarrow$ si y solo si $\Phi_P(x) \downarrow$ y $\Phi_Q(x) \downarrow$. Nos queda que $A \cap B = \{x \ : \ \Psi_R(x) \downarrow\}$, entonces $A \cap B$ es c.e. .

$\boldsymbol{A \cup B}$: Sea el programa R':

$$\begin{aligned} [C] \quad & \text{IF STP}^{(1)}(X, p, T) = 1 \text{ GOTO } E \\ & \text{IF STP}^{(1)}(X, q, T) = 1 \text{ GOTO } E \\ & T \leftarrow T + 1 \\ & \text{GOTO } C \end{aligned}$$

R' irá incrementando T hasta que alguno de los dos programas termine. Notemos que es necesario usar STP e ir ejecutando ambos programas de a poco. Si x no pertenece a ningún conjunto, ambos programas se indefinen y R' nunca encontrará un T que lo haga parar. Si x pertenece a uno solo de los conjuntos, digamos a A, entonces R' incrementará T hasta que P termine y terminará, sin importar que Q no haya terminado en T pasos. Luego el cómputo de R' es tal que $\Psi_{R'}(x) \downarrow$ si y solo si $\Phi_Q(x) \downarrow$ ó $\Phi_P(x) \downarrow$ y podemos usarla para enumerar los elementos de $A \cup B$. □

Teorema 4.4. *A es computable si y solo si A y $\bar{A}$ son c.e.*

DEMOSTRACIÓN

$\Rightarrow$) Por el teorema 4.2, A es c.e. Además, como A es computable sabemos que $\bar{A}$ también lo és (si f es la función carácterística de A, entonces $g(x) = 1 - f(x)$ es la función característica de $\bar{A}$ y g es computable). Luego $\bar{A}$ también es c.e.

$\Leftarrow$) Supongamos que A y $\bar{A}$ son c.e. Entonces existen dos funciones f y g tal que:

$$A = \{x \ : \ f(x) \downarrow\} \quad \text{y} \quad \bar{A} = \{x \ : \ g(x) \downarrow\}$$

Sean P y Q los programas de semi-decisión que computan f y g y números p y q, respectivamente. Vamos a definir un programa R que nos permita computar la función caracterísita de A:

$$\begin{aligned} [C] \quad & \text{IF STP}^{(n)}(X, p, T) = 1 \text{ GOTO } F \\ & \text{IF STP}^{(n)}(X, q, T) = 1 \text{ GOTO } E \\ & T \leftarrow T + 1 \\ & \text{GOTO } C \\ [F] \quad & Y \leftarrow 1 \end{aligned}$$

R está definido para todo x ya que debe pertenecer si o si a alguno de los dos conjuntos.

Si $x \in A$, entonces P termina y se ejecuta la instrucción con etiqueta F. Si $x \in \bar{A}$ entonces Q termina y termina el programa dejando Y en cero. Luego, $\Psi_R(x) = 1$ si $x \in A$ y $\Psi_R(x) = 0$ si no, por lo que Ψ_R nos permite computar el conjunto A. □

4.1. Teorema de la enumeración

Notación: Notamos W_n al conjunto que contiene todos los elementos del dominio del programa P tal que $\#(P) = n$. Ose $W_n = \{x \ : \ \Phi_n(x) \downarrow\}$

Teorema 4.5. *Un conjunto A es c.e. si y solo si existe un n tal que $A = W_n$*

Definición: $K = \{n \ : \ n \in W_n\}$ osea son todos los números de programas auto-referentes que están definidos para si mismos.

$$n \in K \iff n \in W_n \iff \Phi_n(n) \downarrow \iff \text{HALT}(n, n)$$

Teorema 4.6. *K es c.e pero no computable.*

DEMOSTRACIÓN

Por el teorema 3.7, tenemos que ϕ_n es parcial computable.Veamos que $\bar{K}$ no es c.e.
Supongamos que K es computable, entonces $\bar{K}$ tambien lo es. Por el teorema de la numeración (4.5) existe un número de programa e tal que $\bar{K} = W_e$. Vamos a analizar los siguientes dos casos:

- Si $e \in \bar{K} \Rightarrow e \in W_e \Rightarrow e \in K$. Lo que es absurdo ya que solo puede pertenecer a alguno de los dos.
- Si $e \notin \bar{K} \Rightarrow e \notin W_e \Rightarrow e \notin K$. Tambien es absurdo, pues e debe pertenecer a alguno de los dos.

Luego, $\bar{K}$ no es c.e. y, por lo tanto, K no es computable.

Teorema 4.7. *Si A es un c.e., existe un predicado p.r. $R : \mathbb{N}^2 \to \mathbb{N}$ tal que $A = \{x \ : \ (\exists t)\ R(x, t)\}$*

DEMOSTRACIÓN

Como A es c.e, existe un número de programa e tal que 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, es decir:

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

□

Teorema 4.8. *Sea $A \neq \emptyset$ es c.e., existe una función p.r. $f : \mathbb{N} \to \mathbb{N}$ tal que $A = \{f(0), f(1), f(2), \dots\}$, osea que enumera los elementos de A.*

DEMOSTRACIÓN

Por el teorema anterior (4.7), $A = \{x \ : \ (\exists t)R(x,t)\}$ donde R es un predicado primitivo recursivo. Sea $a \in A$, vamos a definir f como:

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

Entonces f es primitiva recursiva (por R lo es). Tenemos que probar dos cosas:

- Si $x \in A$, entonces está en la imagen de f.

$$x \in A \Rightarrow (\exists t)\ R(x,t) \Rightarrow f(\langle x,t\rangle) = x$$

- f solo devuelve elementos de A.

  Sea x tal que $f(u) = x$ para algún u. Entonces $x = a$ (y $a \in A$) o bien u es de la forma $u = \langle x,t\rangle$ y vale $R(x,t)$ por lo que $x \in A$.

Luego, f es una función que nos permite enumerar A *(repitiendo elementos y sin ningún orden en particular).* □

Teorema 4.9. *Si $f : \mathbb{N} \to \mathbb{N}$ es parcial computable, $A = \{f(x) \ : \ f(x)\downarrow\}$ es c.e.*

DEMOSTRACIÓN

Sea p el número del programa que computa f, debemos definir un programa Q que termine si le pasamos un elemento de A y se indefina sinó:

$[A]$	IF $\text{STP}^{(1)}(Z,p,T) = 0$ GOTO B	Dado un valor X, Q busca un valor Z tal que $f(Z) = X$. Para esto va recorriendo cada valor posible y testea el programa durante T pasos. Si el programa terminó en esa cantdidad de pasos y si el resultado es el que queriamos entonces termina. Si no, sigue con el proximo valor de Z con el que hay que probar.
	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.	

Notemos que Z se reinicia cada vez que alcanza T, esto es una forma inteligente de probar todos los pares $\langle Z,T\rangle$ y asegurarnos de que el programa va a terminar para las entradas para los que debe hacerlos.
Tanto el dominio de f, como los valores que puede tomar el tiempo de ejecución son infinitos. Si intentasemos reccorrer todos los valores del dominio antes de aumentar la cantidad de pasos que debe hacer P entonces el programa no terminaría para ningún X.

Si quisieramos recorrer primero todos los tiempos posibles, antes de cambiar Z, el programa aumentaría Z hasta encontrar un Z_0 para el que P se indefine y nunca terminaría para cualquier X tal que $f(Z) = X$ y $Z > 0$.
Luego, si el programa termina en T o menos pasos con entrada Z y $\Phi_P(Z) = X$, termina y $\Psi_Q(X)\downarrow$ y podemos escribir $A = \{x \ : \ \Psi_Q(x)\downarrow\}$. Luego A es c.e. □

Teorema 4.10. *Si $A \neq \emptyset$, son equivalentes:*

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

DEMOSTRACIÓN

(1 ⇒ 2): Por Teorema 4.8
(2 ⇒ 3 ⇒ 4): Trivial
(4 ⇒ 1): Por Teorema 4.9 □

4.2. Teorema de Rice

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

Teorema 4.11. *Si A es un conjunto de índices tal que $\emptyset \neq A \neq \mathbb{N}$ entonces A no es computable.*

DEMOSTRACIÓN

Supongamos una clase $\mathcal{C}$ de funciones $\mathbb{N} \to \mathbb{N}$ parciales computables tal $A = \{x \ : \ \Phi_x \in \mathcal{C}\}$ es computable. Sean f y g funciones parciales computables tal que $f \in \mathcal{C}$ y $g \notin \mathcal{C}$. Definimos la función parcial computable $h : \mathbb{N}^2 \to \mathbb{N}$ de la siguiente forma:

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

Por el teorema de la recursión (3.14), existe un número de programa e tal que $\Phi_e(x) = h(e,x)$. Veamos los siguientes casos:

- Si $e \in A \Rightarrow \Phi_e = g \Rightarrow \Phi_e \notin \mathcal{C}$ (pues $g \notin \mathcal{C}$) $\Rightarrow e \notin A$. Absurdo.
- Si $e \notin A \Rightarrow \Phi_e = f \Rightarrow \Phi_e \in \mathcal{C} \Rightarrow e \in A$. Absurdo.

Luego, A no es computable. □

Este teorema nos da una fuente de conjuntos no computables:

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

Además, $Tot = \{x \ : \ \Phi_x$ no es total $\}$ y $\overline{Tot}$ no son computables, ni c.e.:

DEMOSTRACIÓN

***Tot* no es c.e:** Supongamos que lo es. Entonces existe f computable tal que $Tot = \{f(0), f(1), \dots\}$ (Teorema 4.8). Osea f enumera los elementos de Tot
Sea e el número de un programa tal que $\Phi_e(x) = \Phi_{f(x)}(x) + 1$. Como todos los elementos de la imagen de f computan funciones totales, Φ_e es total ($\Phi_e \in Tot$). Osea que existe u tal que $f(u) = e$ (pues f enumeraba los elementos de Tot).
Luego, $\Phi_{f(u)}(x) = \Phi_e(x) = \Phi_{f(x)}(x) + 1$. Esto genera un absurdo si le pasamos $x = u$ como parámetro. Luego Tot no es c.e.
***$\overline{Tot}$* no es c.e:** Supongamos que lo es, por el teorema 4.5 existe un programa e tal que $\overline{Tot} =$ dom Φ_d. Sea P el siguiente programa:

$$\begin{array}{ll} [C] & \text{IF STP}^{(1)}(X, d, T) = 1 \text{ GOTO } E \\ & T \leftarrow T + 1 \\ & \text{GOTO } C \end{array}$$

Entonces el computo de P:

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

Por el teorema de la recursión (3.14), existe un d tal que

$$\Phi_d(y) = g(d, y) = \left\{ \begin{array}{ll} \uparrow & \text{si } \phi_d \text{ es total} \\ 0 & \text{si no} \end{array} \right.$$

.
La contradicción es clara, pues estamos pidiendo que ϕ_d se indefina si es total y que sea total si no lo es (lo que es absurdo).
Luego $\overline{Tot}$ no es c.e.

Parte II
Lógica

5. Lógica Proposicional (Lenguaje P)

Vamos a definir un lenguaje formal P que nos permitirá evitar las imprecisiones y ambigüedades de un lenguaje natural. Para esto, vamos a definir:

- Un conjunto de símbolos con los que vamos a poder escribir las oraciones del lenguaje: $\neg,\ \rightarrow,\ (,\ ),\ p,'$

  Vamos a llamar símbolos proposicionales a los $p',\ p'',\ p''', \ldots$. Estos símbolos son lo que tienen una traducción directa al lenguaje natural y los llamaremos **variables propocision**.

- Un conjunto de reglas que nos permitirán escribir sentencias *gramaticalmente correctas* (que llamaremos **fórmulas bien formadas**):

  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.

Muchas veces para mejorar la legibilidad de las fórmulas, escribiremos:

- q por p', r por p'', s por p''', etc.
- $(\varphi \vee \psi)$ en lugar de $(\neg\varphi \rightarrow \psi)$.
- $(\varphi \wedge \psi)$ en lugar de $\neg(\varphi \rightarrow \neg\psi)$
- φ en lugar de (φ)

PROP: Es el conjunto de todos los símbolos proposicionales.

FORM: Es el conjunto de todas las formulas.

Interpretación (ó valuación): (o **valuación**) Es una función v : PROP $\rightarrow \{0, 1\}$ que nos indica el valor de verdad de una variable proposicional. La usaremos para decidir si una fórmula es verdadera o no:

Sea $\varphi \in$ FORM (es una fórmula) y v una evaluación, definimos la función ($\vDash$) de la siguiente manera:

1. Sea $p \in$ PROP, $v \vDash p$ si y solo si $v(p) = 1$

2. $v \vDash \neg\varphi$ si y solo si $v \nvDash \varphi$

3. $v \nvDash (\varphi \to \psi)$ si $v \nvDash \varphi$ ó $v \vDash \psi$

Y lo leeremos de la siguiente manera:

- Si $v \vDash \varphi$ leeremos "φ es veradera para v".
- $v \nvDash \varphi$ leeremos "φ es falsa para v".

5.1. Tautologías

Tautología: Es una fórmula φ que es verdad para tod a evaluación v. Escribimos $\vDash \phi$ cuando sucede esto.

Proposición 5.1. *Sea $\varphi \in FORM$ y sean v y w dos evaluaciones tal que $v(p) = w(p)$ para toda variable propocisional que aparece en φ. Entonces $v \vDash \varphi$ si y solo si $s \vDash \varphi$*

Notación: Vamos a notar $V = \langle v(p_1), ..., v(p_n)\rangle$, es decir a la n-upla que contiene en la i-ésima posición el valor en la evaluación v de la i-ésima variable propocisional.

Método de decisión: Es un método que nos permite saber si una fórmula φ es tautología o no.

Supongamos que φ tiene variables proposicionales $p_1, \ldots, p_n$. Sea $\mathcal{P}(\{p_1, \ldots, p_n\}) = \{V_1, \ldots, V_{2^n}\}$ el conjunto de partes de las variables proposicionales. Vamos a definir, para $i \in \{1, \ldots, 2^n\}$, las siguientes valuaciones:

$$v_i(p) = \begin{cases} 1 & \text{si } p \in V_i \\ 0 & \text{si no} \end{cases}$$

Osea definimos 2^n evaluaciones que en conjunto cubren todas las posibles combinaciones de valores para las propocisiones. Por lo tanto, podemos asegurar que φ es una tautología si y solo si $v_i \vDash \varphi$ para todo $i \in \{1, \ldots, 2^n\}$.

5.2. Consecuencia semántica y conjuntos satisfacibles

Conjunto de fórmulas satisfacible: Dado un conjunto de fórmulas $\Gamma \subseteq$ FORMS, decimos que es satisfacible si existe una evaluación v tal que $v \vDash \varphi$ para todo $\varphi \in \Gamma$ y notamos $v \vDash \Gamma$.

Consecuencia semántica: Sea $\Gamma \subseteq$ FORM y $\varphi \in$ FORM entonces φ es consecuencia semántica de Γ ($\Gamma \vDash \varphi$) si para toda interpretación v vale que: $v \vDash \Gamma$ entonces $v \vDash \varphi$.

Propiedades: Sea $\varphi \in$ FORM y $\Gamma, \Delta \subseteq$ FORM entonces valen las siguientes afirmaciones:

- $\emptyset \vDash \varphi$ si y solo si $\vDash \varphi$ (φ es una tautología)
- $\{\varphi\} \vDash \varphi$
- Si $\vDash \varphi$ entonces $\Gamma \vDash \varphi$
- si $\Gamma \subseteq \Delta$ y $\Gamma \vDash \varphi$ entonces $\Delta \vDash \varphi$
- Si $\Gamma \vDash \varphi$ y $\Gamma \vDash (\varphi \to \psi)$ entonces $\Gamma \vDash \psi$

5.3. Mecanimos deductivo SP

Vamos a definir un conjunto de axiomas y reglas que nos permitirán inferir y demostrar el valor de verdad de una fórmula proposicional.

Axiomas: Sean $\varphi, \psi, \rho \in$ FORM:

SP1 $\varphi \to (\psi \to \varphi)$

SP2 $(\varphi \to (\psi \to \rho)) \to ((\varphi \to \psi) \to (\varphi \to \rho))$

SP3 $(\neg\varphi \to \neg\psi) \to (\psi \to \varphi)$

Regla de inferencia:

MP Sean $\varphi, \psi \in$ FORM. ψ es una consecuencia inmediata de $\phi \to \psi$ y ϕ. Osea que si valen $\phi \to \psi$ y ϕ, entonces valoe ψ.

Demostración: Una demostración de φ en SP es una cadena de fórmulas del leguaje proposicional P finita y no vacía $(\varphi_1 \dots \varphi_n)$ tal que:

- $\varphi_n = \varphi$ (termina con la fórmula que queremos demostrar).
- φ_i es un axioma ó
- φ_i es una consecuencia inmediata de φ_k y φ_l con $k, l < i$.

Teorema: Es una fórmula para la cual existe una demostración. Usaremos la notación $\vdash \varphi$ para indicar φ es un teorema.

5.3.1. Consecuencia sintáctica

Sea $\Gamma \subseteq$ FORM y $\varphi \in$ FORM, entonces φ es una **consecuencia sintáctica** de Γ si existe una cadena de fórmulas del leguaje proposicional P finita y no vacía $(\varphi_1 \dots \varphi_n)$ tal que:

- $\varphi_n = \varphi$ (termina con la fórmula que queremos demostrar).
- φ_i es una axioma ó
- $\varphi_i \in \Gamma$ ó
- φ_i es una consecuencia inmediata de φ_k y φ_l con $k, l < i$.

En este caso, la secuencia $\varphi_1 \dots \varphi_n$ se llama **derivación** de φ a partir de Γ y Γ se llama **teoría**.

Además, cuando exista una derivación de ϕ a partir de Γ diremos quer φ es un **teorema de la teoría** Γ y notamos $\Gamma \vdash \varphi$.

5.3.2. Correctitud de SP

Teorema 5.1. *Si $\Gamma \vdash \varphi$ entonces $\Gamma \vDash \varphi$. Osea, si φ es un teorema de la teoría Γ, entonces φ es válido en toda interpretación de Γ.*

DEMOSTRACIÓN

Supongamos que $\Gamma \vdash \varphi$. Es decir, existe una cadena finita y no vacía $\varphi_1, \dots, \varphi_n$ de fórmulas de P tal que:

- $\varphi_n = \varphi$ (termina con la fórmula que queremos demostrar).
- φ_i es una axioma ó
- $\varphi_i \in \Gamma$ ó
- φ_i es una consecuencia inmediata de φ_k y φ_l con $k, l < i$.

Vamos a mostrar por inducción en n (la longitud de la demostración) que $\Gamma \vDash \varphi$.

Caso base ($n = 1$): Sea v una evaluación tal que $v \vDash \Gamma$, queremos ver que $v \vDash \varphi$. Y esto es verdad porque, al ser la demostración de longitud uno, solo tenemos dos posibilades: φ es un axioma de SP ó $\varphi \in \Gamma$.

Paso inductivo ($n \Rightarrow n + 1$): Sea v una evaluación tal que $v \vDash \Gamma$, suponemos que $v \vDash \varphi_n$ (satisface todas las fórmulas que tienen una demostración de longitud $\leq n$). Queremos ver que, dada una derivación de $n + 1$ fórmulas $\varphi_1, \dots, \varphi_n = \varphi$, $v \vDash \varphi$.
En este caso tenemos tres posibilidades:

- Si φ es un axioma de SP ó $\varphi \in \Gamma$ entonces es trivial.
- Si φ es consecuencia inmediata de φ_i y $\varphi_j = \varphi_i \to \varphi$ (con $i, j \leq n$), por HI tenemos que $v \vDash \varphi_i$ y $v \vDash \varphi_j$. Entonces necesariamente $v \vDash \varphi$.

□

5.4. Conjuntos y sistemas consistentes

Conjunto consistente: Un conjunto de fórmulas $\Gamma \subseteq$ FORM es cosistente si no existe $\varphi \in$ FORM tal que $\Gamma \vdash \varphi$ y $\Gamma \vdash \neg\varphi$.

Sistema consistente: Un sistema S es consistentes si no existe $\varphi \in$ FORM tal que $\vdash_S \varphi$ y $\vdash_S \neg\varphi$

Teorema 5.2. *El sitema SP es consistente.*

DEMOSTRACIÓN

Sea v cualquier valuación, por el teorema de correctitud (5.1), vale que:

$$\vdash \varphi \Rightarrow v \vDash \varphi \Rightarrow v \nvDash \neg\varphi \Rightarrow \nvdash \neg\varphi$$

Luego, no puede pasar que φ y $\neg\varphi$ sean teoremas. □

Propiedades: Sean $\varphi, \psi \in \text{FORM}$ y $\Gamma, \Delta \subseteq \text{FORM}$ entonces valen las siguientes afirmaciones:

- $\emptyset \vdash \varphi$ si y solo si $\vdash \varphi$ (φ es teorema)
- $\{\varphi\} \vdash \varphi$
- Si $\vdash \varphi$ entonces $\Gamma \vdash \varphi$
- si $\Gamma \subseteq \Delta$ y $\Gamma \vdash \varphi$ entonces $\Delta \vdash \varphi$
- Si $\Gamma \vdash \varphi$ y $\Gamma \vdash (\varphi \to \psi)$ entonces $\Gamma \vdash \psi$

5.4.1. Teorema de la deducción

Teorema 5.3. *Si* $\Gamma \cup \{\varphi\} \vdash \psi$ *entonces* $\Gamma \vdash \varphi \to \psi$

DEMOSTRACIÓN

Supongamos que $\varphi_1, \ldots, \varphi_n$ es una derivación de ψ a partir de $\Gamma \cup \{\varphi\}$, vamos a demostrar el teorema por inducción:

Caso base (n = 1): La derivación es una sola fórmula $\varphi_1 = \psi$, queremos ver que $\Gamma \vdash \varphi \to \psi$.

- Si ψ es una axioma de SP:

  1. ψ puede ser derivado pues es un axioma.
  2. $\psi \to (\varphi \to \psi)$ por $SP1$.
  3. Por MP con 1 y 2, $\varphi \to \psi$

  Luego, $\vdash \varphi \to \psi$ ($\varphi \to \psi$ es un teorema del sistema), es decir, que se puede derivar a partir de cualquier conjunto Γ de fórmulas.

- Si $\psi \in \Gamma$, entonces:

  1. ψ se puede derivar por pertenecer a Γ.
  2. $\psi \to (\varphi \to \psi)$ por $SP1$.
  3. Por MP con 1 y 2, $\varphi \to \psi$

  Luego, $\Gamma \vdash \varphi \to \psi$.

- $\varphi = \psi$ entonces tenemos que ver existe una derivación para $\psi \to \psi$:

  1. $\psi \to ((\psi \to \psi) \to \psi)$ por SP1.
  2. $(\psi \to ((\psi \to \psi) \to \psi)) \to ((\psi \to (\psi \to \psi)) \to (\psi \to \psi))$ por SP2.
  3. $(\psi \to (\psi \to \psi)) \to (\psi \to \psi)$ por MP entre 1 y 2.
  4. $\psi \to (\psi \to \psi)$ por SP1.
  5. $\psi \to \psi$ por MP entre 3 y 4.

  Luego $\vdash \psi \to \psi$, ($\psi \to \psi$ es un teorema) y, por lo tanto es derivable a partir de Γ.

Caso inductivo ($n \Rightarrow n+1$) : Suponemos que para toda derivación de alguna fórmula ψ' a partir de $\Gamma \cup \{\varphi\}$ de longitud $< n$ tenemos que $\Gamma \vdash \varphi \to \psi'$. Queremos ver que $\Gamma \vdash \varphi \to \psi$:

- Si ψ es una axioma de SP, $\psi \in \Gamma$ ó $\psi = \varphi$, hay que hacer lo mismo que en el caso base.
- Si ψ se infiere por MP de φ_i y φ_j con $i, j < n$ supongamos, sin perdida de generalidad, que $\varphi_j = \varphi_i \to \psi$.

  1. Como $\Gamma \cup \{\varphi\} \vdash \varphi_i$ y la derivación tiene longitud $< n$, por hipotesis inductiva sabemos que $\Gamma \vdash \varphi \to \varphi_i$.
  2. Idem para φ_j. $\Gamma \vdash \varphi \to \varphi_j$, osea que $\Gamma \vdash \varphi \to (\varphi_i \to \psi)$.
  3. Por SP2 tenemos que: $(\varphi \to (\varphi_i \to \psi)) \to ((\varphi \to \varphi_i) \to (\varphi \to \psi))$
  4. Por MP entre 3 y 2 obtenemos: $(\varphi \to \varphi_i) \to (\varphi \to \psi)$
  5. Y por MP entre 4 y 1 obtenemos: $\varphi \to \psi$

  Luego $\Gamma \vdash \varphi \to \psi$. □

5.4.2. Conjuntos consistentes

Proposición 5.2. *Sean $\Gamma \subseteq FORM$ y $\varphi \in FORM$:*

1. $\Gamma \cup \{\neg\varphi\}$ *es inconsistente si y solo si* $\Gamma \vdash \varphi$.
2. $\Gamma \cup \{\varphi\}$ *es inconsistente si y solo si* $\Gamma \vdash \neg\varphi$.

DEMOSTRACIÓN

(⇐) Si $\Gamma \vdash \varphi$, entonces $\Gamma \cup \{\neg\varphi\}$ es inconsistente trivialmente. Pues $\Gamma \cup \{\neg\varphi\} \vdash \varphi$ y $\Gamma \cup \{\neg\varphi\} \vdash \neg\varphi$.

(⇒) : Supongamos que Γ es inconsistente, entonces existe una fórmula ψ tal que $\Gamma \cup \{\neg\varphi\} \vdash \psi$ y $\Gamma \cup \{\neg\varphi\} \vdash \neg\psi$.

Por el teorema de la deducción (5.3) tenemos que $\Gamma \vdash \neg\varphi \to \psi$ y $\Gamma \vdash \neg\varphi \to \neg\psi$

NO ME SALIÓ $- - - >$ Se puede ver que $\vdash (\neg\varphi \to \psi) \to ((\neg\varphi \to \neg\psi) \to \varphi)$

Luego, por MP 2 veces tenemos que $\Gamma \vdash \varphi$. □

Teorema 5.4. *Si $\Gamma \subseteq FORM$ es satisfacible entonces Γ es consistente.*

DEMOSTRACIÓN

Sea v una valuación tal que $v \vDash \Gamma$, supongamos que Γ es inconsistente. Entonces existe una fórmula ψ tal que $\Gamma \vdash \psi$ y $\Gamma \vdash \neg\psi$.

Por correctitud de SP (teorema 5.1), $\Gamma \vDash \psi$ y $\Gamma \vDash \neg\psi$ entonces: $v \vDash \psi$ y $v \vDash \neg\psi$. Esto es absurdo, pues v estaría asignando dos valores de verdad a la misma fórmula y entonces no sería una función.

Conjunto maximal consistente: $\Gamma \subseteq$ FORM es maximal consistente (m.c.) en SP si:

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

Lema de Lindenbaum:

Lema 5.1. *Si $\Gamma \subseteq FORM$ es consistente, existe Γ' m.c. tal que $\Gamma \subseteq \Gamma'$.*

DEMOSTRACIÓN

Sea $\varphi_1, \varphi_2, \ldots$ una enumarición de todas la fórmulas, vamos a definir la siguiente sucesión:

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

Es claro que $\Gamma \subseteq \Gamma'$, veamos que Γ' es maximal consistente:

Γ' es consistente: Supongamos que no, entonces existe ψ tal que $\Gamma' \vdash \psi$ y $\Gamma' \vdash \neg\psi$, entonces existen dos derivaciones que usan las fórmulas $\{\lambda_1, \ldots, \lambda_k\} \subseteq \Gamma'$ para ambas fórmulas.

Esto quiere decir que hubo un paso j de la sucesión en la que se agregó la fórmula φ_j al Γ_{j-1} y $\Gamma_j \vdash \psi$ y $\Gamma_j \vdash \neg\psi$. Pero esto es absurdo ya que, por como definimos la sucessión de Γs, solo hubiesemos agregado la fórmula si se mantiene la consistencia.

Γ' es maximal: Supongamos que $\varphi \notin \Gamma'$. Entonces debe existir n tal que $\varphi_{n+1} = \varphi$ y $\varphi_{n+1} \notin \Gamma_{n+1}$ (porque $\Gamma_n \cup \{\varphi_{n+1}\}$ sería inconsistente). Luego, como $\Gamma_n \subseteq \Gamma'$, si queremos agregarle φ lo haremos inconsistente. □

Proposición 5.3. *Si Γ' es m.c. entonces paratoda $\varphi \in FORM$ pueden pasar dos cosas:*

- *o bien $\varphi \in \Gamma'$*
- *o bien $\neg\varphi \in \Gamma'$*

DEMOSTRACIÓN

Es claro que las dos (φ y $\neg\varphi$) no pueden estar al mismo en Γ' porque sería inconsistente. Vamos a probar que por lo menos hay una de ellas.
Supongamos que ninguna está. Como Γ es maximal entonces:

- $\Gamma' \cup \{\phi\}$ es inconsistente entonces $\Gamma' \vdash \neg\varphi$
- $\Gamma' \cup \{\neg\varphi\}$ es inconsistente entonces $\Gamma' \vdash \varphi$

Osea que $\Gamma' \vdash \neg\varphi$ y $\Gamma' \vdash \varphi$ por lo que es inconsistente. Esto es absurdo ya que Γ' es maximal consistente.

Teorema 5.5. *Si $\Gamma \subseteq FORM$ es consistente entonces Γ es satisfacible.*

DEMOSTRACIÓN

Sea Γ consistente, y Γ' un conjunto maximal consistente que contenga a Γ.
Sea v una evaluación tal que $v(p) = 1$ si y solo si $p \in \Gamma'$. Vamos a demostar por inducción en la **complejidad** de φ (cantidad de $\neg$ ó $\to$ que aparecen φ) la siguiente propiedad:

$$v \vDash \varphi \text{ si y solo si } \varphi \in \Gamma'$$

Si esto es verdad, entonces, como $\Gamma \subseteq \Gamma'$, $v \vDash \Gamma$ trivialmente.

Caso base: $\varphi = p$. Es trivial por definición de v.

Paso inductivo: Supongamos que $v \vDash \varphi$ sii $\varphi \in \Gamma'$ para toda φ de complejidad $< m$. Sea φ de complejidad m, tenemos dos casos:

- $\varphi = \neg\psi$

  $(\Rightarrow)$ $v \vDash \varphi \Rightarrow v \nvDash \psi \overset{HI}{\Rightarrow} \psi \notin \Gamma' \Rightarrow \neg\psi \in \Gamma' \Rightarrow \varphi \in \Gamma'$

  $(\Leftarrow)$ $\varphi \in \Gamma' \Rightarrow \psi \notin \Gamma' \overset{HI}{\Rightarrow} v \nvDash \psi \Rightarrow v \vDash \neg\psi \Rightarrow v \vDash \psi$

- $\varphi = \psi \to \rho$

  ($\Rightarrow$) $v \vDash \varphi \Rightarrow v \vDash (\psi \to \rho) \Rightarrow v \nvDash \psi$ o $v \vDash \rho$

  - Si $v \nvDash \psi \overset{HI}{\Rightarrow} \psi \notin \Gamma' \Rightarrow \Gamma' \vdash \neg\psi$
    Sabemos que $\vdash \neg\psi \to (\psi \to \rho)$ entonces por modus ponens $\Gamma' \vdash \psi \to \rho$, por lo que $\psi \to \rho \in \Gamma'$.
  - Si $v \vDash \rho \overset{HI}{\Rightarrow} \rho \in \Gamma' \Rightarrow \Gamma' \vdash \rho$.
    Por SP1 sabemos que $\vdash \rho \to (\psi \to \rho)$ entonces por MP tenemos que $\Gamma' \vdash \psi \to \rho$.
    Luego $\psi \to \rho \in \Gamma'$.

  ($\Leftarrow$) $\varphi \in \Gamma'$, supongamos que $v \nvDash \varphi \Rightarrow\Rightarrow v \vDash \psi$ y $v \nvDash \rho \overset{HI}{\Rightarrow} \psi \in \Gamma'$ y $\rho \notin \Gamma' \Rightarrow \psi \in \Gamma'$ y $\neg\rho \in \Gamma' \Rightarrow \Gamma' \vdash \psi$ y $\Gamma' \vdash \neg\rho$.

  Sabemos que $\vdash \psi \to (\neg\rho \to \neg(\psi \to \rho))$. Aplicando dos veces MP nos queda $\Gamma' \vdash \neg(\psi \to \rho)$, por loque $\neg\psi \to \rho) \in \Gamma'$, luego $\varphi = \psi \to \rho \notin \Gamma'$. Absurdo. □

Teorema de la completitud fuerte

Corolario 5.1. *Γ es consistente si y solo si Γ es satisfacible.*

Esto se deduce directamente de combinar los teoremas 5.4 y el lema 5.1.

Teorema 5.6. *Si $\Gamma \vDash \varphi$ entonces $\Gamma \vdash \varphi$*

DEMOSTRACIÓN

Supongamos que $\Gamma \vDash \varphi$, entonces $\Gamma \cup \{\neg\varphi\}$ es insatisfacible y, por lo tanto, inconsistente (Corolario 5.1).
Luego, por propocisión 5.2 tenemos que, como $\Gamma \cup \{\neg\varphi\}$ es inconsistente, vale que $\Gamma \vdash \varphi$. □

Corolario 5.2. *$\Gamma \vdash \varphi$ si y solo si $\Gamma \vDash \varphi$*

Corolario 5.3. *$\vdash \varphi$ si y solo si $\vDash \varphi$ (φ es un teorema de SP si y solo si es tautología)*

Teorema de compacidad

Teorema 5.7. *Sea $\Gamma \subseteq FORM$. Si todo subconjunto finito de Γ es satisfacible entonces Γ es satisfacible.*

DEMOSTRACIÓN

Lo demostramos por contraposicin. Supongamos que Γ es insatisfacible, tenemos que ver que es imposible que se cumpla el antecedente. Osea que Γ debe tener algún subcojunto fínito que es insatisfacible.

Como Γ es insatisfacible entonces es inconsistente. Osea que existe una fórmula ψ tal que $\Gamma \vdash \psi$ y $\Gamma \vdash \neg\psi$ en una cantidad finita de pasos. Esto quiere decir que existe un subconjunto finito $\Delta \subseteq \Gamma$ que contiene a las derivaciones de ambas fórmulas y, por lo tanto, $\Delta \vdash \psi$ y $\Delta \vdash \neg\psi$ (es inconsistente e insatifacible).

6. Lógica de primer orden

Hay muchos tipos de inferencias lógicas que no pueden ser justificados usando solo las bases de la lógica propocisional, por ejemplo:

- Todos los humanos son seres racionales.
- Algunos animales son humanos.
- Por lo tanto, algunos animales son seres racionales.

La correctitud de estas inferencias resta, no solo en el significado de los conectivos lógicos, sino en el significado de expresiones como "algunos", "todosz otras expresiones lingüísticas.

Para poder describir este tipo de sentencias agregamos, a los símbolos que ya venimos usando, el símbolo $\forall$ de **cuantificación universal**: $(\forall x)P(x)$ se leerá "La propiedad P es verdadera para todos los valores de x.

Osea que los símbolos lógicos de nuestros lenguajes van a ser: x, $'$, $\forall$, $\neg$, $\rightarrow$, (y).

Además, usaremos los siguientes conjuntos de símbolos:

Usaremos $x, x', x'', \ldots$ para nombrar **variables**.

- Un cojunto $\mathcal{C}$ de símbolos de constantes, es decir que representarán las constantes de nuestro lenguaje.
- Un cojunto $\mathcal{F}$ de símbolos funciones.
- Un cojunto $\mathcal{P}$ de símbolos de predicados.

Tanto $\mathcal{C}$ como $\mathcal{F}$ pueden ser vacíos, sin embargo, como nuestro objetivo es poder describir propiedades de nuestro sistema, no permitiremos que $\mathcal{P}$ lo sea.

Lenguaje de primer orden: Un lenguaje $\mathcal{L}$ de primer orden va a estar definido por $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$ o sea es un conjunto de símbolos de constantes, de funciones y de predicados.

Término: Dado un lenguaje $\mathcal{L}$, definimos sus de la siguiente manera:

- Toda variable es un término.
- Todo símbolo de constante de $\mathcal{L}$ es un término.
- Si f es un símbolo de función n-ádico de $\mathcal{L}$ y $t_1, \ldots, t_n$ son términos de $\mathcal{L}$, entonces $f(t_1, \ldots, t_n)$ es un término de $\mathcal{L}$.
- Nada más es un término de $\mathcal{L}$

Llamaremos TERM($\mathcal{L}$) al conjunto de términos del lenguaje $\mathcal{L}$.

Término cerrado: Es un término que no tiene variables.

Fórmulas: Dado un lenguaje $\mathcal{L}$, definimos sus fórmulas de la siguiente manera:

- Si P es un símbolo de predicado n-ádico de $\mathcal{L}$ y $t_1, \ldots, t_n$ son términos de $\mathcal{L}$, entonces $P(t_1, \ldots, t_n)$ es una **fórmula atómica** de $\mathcal{L}$
- Si φ es una fórmula de $\mathcal{L}$ entonces $\neg\varphi$ es una fórmula de $\mathcal{L}$.
- Si φ y ψ son fórmulas de $\mathcal{L}$ entonces $(\varphi \to \psi)$ es una fórmula de $\mathcal{L}$.
- Si φ es una fórmula de $\mathcal{L}$ y x un variable entonces $(\forall x)\varphi$ es una fórmula de $\mathcal{L}$.
- Nada más es una fórmula de $\mathcal{L}$.

Llamaremos FORM($\mathcal{L}$) al conjunto de fórmulas de $\mathcal{L}$.

Convenciones de notación: Usaremos:

- letras minúsculas para variables, símbolos constantes y símbolos de funciones.
- letras mayúsculas para predicados.
- $(\exists x)\varphi$ en lugar de $\neg(\forall x)\neg\varphi$
- $(\varphi \vee \psi)$ en lugar de $(\neg\varphi \to \psi)$.
- $(\varphi \wedge \psi)$ en lugar de $\neg(\varphi \to \neg\psi)$
- φ en lugar de (φ)

Variables libres y ligadas: Una aparición de una variable x es una fórmula está **ligada** si está dentro del alcance de un cuantificador. En caso contrario dicha aparición estarpá **libre**.

Si todas sus apariciones en una fórmula están libres, entonces la variable está libre. Si todas sus apariciones están ligadas, entonces está ligada.

Sentencia: Es una fórmula tal que todas sus variables son ligadas.

6.1. Interpretación de un lenguaje

Hasta ahora, definimos los símbolos de un lenguaje y como formar sus términos y fórmulas. Ahora, debemos asiganarle un significado a cada uno de esos símbolos.

$\mathcal{L}$-estructura (o interpretación de $\mathcal{L}$): Una $\mathcal{L}$-estructura (o interpretación) $\mathcal{A}$ de un lenguaje $\mathcal{L} = \mathbf{C} \cup \mathcal{F} \cup \mathcal{P}$ consiste en:

- Un conjunto no vacío A llamado el **dominio** o **universo** de la interpretación.
- Para cada símbolo de constante $c \in \mathcal{C}$, una asignación de un elemento fijo $c_{\mathcal{A}} \in A$

- Para cada símbolo de función n-aria $f \in \mathcal{F}$, una función $f_{\mathcal{A}} : A^n \to A$
- Para cada símbolo de predicado n-ario $P \in \mathcal{P}$, una relación $P_{\mathcal{A}} \subset A^n$ de tal forma que $P_{\mathcal{A}}$ contenga a los elementos que la hacen verdadera.

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

Dada tal interpretación, se considera que las variables pueden tomar cualquier valor del dominio.

Ejemplo de interpretación: Sea $\mathcal{L} = \mathcal{C} \cup \mathcal{F} \cup \mathcal{P}$ con $\mathcal{C} = \{c, d\}$, $\mathcal{F} = \{f, g\}$, $\mathcal{P} = \{P\}$. Si f unaria, g binaria y P binario, una posible interpretación $\mathcal{A}$ es:

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

Valuaciones: Sea $\mathcal{A}$ una $\mathcal{L}$-estructura con dominio A. Definimos una **valuación** para $\mathcal{A}$ como una función $v : \text{VAR} \to A$ que asigna un valor de A a una variable del lenguaje.

Por ejemplo, la valuación $v(x = a)$ que asigna el valor $a \in A$ a x se define de la siguiente manera:

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

Además, podemos definir $\tilde{v} : \text{TERM}(\mathcal{L}) \to A$ como una extensión de v que nos permite interpretar un término del lenguaje:

- Si $t = x$ es una variable entonces $\tilde{v}(t) = v(x)$
- Si $t = c$ es una constante entonces $\tilde{v}(t) = c_{\mathcal{A}}$
- Si $t = f(t_1, \ldots, t_n)$ es una función entonces $\tilde{v}(c) = f_{\mathcal{A}}(\tilde{v}(t_1), \ldots, \tilde{v}(t_n))$

Interpretación de una fórmula Sea $\mathcal{A}$ una $\mathcal{L}$-estructura con dominnio A y v una valuación de $\mathcal{A}$. Vamos a definir cuado una fórmula φ del lenguaje $\mathcal{L}$ es verdadera en $\mathcal{A}$ bajo la evaluación v ($\mathcal{A} \vDash \varphi[v]$):

1. $\mathcal{A} \vDash P(t_1, t_1, \ldots, t_n)[v]$ si y solo si $(\tilde{v}(t_1), \ldots, \tilde{v}(t_n)) \in P_{\mathcal{A}}$.
2. $\mathcal{A} \vDash \neg\psi[v]$ si y solo si $\mathcal{A} \nvDash \psi[v]$
3. $\mathcal{A} \vDash (\forall x)\psi[v]$ si y solo si $\mathcal{A} \vDash \psi[v(x = a)]$ para cualquier $a \in A$.
4. $\mathcal{A} \vDash (\psi \vee \rho)[v]$ si y solo si $\mathcal{A} \vDash \psi[v]$ o $\mathcal{A} \vDash \rho[v]$.
5. $\mathcal{A} \vDash (\psi \wedge \rho)[v]$ si y solo si $\mathcal{A} \vDash \psi[v]$ y $\mathcal{A} \vDash \rho[v]$.
6. $\mathcal{A} \vDash (\exists x)\psi[v]$ si y solo si hay algún $a \in A$ tal que $\mathcal{A} \vDash \psi[v(x = a)]$.

6.2. Satisfacibilidad y Validez

Sea φ una fórmula del lenguaje $\mathcal{L}$:

- φ es **satisfacible** si existe una $\mathcal{L}$-estructura $\mathcal{A}$ y una valuación v tal que $\mathcal{A} \vDash \varphi[v]$.
- φ es **verdadera (o válida) en una $\mathcal{L}$-estructura** $\mathcal{A}$ ($\mathcal{A} \vDash \varphi$) si $\mathcal{A} \vDash \varphi[v]$ para toda valuación v. En este caso decimos que $\mathcal{A}$ es un modelo de φ.
- φ es **universalmente válida** ($\vDash \varphi$) si $\mathcal{A} \vdash \varphi[v]$ para toda $\mathcal{L}$-estructura $\mathcal{A}$ y toda evaluación v de $\mathcal{A}$.

Proposición 6.1. *Una fórmula φ es **universalmente válida** si y solo si $\neg\varphi$ es insatisfacible (no existe ninguna $\mathcal{L}$-estructura $\mathcal{A}$, ni ninguna evaluación v tal que $\mathcal{A} \vDash \varphi[v]$)*

Notación: Sea $\Gamma \in \text{FORMS}(\mathcal{L})$ y $\mathcal{A}$ una $\mathcal{L}$-estructura, notamos $\mathcal{A} \vDash \Gamma[v]$ cuando $\mathcal{A} \vDash \psi[v]$ para toda fórmula $\psi \in \Gamma$.

Consecuencia semántica: Sea $\Gamma \subseteq \text{FORM}(\mathcal{L})$ y $\varphi \in \text{FORM}(\mathcal{L})$, entonces φ es **consecuencia semántica** de Γ ($\Gamma \vDash \varphi$) si para toda $\mathcal{L}$-estructura $\mathcal{A}$ y toda valuación v de $\mathcal{A}$, vale:

$$\mathcal{A} \vDash \Gamma[v] \Rightarrow \mathcal{A} \vDash \varphi[v]$$

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

6.2.1. Lema de la sustitución

Remplazo de variables libres por términos: Sea $\mathcal{L}$ un lenguaje fijo, $\varphi \in \text{FORM}(\mathcal{L})$, $t \in \text{TERM}(\mathcal{L})$ y $x \in \text{VAR}$ entonces: $\boldsymbol{\varphi[x/t]}$ es la fórmula obtenida a partir de *varphi* sustituyendo todas las apariciones libres de la variable x por t.

Variable remplazable por un término: Sea $t \in \text{TERM}(\mathcal{L})$ y $x \in \text{VAR}$ y $\varphi \in \text{FORM}(\mathcal{L})$. Decimos que x es **remplazable** por t en φ cuando:

- t es un término cerrado (no tiene variables) ó
- t tiene variables pero ninguna de ellas queda ligada por un cuantificador en el remplazo $\varphi[x/t]$.

Ejemplo: Sea $\varphi = (\forall y)(((\forall x)P(x)) \to P(x))$, notemos que el primer $P(x)$ tiene ligada x al cuantificador mientras que en el segundo está libre. Entonces:

- x es remplazable por z: $\varphi[x/z] = (\forall y)(((\forall x)P(x)) \to P(z))$
- x es remplazable por $f(x,z)$: $\varphi[x/f(x,z)] = (\forall y)(((\forall x)P(x)) \to P(f(x,z)))$

- x no es remplazable por $f(x,y)$ porque y está ligada al primer cuantificador y si hacemos la sustitución $\varphi[x/f(x,y)] = (\forall y)(((\forall x)P(x)) \to P(f(x,y)))$, entonces el y que pasamos como parámetro a f queda atrapado por el primero cuantificador.

Lema 6.1. *Si x es remplazable por t en φ entonces $\mathcal{A} \vDash (\varphi[x/t])[v]$ si y solo si $\mathcal{A} \vDash \varphi[v(x = \tilde{v}(t))]$*
Osea si $\mathcal{A}$ satisface φ con la valuación que asigna a x el valor t.

DEMOSTRACIÓN

Lo vamos a hacer por inducción en la complejidad de φ:

Caso base (fórmula atómica): Sea $\varphi = P(u)$ con u un término del lenguaje. Entonces $\mathcal{A} \vDash P(u[x/t]) \iff \tilde{v}(u[x/t]) \in P_{\mathcal{A}} \iff v(x = \tilde{v}(t))(u) \in P_{\mathcal{A}} \iff \mathcal{A} \vDash \varphi[v(= \tilde{v}(t))]$

Caso $\varphi = \neg\psi$ y $\varphi = \psi \to \rho$: Trivial.

Caso $\varphi = (\forall y)\psi$: Queremos ver que $\mathcal{A} \vDash (\varphi[x/t])[v]$ si y solo si $\mathcal{A} \vDash \varphi[v(x = \tilde{v}(t))]$

- Supongamos que x no aparece libre en φ. Entonces v y $v[x = v(t)]$ coinciden en todas las variables que aparecen libres en φ. Además, $\varphi = \varphi[x/t]$. Es trivial ver que vale el lema.
- Ahora supongamos x aparece libre en φ, entonces x aparece libre en ψ. Como x es remplazable por t, la variable y no aparece en t (porque sino no la variable quedaría atrapada por el cuantificador). Luego para todo $d \in A$, $w(t) = v(y = d)(t)$ (no importa que valor le asignemos a y, el valor de t no va a cambiar). Y como $x \neq y$, $\varphi[x/t] = ((\forall y)\psi)[x/t] = (\forall y)(\psi[x/t])$.

$$\begin{aligned}
\mathcal{A} \vDash \varphi[x/t][v] \text{ sii para todo } d \in A,\ & \mathcal{A} \vDash \psi[x/t][v(y = d)] \\
\text{sii para todo } d \in A,\ & \mathcal{A} \vDash \psi[v(y = d)(x = v(y = d)(t))] \\
\text{sii para todo } d \in A,\ & \mathcal{A} \vDash \psi[v(y = d)(x = v(t))] \\
\text{sii para todo } d \in A,\ & \mathcal{A} \vDash \psi[v(x = v(t))(y = d)] \\
\text{sii } & \mathcal{A} \vDash \varphi[v(x = v(t))]
\end{aligned}$$

□

6.3. Mecanismo deductivo SQ

Dado un lenguaje $\mathcal{L}$, sean $\varphi, \psi, \rho \in \text{FORM}(\mathcal{L})$, $x \in \text{VAR}$, $t \in \text{TERM}(\mathcal{L})$, tendremos como axiomas:

SQ1 $\varphi \to (\psi \to \varphi)$

SQ2 $(\varphi \to (\psi \to \rho)) \to ((\varphi \to \psi) \to (\varphi \to \rho))$

SQ3 $(\neg\varphi \to \neg\psi) \to (\psi \to \varphi)$

SQ4 $(\forall x)\varphi \to \varphi[x/t]$ si x es remplazable por t en φ.

SQ5 $\varphi \to (\forall x)\varphi$ si x no aparece libre en φ.

SQ6 $(\forall x)(\varphi \to \psi) \to ((\forall x)\varphi \to (\forall x)\psi$.

SQ7 Si φ es una axioma entonces $(\forall x)\varphi$ también es un axioma.

Además, seguimos pudiendo usar el **modus ponens** como regla de inferencia:

MP Sean $\varphi, \psi \in$FORM($\mathcal{L}$). ψ es una consecuencia inmediata de $\varphi \to \psi$ y ψ.

6.3.1. Consecuencia sintáctica, demostraciones, teoremas y teorías

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

Demostración: Una demostración de φ en SQ es una cadena finita y no vacía $\varphi_1, \ldots, \varphi_n$ de fórmulas de $\mathcal{L}$ tal que $\varphi_n = \varphi$ y

- φ_i es un axioma ó
- φ_i es una consecuencia inmediata de φ_k y φ_l con $k, l < i$.

Teorema ($\vdash \varphi$): Es una fórmula φ para la que existe una demostración en SP.

Consecuencia sintáctica ($\Gamma \vdash \varphi$): φ es una consecuencia sintáctica de Γ si existe una cadena finita y no vacía $\varphi_1, \ldots, \varphi_n$ de fórmulas de $\mathcal{L}$ tal que $\varphi_n = \varphi$ y

- φ_i es un axioma ó
- $\varphi_i \in \Gamma$ ó
- φ_i es una consecuencia inmediata de φ_k y φ_l con $k, l < i$.

Aquí $\varphi_1, \ldots, \varphi_n$ se llama **derivacíón** de φ a partir de Γ, Γ se llama **teoría** y decimos que φ es un **teorema de la teoría** Γ.

Teorema 6.1. *El sistema SQ es correcto. Es decir que si* $\Gamma \vdash \varphi$ *entonces* $\Gamma \vDash \varphi$

Teorema 6.2. *El sistema SQ es consistente. Es decir que no existe* $\varphi \in$*FORM($\mathcal{L}$) tal que* $\vdash \varphi$ *y* $\vdash \neg\varphi$.

Teorema 6.3. *Si* $\Gamma \cup \{\varphi\} \vdash \psi$ *entonces* $\Gamma \vdash \varphi \to \psi$

Conjunto consistente: $\Gamma \subseteq \text{FORM}(\mathcal{L})$ es consistente si no existe $\varphi \in \text{FORM}(\mathcal{L})$ tal que $\Gamma \vdash \varphi$ y $\Gamma \vdash \neg\varphi$.

Proposición 6.2. *Sea $\Gamma \subseteq FORM(\mathcal{L})$ y $\varphi \in FORM(\mathcal{L})$*

- $\Gamma \cup \{\varphi\}$ *es incosistente si y solo si* $\Gamma \vdash \neg\varphi$
- $\Gamma \cup \{\neg\varphi\}$ *es incosistente si y solo si* $\Gamma \vdash \varphi$

Teorema 6.4. *Si Γ es satisfacible entonces Γ es consistente.*

Teorema 6.5. *Si Γ es inconsistente, entonces existe un subconjunto finito de Γ que es inconsistente.*

Instancia de esquema tautológico: Sea $\varphi(p_1, \ldots, p_n)$ una tautológia de P con variables proposicionales $p_1, \ldots, p_n$. Sean $\psi_1, \ldots, \psi_n$ fórmulas cualesquiera de primer orden, notamos $\varphi(\psi_1, \ldots, \psi_n)$ a la fórmula que se obtiene de remplazar p_i por ψ_i y la llamos **instancia de un esquema tautológico**.

Proposición 6.3. *Si φ es una instancia de un esquema tautológico entonces $\vdash \varphi$.*

Variantes alfabéticas Una fórmula φ' es una variante alfabética de φ si φ' se puede obtener a partir de φ renombrando sus variables. Por ejemplo:

$$\varphi = x \neq 0 \to (\exists y)x = S(y) \qquad \varphi' = x \neq 0 \to (\exists w)x = S(w)$$

Lema 6.2. *Sea $\varphi \in FORM(\mathcal{L})$. Dados $x \in VAR$ y $t \in TERM(\mathcal{L})$ podemos encontrar φ' (variante alfabética de φ) tal que:*

- $\{\varphi\} \vdash \varphi'$ *y* $\{\varphi'\} \vdash \varphi$
- *x es remplazable por t en φ'*

6.3.2. Teorema de la generalización (TG)

Teorema 6.6. *Si $\Gamma \vdash \varphi$ y x no aparece libre en ninguna fórmula de Γ, entonces $\Gamma \vdash (\forall x)\varphi$*

DEMOSTRACIÓN

Vamos a demostrarlo por inducción en la longitud de la derivación de ψ a partir de Γ.
Sea ψ, Γ y x tal que $\Gamma \vdash \psi$ y x no aparece libre en ninguna fórmula de Γ, si $\psi_1, \ldots, \psi_n$ es una derivación de ψ:

Caso base ($n = 1$): Queremos ver que $\Gamma \vdash (\forall x)\psi$. Tenemos dos posibilidades:

- ψ es un axioma de SQ $\overset{SQ7}{\Rightarrow} \vdash (\forall x)\psi \Rightarrow \Gamma \vdash (\forall x)\psi$
- $\psi \in \Gamma$. Por SQ5, como x no aparece libre en ψ, sabemos que $\vdash \psi \to (\forall x)\psi$.

  Luego, usando modus ponens con $\Gamma \vdash \psi$ y $\vdash \psi \to (\forall x)\psi$, obtenemos que $\Gamma \vdash (\forall x)\psi$.

Paso inductivo: Nuestra hipotesis inductiva es que para toda fórmula ψ tal $\Gamma \vdash \psi$ tiene una derivación de longitud $m < n$, vale que $\Gamma \vdash (\forall x)\psi$ para toda x que no aprece libre en ninguna fórmula de Γ.
Si φ es un axioma de SQ ó $\varphi \in \Gamma$ es lo mismo que en el caso base.
Si φ se obtiene por modus ponens de φ_i y φ_j con $i, j < n$. Supongamos que $\varphi_j = \varphi_i \to \varphi$:

- Como $i < n$, vale la hipotesis inductiva y $\Gamma \vdash (\forall x)\varphi_i$
- Como $j < n$, vale la hipotesis inductiva y $\Gamma \vdash (\forall x)(\varphi_i \to \varphi)$

Por SQ6, $\vdash (\forall x)(\varphi_i \to \varphi) \to ((\forall x)\varphi_i \to (\forall x)\varphi)$. Luego, usando modus ponensdos veces, obtenemos $\Gamma \vdash (\forall x)\varphi$.

6.3.3. Teorema de generalización en constantes (TGC)

Teorema 6.7. *Supongamos que $\Gamma \vdash \varphi$ y c es un símbolo de constante que no aparece en Γ. Entonces exite una variable x que no aprece en φ tal que $\Gamma \vdash (\forall x)(\varphi[c/x])$. Más aún, hay una derivación de $(\forall x)(\varphi[c/x])$ a partir de Γ en donde c no aparece.*

DEMOSTRACIÓN

IDEA DE LA DEMOSTRACION: Sea $\varphi_1, \ldots, \varphi_n$ una derivación de φ a partir de Γ. Sea x la primer variable que no aparece en ninguna de las φ_i:

1. Demostrar por inducción en n que $\varphi_1[c/x], \ldots, \varphi_n[c/x]$ es una derivación de $\varphi[c/x]$ a partir de Γ y luego que no contiene al símbolo de constante c.
2. Hay un $\Delta \subseteq \Gamma$ finito tal que $\Gamma \vdash \varphi[c/x]$ con derivación que no usa c y tal que x no aparece libre en ninguna fórmula de Δ.
3. Por el teorema de la generalización, $\Delta \vdash (\forall x)(\varphi[c/x])$ con derivación que no usa c □

Corolario 6.1. *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 aún, hay una derivación de $(\forall z)\varphi$ a partir de Γ en donde c no aparece.*

DEMOSTRACIÓN

Por TGC (6.7), existe un x que no aparece en $\varphi[z/c]$ tal que $\Gamma \vdash (\forall x)(\varphi[z/c][c/x])$ y c no aparece en la derivación de esta fórmula.
Como c no aparece en φ, $\varphi[z/c][c/x] = \varphi[z/x]$, entonces $\Gamma \vdash (\forall x)(\varphi[z/x])$.
Sabemos que $\vdash (\forall x)(\varphi[z/x]) \to (\forall z)\varphi$, entonces por modus ponens obtenemos $\Gamma \vdash (\forall z)\varphi$. □

6.3.4. Lenguajes con igualdad

Dado un lenguaje $\mathcal{L}$ con igualdad, debemos considerar el sistema deductivo $\mathrm{SQ}^{=}$ que extiene a SQ con los siguientes axiomas:

SQ$^=$1 $x = x$

SQ$^=$2 $x = y \to (\varphi \to \psi)$ donde φ es atómica y ψ se obtiene de φ remplazando x por y en cero o más lugares.

Se puede probar que SQ$^=$ es consistente y que si hay una derivación de φ en SQ$^=$ entonces φ es verdadera en toda $\mathcal{L}$-estructura en donde el $=$ se interpreta como igualdad.

6.4. Consistencia implica satisfacibilidad

Teorema 6.8. *Sea $\Gamma \subseteq FORM(\mathcal{L})$ consistente y $\mathcal{C}$ 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 que Γ es inconsistente en el nuevo lenguaje $\mathcal{L}'$. Entonces existe $\varphi \in$FORMS($\mathcal{L}$) tal que $\Gamma \vdash \varphi$ y $\Gamma \vdash \neg\varphi$.
Luego, existen dos derivaciones finitas que usan fórmulas en FORM($\mathcal{L}'$) que terminan en φ y $\neg\varphi$.
Por TGC (teorema **??**), cada constante nueva utilizada puede remplazarse por una variable nueva.
Luego obtenemos dos derivaciones: $\Gamma \vdash \varphi[c_1, \dots, c_n/x_1, \dots, x_n]$ y $\Gamma \vdash \neg\varphi[c_1, \dots, c_n/x_1, \dots, x_n]$ en el lenguaje original $\mathcal{L}$. Entonces Γ era inconsistente en $\mathcal{L}$, lo que es absurdo.

Téstigo: Sea $\Gamma \subseteq$FORM($\mathcal{L}$) consistente y $\mathcal{C}$ un conjunto de nuevas constantes que no aparecen en $\mathcal{L}$ y $\mathcal{L}' = \mathcal{L} \cup \mathcal{C}$. Sea $\langle \varphi_1, x_1 \rangle, \langle \varphi_2, x_2 \rangle, \dots$ una enumeración de FORM($\mathcal{L}$)$\times$VAR, un **testigo** es una fórmula de la siguiente forma:

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

donde c_n es la primera constante de $\mathcal{C}$ que no aparece en φ_n y no aparece en $\theta_1, \dots, \theta_{n-1}$.

Este tipo de fórmulas son llamadas **testigos** porque si ocurre $\neg(\forall x)\varphi$ entonces hay una constante c que atestigua que φ no vale para todo x.

Llamaremos $\Theta = \{\theta_1, \theta_2, \dots\}$ al conjunto de todos los testigos.

Teorema 6.9. *Sea $\Gamma \subseteq FORM(\mathcal{L})$ consistente y $\mathcal{C}$ un conjunto de nuevas constantes que no aparecen en $\mathcal{L}$ y $\mathcal{L}' = \mathcal{L} \cup \mathcal{C}$. Entonces $\Gamma \cup \Theta \subseteq FORM(\mathcal{L}')$ es consistente.*

DEMOSTRACIÓN

Supongamos que $\Gamma \cup \Theta$ es inconsistente, entonces debe existir i tal que $\Gamma \cup \{\theta_1, \dots, \theta_{i+1}\}$ es inconsistente.
Sea n el mínimo i para el que pasa esto, entones $\Gamma \cup \{\theta_1, \dots, \theta_n\}$ es consistente.
Luego $\Gamma \cup \{\theta_1, \dots, \theta_n\} \vdash \neg\theta_{n+1}$, osea $\Gamma \cup \{\theta_1, \dots, \theta_n\} \vdash \neg(\neg(\forall x)\varphi \to \neg(\varphi[x/c]))$ donde c no aparece en ningun θ_k con $k \leq n$.

$$\neg\theta_{n+1} = \neg(\neg(\forall x)\varphi \to \neg(\varphi[x/c])) = \neg(\neg(\neg(\forall x)\varphi) \vee \neg(\varphi[x/c])) = \neg((\forall x)\varphi) \vee \neg(\varphi[x/c]))$$
$$= \neg(\forall x)\varphi) \wedge \neg\neg(\varphi[x/c])) = \neg(\forall x)\varphi \wedge (\varphi[x/c]))$$

Luego, $\neg\theta_{n+1} \to \neg(\forall x)\varphi$ y $\neg\theta_{n+1} \to (\varphi[x/c])$ son instancias de esquemas tautológicos y, por lo tanto:

1. $\vdash \neg\theta_{n+1} \to \neg(\forall x)\varphi \overset{MP}{\Rightarrow} \Gamma \cup \{\theta_1, \ldots, \theta_n\} \vdash \neg(\forall x)\varphi$
2. $\vdash \neg\theta_{n+1} \to (\varphi[x/c]) \overset{MP}{\Rightarrow} \Gamma \cup \{\theta_1, \ldots, \theta_n\} \vdash \varphi[x/c]$.

Como c no aparece en $\Gamma \cup \{\theta_1, \ldots, \theta_n\}$, por (2) y el corolario 6.1 vale que $\Gamma \cup \{\theta_1, \ldots, \theta_n\} \vdash (\forall x)\varphi$. Esto contradice (1). Luego $\Gamma \cup \{\theta_1, \ldots, \theta_n\}$ es inconsistente. Absurdo.

6.4.1. Lema de Lindenbaum para $\Gamma \cup \Theta$

Teorema 6.10. *Sean Γ y Θ como en los teoremas anteriores. Existe un conjunto Δ que contiene $\Gamma \cup \Theta$ tal que Δ es maximal consistente.*

La demostración de este lema es igual que la del lema de Lindebaum para el caso proposicional.

6.4.2. Satisfacibilidad de un model canónico

Modelo canónico: Sea $\mathcal{L}' = \mathcal{L} \cup \mathcal{C}$, un modelo canónico es una $\mathcal{L}$-estrucutra $\mathcal{A}$ tal que:

- $A =$TERM$(\mathcal{L}')$
- Para cada símbolo de función n-aria $f \in \mathcal{L}'$, $f_{\mathcal{A}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n) \in A$
- Para cada símbolo de constante $c \in \mathcal{L}'$, $c_{\mathcal{A}} = c \in A$
- Si Δ es un conjunto maximal consistente, entonces: $(t_1, \ldots, t_n) \in P^{\mathcal{A}}$ si y solo si $P(t_1, \ldots, t_n) \in \Delta$,

Y definimos la evaluación v :VAR$\to$TERM$(\mathcal{L}')$ como: $v(x) = x$

Lema 6.3. *Para todo $t \in TERM(\mathcal{L}')$, $\bar{v}(t) = t$*

Lema 6.4. *Para toda $\varphi \in FORM(\mathcal{L}')$, $\mathcal{A} \vDash \varphi[v]$ si y solo si $\varphi \in \Delta$*

DEMOSTRACIÓN

Por indcucción en la longitud de la fórmula:

Caso base $\varphi = P(t_1, \ldots, t_n)$: φ es una fórmula atómica. $\mathcal{A} \vDash P(t_1, \ldots, t_n)[v]$ sii $(\bar{v}(t_1), \ldots, \bar{v}(t_n)) \in P^{\mathcal{A}}$ sii $(t_1, \ldots, t_n) \in P^{\mathcal{A}}$ (pues $\bar{v}(t) = t$) sii $P(t_1, \ldots, t_n) \in \Delta$

Hipotesis inductiva: Para toda fórmula ψ de menos longitud que φ vale que $\mathcal{A} \vDash \psi[v]$ sii $\psi \in \Delta$

Caso $\varphi = \neg\psi$**:** $\mathcal{A} \vDash \varphi[v]$ sii $\mathcal{A} \nvDash \varphi[v]$ sii (por hipotesis inductiva) $\psi \notin \Delta$ sii $\neg\psi \in \Delta$ (pues Δ es maximal consistente).

Caso $\varphi = \psi \to \rho$**:**

$\Rightarrow$) $\mathcal{A} \vDash \varphi[v] \iff \mathcal{A} \nvDash \psi[v]$ o $\mathcal{A} \nvDash \rho[v]$, por hipotesis inductiva esto pasa sii $\psi \notin \Delta$ ó $\rho \in \Delta \iff \neg\psi \in \Delta$ ó $\rho \in \Delta$ (porque Δ es maximal consistente) $\Rightarrow \Delta \vdash \psi \to \rho \Rightarrow \psi \to \rho \in \Delta$.

$\Leftarrow$) $\varphi \in \Delta \Rightarrow \psi \notin \Delta$ ó ($\psi\Delta$ y $\Delta \vdash \rho$) $\Rightarrow \psi \notin \Delta$ ó ($\psi \in \Delta$ y $\rho \in \Delta$) $\Rightarrow \psi \notin \Delta$ ó $\rho \in \Delta \iff \mathcal{A} \nvDash \psi[v]$ ó $\mathcal{A} \vdash \rho[v]$ (esto vale por hipotesis inductiva) $\iff \mathcal{A} \vDash \psi \to \rho[v]$

Caso $(\forall x)\psi$**:**

$\Rightarrow$) Supongamos que $\mathcal{A} \vDash (\forall x)\psi[v]$. Entonces, para todo $t \in A$, $\mathcal{A} \vDash \psi[v(x = t)]$, en particular $\mathcal{A} \vDash \psi[v(x = c)]$ para $c \in \mathcal{C}$.

Por definición de v, $\mathcal{A} \vDash \psi[v(x = \bar{v}(c))]$ y, por el lema de sustitución (6.1) $\mathcal{A} \vDash (\psi[x/c])[v]$.

Luego, por hipotesis inductiva $(\psi[x/c])[v] \in \Delta$.

Sea $\theta = \neg(\forall x)\psi \to \neg(\psi[x/c]) \in \Theta$, el testigo agregado par la fórmula ψ. Para que θ sea verdadera tiene que valer que $\neg(\forall x)\psi \notin \Delta$ (si estuviese, entonces el consecuente tambien debería pertenecer pero esto es absurdo, pues $(\psi[x/c])[v] \in \Delta$).

> Supongamos que $\neg(\forall x)\psi \in \Delta$, entonces $\Delta \vdash \neg(\forall x)\psi$. Además, como $\Theta \subseteq \Delta$, $\theta \in \Delta$ por lo que $\Delta \vdash \theta$. Por Modus Ponens entre θ y $\neg(\forall x)\psi$, tenemos que $\Delta \vdash \neg(\psi[x/c])$. Luego $\neg(\psi[x/c]) \in \Delta$. Absurdo, habiamos demostrado que $\psi[x/c] \in \Delta$.

Luego $(\forall x)\psi \in \Delta$

$\Leftarrow$) Supongamos que $\mathcal{A} \nvDash (\forall x)\psi[v]$. Entonces, existe $t \in A$, $\mathcal{A} \nvDash \psi[v(x = t)]$. Sea ψ' una variate alfabética de ψ tal que x sea remplazable por t en ψ', $\mathcal{A} \nvDash \psi'[v(x = t)]$.

Como $\bar{v}(t) = t$, $\mathcal{A} \nvDash \psi'[v(x = \bar{v}(t))]$ y, por el lema de sustitución (6.1) $\mathcal{A} \nvDash (\psi'[x/t])[v]$.

Luego, por hipotesis inductiva $(\psi'[x/t])[v] \notin \Delta$.

Tenemos que ver que $(\forall x)\psi \notin \Delta$, (sino no valdría θ).

> Supongamos que $(\forall x)\psi' \in \Delta$, entonces $\Delta \vdash (\forall x)\psi'$. Por **SQ4**, sabemos que $\vdash (\forall x)\varphi' \to \varphi'[x/t]$. Luego, por modus ponens podemos concluir que $\Delta \vdash \varphi[x/t]$ y $\varphi'[x/t] \in \Delta$. Lo que es absurdo.

Luego, $(\forall x)\psi' \notin \Delta$ y, por equivalencia de variantes alfabéticas $(\forall x)\psi \notin \Delta$

□

6.4.3. Satisfacibilidad de un lenguaje $\mathcal{L}$ cualqiera

Teorema 6.11. *Sea $\Gamma \subseteq FORM(\mathcal{L})$ consistente. Entonces existe una L-estructura $\mathcal{B}$ y una evaluación v de $\mathcal{B}$ tal que $\mathcal{B} \vDash \varphi[v]$ para toda $\varphi \in \Gamma$*

DEMOSTRACIÓN

Dado un lenguaje $\mathcal{L}$ y su módelo cánonico $\mathcal{A}$, definimos $\mathcal{B}$ como el modelo que restringe $\mathcal{A}$ a $\mathcal{L}$. Del lema 6.4, sabemos que para toda fórmula $\varphi \in$FORM($\mathcal{L}'$), $\mathcal{A} \vDash \varphi$ sii $\varphi \in \Delta$. Como $\Gamma \subseteq \Delta$, si $\varphi \in \Gamma$, tenemos que $\mathcal{A} \vdash \varphi[v]$ sii $\mathcal{B} \vDash \varphi[v]$.
Luego, encontramos la L-estructura $\mathcal{B}$ y una evaluación v de $\mathcal{B}$ tal que $\mathcal{B} \vDash \varphi[v]$ para toda $\varphi \in \Gamma$.
Entonces podemos concluir que Γ es satisfacible.

6.4.4. Teorema de Löwenheim-Skolem

Corolario 6.2. Γ *es consistente sii* Γ *es satisfacible*

Teorema 6.12. *Sea $\mathcal{L}$ un lenguaje numerable y sin igualdad. Si $\Gamma \subseteq FORM(\mathcal{L})$ es satisfacible, es satisfacible en un modelo infinito numerable.*

Teorema 6.13. *Sea $\mathcal{L}$ un lenguaje numerable con igualdad. Si $\Gamma \subseteq FORM(\mathcal{L})$ es satisfacible, es satisfacible en un modelo finito o infinito numerable.*

Teorema 6.14. *Sea $\mathcal{L}$ un lenguaje numerable y $\Gamma \subseteq FORM(\mathcal{L})$ tiene modelo infinito entonces tiene modelo de cualquier cardinalidad.*

6.5. Completitud y compacidad

Teorema 6.15. *Si $\Gamma \vDash \varphi$ entonces $\Gamma \vdash \varphi$*

Corolario 6.3. $\Gamma \vDash \varphi$ *sii* $\Gamma \vdash \varphi$

Teorema 6.16. *Sea $\Gamma \subseteq FORM(\mathcal{L})$. Si todo subconjunto finito Δ de Γ ($\Delta \subseteq \Gamma$) es satisfacible, entonces Γ es satisfacible.*

Teorema 6.17. *Si Γ tiene modelos arbitrariamente grandes, tiene modelo infinito.*

DEMOSTRACIÓN

Definimos (en un lenguaje con solo la igualdad) las siguientes fórmulas:

$$\varphi_2 = (\exists x)(\exists y)x \neq y$$
$$\varphi_3 = (\exists x)(\exists y)(\exists z)(x \neq y \wedge x \neq z \wedge y \neq z)$$
$$\vdots$$
$$\varphi_n = \text{"hay al menos n elementos"}$$

Si un lenguaje cumple todas las fórmulas $\varphi_1, \ldots, \varphi_n$, quiere decir que tiene n elementos distintos. Ahora, supongamos que Γ tiene modelos arbitrariamente grandes, entonces todo subconjunto finito de $\Gamma \cup \{\varphi_i \mid i \geq 2\}$ tiene modelo, osea es satisfacible.
Luego, por el teorema de la compacidad (6.16), $\Gamma \cup \{\varphi_i \mid i \geq 2\}$ es satisfacible por lo que existe un modelo $\mathcal{M}$ que lo satisface. Notemoas que $\{\varphi_i \mid i \geq 2\}$ que como el modelo satisface todas las formulas con $i \geq 2$, tiene infinitos elementos distintos. Luego $\mathcal{M}$ tiene que ser infinito.

Proposición 6.4. *$\mathcal{A}$ es infinito sii $\mathcal{A} \vDash \{\varphi_i \mid i \geq 2\}$*

6.6. Indecidibilidad de primer orden

Vamos a demostrar que el problema de decidir si una fórmula puede ser derivada o no, no es computable.

Para esto vamos a armar un lenguaje y una interpretación que nos permitan traducir facilmente una fórmula en un programa de una máquina de turing y vicevers.

El lenguaje: Sea $\mathcal{L}$ el siguiente lenguaje:

- ϵ es el único símbolo de constante.
- 1 y $*$ son símbolos de funciones unarias.
- Sea $E = \{q_0, q_f, p, q, r, \ldots\}$ un conjunto infinito de estados de una máquina de turing, entonces los símbolos de relación son: $R_{q_0}, R_{q_f}, R_p, R_q, R_r, \ldots$

Si t es un término de $\mathcal{L}$, vamos a notar:

- $1(t)$ como $1t$
- $*(t)$ como $*t$

La interpretación: Dada una máquina de Turing $\mathcal{M} = (\Sigma, Q, T, q_0, q_f)$ y una entrada $w \in \{1\}^+$ (string de unos). Vamos a definir una interpretación $\mathcal{A}_{M,w}$ de la siguiente forma:

- El universo $A_{M,w} = \{1, *\}$ son las cadenas finitas formadas por los carácteres 1 y $*$
- $\epsilon_{\mathcal{A}}$ es la cadena vacía.
- Las funciones $1_{\mathcal{A}} : A \to A$ y $*_{\mathcal{A}} : A \to A$, agregar el respectivo caracter adelante de un string ($1_{\mathcal{A}}(*11*1) = 1*11*1$).
- Para cada estado $q \in Q$, $(R_q)_{\mathcal{A}}(x, y)$ es verdadero sii la máquina M con entrada w llega al estado q con x escrito en orden inverso y a continuación y y la cabeza de M apunta al primer carácter de y.

La fórmula programa: Debemos definir una fórmula lógica que nos permita describir la máquina de turing mencionda:

- $\varphi_{=}R_{q_0}(1\ldots1\epsilon,\epsilon)$ indica que se puede alcanzar el estado inicial. $\mathcal{A}\vDash\varphi_0$
- $\varphi_f=(\exists x)(\exists y)R_{q_f}(x,y)$ indica que el estado final es alcanzable. $\mathcal{A}\vDash\varphi_f$ sii $M(w)\downarrow$ (si la máquina término cuando se le pasa w como entrada)
- Para cada instrucción $I\in T$ generamos una fórmula ψ_I que describe las transiciones realizadas.

Entonces definimos la **fórmula programa** como:

$$\varphi_{M,w}=(\varphi_0\wedge\bigwedge_{i\in T}\psi_I)\to\varphi_f$$

Proposición 6.5. *$\mathcal{A}\vDash\varphi_{M,w}$ sii $M(w)\downarrow$*

DEMOSTRACIÓN

Sabemos que $\mathcal{A}\vDash\varphi_0$ y que $\mathcal{A}\vDash\varphi_f$ sii $M(w)\downarrow$. Tambien que $\mathcal{A}\vDash\psi_I$ para $I\in T$.
Luego $\mathcal{A}\vDash\varphi_{M,w}$ sii $\mathcal{A}\vDash\varphi_f$ sii $M(w)\downarrow$

Teorema 6.18. *$\vdash\varphi_{M,w}$ sii $M(w)\downarrow$*

DEMOSTRACIÓN

($\Rightarrow$) Si $\vdash\varphi_{M,w}$ entonces $\vDash\varphi_{M,w}$, es decir, $\varphi_{M,w}$ es veradera en toda interpreación. En particular, $\mathcal{A}\vDash\varphi_{M,w}$. Luego $M(w)\downarrow$.
($\Leftarrow$) Idea. Si $M(w)\downarrow$ entonces existe un computo de $M(w)$. Entonces dada una configuración inicial, solo falta escribir la secuencia de configuraciones que realiza M hasta llegar al estado final. Como el estado inicial, el final y cada una de las transiciones utilizadas se corresponden directamente con una de las fórmulas descriptas más arribas, entonces, el cómputo nos sirve como demostración $\varphi_{M,w}$. □

Teorema 6.19. *Sea $\psi\in$FORM($\mathcal{L}$). El problema de decidir si $\vdash\psi$ o $\nvdash\psi$ no es computable.*

DEMOSTRACIÓN

Supongamos que hay un programa que dada $\psi\in$FORM($\mathcal{L}$) devuelve verdadero sii $\vdash\psi$
Entonces habría un procedimiento para decidir si $M(w)\downarrow$ ó $M(w)\uparrow$:

1. Construir $\varphi_{M,w}$
2. Si $\vdash\varphi_{M,w}$ entonces $M(w)\downarrow$; si no $M(w)\uparrow$.

□

Apéndices

A. Otras funciones primitivas recursivas:

A.1. Division

$y \text{ div } y = \min_{t \leq x} x * (t+1) > y$

A.2. Divisor

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

A.3. i-ésimo primo

$p_0 = 2$
$p_{t+1} = \min_{k \leq p_t! + 1}(k > p_t \wedge \text{esPrimo}(p_k)$

B. Otras Macros

B.1. $V \leftarrow 0$

$[A]$ $V \leftarrow V - 1$
IF $V \neq 0$ GOTO A

La macro resta uno a V hasta que sea cero.

B.2. $V \leftarrow k$

$V \leftarrow 0$
$V \leftarrow V + 1$
$V \leftarrow V + 1$
$\vdots$
$V \leftarrow V + 1$

La macro asigna cero a V y luego la incrementa en uno k veces.

B.3. $x = 0$

IF $X \neq 0$ GOTO E
$Y \leftarrow Y + 1$

Este programa recibe X como parámetro. Devuelve 1 si X es igual a cero y devuelve 0 si no.

B.4. $V \leftarrow f(V_1, ..., V_n)$

Dada f una función parcialmente computable queremos asignar a V el resultado de pasarle como parámetro las variables $V_1, \ldots, V_n$.

Como f es parcial computable, sabemos que existe un programa P tal que $f(x_1, \ldots, x_n) = \Psi_P^{(m)}(x_1, \ldots, x_n)$. Asumamos que P usa las variables $X_1, \ldots, X_m$, $Z_1, \ldots, Z_k$ e Y, además de las etiquetas $E, A_1, \ldots, A_l$.

Además, supongamos que para cada instrucción de la forma IF $V \neq 0$ GOTO A_i, la etiqueta referenciada se encuentra dentro del programa P, por lo que la única etiqueta de *salida* es E.

Entonces, si tenemos un programa P_0 en el que queremos almacenar el resultado de una función, solo hace falta renombrar las variables y etiquetas usadas por P a variables y etiquetas que no hayan sido usadas por P_0.

Sea m un número lo suficientemente grande como para que las variables y etiquetas que remplazamos no estén usadas por P_0 entonces, podemos hacer los siguientes remplazos: $Y \to Z_m$, $X_i \to Z_{m+i+1}$, $Z_i = Z_{m+n+1+i}$ y $A_i \to A_{m+i}$.

$$
\begin{array}{ll}
 & Z_m \leftarrow 0 \\
 & Z_{m+1} \leftarrow V_1 \\
 & Z_{m+2} \leftarrow V_2 \\
 & \vdots \\
 & Z_{m+n} \leftarrow V_n \\
 & Z_{m+n+1} \leftarrow 0 \\
 & \vdots \\
 & Z_{m+n+k} \leftarrow 0 \\
 & P \\
[E_m] & W \leftarrow Z_m
\end{array}
$$

Primero *inicializamos* las variables de P. Z_m es donde el programa P (con los remplazos) va a guardar su resultado y Z_{m+1} a Z_{m+n} son las variables que remplazaron a los parámetros y Z_{m+n+1} a Z_{m+n+k} las variables locales. Además, de los remplazos de las etiquetas A_i dentro de P, remplazamos E por E_m.

De esta forma, cuando P termina tanto porque se acabó la lista de instrucciones o porque hizo un salto a E_m asignamos el valor de Z_m a W.

Si quisieramos ejecutar P dentro de un loop, cada vez que termine va a dejar los valores de la última ejecución en las variables que use. Por esta razón es necesario realizar la *inicialización*.

B.5. IF $p(V_1, \ldots, V_n)$ GOTO L

$$
\begin{array}{l}
Z \leftarrow p(V_1, \ldots, V_n) \\
\text{IF } Z \neq 0 \text{ GOTO } L
\end{array}
$$

Usando la macro definida en B.4, asignamos a una variable Z el resultado de ejecutar el predicado p con $V_1, \ldots, V_n$ como variables de entrada. Luego ejecutamos la instrucción IF.

C. Otras definiciones

C.1. La función de Ackerman

Es una función que no es primitiva recursiva.

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

C.1.1. Versión de Robinson & Peter

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