Notas de Lógica y Computabilidad

(Basadas en el curso de 2013 de A. Petrovich)

Nahuel Albarracín
E-mail: nahuelnehuen89@gmail.com

Tabla de Contenidos

1 Nociones preliminares: alfabeto, expresión y lenguaje

Definición (Expresión)**.** Sea A un conjunto no vacío. Una **expresión sobre** A es una sucesión finita de elementos de A (i.e, una función $f : I_k \longrightarrow A$, donde $I_k = \{0, 1, \ldots, k-1\}, k \geq 1$). Si notamos $f(i) = a_i$, la sucesión finita es $a_0 a_1 \ldots a_{k-1}$

Notamos con A^* al conjunto de todas las expresiones definidas sobre A. Notar que A^* es siempre infinito (al conjunto A también se lo denomina **alfabeto**). En efecto, si $a \in A$, se puede generar las expresiones $a, aa, \ldots, \underbrace{aaa\ldots a}_{k \text{ veces}}$

Si $E \in A^*$, se define la longitud de E (notamos $\ell(E)$), como el número de símbolos que aparecen en E.

Observación
Si $E, F \in A^*$, $E = a_0 a_1 \ldots a_{k-1}$, $F = b_0 b_1 \ldots b_{n-1}$, entonces $E = F$ si y solo si $\ell(E) = \ell(F)$ y $a_i = b_i$ para todo $0 \leq i \leq k-1$.

Definición (Concatenación de expresiones)**.** Si $E, F \in A^*$, $E = \underbrace{a_0 a_1 \ldots a_{k-1}}_{f}$, $F = \underbrace{b_0 b_1 \ldots b_{n-1}}_{g}$, se define $EF = a_0 a_1 \ldots a_{k-1} b_0 b_1 \ldots b_{n-1}$, que se corresponde con la siguiente función $h : I_{n+k} \longrightarrow A$:

$$h(j) = \begin{cases} f(j) & 0 \leq j \leq k-1 \\ g(j-k) & k \leq j \leq n+k-1 \end{cases}$$

Observaciónes
Es fácil ver que la concatenación de expresiones es asociativa. Como una consecuencia inmediata, si $E \leq F$, entonces $GE \leq GF \; \forall \; G \in A^*$.
Si $E, F \in A^*$, tenemos que $\ell(EF) = \ell(E) + \ell(F)$, y en general $EF \neq FE$.

Definición. Si $E, F \in A^*$, decimos que $E \leq F$ si existe $G \in A^*$ tal que $F = EG$ (decimos en tal caso que E es sección inicial de F)

Proposición. Si A es un conjunto no vacío, se verifican las siguientes propiedades:

1) $\leq$ es un orden parcial.

2) Sean $E, F, G, H \in A^*$ tales que $EF = GH$. Entonces $E \leq G$ o $G \leq E$

Demostración.
• Reflexividad: $E \leq E$ pues $G = \emptyset$ cumple que $EG = E$.
• Antisimetría: $E \leq F$ y $F \leq E$, tenemos que $F = EG$, y $E = FG'$, de manera que $F = (FG')G$, y entonces $\cancel{\ell(F)} = \cancel{\ell(F)} + \ell(G') + \ell(G)$, de modo que $\ell(G) + \ell(G') = 0$, o sea, $G = G' = \emptyset$, y por ende $E = F$.
• Transitividad: Si $E, F, G \in A^*$ son tales que $E \leq F \leq G$, existen expresiones $H_1, H_2 \in A^*$ tales que $F = EH_1$ y $G = FH_2 = (EH_1)H_2 = E(H_1 H_2)$. Luego, $E \leq F$

(2) Supongamos que $EF = GH$ (podemos suponer que $E, G \neq \emptyset$). Sea $p(n)$ la siguiente sentencia:

$$p(n) := \text{" Si } E \in A^* \text{ con } \ell(E) = n \text{ y } F, G, H \in A^* \text{ son tales que } EF = GH, \text{ entonces } E \leq G \text{ o } G \leq E \text{ "}$$

$p(1)$ es verdadera: en efecto, si $E = a$, con $a \in A$, tal que $aF = GH$, entonces $G = aG'$ para cierta $G' \in A^*$, así que $E \leq G$.

Supongamos que $p(n)$ es verdadera, y probemos $p(n+1)$: sea $E \in A^*$ con $\ell(E) = n+1$, y sean $F, G, H \in A^*$ tales que $EF = GH$. Podemos escribir $E = aE'$, con $a \in A$, y $E' \in A^*$. Nos queda así que $aE'F = GH$, de modo que $G = aG'$ para cierta $G' \in A^*$. Luego, $aE'F = aG'H$, de modo que $E'F = G'H$, con $\ell(E') = n$. Por hipótesis inductiva, tenemos que $E' \leq G'$ o $G' \leq E'$. Suponiendo que $E' \leq G'$, tenemos que $G' = E'\widetilde{G}$, y entonces (concatenando a por detrás) $G = E\widetilde{G}$, de modo que $E \leq G$. Análogamente, si $G' \leq E'$, entonces $E' = G'\widehat{G}$; concantenando a, se obtiene que $E = G\widehat{G}$, de modo que $G \leq E$. □

Definición (Lenguaje)**.** Si $A \neq \emptyset$, un **lenguaje sobre** A es un subconjunto no vacío $\mathcal{L}$ de A^*. A los elementos (expresiones) de $\mathcal{L}$ las llamamos **palabras de** $\mathcal{L}$.

Ejemplos
(1) $\mathcal{L} = A^*$, con $A \neq \emptyset$
(2) A el alfabeto del idioma español, y $\mathcal{L} \subseteq A^*$ el conjunto de palabras que aparecen en la última edición del RAE.
(3) $A = \{\mathbf{A}, \mathbf{C}, \mathbf{G}, \mathbf{T}\}$, y $\mathcal{L}$ el lenguaje de los genes.
(4) $A = \{0, 1, 2, \ldots, 9\}$ y $\mathcal{L} = \{x \in A^* : x \text{ representa en decimal un número natural }\}$. Concretamente, para una expresión $E = a_0 a_1 \ldots a_n \in A^*$,

$$E \in \mathcal{L} \Longleftrightarrow a_0 \neq 0 \text{ o } a_0 = 0 \text{ y } n = 0$$

(5) Para $A = I_{10} \cup \{\mathbf{,}\} \cup \{\frown\}$, definamos un lenguaje $\mathcal{L} \subseteq A^*$ de la siguiente manera: sea $\mathcal{L}' \subseteq (I_{10})^*$ el lenguaje del ítem anterior. Una expresión $E \in A^*$ es una palabra de $\mathcal{L}$ sii vale una y solo una de las siguientes condiciones:

i) $E \in \mathcal{L}'$

ii) $E = F\mathbf{,}G$ donde $F \in \mathcal{L}'$ y $G \in (I_{10})^*$ no es una tira de ceros.

iii) $E = F\mathbf{,}G \frown H$, donde $F \in \mathcal{L}'$ y $G, H \in (I_{10})^*$, y H no es una tira de ceros, ni una tira de nueves.

El lenguaje $\mathcal{L}$ así definido representa al conjunto de las racionales positivos.

2 Sintaxis de la Lógica Proposicional: nociones básicas

Definimos el alfabeto de la lógica proposicional como la unión de los siguientes conjuntos:
(1) Un conjunto infinito $\{p_0, p_1, \dots, p_n, \dots\}$ cuyos elementos se llaman *variables proposicionales*.
(2) Un paréntesis de apertura y uno de clausura: (,).
(3) Un conjunto de 4 símbolos: $\neg$, $\vee$, $\wedge$, $\Rightarrow$ (llamados conectivos), que se los denomina, respectivamente, negación, disjunción, conjunción e implicación. Los últimos tres se llaman conectivos binarios.

Definición.
Si A es el alfabeto de la lógica proposicional, una CADENA DE FORMACIÓN es una sucesión finita $x_1, x_2, \dots, x_n$ de elementos de A^* (llamados eslabones) que verifica lo siguiente: dado un índice $1 \leq i \leq n$, entonces x_i es una variable proposicional, o existe $j < i$ tal que $x_i = \neg x_j$, o existen $j, k < i$ y $* \in \{\vee, \wedge, \Rightarrow\}$ tal que $x_i = (x_j * x_k)$.

Llamaremos FÓRMULA a una expresión $\alpha \in A^*$ tal que existe una cadena de formación $x_1, x_2, \dots, x_n$ con $x_n = \alpha$.

Ejemplo.
Sea $\alpha = ((p_1 \Rightarrow p_2) \vee \neg p_3)$. Si tomamos:
$x_1 = p_1$, $x_2 = p_2$, $x_3 = \underbrace{(p_1 \Rightarrow p_2)}_{=(x_1 \Rightarrow x_2)}$, $x_4 = p_3$, $x_5 = \underbrace{\neg p_3}_{=\neg p_4}$, $x_6 = (x_3 \vee x_5)$., tenemos que $(x_i)_{i=1}^6$ es una cadena de formación de α. ▲

El lenguaje de la lógica proposicional consiste del conjunto de todas las fórmulas, y lo denotamos FORM

Observaciones
(1) Una fórmula admite en general más de una cadena de formación: en el ejemplo anterior, esto se puede ver permutando x_1 con x_2
(2) Si $x_1, \dots, x_n$ es cadena de formación, entonces x_i es fórmula para todo $1 \leq i \leq n$.
(3) Una cadena de formación puede tener eslabones "superfluos": por ejemplo, en la cadena de formación $p_1, p_2, \neg p_1$, el segundo eslabón es superfluo. (si lo sacamos, la fórmula final es la misma). Se verá más adelante el concepto de minimalidad de cadenas de formación.

La siguiente propiedad de FORM es naturalmente esperada

Proposición. El lenguaje FORM es cerrado por conectivos, es decir:
(a) Si $\alpha \in$ FORM, entonces $\neg\alpha \in$ FORM
(b) Si $\alpha, \beta \in$ FORM y $* \in \{\vee, \wedge, \Rightarrow\}$, entonces $(\alpha * \beta) \in$ FORM
Recíprocamente, si $S \subseteq$ FORM contiene a todas las variables proposicionales y es cerrado por conectivos, entonces $S =$ FORM.

Demostración.
(a) Si $x_1, \dots, x_n$ es una cadena de formación de α, entonces $x_1, \dots, x_n, \neg x_n$ es una cadena de formación de $\neg\alpha$.
(b) Si $x_1, \dots, x_n$ e $y_1, \dots, y_m$ son cadenas de formación de α y β, respectivamente, entonces $x_1, \dots x_n, y_1, \dots, y_m, (x_n * y_m)$ es cadena de formación de $(\alpha * \beta)$.

Sea $S \subseteq$ FORM tal que $p_i \in S \ \forall \ i \in \mathbb{N}$ y que es cerrado por conectivos. Sea $x \in$ FORM. Probaremos por inducción en $\ell(x)$ que $x \in S$: si $\ell(x) = 1$, entonces $x = p_i \in S$ para algún i. Supongamos ahora que $\ell(x) > 1$. Hay dos posibilidades:
- $x = \neg x_1$, con $x_1 \in$ FORM
- $x = (x_1 * x_2)$ con $x_1, x_2 \in$ FORM y $*$ un conectivo binario.

En cualquiera de los casos, x_1 y x_2 son de longitud menor que x, así que por hipótesis inductiva, están en S, que por ser cerrado por conectivos, implica que $x \in S$. □

Definición. Si $C = \{x_1, \ldots, x_n\}$ es una cadena de formación, una subcadena (de formación) de C es una subsucesión $x_{j_1}, \ldots, x_{j_k}$ (donde $1 \leq j_1 < j_2 < \cdots < j_k \leq n$) que también es cadena de formación.

Ejemplos
(1) Si $C = p_1, p_2.(p_1 \vee p_2)$, y $C' = p_1, (p_1 \vee p_2)$, tenemos que C' no es subcadena de formación de C
(2) Si $C = p_1, p_2, \neg p_1, \neg\neg p_1$, $C' = p_1, p_2, \neg p_1$ es subcadena de formación de C.

Definición. Si C es una cadena de formación de una fórmula α, diremos que C es minimal si la única subcadena de formación de C para α es ella misma.

Proposición. Si C es una cadena de formación de una fórmula α, existe una subcadena C' (de C) de formación de α que es minimal.

Demostración. Haremos inducción en n, siendo n la longitud de la cadena:
Si $n = 1$, entonces $C = x_1 = p_i$ ($i \in \mathbb{N}$), así que en este caso la minimalidad es trivial
Supongamos que el resultado es verdadero para todo $1 \leq k < n$, y veamos que vale para n: sea $C = x_1, \ldots, x_n$ una cadena de formación. Si la cadena es minimal, simplemente tomamos $C' = C$. Si no, existe una subcadena C' de formación de α, con $C' \neq C$; como $\ell(C') < \ell(C) = n$, por hipótesis inductiva, tenemos que C' admite una subcadena C'' de formación de α que es minimal (en particular, C'' es subcadena minimal de C) □

Definición. Si $\alpha \in$ FORM, la COMPLEJIDAD de α es el número de conectivos que aparecen en α (contados con repetición), y se la denota $c(\alpha)$. La complejidad binaria es el número de conectivos binarios que aparecen en α, y se lo denota $c_b(\alpha)$
Si $E \in A^*$, el PESO de E (denotamos $p(E)$) se define como el número de paréntesis de apertura menos el número de paréntesis de clausura que figuran en E.

Proposición.

a) Si $\alpha \in$ FORM, entonces $p(\alpha) = 0$ (las fórmulas tienen peso cero)

b) Si $\alpha \in$ FORM y $*$ es un conectivo binario que aparece en α, entonces $p(E) > 0$, donde E es la expresión que aparece a la izquierda de $*$ en α.

Demostración.
(a) Sea $x_1, \ldots, x_n$ una cadena de formación de α. Probaremos por inducción en i que $p(x_i) = 0 \ \forall \ 1 \leq i \leq n$. Para $i = 1$, como x_1 es variable proposicional, tiene peso nulo. Sea ahora $1 < j \leq n$.

Si x_j es variable proposicional, tiene peso nulo.

Si $x_j = \neg x_\ell$, con $\ell < j$, entonces $p(x_j) = p(x_\ell) \underbrace{=}_{HI} 0$.

Finalmente, si $x_j = (x_\ell * x_k)$, con $*$ conectivo binario y $k, \ell < j$, tenemos que

$$p(x_j) = 1 + p(x_\ell) + p(x_k) - 1 = p(x_\ell) + p(x_k) \underbrace{=}_{HI} 0$$

Luego, $p(x_i) = 0 \; \forall \; 1 \leq i \leq n$, y en particular, $p(\alpha) = 0$.

(b) Procederemos por inducción, como en el caso anterior:

Si $i = 1$, x_1 es una variable proposicional, así que cumple la propiedad trivialmente.

Sea ahora $1 < j \leq n$. Analicemos por casos:

• x_j es una variable proposicional: cumple así la propiedad de forma trivial.
• $x_j = \neg x_\ell$: si $*$ es un conectivo binario que aparece en x_j, entonces aparece en x_ℓ, de modo que $x_\ell = E * G$, con E, G expresiones. Por hipótesis inductiva, tenemos que $p(E) > 0$. La expresión a la izquierda de $*$ en x_j es $\neg E$, y $p(\neg E) = p(E) > 0$.
• $x_j = (x_\ell \circ x_k)$, con $\circ$ conectivo binario, $\ell, k < j$: Si $*$ es un conectivo binario que aparece en x_j, entonces:

-Si $* = \circ$:, la expresión E que aparece en x_j a la izquierda de $*$ es $E = (x_\ell$, cuyo peso es $p(E) = 1 + \underbrace{p(x_\ell)}_{=0} = 1 > 0$ (donde x_ℓ tiene peso nulo, por ser fórmula)
-Si $*$ aparece en x_ℓ: en tal caso, $x_\ell = E * G$. La expresión que aparece a la izquierda de $*$ es x_j es $(E$, cuyo peso es $1 + p(E)$. Por hipótesis inductiva sobre x_ℓ, $p(E) > 0$, así que $1 + p(E) > 0$.
-Si $*$ aparece en x_k: en tal caso, $x_k = F * H$, y por hipótesis inductiva, $p(F) > 0$. La expresión a la izquierda de $*$ en x_j es $(x_\ell \circ F$, cuyo peso es $1 + p(x_\ell) + p(F) = 1 + 0 + p(F) = 1 + p(F) > 0$. □

Corolario. Si $\alpha \in$ FORM, se cumple una y solo una de las siguientes condiciones:

1) α es una variable proposicional.

2) Existe una única fórmula β tal que $\alpha = \neg\beta$.

3) Existe un único conectivo binario $*$ y únicas fórmulas α_1, α_2 tal que $\alpha = (\alpha_1 * \alpha_2)$

Demostración.
Si $\alpha = \neg\beta = \neg\beta'$, con $\beta, \beta' \in$ FORM, es obvio de esta igualdad que $\beta = \beta'$.
Si $\alpha = (\alpha_1 * \alpha_2) = (\beta_1 \circ \beta_2)$, con $\alpha_1, \alpha_2, \beta_1, \beta_2 \in$ FORM, y $*, \circ$ conectivos binarios, de esta igualdad tenemos que $\alpha_1 * \alpha_2 = \beta_1 \circ \beta_2$. Usaremos el siguiente hecho: si E, F, G, H son expresiones sobre un alfabeto tales que $EF = GH$, entonces $E \leq G$ o $G \leq E$; tomemos $E = \alpha_1$, $F = *\alpha_2$, $G = \beta_1$ y $H = \circ\beta_2$ (así tenemos $EF = GH$, y entonces $\alpha_1 \leq \beta_1$ o $\beta_1 \leq \alpha_1$). Si $\alpha_1 \neq \beta_1$, entonces $\beta_1 = \alpha_1 \tilde{*} X$ o $\beta_1 = \alpha_1 \tilde{\circ} Y$, con X, Y expresiones. Por la proposición anterior, $p(\alpha_1) > 0$ o $p(\beta_1) > 0$, lo cual es absurdo, pues las fórmulas tiene peso nulo. Por consiguiente, $\alpha_1 = \beta_1$, con lo cual $*\alpha_2 = \circ\beta_2$, y así $* = \circ$ y $\alpha_2 = \beta_2$. □

Definición (SUBFÓRMULAS)**.**
Si α es una fórmula, diremos que β es una SUBFÓRMULA de α si se verifica una y solo una de las siguientes condiciones:

1) Si α es variable proposicional, β es subfórmula de α sii $\beta = \alpha$.

2) Si α es la negación de una fórmula α_1 (i.e, $\alpha = \neg\alpha_1$), β es subfórmula de α sii $\beta = \alpha$ o β es sufórmula de α_1

3) Si α es la conexión de dos fórmulas α_1, α_2 (i.e, $\alpha = (\alpha_1 * \alpha_2)$), entonces β es subfórmula de α sii $\beta = \alpha$, o es subfórmula de α_1, o es subfórmula de α_2.

Ejemplo.
Sea $\alpha = ((\neg p_1 \Rightarrow \neg\neg p_2) \Rightarrow p_4)$. Las subfórmulas de α son:

$$\alpha,\ p_1,\ p_2,\ p_4,\ \neg p_1,\ \neg p_2,\ \neg\neg p_2,\ (\neg p_1 \Rightarrow \neg\neg p_2)$$

▲

Proposición.
Sea $\alpha \in$ FORM.

1) Si β es una subfórmula de α, entonces β aparece como eslabón en toda cadena de formación de α.

2) Si $C = \{x_1, \ldots, x_n\}$ es cadena de formación minimal de α, entonces x_i es subfórmula de α para todo $1 \leq i \leq n$

Demostración.
1) Procedemos por inducción en la longitud de α: si α es variable proposicional, no hay nada que decir.
Supongamos ahora que $\ell(\alpha) > 1$, y sean $\{x_1, \ldots, x_n\}$ una cadena de formación de α, y β subfórmula de α. (supongamos además que $\beta \neq \alpha$, pues si $\beta = \alpha$, no hay nada que decir)
Si α es la negación de una fórmula, entonces existe $j < n$ tal que $\alpha = \neg x_j$; como β es subfórmula de x_j (y $x_1, \ldots, x_j$ es cadena de formación de x_j), por hipótesis inductiva, β es igual a uno de estos eslabones.
Si α es la conexión de dos fórmulas, existen $k, j < n$ y $*$ conectivo binario tales que $\alpha = (x_k * x_j)$. Como β es subfórmula de x_k o x_j (y $x_1, \ldots, x_k$ y $x_1, \ldots, x_j$ son cadenas de formación de x_k y x_j, resp), de nuevo por hipótesis inductiva concluimos $\beta = x_i$ para algún $1 \leq i \leq n$.

(2) Para el caso $n = 1$, no hay nada que decir. Asumamos ahora que $n > 1$. Procederemos por inducción, empezando desde el eslabón final de la cadena C: Como $x_n = \alpha$, x_n es trivialmente subfórmula de α. Sea $1 \leq i < n$. Existe algún $i < k \leq n$ tal que x_k depende de x_i (en caso contrario, se tendría que la cadena resultante de sacar x_i de C es cadena de formación de α, lo cual contradice su minimalidad). Tenemos así que x_i es subfórmula de x_k, que por hipótesis inductiva es subfórmula de α. Por consiguiente, x_i es subfórmula de α. □

Una consecuencia inmediata de esta proposición es que los eslabones de cualquier cadena de formación minimal de una fórmula α son las subfórmulas de α (y la longitud de una tal cadena es el número de subfórmulas de α, que denotamos $s(\alpha)$.)

3 Semántica de la Lógica Proposicional: nociones básicas

3.1 Valuaciones; Tautología, Contradicción y Contingencia; Equivalencia Semántica

Denotaremos con 2 al conjunto de dos elementos $\{0, 1\}$ (es un conjunto ordenado con el orden usual $0 < 1$). En él, están bien definidas las operaciones binarias internas de máximo y mínimo, y la operación de negación, dada por $\neg x := 1 - x$ para $x \in 2$.

Definición.
Una VALUACIÓN es una función $v : \text{FORM} \longrightarrow 2$ tal que $\forall\ \alpha, \beta \in \text{FORM}$, se cumplen las siguientes propiedades:

1) $v(\alpha \vee \beta) = \max(v(\alpha), v(\beta))$

2) $v(\alpha \wedge \beta) = \min(v(\alpha), v(\beta))$.

3) $v(\alpha \Rightarrow \beta) = \max(1 - v(\alpha), v(\beta))$.

4) $v(\neg\alpha) = 1 - v(\alpha)$.

Notación
1) Notaremos con VAR al conjunto de todas las variables proposicionales.
2) Para una fórmula α, $\text{VAR}(\alpha)$ denota el conjunto de las variables proposicionales que figuran en α.
3) Para $n \in \mathbb{N}$, FORM_n denotará el conjunto de fórmulas de complejidad menor o igual que n (observar que $\text{FORM}_0 = \text{VAR}$).

Proposición. Sea $\alpha \in \text{FORM}$ y sean v_1, v_2 valuaciones cuyas restricciones a $\text{VAR}(\alpha)$ son iguales. Entonces $v_1(\alpha) = v_2(\alpha)$

Demostración. Hacer inducción completa en la complejidad, o bien en la longitud de α (en el paso inductivo, hay que usar por supuesto que v_1, v_2 son valuaciones, y que $\text{VAR}(\neg\alpha) = \text{VAR}(\alpha)$, y $\text{VAR}(\alpha_1 * \alpha_2) = \text{VAR}(\alpha_1) \cup \text{VAR}(\alpha_2)$) □

Teorema.
Dada una función $f : \text{VAR} \longrightarrow 2$, existe una única valuación $v_f : \text{FORM} \longrightarrow 2$ que extiende a f, es decir, tal que $v_f(p_i) = f(p_i)\ \forall\ i \in \mathbb{N}$.

Demostración.
Unicidad: Es una consecuencia inmediata de la proposición anterior.
Existencia: Notemos primero que $\text{FORM} = \bigcup_{n \geq 0} \text{FORM}_n$. Construiremos inductivamente una familia de funciones $\{v_n : \text{FORM}_n \longrightarrow 2\}_{n \geq 0}$ que verifica:

i) $v_n(\alpha) = v_{n-1}(\alpha)$ si $\alpha \in \text{FORM}_{n-1}$ y $n > 0$.

ii) v_n cumple las propiedades de la definición de una valuación sobre las fórmulas de complejidad menor o igual que n.

Para $n = 0$, simplemente tomamos $v_0 = f$. Asumiendo que $n > 0$ y que ya hemos construido v_k para $0 \leq k < n$, queremos definir $v_n(\alpha)$ para $\alpha \in \text{FORM}_n$: si $\alpha \in \text{FORM}_{n-1}$, debemos definir $v_n(\alpha) = v_{n-1}(\alpha)$. Si $\alpha \notin \text{FORM}_{n-1}$, se define $v_n(\alpha)$ según α sea una negación, disyunción, conjunción o implicación, de la manera obvia.

Finalmente, definimos $v_f : \text{FORM} \longrightarrow 2$ como $v_f(\alpha) = v_n(\alpha)$ para $\alpha \in \text{FORM}_n$. □

Definición. Si $\alpha \in$ FORM, diremos que α es una TAUTOLOGÍA si $v(\alpha) = 1$ para toda valuación v. Diremos que es CONTRADICCIÓN si $\neg\alpha$ es una tautología. Diremos que es CONTINGENCIA si no es tautología ni contradicción.

Ejemplo. $(\alpha \vee \neg\alpha)$ es una tautología para toda fórmula α. ▲

El siguiente ejemplo requerirá una demostración:

Ejemplo. Sea $\gamma \in$ FORM tal que ninguna de sus variables proposicionales aparece más de una vez. Entonces γ es una contingencia. ▲

Demostración. Lo probaremos por inducción en la longitud de γ:
Si $\ell(\gamma) = 1$, entonces $\gamma = p_i$ variable proposicional, y basta tomar valuaciones u, v tal que $u(p_i) \neq v(p_i)$.
Asumamos ahora que $\ell(\gamma) > 1$, y separemos en los siguientes casos:

1) $\gamma = \neg\alpha$

2) $\gamma = (\alpha \vee \beta)$

3) $\gamma = (\alpha \wedge \beta)$

4) $\gamma = (\alpha \Rightarrow \beta)$

Caso (1): Por hipótesis inductiva, α es una contingencia; si tomamos u, v valuaciones tal que $u(\alpha) \neq v(\gamma)$, entonces también $u(\gamma) \neq v(\gamma)$.

Observar que en cualquiera de los tres casos restantes, por hipótesis sobre γ y por hipótesis inductiva, tenemos que $\text{VAR}(\alpha) \cap \text{VAR}(\beta) = \emptyset$ y α, β son contingencias. En cada caso, tomaremos $u, v, \widetilde{u}, \widetilde{v}$ valuaciones tal que $\begin{cases} u(\alpha) = 1 & , \ v(\alpha) = 0 \\ \widetilde{u}(\beta) = 1 & , \ \widetilde{v}(\beta) = 0 \end{cases}$

Caso (2): Observar que $u(\gamma) = 1$. Así, es suficiente encontrar una valuación U tal que $U(\gamma) = 0$, equivalentemente, $U(\alpha) = U(\beta) = 0$. Basta definir

$$U(p) = \begin{cases} v(p) & , \ p \in \text{VAR}(\alpha) \\ \widetilde{v}(p) & , \ p \in \text{VAR}(\beta) \\ 0 & \text{en otro caso} \end{cases}$$

Caso (3): Observar que $v(\gamma) = 0$. Así, es suficiente encontrar una valuación U tal que $U(\gamma) = 1$, es decir, $U(\alpha) = U(\beta) = 1$. Basta definir

$$U(p) = \begin{cases} u(p) & , \ p \in \text{VAR}(\alpha) \\ \widetilde{u}(p) & , \ p \in \text{VAR}(\beta) \\ 0 & \text{en otro caso} \end{cases}$$

Caso (4): Observar que $\widetilde{u}(\gamma) = 1$. Así, basta hallar una valuación U tal que $U(\gamma) = 0$, equivalentemente, $U(\alpha) = 1$ y $U(\beta) = 0$. Definimos

$$U(p) = \begin{cases} u(p) & , \ p \in \text{VAR}(\alpha) \\ \widetilde{v}(p) & , \ p \in \text{VAR}(\beta) \\ 0 & \text{en otro caso} \end{cases}$$

□

Definición (EQUIVALENCIA SEMÁNTICA)**.** Si $\alpha, \beta \in$ FORM, diremos que son (semánticamente) equivalentes si $v(\alpha) = v(\beta)$ para toda valuación v (notamos en tal caso $\alpha \equiv \beta$).

Es evidente que se trata de una relación de equivalencia en FORM. Para $\alpha \in$ FORM, denotamos $|\alpha|$ a la clase de equivalencia de α. El conjunto cociente $\text{FORM}/_{\equiv}$ se denomina Álgebra de Boole asociada a FORM (también se denomina álgebra de Lindembaum-Tarski)

Ejemplo. Si α, β son tautologías (o contradicciones), entonces $\alpha \equiv \beta$. ▲

Observación: Si $\alpha, \beta \in$ FORM, se tiene que $\alpha \equiv \beta$ si y solo si $(\alpha \Rightarrow \beta)$ y $(\beta \Rightarrow \alpha)$ son tautologías. En efecto, notando primero que $v(\alpha \Rightarrow \beta) = 1$ sii $v(\alpha) \leq v(\beta)$, tenemos que

$$\begin{aligned}\alpha \equiv \beta &\iff v(\alpha) = v(\beta)\ \forall\ v \iff v(\alpha) \leq v(\beta) \text{ y } v(\beta) \leq v(\alpha)\ \forall\ v\\ &\iff v(\alpha \Rightarrow \beta) = 1 \text{ y } v(\beta \Rightarrow \alpha) = 1\ \forall\ v\\ &\iff (\alpha \Rightarrow \beta) \text{ y } (\beta \Rightarrow \alpha) \text{ son tautologías}\end{aligned}$$

3.2 Funciones booleanas y tablas de verdad

Definición.
Si $n \geq 1$, una FUNCIÓN BOOLEANA de n-variables es una función de 2^n a 2.

Ejemplo.
Si $\alpha = (p_1 \vee p_2)$, le podemos asociar la siguiente función booleana $f : 2^n \longrightarrow 2$, que se corresponde en forma natural con la tabla de verdad de α: $f(0,0) = 0$ y $f(1,0) = f(0,1) = f(1,1) = 1$. ▲

En general, si α es una fórmula, y $\text{VAR}(\alpha) = \{p_{i_1}, \ldots, p_{i_n}\}$ (con $0 \leq i_1 < \cdots < i_n$), queremos explicitar de alguna manera la función booleana f_α que se corresponde con la tabla de verdad de α. Hacemos lo siguiente: dado $\mathbf{a} = (a_1, \ldots, a_n) \in 2^n$, sea $v_{\mathbf{a}}$ la única valuación tal que $v_{\mathbf{a}}(q) = a_j$ si $q = p_{i_j}$ para algún $1 \leq j \leq n$, y vale 0 en otro caso. Definimos $f_\alpha(\mathbf{a}) = v_{\mathbf{a}}(\alpha)$.

Dadas α, β fórmulas tal que $\text{VAR}(\alpha) = \text{VAR}(\beta)$, se tiene que $\alpha \equiv \beta$ sii $f_\alpha = f_\beta$. (esto no es más que decir que dos fórmulas con las mismas variables proposicionales son equivalentes sii tienen la misma tabla de verdad).

Dadas $q_1, \ldots, q_n$ variables proposicionales todas distintas, denotemos

$$\text{FORM}(q_1, \ldots, q_n) = \{\alpha \in \text{FORM} : \text{VAR}(\alpha) = \{q_1, \ldots, q_n\}\}$$

Acabamos de observar que la aplicación

$$\begin{aligned}\text{FORM}(q_1, \ldots, q_n)/_{\equiv} &\longrightarrow \{f : 2^n \longrightarrow 2\}\\ |\alpha| &\mapsto f_\alpha\end{aligned}$$

está bien definida y es inyectiva. Veamos que es una biyección:

Teorema.
Sean $q_1, \ldots, q_n$ variables proposicionales todas distintas y $f : 2^n \longrightarrow 2$ una función booleana. Entonces existe $\alpha \in$ FORM tal que $\text{VAR}(\alpha) = \{q_1, \ldots, q_n\}$ y $f_\alpha = f$

Demostración.
Si $f \equiv 0$, basta tomar $\alpha = \bigwedge_{i=1}^{n} (q_i \wedge \neg q_i)$.

Si no, consideremos el conjunto no vacío $T = \{\mathbf{a} \in 2^n : f(\mathbf{a}) = 1\}$.
Dado $\mathbf{a} \in 2^n$, sea $\alpha_{\mathbf{a}} := \bigwedge_{i=1}^{n} a_i q_i$, donde $a_i q_i := \begin{cases} q_i & a_i = 1 \\ \neg q_i & a_i = 0 \end{cases}$
Si k es el número de elementos de T, denotemos $T = \{\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(k)}\}$. Como para $\mathbf{a} \in 2^n$, la fórmula $\alpha_{\mathbf{a}}$ tiene valor de verdad 1 sii $q_1, \ldots, q_n$ tienen valor de verdad $a_1, \ldots, a_n$ respectivamente, vemos que basta tomar $\alpha = \bigvee_{i=1}^{k} \alpha_{\mathbf{a}^{(i)}}$. □

Queremos hacer aquí un pequeño paréntesis para definir el concepto que entró en juego en la demostración previa:

Definición.
Dado $\alpha \in$ FORM con VAR$(\alpha) = \{q_1, \ldots, q_n\}$, decimos que α está en la FORMA NORMAL DISYUNTIVA si existen $\mathbf{a}^{(1)}, \ldots, \mathbf{a}^{(m)} \in 2^n$ de manera que $\alpha = \bigvee_{i=1}^{m} (\bigwedge_{j=1}^{n} \mathbf{a}_j^{(i)} q_i)$

Del teorema y comentarios previos, deducimos $\#\left(\text{FORM}(q_1, \ldots, q_n)/_{\equiv}\right) = 2^{2^n}$.
Dado que para cada fórmula α, con VAR$(\alpha) \subseteq \{q_1, \ldots, q_n\}$, podemos tomar un representante en la clase de equivalencia semántica de α que tenga exactamente las n variables $q_1, \ldots, q_n$, deducimos el siguiente resultado:

Proposición. La cantidad de clases de equivalencia semántica en el lenguaje del cálculo proposicional con n variables proposicionales es 2^{2^n}

Ejemplo. Para $n = 1$ y una variable proposicional p, las $2^{2^1} = 4$ fórmulas no equivalentes que solo tienen la variable p son: p, $\neg p$, $(p \vee \neg p)$ y $(p \wedge \neg p)$. ▲

3.3 Conjuntos adecuados de conectivos

Definición. Un conjunto de conectivos C (esto es, un conjunto de operaciones internas 1-arias y/o binarias en FORM) se dice ADECUADO si para toda $\alpha \in$ FORM, existe $\beta \in$ FORM tal que $\alpha \equiv \beta$ y β usa solo conectivos de C

Es trivial que si C, C' son conjuntos de conectivos tales que $C \subseteq C'$ y C es adecuado, entonces C' también lo es.

Ejemplos
1) $\{\neg, \vee, \wedge, \Rightarrow\}$ es trivialmente adecuado.
2) $\{\neg, \vee\}$ es adecuado. En efecto, dadas $\alpha, \beta \in$ FORM, se tiene que

$$\alpha \Rightarrow \beta \equiv \neg\alpha \vee \beta$$
$$\alpha \wedge \beta \equiv \neg(\neg\alpha \vee \neg\beta)$$

Similarmente, $\{\neg, \wedge\}$ y $\{\neg, \Rightarrow\}$ también son adecuados.
3) Para $\alpha, \beta \in$ FORM, se define $\alpha|\beta = \neg\alpha \vee \neg\beta$ (este conectivo se conoce como NAND, por ser la negación de la conjunción). $\{|\}$ es adecuado; en efecto, basta ver que

la negación y la disyunción puede expresarse usando solo este conectivo:

$$\neg\alpha \equiv \neg\alpha \vee \neg\alpha \equiv \alpha|\alpha$$
$$\alpha \vee \beta \equiv \neg\neg\alpha \vee \neg\neg\beta \equiv \neg(\alpha|\alpha) \vee \neg(\beta|\beta) \equiv \big((\alpha|\alpha)\big|(\beta|\beta)\big)$$

Similarmente, si definimos $\alpha \downarrow \beta = \neg\alpha \wedge \neg\beta$ (conectivo llamado NOR), tenemos que $\{\downarrow\}$ también es adecuado.
4) $\{\wedge\}$, $\{\vee\}$ y $\{\Rightarrow\}$ no son adecuados. En efecto, si v es la valuación tal que $v(p_i) = 1 \; \forall \; i \in \mathbb{N}$, se tiene que $v(\alpha) = 1$ para toda fórmula α que usa uno (y solo uno) de esos conectivos, y $\neg p_1$ es una fórmula tal que $v(\neg p_1) = 0$. También es evidente que $\{\neg\}$ no es adecuado.
5) $\{\vee, \wedge, \Rightarrow\}$ no es adecuado. En efecto, si β es una fórmula en la que no aparece la negación, y v es la valuación tal que $v(p_i) = 1 \; \forall \; i \in \mathbb{N}$, se tiene que $v(\beta) = 1$ (probarlo, si se quiere, por inducción). En particular, una contradicción no es equivalente a ninguna fórmula que no use la negación.

PROBLEMA: Dado $n \in \mathbb{N}_{\geq 1}$, determinar cuáles son las funciones $f : 2^n \longrightarrow 2$ tales que existe una fórmula α tal que $f_\alpha = f$, y en α no aparece el conectivo de la negación. Calcular cuántas fórmulas no equivalentes hay con exactamente n-variables proposicionales fijas, y en las que no aparece dicho conectivo.

4 Consecuencia lógica y el Teorema de Compacidad

Definición. Si Γ es un conjunto de fórmulas de la lógica proposicional, y v es una valuación, diremos que v satisface Γ si $v(\gamma) = 1 \; \forall \; \gamma \in \Gamma$. Diremos que Γ es SATISFACIBLE si existe una valuación v que satisface Γ; en caso contrario, se dice insatisfacible.

Ejemplos
1) FORM es insatisfacible. Más generalmente, si Γ es un conjunto de fórmulas que contiene a una cierta fórmula y su negación, entonces Γ es insatisfacible.
2) $\emptyset$ es satisfacible (toda valuación lo satisface)
3) VAR es satisfacible (la valuación que toma el valor 1 en todas las variables proposicionales trivialmente satisface VAR)
4) Un conjunto finito de fórmulas $\{\alpha_1, \ldots, \alpha_n\}$ es satisfacible sii $\bigwedge_{i=1}^{n} \alpha_i$ no es una contradicción.

Definición. Si Γ es un conjunto de fórmulas, diremos que una fórmula α es consecuencia lógica de Γ si toda valuación que satisface Γ satisface α. Denotamos $C(\Gamma)$ al conjunto de fórmulas que son consecuencia lógica de Γ.

Ejemplos
1) $C(\emptyset) = \{\alpha \in \text{FORM} : \alpha \text{ es tautología}\}$ y $C(\text{FORM}) = \text{FORM}$
2) Si tenemos finitas fórmulas $\alpha_1, \ldots, \alpha_n$, entonces

$$C(\{\alpha_1, \ldots, \alpha_n\}) = C(\{\bigwedge_{i=1}^{n} \alpha_i\}) = \{\alpha \in \text{FORM} : v\Big(\bigwedge_{i=1}^{n} \alpha_i \Rightarrow \alpha\Big) = 1 \; \forall \; v\}$$
$$= \{\alpha \in \text{FORM} : \bigwedge_{i=1}^{n} \alpha_i \Rightarrow \alpha \text{ es tautología } \}$$

Proposición.
Sea $\Gamma \subseteq$ FORM y α, β fórmulas. Se tiene que

1) $\alpha \in C(\Gamma)$ si y solo si $\Gamma \cup \{\neg\alpha\}$ es insatisfacible.

2) $\beta \in C(\Gamma \cup \{\alpha\})$ si y solo si $(\alpha \Rightarrow \beta) \in C(\Gamma)$.

Demostración.
1) Efectivamente, tenemos que

$$\begin{aligned}
\alpha \in C(\Gamma) &\iff \forall\, v\,, v \text{ satisface } \Gamma \text{ implica } v(\alpha) = 1 \\
&\iff \forall\, v,\ v \text{ no satisface } \Gamma \text{ o } v(\alpha) = 1 \\
&\iff \forall\, v,\ v \text{ no satisface } \Gamma \text{ o } v(\neg\alpha) = 0 \\
&\iff \Gamma \cup \{\neg\alpha\} \text{ es insatisfacible}
\end{aligned}$$

2) Usando el ítem anterior, tenemos que

$$\begin{aligned}
\beta \in C(\Gamma \cup \{\alpha\}) &\iff \Gamma \cup \{\alpha\} \cup \{\neg\beta\} \text{ es insatisfacible} \\
&\iff \Gamma \cup \{\underbrace{\alpha \wedge \neg\beta}_{\equiv\neg(\alpha\Rightarrow\beta)}\} \text{ es insatisfacible} \\
&\iff \Gamma \cup \{\neg(\alpha \Rightarrow \beta)\} \text{ es insatisfacible} \\
&\iff (\alpha \Rightarrow \beta) \in C(\Gamma)
\end{aligned}$$

□

Notar que aplicando sucesivamente la propiedad (2) de la proposición precedente, tenemos que $\beta \in C(\{\alpha_1, \ldots, \alpha_n\})$ si y solo si $(\alpha_1 \Rightarrow (\alpha_2 \Rightarrow \ldots (\alpha_n \Rightarrow \beta)) \ldots)$ es tautología.

Diremos que un conjunto de fórmulas Γ es finitamente satisfacible (a veces abreviaremos F.S) si todo subconjunto finito de Γ es satisfacible.

Teorema (TEOREMA DE COMPACIDAD)**.**
Un conjunto de fórmulas Γ es satisfacible sii es finitamente satisfacible.

Demostración. Dado que la ida es evidente, solo hemos de probar la vuelta.

Afirmación: Dado $p \in$ VAR, se tiene que $\Gamma \cup \{p\}$ es finitamente satisfacible, o $\Gamma \cup \{\neg p\}$ es finitamente satisfacible (obviamente, solo uno de ellos lo es). En efecto, de lo contrario existen $\gamma_1, \ldots, \gamma_n, \gamma'_1, \ldots, \gamma'_m \in \Gamma$ tal que $\{\gamma_1, \ldots, \gamma_n, p\}$ y $\{\gamma'_1, \ldots, \gamma'_m, \neg p\}$ son insatisfacibles, equivalentemente, $p \wedge \underbrace{\bigwedge_{i=1}^{n} \gamma_i}_{=\alpha}$ y $\neg p \wedge \underbrace{\bigwedge_{i=1}^{m} \gamma'_i}_{=\beta}$ son contradicciones. Si v es una valuación tal que $v(\alpha) = v(\beta) = 1$ (existe, pues Γ es finitamente satisfacible), se tiene entonces que $v(p) = v(\neg p) = 0$, lo cual es absurdo.

Paso siguiente, definimos la siguiente sucesión de conjuntos:
$\Gamma_0 = \Gamma$ (es finitamente satisfacible por hipótesis)
Para $n > 0$, $\Gamma_n = \begin{cases} \Gamma_{n-1} \cup \{p_n\} \text{ si éste es F.S} \\ \Gamma_{n-1} \cup \{\neg p_n\} \text{ si éste es F.S} \end{cases}$

Tenemos que $(\Gamma_n)_{n\geq 0}$ es una sucesión creciente de conjuntos finitamente satisfacibles, y por lo tanto $\widehat{\Gamma} := \bigcup_{n=0}^{\infty} \Gamma_n$ es finitamente satisfacible. Además, por construcción se tiene que $\forall\ i \in \mathbb{N}$, $p_i \in \widehat{\Gamma}$ o $\neg p_i \in \widehat{\Gamma}$. (pero no ambos)
Se define ahora la siguiente valuación:

$$v(p_i) = \begin{cases} 1 & \text{si } p_i \in \widehat{\Gamma} \\ 0 & \text{si } \neg p_i \in \widehat{\Gamma} \end{cases}$$

Veamos que esta valuación satisface $\Gamma_0 = \Gamma$: sea $\alpha \in \Gamma$ y sean $q_1, \ldots, q_n$ las variables proposicionales que aparecen en α. Consideremos ahora el conjunto finito $\Gamma' = \{\alpha, \pm q_1, \ldots, \pm q_n\}$ donde $\pm q_i = \begin{cases} q_i & \text{si } q_i \in \widehat{\Gamma} \\ \neg q_i & \text{si } \neg q_i \in \widehat{\Gamma} \end{cases}$. Como $\widehat{\Gamma}$ es finitamente satisfacible, existe una valuación v' que satisface Γ'. Por definición de v, tenemos que $v(q_i) = v'(q_i)\ \forall\ 1 \leq i \leq n$. Como v, v' coinciden en $\text{VAR}(\alpha)$, concluimos que $v(\alpha) = v'(\alpha) = 1$, lo cual concluye la demostración. □

Evidentemente, el enunciado del teorema de compacidad es equivalente a que todo subconjunto insatisfacible de fórmulas contiene un subconjunto finito insatisfacible, que equivale a decir que $\forall\ \Gamma \subseteq \text{FORM}$ y $\forall\ \alpha \in \text{FORM}$,

$\underbrace{\Gamma \cup \{\neg\alpha\} \text{ insatisfacible}}_{\text{que es decir } \alpha \in C(\Gamma)}$ implica que $\exists\ \Gamma' \subseteq \Gamma$ finito tal que $\underbrace{\Gamma' \cup \{\neg\alpha\} \text{ es insatisfacible}}_{\text{que es decir } \alpha \in C(\Gamma')}$

Reescribiendo, el teorema de compacidad no es más que la igualdad

$$C(\Gamma) = \bigcup_{\substack{\Gamma' \subseteq \Gamma \\ \text{finito}}} C(\Gamma')$$

El teorema de compacidad se puede formular en términos topológicos:

Para cada $\varphi \in \text{FORM}$, sea $X_\varphi = \{v \in \text{VAL} : v(\varphi) = 1\}$. Es trivial ver que:
• $\text{VAL} = \bigcup_{\varphi \in \text{FORM}} X_\varphi$.
• $X_\varphi \cap X_\psi = X_{\varphi \wedge \psi}$ y $X_\varphi^c = X_{\neg\varphi}$ para todas $\varphi, \psi \in \text{FORM}$.
Luego, $(X_\varphi)_{\varphi \in \text{FORM}}$ es una base para una topología en FORM (tal que X_φ es cerrado para toda φ). Los abiertos son por definición las uniones de conjuntos X_φ. Así, con esta topología, la compacidad de VAL equivale a que todo cubrimiento por conjuntos de la base $\{X_\phi\}_{\varphi \in \text{FORM}}$ admita un subcubrimiento finito.

Tenemos pues que: VAL es compacto $\iff \forall\ \Phi \subseteq \text{FORM}$ tal que $\text{VAL} = \bigcup_{\varphi \in \Phi} X_\varphi$, existe $\Phi_0 \subseteq \Phi$ finito tal que $\text{VAL} = \bigcup_{\varphi \in \Phi_0} X_\varphi \underbrace{\iff}_{\text{tomando complemento}} \forall\ \Phi \subseteq \text{FORM}$ tal que $\varnothing = \bigcap_{\varphi \in \Phi} X_\varphi$, existe $\Phi_0 \subseteq \Phi$ finito tal que $\varnothing = \bigcap_{\varphi \in \Phi_0} X_\varphi \iff$ Todo conjunto insatisfacible de fórmulas tiene un subconjunto finito insatisfacible, que equivale al enunciado del teorema de compacidad

PROBLEMA

Dados $\Gamma, \Gamma' \subseteq$ FORM, diremos que son (semánticamente) equivalentes (y notamos $\Gamma \equiv \Gamma'$) si $\Gamma \subseteq C(\Gamma')$ y $\Gamma' \subseteq C(\Gamma)$ (o lo que es lo mismo, $C(\Gamma) = C(\Gamma')$, o también, toda valuación cumple que satisface Γ sii satisface Γ'). Considere la aplicación

$$\begin{array}{c} \mathcal{P}(\text{FORM})/_{\equiv} \xrightarrow{\Phi} \{f : 2^{\mathbb{N}} \longrightarrow 2\} \\ [\Gamma] \longmapsto f_\Gamma \end{array}$$

donde $f_\Gamma : 2^{\mathbb{N}} \longrightarrow \mathbb{N}$ está definida como $f_\Gamma(\mathbf{a}) = 1$ si y solo si $v_{\mathbf{a}}$ satisface Γ, y $v_{\mathbf{a}}$ es la valuación definida como $v_{\mathbf{a}}(p_i) = a_i$ para $i \geq 0$. Es claro que Φ está bien definida y es inyectiva. Determinar si es suryectiva.

5 Lógica de Primer Orden (sintaxis)

Definición. Un alfabeto de una lógica de primer orden consiste de los siguientes símbolos:
1) Un conjunto infinito numerable de símbolos $X_0, \ldots, X_n, \ldots$ que se denominan variables
2) Un paréntesis de apertura y uno de clausura: (,)
3) Un conjunto de 6 símbolos $\{\vee, \wedge, \neg, \Rightarrow, \forall, \exists\}$, donde los primeros cuatro son los conectivos de la lógica proposicional y los símbolos $\forall$ y $\exists$ se denominan cuantificador universal y existencial, respectivamente.
4) Un conjunto $\mathcal{C}$ (eventualmente vacío), cuyos elementos se denominan símbolos de constante.
5) Un conjunto $\mathfrak{F}$ (eventualmente vacío) cuyos elementos se denominan símbolos de función. Para $k \geq 1$, denotamos $\mathcal{F}^k$ a un símbolo de función k-ario (si hay varios símbolos de funcion k-ario, se los distingue con subíndices)
6) Un conjunto $\mathfrak{P}$ no vacío, cuyos elementos se denominan símbolos de predicado. Para $k \geq 1$, denotamos $\mathcal{P}^k$ a un símbolo de función k-ario (si hay varios, se los distingue con subíndices)

Definición. Si A es un alfabeto de una lógica de primer orden, un TÉRMINO es una expresión sobre A que se define inductivamente como sigue:
1) Las variables X_i son términos.
2) Los símbolos de constante son términos.
3) Si $\mathcal{F}^k \in \mathfrak{F}$ y $t_1, \ldots, t_k$ son términos, entonces $\mathcal{F}^k(t_1 \ldots t_k)$ es un término.

Para precisar, introducimos el concepto de cadena de formación términos: Una cadena de formación de términos es una sucesión finita $x_1, \ldots, x_n$ de expresiones sobre A que verifica que para todo $1 \leq i \leq n$, x_i es una variable o un símbolo de constante, o bien existe $\mathcal{F}^k \in \mathfrak{F}$ tal que $x_i = \mathcal{F}^k(x_{i_1} \ldots x_{i_k})$, con $i_1 < i_2 < \cdots < i_k < i$ Así, un término t es tan solo una expresión que admite una cadena de formación $x_1, \ldots, x_n$ con $x_n = t$.

Ejemplo. Sea A un alfabeto (de lógica de primer orden) que tiene un símbolo de constante c, y un símbolo de función ternario $\mathcal{F}^3$. Las siguientes expresiones son términos:

$$X_1,\ c,\ X_7,\ \mathcal{F}^3(X_1\ c\ X_7),\ \mathcal{F}^3(X_8\ X_8\ \mathcal{F}^3(X_1\ c\ X_7))$$

Observar que ésta es una cadena de formación de $\mathcal{F}^3(X_8\ X_8\ \mathcal{F}^3(X_1\ c\ X_7))$. Concretamente, si denotamos E_i al i-ésimo eslabón, se tiene que $E_5 = \mathcal{F}^3(E_4\ E_2\ E_3)$ ▲

Definición. Una FÓRMULA ATÓMICA es una expresión del tipo $\alpha = \mathcal{P}^k(t_1 \ldots t_k)$, donde $\mathcal{P}^k$ es un símbolo de predicados k-ario, y $t_1, \ldots, t_k$ representan términos.

Definición. Una CADENA DE FORMACIÓN DE FÓRMULAS es una sucesión finita $x_1, \ldots, x_n$ de expresiones sobre un alfabeto de una lógica de primer orden, que verifica: para todo $1 \leq i \leq n$, x_i es una fórmula atómica, o existe $j < i$ tal que $x_i = \neg x_j$, o existen $k, j < i$ y $* \in \{\vee, \wedge, \Rightarrow\}$ tal que $x_i = (x_j * x_k)$, o bien existe $j < i$ y una variable X_ℓ de manera que $x_i = \forall\ X_\ell\ x_j$, o $x_i = \exists\ X_\ell\ x_j$.
Una expresión α se dice FÓRMULA si existe una cadena de formación de fórmulas $x_1, \ldots, x_n$ tal que $x_n = \alpha$.

Si A es un alfabeto de primer orden, el lenguaje de primer orden sobre A es el conjunto de fórmulas sobre el alfabeto A.

Ejemplo. Consideremos el lenguaje de primer orden $\mathcal{L} =< \mathcal{F}^2, c, \mathcal{P}^2 >$.
Las siguientes expresiones son fórmulas atómicas en $\mathcal{L}$:

$$\mathcal{P}^2(X_1\ X_2),\ \mathcal{P}^2(c\ X_3),\ \mathcal{P}^2(\mathcal{F}^2(X_1\ X_2)\ X_7)$$

Las siguientes expresiones son fórmulas no atómicas:

$$\exists\ X_1\ \mathcal{P}^2(X_1\ X_2),\ \exists\ X_3\ \mathcal{P}^2(X_1\ X_2)$$

▲

Definición. Sea $\mathcal{L}$ un lenguaje de primer orden, sea α una fórmula de $\mathcal{L}$, y sea X_i una variable que aparece en α. Definimos que X_i es LIBRE en α como sigue:
1) Si α es una fórmula atómica, X_i es libre en α.
2) $\alpha = \neg\beta$, y X_i es libre en α sii es libre en β.
3) Si $\alpha = (\alpha_1 * \alpha_2)$, X_i es libre en α sii es libre en α_1 o es libre en α_2.
4) Si $\alpha = \forall\ X_j\beta$ o $\alpha = \exists\ X_j\beta$, X_i es libre en α sii $i \neq j$ y X_i es libre en β.

Diremos que X_i es LIGADA en α si no es libre en α.

Ejemplo. Dado el lenguaje de primer orden $\mathcal{L} =< \mathcal{P}^2, \mathcal{F}^2, c >$, sea

$$\alpha = (\exists\ X_1\ \exists\ X_2\ \mathcal{P}^2(X_1\ X_3) \Rightarrow \forall\ X_1 \exists\ X_2\ \mathcal{P}^2(X_2\ X_3))$$

Las variables que aparecen en α son X_1, X_2, X_3. X_1 y X_2 son ligadas, y X_3 es libre. ▲

Definición. Un ENUNCIADO (o SENTENCIA) es una fórmula tal que todas sus variables son ligadas.

Si α es una fórmula, observar que
- Si α es atómica, entonces α es una sentencia sii en α no aparecen variables.
- Si $\alpha = \neg\beta$, α es una sentencia sii β es una sentencia.
- Si $\alpha = (\alpha_1 * \alpha_2)$, entonces α es una sentencia sii α_1 y α_2 son sentencias.
- Si $\alpha = \forall\ X_j\ \beta$ o $\alpha = \exists\ X_j\ \beta$, entonces α es una sentencia sii β es una fórmula que tiene a lo sumo como variable libre a X_j.

Ejemplo. Si $\mathcal{L} =< \mathcal{F}^2, \mathcal{P}^2 >$, la siguiente fórmula es un enunciado:

$$\forall\ X_1\ \forall\ X_2\ \mathcal{P}^2(\mathcal{F}^2(X_1\ X_2), \mathcal{F}^2(X_2\ X_1))$$

▲

6 Interpretación de lenguajes de primer orden

6.1 Interpretaciones, ejemplos

Definición. Sea $\mathcal{L}$ un lenguaje de primer orden. Una interpretación I consiste de un conjunto no vacío U, que llamaremos *universo de interpretación* o *domino de interpretación*, y funciones $\mathcal{C} \longrightarrow U$, $\{\mathfrak{F}^n \longrightarrow U^{U^n}\}_{n\geq 0}$ y $\{\mathfrak{P}^n \longrightarrow \mathcal{P}(U^n)\}_{n\geq 0}$. En

otros términos:
1) Un símbolo de constante c se interpreta como un elemento $c_I \in U$
2) Un símbolo de función n-ario $\mathcal{F}^n$ se interpreta como una función $\mathcal{F}^n_I : U^n \longrightarrow U$.
3) Un símbolo de predicados n-ario $\mathcal{P}^n$ se interpreta como una relación P^n_I de n variables (un subconjunto $P^n_I \subseteq U^n$).

Ejemplo. Si $\mathcal{L} =< \mathcal{F}^2, c, \mathcal{P}^2 >$, las siguientes son interpretaciones de $\mathcal{L}$:
1) $U = \mathbb{N}$, $\mathcal{F}^2_I =$ " $+$ ", $c_I = 17$, $P^2_I =$ " $<$ "
(o alternativamente, $P^2_I = \{(x, y) \in \mathbb{N}^2 : x < y\}$)
2) $U = \mathbb{R}$, $F^2_I(x, y) = x.y$, $c_I = 0$, $P^2_I = \{(x, y) \in \mathbb{R}^2 : x^2 + y^2 = 1\}$.
3) U el universo de las personas, $F^2_I(x, y) = x$, $c_I =$ Marilyn Monroe, y
$P^2_I = \{(x, y) \in U^2 :$ la edad de x es menor que la de $y\}$ ▲

Un lenguaje con igualdad es un lenguaje en el que aparece como símbolo de predicado binario el símbolo $=$, y se usa para aclarar que se interpretará en la forma natural en las interpretaciones en consideración.

Ejemplo. Si $\mathcal{L} =< \mathcal{F}^1, =>$, dos interpretaciones de $\mathcal{L}$ son:
- $U = \mathbb{R}_{<0}$, $F^1_I(x) = x$
- $U = \mathbb{R}_{>0}$, $F^2_I(x, y) = x^3 + 1$. ▲

Ejemplo (de lo que no es intepretación)**.**
Si $\mathcal{L} =< \mathcal{F}^1, =>$, las siguientes no son interpretaciones de $\mathcal{L}$:
- $I = (\mathbb{N}, \mathcal{F}^1_I(x) = \dfrac{x}{2})$ (el codominio de $\mathcal{F}^1_I$ debe ser $\mathbb{N}$)
- $I = (\mathbb{N}, \mathcal{F}^1_I(x, y) = x + y)$ ($\mathcal{F}^1_I$ debe ser 1-aria) ▲

6.2 Valor de verdad de un enunciado; variables libres, equivalencia semántica, etc

Si $\mathcal{L}$ es un lenguaje de primer orden e I es una interpretación de $\mathcal{L}$, le asignaremos a cada término sin variables t (por ejemplo, $t = \mathcal{F}^2(c_1\ c_2)$) un elemento t_I del universo U, y a cada enunciado α un valor de verdad en $\{0, 1\}$.

Ejemplo. ¿Cómo definimos el valor de verdad de un enunciado? Consideremos el enunciado $\alpha = \exists X_2\ \forall\ X_1\ \mathcal{P}^2(X_1\ X_2)$. Con la intepretación $(U = \{1, 2, 3\}, \leq)$, α es verdadero, pero con la interpretación $(U = \mathbb{N}, \leq)$, α es falsa. ▲

Sea $\mathcal{L}$ un lenguaje de primer orden, e I una interpretación de $\mathcal{L}$. Si φ es una sentencia de $\mathcal{L}$, vamos a definir inductivamente $V_I(\varphi) \in \{0, 1\}$.
Denotemos n a la complejidad de φ (el número de conectivos que aparecen en φ).

Si $n = 0$, φ es una fórmula atómica sin variables, digamos $\varphi = P^k(t_1 \dots t_k)$, con los t_i's términos sin variables. Para poder definir $V_I(\varphi)$, vamos a extender el lenguaje $\mathcal{L}$ del siguiente modo: sea U_I el universo de interpretación de I. Para cada $a \in U_I$, sea c_a un símbolo de constante (que agregaremos al lenguaje $\mathcal{L}$). Denotemos $\widehat{\mathcal{L}} = \mathcal{L} \cup \{c_a\}_{a \in U_I}$, y el símbolo c_a se interpretará como a.
Si s es un término sin variables, sea m la complejidad de s (el número de símbolos de función que aparece en s). Definiremos inductivamente $s_I \in U_I$:
Si $m = 0$, s es un símbolo de constante c, y definimos $s_I = c_I$. Si $m > 0$, entonces $s = \mathcal{F}^k(s_1 \dots s_k)$, siendo $\mathcal{F}^k$ un símbolo de función k-ario y $s_1, \dots, s_k$ términos (sin variables y de complejidad menor que m), y definimos $s_I \in U$ como

$$s_I = \mathcal{F}^k_I(s_{1I} \dots s_{kI})$$

Teniendo esto, definimos $V_I(\varphi) = \begin{cases} 1 & \text{si } (t_{1I} \dots t_{kI}) \in \mathcal{P}_I^k \\ 0 & \text{si no} \end{cases}$

Supongamos ahora que $n > 0$:
- Si $\varphi = \neg\alpha$, se define $V_I(\varphi) = 1 - V_I(\varphi)$
- Si $\varphi = (\alpha_1 \vee \alpha_2)$, se define $V_I(\varphi) = \max(V_I(\alpha_1), V_I(\alpha_2))$
- Si $\varphi = (\alpha_1 \wedge \alpha_2)$, se define $V_I(\varphi) = \min(V_I(\alpha_1), V_I(\alpha_2))$
- Si $\varphi = (\alpha_1 \Rightarrow \alpha_2)$, se define $V_I(\varphi) = \max(1 - V_I(\alpha_1), V_I(\alpha_2))$
- Si φ comienza con un cuantificador universal o existencial, analizamos este caso a continuación:

Definición. Sea $\mathcal{L}$ un lenguaje de primer orden, sea $\alpha = \alpha(X_i)$ una fórmula, donde X_i es una variable libre de α, y sea n su complejidad. Sea t un término sin variables. Definimos inductivamente la fórmula $\alpha(^{X_i}\!/_t)$ (la sustitución por t en todas las ocurrencias libres de X_i en α) como sigue:
-Si $n = 0$, α es una fórmula atómica, digamos $\alpha = \mathcal{P}^k(t_1 \dots t_k)$, y se define

$$\alpha(^{X_i}\!/_t) = \mathcal{P}^k(t_1(^{X_i}\!/_t) \dots t_k(^{X_i}\!/_t))$$

donde $t_j(^{X_i}\!/_t)$ es el término que resulta de reemplazar X_i por t en t_j.
- Si $\alpha = \neg\beta$, se define $\alpha(^{X_i}\!/_t) = \neg\beta(^{X_i}\!/_t)$.
- Si $\alpha = (\alpha_1 * \alpha_2)$, se define $\alpha(^{X_i}\!/_t) = (\alpha_1(^{X_i}\!/_t) * \alpha_2(^{X_i}\!/_t))$.
- Si $\alpha = \exists\ X_j\ \beta$, se define $\alpha(^{X_i}\!/_t) = \exists\ X_j\ \beta(^{X_i}\!/_t)$.
- Si $\alpha = \forall\ X_j\ \beta$, se define $\alpha(^{X_i}\!/_t) = \forall\ X_j\ \beta(^{X_i}\!/_t)$.

Volviendo a nuestro asunto:
- Si $\varphi = \exists\ X_i\ \alpha$, se define $V_I(\varphi) = \sup\{V_I(\alpha(^{X_i}\!/_{c_a})) : a \in U_I\}$ (observar que $V_I(\varphi) = 1$ si y solo si $V_I(\alpha(^{X_i}\!/_{c_a})) = 1$ para algún $a \in U_I$.)
- Si $\varphi = \forall\ X_i\ \alpha$, se define $V_I(\varphi) = \inf\{V_I(\alpha(^{X_i}\!/_{c_a})) : a \in U_I\}$ (observar que $V_I(\varphi) = 1$ si y solo si $V_I(\alpha(^{X_i}\!/_{c_a}) : a \in U_I) = 1$ para cualquier $a \in U_I$.

Ejemplo.
Sea el lenguaje $\mathcal{L} =< \mathcal{P}^2, \mathcal{F}^2 >$, y la sentencia

$$\alpha = \exists\ X_1\ \exists\ X_2\ \mathcal{P}^2(X_1, \mathcal{F}^2(X_1\ X_2))$$

Consideremos la interpretación dada por

$$U_I = \mathbb{R},\ \mathcal{P}_I^2 =<,\ \mathcal{F}^2(x, y) = xy$$

La interpretación de α es $\exists\ X_1 \exists\ X_2\ X_1 < X_1 X_2$.
Sea $\widehat{\mathcal{L}} = \mathcal{L} \cup \{c_a\}_{a\in\mathbb{R}}$. Tenemos que

$$\begin{aligned} V_I(\alpha) = 1 &\iff \text{existe } a \in \mathbb{R} \text{ tal que } V_I(\exists X_2\ \mathcal{P}^2(c_a\ \mathcal{F}^2(c_a\ X_2))) = 1 \\ &\iff \text{existen } a, b \in \mathbb{R} \text{ tal que } V_I(\mathcal{P}^2(c_a\ \mathcal{F}^2(c_a\ c_b))) = 1 \\ &\iff \text{existen } a, b \in \mathbb{R} \text{ tal que } a < ab \text{ (que se cumple con } a = 1,\ b = 2) \end{aligned}$$

, de modo que $V_I(\alpha) = 1$. ▲

Ejemplo. Sea el lenguaje $\mathcal{L} =< \mathcal{P}_1^1, \mathcal{P}_2^1, =, \mathcal{F}^2 >$, y la sentencia α dada por

$$\alpha = \forall\ X_1\ (\mathcal{P}_2^1(X_1) \Rightarrow \exists\ X_2\ \exists\ X_3\ (\mathcal{P}_1^1(X_2) \wedge P_1^1(X_3) \wedge X_1 = \mathcal{F}^2(X_2\ X_3)))$$

Si consideramos la interpretación $I:$ ($\mathbb{N}$, es primo positivo , es par mayor que 2 , $+$), α se interpreta como " todo número par mayor que 2 se escribe como suma de dos primos positivos". (la famosa conjetura de Goldbach) ▲

Definición (validez de una fórmula)**.**
Si $\mathcal{L}$ es un lenguaje de primer orden, y φ es un enunciado de $\mathcal{L}$, diremos que φ es VÁLIDA en una interpretación I si $V_I(\varphi) = 1$.
φ se dice UNIVERSALMENTE VÁLIDA si $V_I(\varphi) = 1$ para toda interpretación I.

Ejemplos
1) Si φ es un enunciado, $(\varphi \vee \neg\varphi)$ es universalmente válida.
2) Sean φ_1, φ_2 fórmulas, donde el conjunto de variables libres de φ_i está contenido en $\{X\}$. Las siguientes sentencias son universalmente válidas:
- $(\forall\, X\ (\varphi_1 \wedge \varphi_2) \Rightarrow (\forall\, X\ \varphi_1 \wedge \forall\, X\ \varphi_2))$
- $((\forall\, X\ \varphi_1 \wedge \forall\, X\ \varphi_2) \Rightarrow \forall\, X(\varphi_1 \wedge \varphi_2))$.
- $(\exists\, X\ (\varphi_1 \wedge \varphi_2) \Rightarrow\ (\exists\, X\ \varphi_1 \wedge\ \exists\, X\ \varphi_2))$

3) Con φ_1, φ_2 como antes, la sentencia $((\exists\, X\ \varphi_1 \wedge\ \exists\, X\ \varphi_2) \Rightarrow\ \exists\, X\ (\varphi_1 \wedge \varphi_2))$ no es universalmente válida. En efecto, si consideramos el lenguaje $\mathcal{L} = \{\mathcal{P}^1\}$, la interpretación $I = (\mathbb{N}, x$ es par), y las fórmulas $\varphi_1 = \mathcal{P}^1(X)$ y $\varphi_2 = \neg\varphi_1$, tenemos que $V_I(\exists\, X\ (\varphi_1 \wedge \varphi_2)) = 0$ y $V_I(\exists\, X\ \varphi_1 \wedge\ \exists\, X\ \varphi_2) = 1$

Definición (equivalencia de enunciados)**.**
Sea $\mathcal{L}$ un lenguaje de primer orden y sean φ_1, φ_2 enunciados de $\mathcal{L}$. Decimos que φ_1 y φ_2 son equivalentes (y notamos en tal caso $\varphi_1 \equiv \varphi_2$) si $V_I(\varphi_1) = V_I(\varphi_2)$ para toda interpretación I de $\mathcal{L}$, o equivalentemente, $(\varphi_1 \Rightarrow \varphi_2)$ y $(\varphi_2 \Rightarrow \varphi_1)$ son universalmente válidas.

Ejemplo. Con φ_1, φ_2 como en los ejemplos previos, tenemos que

$$\forall\, X\ (\varphi_1 \wedge \varphi_2) \equiv (\forall\, X\ \varphi_1 \wedge \forall\, X\ \varphi_2)$$

▲

6.3 Modelos

Definición (Modelo)**.**
Si $\mathcal{L}$ es un lenguaje de primer orden, I es una interpretación de $\mathcal{L}$, y Γ es un conjunto de fórmulas, diremos que I es un MODELO de Γ si I satisface a γ para todo $\gamma \in \Gamma$.

Ejemplos
1) (Grupos) Dado el lenguaje $\mathcal{L} =< \mathcal{F}^2, c, =>$, consideremos los siguientes tres enunciados:

$\varphi_1 = \forall\, X_1\ \forall\, X_2\ \forall\, X_3\ \mathcal{F}^2(X_1, \mathcal{F}^2(X_2, X_3)) = \mathcal{F}^2(\mathcal{F}^2(X_1, X_2), X_3)$ (asociatividad)
$\varphi_2 = \forall\, X_1\ (\mathcal{F}^2(X_1, c) = X_1 \wedge \mathcal{F}^2(c, X_1) = X_1)$ (elemento neutro)
$\varphi_3 = \forall\, X_1\ \exists\, X_2\ (\mathcal{F}^2(X_1, X_2) = c \wedge \mathcal{F}^2(X_2, X_1) = c)$ (existencia de inverso)

Un modelo de $\{\varphi_1, \varphi_2, \varphi_3\}$ se denomina un grupo.

2) (Conjuntos con orden total denso) Dado el lenguaje $\mathcal{L} =< \mathcal{P}^2, =>$, considere el

conjunto Γ que consta de los siguientes enunciados:

$$\varphi_1 : \forall\, X\ \neg \mathcal{P}^2(X, X)$$
$$\varphi_2 : \forall\, X\ \forall\, Y\ (\mathcal{P}^2(X, Y) \Rightarrow \neg \mathcal{P}^2(Y, X))$$
$$\varphi_3 : \forall\, X\ \forall\, Y\ \forall\, Z\ ((\mathcal{P}^2(X, Y) \wedge \mathcal{P}(Y, Z)) \Rightarrow \mathcal{P}^2(X, Z))$$
$$\varphi_4 : \forall\, X\ \forall\, Y\ ((\neg X = Y) \Rightarrow (\mathcal{P}^2(X, Y) \vee \mathcal{P}^2(Y, X)))$$
$$\varphi_5 : \forall\, X\ \forall\, Y\ (\mathcal{P}^2(X, Y) \Rightarrow \exists\, Z\ (\mathcal{P}^2(X, Z) \wedge \mathcal{P}^2(Z, Y)))$$

Los modelos de Γ son los conjuntos con un orden total denso. Por ejemplo, $\mathbb{Q}$ con el orden usual es un modelo de Γ. Notar que ningún modelo finito (esto es, interpretación con universo finito) satisface Γ

Teorema (Teorema de Compacidad)**.**
Si $\mathcal{L}$ es un lenguaje de primer orden, y Γ es un conjunto de fórmulas de $\mathcal{L}$, entonces Γ tiene un modelo sii todo subconjunto finito de Γ tiene un modelo.

6.4 Isomorfismos

Definición (ISOMORFISMO DE INTERPRETACIONES)**.**
Sean $\mathcal{A}, \mathcal{B}$ dos interpretaciones de un lenguaje de primer orden $\mathcal{L}$, con universos respectivos A y B. Una función $\pi : A \longrightarrow B$ se dice un isomorfismo de $\mathcal{A}$ sobre $\mathcal{B}$ si es biyectiva y

1) $\pi(c_{\mathcal{A}}) = \pi(c_{\mathcal{B}})$ para todo símbolo de constante c.

2) Para toda símbolo de predicado n-ario $\mathcal{P}^n$ y $a_1, \ldots, a_n \in A$, se tiene $\mathcal{P}^n_{\mathcal{A}}(a_1, \ldots, a_n)$ es verdadero si y solo si $\mathcal{P}^n_{\mathcal{B}}(\pi(a_1), \ldots, \pi(a_n))$ es verdadero.

3) Para todo símbolo de función n-ario $\mathcal{F}^n$ y $a_1, \ldots, a_n \in A$,

$$\pi(\mathcal{F}^n_{\mathcal{A}}(a_1, \ldots, a_n)) = \mathcal{F}^n_{\mathcal{B}}(\pi(a_1), \ldots, \pi(a_n))$$

Cuando existe tal π, decimos que $\mathcal{A}$ y $\mathcal{B}$ son isomorfas.

Ejemplos (de estructuras isomorfas)
1) Las estructuras $(\mathbb{N}, +, 0)$ y $(2\mathbb{N}, +, 0)$ son isomorfas: la aplicación $\pi : \mathbb{N} \longrightarrow 2\mathbb{N}$ dada por $\pi(n) = 2n$ es un isomorfismo entre las mismas. (preserva la suma, es decir, $\pi(n + m) = \pi(n) + \pi(m)$, y manda el 0 en sí mismo).

2) Las estructuras $(\mathbb{Z}, <)$ y $(\mathbb{Z}, >)$ son isomorfas: la aplicación $\pi : \mathbb{Z} \longrightarrow \mathbb{Z}$ dada por $\pi(x) = -x$ es un isomorfismo.

Teorema (LEMA DE ISOMORFISMO)**.**
Si $\mathcal{A}, \mathcal{B}$ son dos interpretaciones isomorfas de un lenguaje de primer orden $\mathcal{L}$, entonces para todo enunciado φ, se tiene que

$$\varphi \text{ es verdadera en } \mathcal{A} \text{ si y solo si } \varphi \text{ es verdadera en } \mathcal{B}$$

Más aún, si π es un isomorfismo de $\mathcal{A}$ a $\mathcal{B}$, entonces para toda fórmula $\varphi(v_0, \ldots, v_{n-1})$ (donde $v_0, \ldots, v_{n-1}$ son las variables libres de φ, y $n \geq 0$), y $a_0, \ldots, a_{n-1}$ en el dominio de $\mathcal{A}$, se tiene que

$\varphi(a_0, \ldots, a_{n-1})$ es verdadera en $\mathcal{A}$ si y solo si $\varphi(\pi(a_0), \ldots, \pi(a_{n-1}))$ es verdadera en $\mathcal{B}$

Exhibimos ejemplos de estructuras no isomorfas:

Ejemplos (de estructuras no isomorfas)
1) Las interpretaciones $(\mathbb{N}, <)$ y $(\mathbb{N}, \leq)$ no son isomorfas: si $\mathcal{P}$ es el símbolo de predicado que interpretan, la sentencia $\varphi = \exists\ x \forall\ y\ \mathcal{P}(x, y)$ es falsa en $(\mathbb{N}, <)$ pero verdadera en $(\mathbb{N}, \leq)$.
De un modo ingenuo: si tuviera una función $\pi : (\mathbb{N}, \leq) \longrightarrow (\mathbb{N}, <)$ que preserva la relación, en particular $\pi(1) < \pi(1)$, lo cual es absurdo.

2) Las interpretaciones $(\mathbb{Z}, <)$ y $(\mathbb{N}, <)$ no son isomorfas: si φ es el enunciado "existe un mínimo", que podemos escribir como

$$\varphi = \exists\ x\ \forall\ y\ (\underbrace{((x < y) \vee (y < x))}_{\text{una forma de escribir } x \neq y} \Rightarrow x < y)$$

, el mismo es verdadera en $(\mathbb{N}, <)$ y falso en $(\mathbb{Z}, <)$.

3) Las intepretaciones $(\mathbb{N}, +)$ y $(\mathbb{N}, \cdot)$ no son isomorfas: si $\mathcal{F}$ es el símbolo de función que interpretan, la sentencia $\varphi = \exists x \forall y (\mathcal{F}(x, y) = x)$ ("existe un elemento absorbente") es verdadera en $(\mathbb{N}, \cdot)$ y falsa en $(\mathbb{N}, +)$.
De un modo ingenuo: si $\pi : \mathbb{N} \longrightarrow \mathbb{N}$ es una función tal que $\pi(nm) = \pi(n) + \pi(m)\ \forall\ n, m \in \mathbb{N}$, entonces $\pi(0) = \pi(0.0) = \pi(0) + \pi(0)$, con lo cual $\pi(0) = 0$, y entonces, para todo $n \in \mathbb{N}$, tenemos que

$$0 = \pi(0) = \pi(0.n) = \underbrace{\pi(0)}_{=0} + \pi(n) = \pi(n)$$

así que $\pi \equiv 0$ (en particular, no es biyectiva)

6.5 Conjuntos definibles

Definición (conjuntos definibles)**.**
Sea $\mathcal{L}$ un lenguaje de primer orden, sea $\varphi(X)$ una fórmula con una única variable libre X, e I una interpretación de $\mathcal{L}$ con universo U_I. Dado $A \subseteq U_I$, diremos que A es DEFINIBLE (también se dice DISTINGUIBLE o EXPRESABLE) si existe una fórmula φ con una única variable libre X tal que $A = \{a \in U_I : \varphi(^{X}/_{c_a})$ es válida en $I\}$, y diremos en tal caso que φ define o expresa a A.
Un elemento $u \in U_I$ se dirá DISTINGUIBLE si $\{u\}$ es distinguible.

Ejemplos

1) Sea $\mathcal{L} =<=>$, y la interpretación con universo U_I, un conjunto arbitrario. Considere la fórmula $\varphi(X) : X = X$. Es claro que φ expresa a U_I, y $\neg\varphi$ expresa a $\emptyset$.

2) Sea $\mathcal{L} =< \mathcal{F}^1, c, =>$ y la interpretación $I :\ (\mathbb{N}, \text{suc}, 0)$, donde suc $: \mathbb{N} \longrightarrow \mathbb{N}$

es la función sucesor ($\text{suc}(x) = x + 1$). Dado $n \in \mathbb{N}$, podemos escribirlo como $n = \text{suc}^n(0)$. Consideremos la fórmula

$$\varphi_n(X) : \ (\mathcal{F}^1)^n(c) = X$$

(donde es claro que se entiende por $(\mathcal{F}^1)^n(c)$). La fórmula φ_n expresa a n (así que n es distinguible para todo $n \in \mathbb{N}$ bajo esta interpretación)

Observaciones
Sea $\mathcal{L}$ un lenguaje de primer orden e I una interpretación de $\mathcal{L}$ con universo U_I. Entonces:
• Si $A_1, \ldots, A_n \subseteq U_I$ son definibles (en $\mathcal{L}$), entonces $\bigcup_{i=1}^n A_i$ y $\bigcap_{i=1}^n A_i$ son definibles.
• Si $A \subseteq U_I$ es definible, entonces A^c es definible.
• Si $\pi : U_I \longrightarrow U_I$ es un isomorfismo (de la interpretación I sobre I), y $A \subseteq U_I$ es definible, entonces $f(A) \subseteq A$: efecto, si $\varphi(X)$ es una fórmula que expresa a A, y $a \in A$, como $\varphi(a)$ es verdadero (en I) y π es iso, se tiene que $\varphi(f(a))$ es verdadero, esto es, $f(a) \in A$.

7 Árboles

Definición. Un ÁRBOL A es un conjunto parcialmente ordenado con primer elemento (llamado RAÍZ del árbol) tal que $\forall\ x \in A$, el conjunto $\{y \in A : y \leq x\}$ es finito y totalmente ordenado.
Un elemento $x_0 \in A$ se dice NODO TERMINAL si x_0 es maximal en A.

Definición (extensión inmediata de un árbol)**.**
Si A es un árbol de un enunciado φ, diremos que un árbol A' es de EXTENSIÓN INMEDIATA de A, si se obtiene agregando a un nodo terminal de A una fórmula del siguiente tipo:
1) Si la fórmula que corresponde a dicho nodo es del tipo $\neg\neg\alpha$, agregamos α.
2) Si es del tipo $(\alpha \wedge \beta)$, $\underbrace{\neg(\alpha \vee \beta)}_{\equiv \neg\alpha \wedge \neg\beta}$, $\underbrace{\neg(\alpha \Rightarrow \beta)}_{\equiv \alpha \wedge \neg\beta}$, agregamos respectivamente α o β, $\neg\alpha$ o $\neg\beta$, α o $\neg\beta$.
3) Si es del tipo $(\alpha \vee \beta)$, $\underbrace{\neg(\alpha \wedge \beta)}_{\equiv \neg\alpha \vee \neg\beta}$, $\underbrace{(\alpha \Rightarrow \beta)}_{\equiv \neg\alpha \vee \beta}$, agregamos respectivamente α y β, $\neg\alpha$ y $\neg\beta$, $\neg\alpha$ y β.
4) Si es del tipo $\exists\ X\ \alpha$ o $\neg\ \forall\ X\ \alpha$, agregamos respectivamente $\alpha(^X\!/_c)$, $\neg\alpha(^X\!/_c)$, con c un símbolo de constante de $\mathcal{L}_{\text{par}}$ que no aparece en A.
5) Si es del tipo $\forall\ X\ \alpha$ o $\neg\exists\ X\ \alpha$, agregamos respectivamente $\alpha(^X\!/_t)$, $\neg\alpha(^X\!/_t)$, con t un término sin variables.

Definición (Árbol de refutación)**.**
Un árbol de refutación A de un enunciado φ es un árbol que se obtiene a partir de A en un número finito de pasos, mediante la construcción de extensiones inmediatas, es decir, existe árboles $A_1, \ldots, A_n$ tales que $A_1 = \{\varphi\}$, y A_{i+1} es extensión inmediata de A_i para todo $1 \leq i \leq n-1$.
Un árbol de refutación de un conjunto finito de enunciados $\{\varphi_1, \ldots, \varphi_n\}$ es un árbol de refutación de $\varphi = \bigwedge_{i=1}^{n} \varphi_i$.

Definición (Conjuntos saturados de enunciados)**.**
Si H es un conjunto de enunciados de un lenguaje de primer orden (esto es, una teoría), H se dice SATURADO si verifica las siguientes propiedades:
1) Si $\varphi \in H$, entonces $\neg\varphi \notin H$.
2) Si $\neg\neg\varphi \in H$, entonces $\varphi \in H$
3) Si $(\alpha \vee \beta) \in H$, entonces $\alpha \in H$ o $\beta \in H$
4) Si $(\alpha \wedge \beta) \in H$, entonces $\alpha, \beta \in H$.
5) Si $(\alpha \Rightarrow \beta) \in H$, entonces $\neg\alpha \in H$ o $\beta \in H$.
6) Si $\neg(\alpha \Rightarrow \beta) \in H$, entonces $\alpha, \neg\beta \in H$.
7) Si $\neg(\alpha \vee \beta) \in H$, entonces $\neg\alpha, \neg\beta \in H$
8) Si $\neg(\alpha \wedge \beta) \in H$, entonces $\neg\alpha \in H$ o $\neg\beta \in H$.
9) Si $\exists\, X\ \varphi \in H$, entonces $\varphi(^{X}/_{c}) \in H$, con c un símbolo de constante.
10) Si $\forall\, X\ \varphi \in H$, entonces $\varphi(^{X}/_{t}) \in H$ para todo término sin variables t.
11) Si $\neg\forall\, X\ \varphi \in H$, entonces $\neg\ \varphi(^{X}/_{c}) \in H$, siendo c un símbolo de constante.
12) Si $\neg\exists\, X\ \varphi \in H$, entonces $\neg\varphi(^{X}/_{t}) \in H$ para todo término sin variables t.

Es claro qué se entiende por conjunto saturado de fórmulas de la lógica proposicional (mirar solo las condiciones (1) a (8) en la definición anterior)

Ejemplo.
El conjunto $H = \{\neg\neg p_2, (p_1 \Rightarrow p_3), p_2, p_3\}$ es saturado. Por otra parte, claramente es satisfacible. ▲

Proposición.
Sea $S \subseteq$ FORM saturado. Entonces S es satisfacible.

Demostración.
Sea v : FORM $\longrightarrow 2$ la siguiente valuación: para una variable proposicional p,

$$v(p) := \begin{cases} 1 & \text{si } p \in S \\ 0 & \text{si } \neg p \in S \\ 0 & \text{si } p \notin S \text{ y } \neg p \notin S \end{cases}$$

Está bien definida porque S es saturado. Probaremos por inducción que v satisface S.
Sea $\alpha \in S$. Si α es una variable proposicional o la negación de una variable proposicional, por definición de v tenemos que $v(\alpha) = 1$.
Si no, α es de alguno de los siguientes tipos:
1) $\alpha = \neg\neg\beta$: como S es saturado, $\beta \in S$, así que por hipótesis inductiva $v(\beta) = 1$, y por ende $v(\alpha) = 1$
2) α es una conjunción, disyunción, implicación, o una negación de alguno de estos tipos: usando la hipótesis inductiva, se desprende inmediatamente que $v(\alpha) = 1$ en cualquiera de estos casos. □

A continuación enunciamos y probamos el resultado análogo para conjuntos saturados de enunciados:

Teorema.
Si $\mathcal{L}$ es un lenguaje de primer orden, y S es un conjunto saturado de enunciados de $\mathcal{L}$, entonces S es satisfacible

Demostración.
Sea I la siguiente interpretación (llamada *interpretación de Herbrand* asociada a S):

- U_I es el conjunto de términos sin variables de $\mathcal{L}$.
- Si c es un símbolo de constante, $c_I = c$.
- Si $\mathcal{F}^k$ es un símbolo de función k-ario, $\mathcal{F}^k_I$ se define como

$$\mathcal{F}^k_I(t_1, \ldots, t_k) = \mathcal{F}^k(t_1 \ldots t_k)$$

(de manera que en general, $t_I = t$ para todo término t sin variables)
- Si $\mathcal{P}^k$ es un símbolo de predicado k-ario, se define

$$\mathcal{P}^k_I = \{(t_1, \ldots, t_k) \in U^k_I : \mathcal{P}^k(t_1 \ldots t_k) \in S\}$$

Tomemos $\alpha \in S$ y probemos que $V_I(\alpha) = 1$

1) Si α es fórmula atómica, digamos $\alpha = \mathcal{P}^k(t_1 \ldots t_k)$, con $t_1, \ldots, t_k$ términos sin variables, tenemos que $V_I(\alpha) = 1$ si y solo si $(t_{1I}, \ldots, t_{kI}) \in \mathcal{P}^k_I$ si y solo si $\alpha = \mathcal{P}^k(t_1 \ldots t_k) \in S$, que es la hipótesis.
2) Si $\alpha = \exists\ X\ \beta$, entonces $\beta(^X\!/_c) \in S$ para algún símbolo de constante c. Por hipótesis inductiva, $V_I(\beta(^X\!/_c)) = 1$, y luego $V_I(\exists\ X\ \beta) = 1$.
3) Si $\alpha = \forall\ X\ \beta$, entonces $\beta(^X\!/_t) \in S$ para todo término sin variables t. Por hipótesis inductiva, se tiene que $V_I(\beta(^X\!/_t)) = 1$ para todo término sin variables t, y entonces $V_I(\alpha) = 1$.
4) Si $\alpha = (\alpha_1 \vee \alpha_2)$, por ser S saturado, tenemos que $\alpha_1 \in S$ o $\alpha_2 \in S$, así que por hipótesis inductiva, uno de ellos tiene valor de verdad 1, de modo que $V_I(\alpha) = 1$.
5) Si $\alpha = (\alpha_1 \wedge \alpha_2)$, por ser S saturado tenemos que $\alpha_1, \alpha_2 \in S$, así que por hipótesis inductiva $V_I(\alpha_i) = 1$, $i = 1, 2$, y luego $V_I(\alpha) = 1$.
6) Supongamos ahora que $\alpha = \neg\alpha_1$. Como S es saturado, tenemos que $\alpha_1 \notin S$
Si α_1 es fórmula atómica, digamos $\alpha_1 = \mathcal{P}^k(t_1 \ldots t_k) \notin S$, por definición de la interpretación del predicado se tiene que $V_I(\alpha_1) = 0$, y luego $V_I(\alpha) = 1$.
Si α_1 comienza con el cuantificador universal o existencial, entonces $\alpha = \neg\alpha_1$ es equivalente a un enunciado que comienza con el cuantificador existencial o universal respectivamente (y de igual complejidad que α), así que nos reducimos a los casos **(2)** y **(3)**.
Si $\alpha_1 = \neg\beta$, entonces $\alpha = \neg\neg\beta$, de modo que por saturación de S se tiene que $\beta \in S$. Por hipótesis inductiva, $V_I(\beta) = 1 = V_I(\neg\neg\beta) = V_I(\alpha)$.
Si $\alpha_1 = (\beta_1 \vee \beta_2)$, entonces $\alpha = \neg(\beta_1 \vee \beta_2)$, de modo que $\neg\beta_1, \neg\beta_2 \in S$. Por hipótesis inductiva, $V_I(\neg\beta_i) = 1$ ($i = 1, 2$), implicando $V_I(\alpha) = 1$.
El caso en que α_1 es una conjunción o una implicación se resuelve análogo al anterior. En definitiva, probamos que si $\alpha \in S$ es la negación de un enunciado, entonces es satisfacible.
7) Si $\alpha = (\alpha_1 \Rightarrow \alpha_2)$, entonces $\neg\alpha_1 \in S$ o $\alpha_2 \in S$. En el segundo caso, la hipótesis inductiva ya concluye. En el primer caso, observar que $\beta := \neg\alpha_1$ es un enunciado de complejidad menor o igual que α, así que se aplica el caso **(6)** para concluir que $V_I(\neg\alpha_1) = 1$, implicando $V_I(\alpha) = 1$. □

Corolario.
Si α es un enunciado, o una fórmula de la lógica proposicional, y A es un árbol de α que tiene una rama saturada, entonces α es satisfacible.

Definición.
Dado un árbol A de un enunciado o de un elemento de FORM, una rama R de A se dice CERRADA si existe $\varphi \in A$ tal que $\varphi \in R$ y $\neg\varphi \in R$.
El árbol A se dice COMPLETO si todas sus ramas son saturadas o cerradas.

Observación:
Si α es un enunciado o una fórmula de la lógica proposicional que admite un árbol cerrado, entonces α es universalmente inválida (respectivamente, α es contradicción)

Proposición.
Si α es una fórmula de la lógica proposicional, entonces α admite un árbol de refutación completo.

Demostración.
Inducción en $\ell_m(\alpha) := c(\alpha) + \#\text{VAR}R(\alpha))$ (llamada longitud modificada de α) $\square$

Como consecuencia de los resultados previos, tenemos el siguiente corolario:

Corolario.
Sea $S \subseteq$ FORM finito. Son equivalentes:
(1) S es satisfacible
(2) Existe un árbol de refutación completo de S con una rama abierta (o sea, una rama saturada, por ser completo).
(3) Todo árbol de refutación de S admite una rama abierta.

8 Lenguaje de programación $\mathscr{S}$

El lenguaje de programación $\mathscr{S}$ posee

$$\begin{cases} \bullet \text{ variables de entrada} \\ \bullet \text{ variables } \textit{locales} \\ \bullet \text{ 1 variable de salida que usualmente notamos con la letra } Y \end{cases}$$

Todas las variables son números enteros no negativos.
Todas las variables no ingresadas tiene valor inicial 0.
No hay límite en cuanto a la cantidad de variables a ingresar.
La sintaxis del lenguaje $\mathscr{S}$, además de variables, posee *etiquetas*.
Un programa en $\mathscr{S}$ consiste en una lista (sucesión finita de instrucciones).
En $\mathscr{S}$, tenemos permitidas las siguientes 3 tipos de instrucciones, que pueden o no estar etiquetadas:

- $V \leftarrow V + 1$ (incrementar en 1 la variable V)
- $V \leftarrow \begin{cases} V - 1 & \text{si } V \neq 0 \\ 0 & \text{si } V = 0 \end{cases}$ (resta truncada por 1, que usualmente notaremos $V \leftarrow V - 1$)
- if $V \neq 0$, go to [L], donde [L] es una etiqueta

Más adelante, agregaremos la instrucción $V \leftarrow V$.
Usualmente, notamos [E] (exit) al salir del programa.

Definición (función parcial)**.**
Una FUNCIÓN PARCIAL f sobre un conjunto A es una función tal que dom$(f) \subseteq A$.
Si f es una función parcial sobre un conjunto A y $a \in A$, entonces:
- Si $a \in$ dom(f), decimos que $f(a)$ está definida y notamos $f(a)\downarrow$
- Si $a \notin$ dom(f), decimos que $f(a)$ está indefinida, y notamos $f(a)\uparrow$.
Si dom$(f) = A$, decimos que f es una función total.

Definición.
Decimos que un programa computa a la función parcial $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ si dadas las entradas $(a_1, \ldots, a_k) \in$ dom(f), el programa devuelve $f(a_1, \ldots, a_k)$, y si $(a_1, \ldots, a_k) \notin$ dom(f), el programa no está definido con esas entradas.

Ejemplos
1) El siguiente programa computa a la función constantemente igual a n:

$$n \text{ veces } \begin{cases} Y \leftarrow Y + 1 \\ Y \leftarrow Y + 1 \\ \vdots \\ Y \leftarrow Y + 1 \end{cases}$$

2) Considere el siguiente programa:

```
if X ≠ 0, go to [A]
      go to [E]
[A]   Y ← Y + 1
      X ← X − 1
      if X ≠ 0, go to [A]
```

Este programa computa a la función $id : \mathbb{N} \longrightarrow \mathbb{N}$.

En el ejemplo anterior, hemos hecho uso de la pseudoinstrucción go to [E]. La misma se denomina una *macro*. Una macro es sencillamente una pseudoinstrucción abreviada. La parte del programa que se abrevia se denomina *expansión de la macro*.

Ejemplos de macros
1) (Salto incondicional go to [L])
Una posible expansión de go to [L] es como sigue:

```
V ← V + 1
if V ≠ 0, go to [L]
```

(donde V es una variable no usada anteriormente)

2) ($Z \leftarrow 0$: asignar el valor 0 a la variable Z):
Una posible expansión de esta macro es como sigue:

```
[A] Z ← Z − 1
    if Z ≠ 0, go to [A]
```

3) ($Z \leftarrow X$: copiar el valor de X en Z) Una posible expansión de esta macro es la siguiente:

```
Z ← 0
if X ≠ 0, go to [A]
  go to [E]
[A] Z ← Z + 1
    X ← X − 1
    if X ≠ 0, go to [A]
```

La desventaja de esta macro es que no mantiene el valor de X. A continuación, exhibimos una macro de $Z \leftarrow X$ que mantiene el valor de X:

```
Z ← 0
W ← 0
if X ≠ 0, go to [A]
  go to [E]
[A] Z ← Z + 1
    W ← W + 1
    X ← X − 1
    if X ≠ 0, go to [A]
[B] W ← W − 1
    X ← X + 1
    if W ≠ 0, go to [B]
```

5) ($Z \leftarrow X_1 + X_2$: sumar dos variables) Podemos considerar la siguiente expansión:

```
Z ← X1
if X2 ≠ 0, go to [A]
  go to [E]
[A] Z ← Z + 1
    X2 ← X2 − 1
    if X2 ≠ 0, go to [A]
```

Ejercicio

Computar la función parcial $f : \mathbb{N}^2 \longrightarrow \mathbb{N}$ dada por $f(x_1, x_2) = \begin{cases} x_1 - x_2 & \text{si } x_2 \leq x_1 \\ \uparrow & \text{si no} \end{cases}$

Solución

El siguiente programa en $\mathscr{S}$ computa f:

```
Y ←X1
Z ←X2
if Z ≠ 0, go to [A]
  go to [E]

[A] if Z ≠ 0, go to [B]
    go to [E]

[B] if Y ≠ 0, go to [C]
    go to [B]

[C] Y ← Y − 1
    Z ← Z − 1
    go to [A]
```

Proposición.

Sea $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ una función de k variables, $g_1, g_2, \ldots, g_k : \mathbb{N}^n \longrightarrow \mathbb{N}$ de n variables, y sea $h : \mathbb{N}^n \longrightarrow \mathbb{N}$ dada por $h(x) = f(g_1(x), \ldots, g_k(x))$.
Si $f, g_1, \ldots, g_k$ son (parcialmente) computables, entonces h es (parcialmente) computable.

Demostración.

El siguiente programa computa a h:

$$
\begin{aligned}
&Z_1 \leftarrow g_1(X_1, \ldots, X_n) \\
&Z_2 \leftarrow g_2(X_1, \ldots, X_n) \\
&\vdots \\
&Z_k \leftarrow g_k(X_1, \ldots, X_n) \\
&Y \leftarrow f(Z_1, \ldots, Z_k)
\end{aligned}
$$

□

9 Recursión, funciones recursivas primitivas, etc

Definición.
Dado $k \in \mathbb{N}$ y $H : \mathbb{N} \times \mathbb{N} \longrightarrow$ una función, decimos que la función $f : \mathbb{N} \longrightarrow \mathbb{N}$ se obtiene por recursión a partir de k y H si se verifica:
(i) $f(0) = k$
(ii) $f(n+1) = H(n, f(n))$

Ejemplo.
Sea la función $a : \mathbb{N} \longrightarrow \mathbb{N}$ dada por $\begin{cases} a_0 = 3 \\ a_{n+1} = 3a_n + 1 \end{cases}$
En este ejemplo, si llamamos $k = 3$ y $H(x, y) = 3y + 1$, tenemos que a se obtiene por recursión a partir de k y H. ▲

Ejemplo.
Sea $k = 0$ y $H(x, y) = x + y$. Calculemos la función f definida por recursión a partir de k y H:

$$f(n+1) = H(n, f(n)) = n + f(n) = n + (n-1) + f(n-1) + \cdots + 0 + \underbrace{f(0)}_{=0}$$

$$= \sum_{k=0}^{n} k = \frac{n(n+1)}{2}$$

Luego, $f(n) = \dfrac{n(n-1)}{2} \ \forall\, n \in \mathbb{N}$ ▲

Proposición.
Si $H : \mathbb{N} \times \mathbb{N}$ es una función computable, entonces una función f definida por recursión a partir de un $k \in \mathbb{N}$ y H es computable.

Demostración.
El siguiente programa en el lenguaje $\mathscr{S}$ computa a f:

```
     Y ← k
[C]  if X ≠ 0, go to [A]
     go to [E]
[A]  Y ← H(Z,Y)
     Z ← Z + 1
     X ← X − 1
     go to [C]
```

□

De forma análoga, se define una función de k variables por recursión:

Definición.
Sean $g : \mathbb{N}^k \longrightarrow \mathbb{N}$ y $H : \mathbb{N}^{k+2} \longrightarrow \mathbb{N}$ (con $k \geq 0$). Decimos que una función $f : \mathbb{N}^{k+1} \longrightarrow \mathbb{N}$ se obtiene por recursión a partir de g y H si se verifica:
(i) $f(x, 0) = g(x) \ \forall\, x \in \mathbb{N}^k$
(ii) $f(x, n+1) = H(x, n, f(x, n)) \ \forall\, x \in \mathbb{N}^k$, $n \in \mathbb{N}$.

Proposición.
Si $g : \mathbb{N}^k \longrightarrow \mathbb{N}$ y $H : \mathbb{N}^{k+2} \longrightarrow \mathbb{N}$ son funciones computables, entonces la f obtenida por recursión a partir de g y H es computable.

Demostración. La prueba es similar a la anterior. □

Ejemplo.
Consideremos la función suma $S(x_1, x_2) = x_1 + x_2$. Tenemos que

$$S(x_1, 0) = x_1 \text{ (i.e, } S(-, 0) = id_{\mathbb{N}}\text{, que es computable)}$$
$$S(x_1, x_2 + 1) = x_1 + (x_2 + 1) = (x_1 + x_2) + 1 = S(x_1, x_2) + 1 = H(x_1, x_2, S(x_1, x_2))$$

con $H(x_1, x_2, x_3) = x_3 + 1$ (computable).
Luego, S es computable. ▲

Definición (Funciones iniciales)**.**
Llamaremos FUNCIÓN INICIAL a una función que corresponde a uno de los siguientes tipos:

(1) cero$(x) = 0 \ \forall \ x$

(2) suc$(x) = x + 1$ (la función sucesor)

(3) Proyecciones: si $k \in \mathbb{N}$ y $0 \leq n \leq k$, se define $\Pi^n_k(x_1, \ldots, x_k) = x_n$.

Definición (Funciones recursivas primitivas)**.**
Una función $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ se dice RECURSIVA PRIMITIVA si se obtiene en un número finito de pasos a partir de las funciones iniciales usando las operaciones de composición y/o recursión.

Observación
Toda función recursiva primitiva es computable.

Ejemplos
1) En el anterior ejemplo, $H(x_1, x_2, x_3) = \text{suc}(\Pi^3_3(x_1, x_2, x_3))$, de manera que la función suma S es recursiva primitiva.

2) La función producto prod$(x_1, x_2) = x_1x_2$ es recursiva primitiva. En efecto, tenemos que

$$\begin{aligned} \text{prod}(-, 0) &= 0 \\ \text{prod}(x_1, x_2 + 1) &= \text{prod}(x_1, x_2) + x_1 \\ &= H(x_1, x_2, \text{prod}(x_1, x_2)) \text{ , con } H = \text{suma}(\Pi^1_3, \Pi^3_3) \end{aligned}$$

3) Sea la función $\text{pot}(x_1, x_2) = \begin{cases} 0 & \text{si } x_2 = x_1 = 0 \\ x_2^{x_1} & \text{en caso contrario} \end{cases}$.

Tenemos que $\text{pot}(0, x_2) = \begin{cases} 0 & x_2 = 0 \\ 1 & x_2 \neq 0 \end{cases}$, que es recursiva primitiva como función de x_2, dado que se obtiene por recursión a partir de 0 y 1.
Tenemos que

$$\text{pot}(x_1 + 1, x_2) = x_2^{x_1+1} = x_2^{x_1}x_2 = \text{prod}(\text{pot}(x_1, x_2), x_2) = H(x_1, x_2, \text{pot}(x_1, x_2))$$

donde $H(x_1, x_2, x_3) = \text{prod}(\Pi^3_3(x_1, x_2, x_3), \Pi^2_3(x_1, x_2, x_3))$.
Luego, la función pot es recursiva primitiva.

4) Sea la función $f(n) = \underbrace{n^{n^{\cdot^{\cdot^{\cdot^{n}}}}}}_{n \text{ veces}}$. Así, por ejemplo

$$f(1) = 1,\ f(2) = 2^2 = 4,\ f(3) = 3^{3^3} = 3^{27}$$

Llamemos $G(n, m) = \underbrace{n^{n^{\cdot^{\cdot^{\cdot^{n}}}}}}_{m \text{ veces}}$. Tenemos que $G(n, 0) = 1$ y

$$G(n, m+1) = n^{G(n,m)} = \text{pot}(G(n, m), n) = H(n, m, G(n, m))$$

con $H(x, y, z) = \text{pot}(z, x)$. Luego, G es recursiva primitiva, y como $f(n) = G(n, n)$, concluimos que f es recursiva primitiva.

5) Sea la función predecesor definida por $\text{pred}(n) = \begin{cases} 0 & n = 0 \\ n - 1 & n > 0 \end{cases}$.
Tenemos que $\text{pred}(0) = 0$ y

$$\text{pred}(n+1) = n = H(n, \text{pred}(n)) \text{ con } H(x, y) = x$$

Luego, pred es recursiva primitiva.

6) Sea la resta truncada, definida por $R(m, n) = \begin{cases} m - n & m \geq n \\ 0 & m < n \end{cases}$.
Tenemos que $R(-, 0) = id_{\mathbb{N}}$ y

$$\begin{aligned} R(m, n+1) &= \begin{cases} (m-n) - 1 & m - n > 0 \\ 0 & m - n \leq 0 \end{cases} \\ &= \text{pred}(R(m, n)) \\ &= H(m, R(m, n)) \end{aligned}$$

con $H(x, y) = \text{pred}(y)$. Luego, la resta truncada es recursiva primitiva.
Usualmente, denotaremos $R(x, y) = x \dot{-} y$.

7) Para $x, y \in \mathbb{N}$, tenemos $d(x, y) := |x - y| = x \dot{-} y + y \dot{-} x$. Luego, d es recursiva primitiva.

8) Sea la función $\text{eq}(x, y) = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases}$. Tenemos que:

$$\text{eq}(x, y) = 1 \iff |x - y| = 0 \iff 1 \dot{-} |x - y| = 1$$

, esto es, $\text{eq}(x, y) = 1 \dot{-} d(x, y)$. Luego, eq es recursiva primitiva.

9) La función $\text{menor}(x, y) = \begin{cases} 1 & x < y \\ 0 & x \geq y \end{cases}$ es recursiva primitiva. En efecto, tenemos que

$$\text{menor}(x, y) = 1 \iff x < y \iff y \dot{-} x \neq 0 \iff \text{eq}(y \dot{-} x, 0) = 0$$

, esto es, $\text{menor}(x, y) = 1 \dot{-} \text{eq}(y \dot{-} x, 0)$.

Definición.
Si $P \subseteq \mathbb{N}^n$ (un predicado de n-variables), diremos que P es computable (respectivamente, recursivo primitivo) si su función característica $C_P : \mathbb{N}^n \longrightarrow \mathbb{N}$ es computable (resp, recursiva primitiva)

Proposición.
Si P, Q son dos predicados de n-variables, se verifican las siguientes condiciones:
(1) Si P, Q son computables, entonces $(P \vee Q)$, $(P \wedge Q)$ y $(P \Rightarrow Q)$ son computables.
(2) Si P es computable, entonces $\neg P$ es computable.
(3) Los ítems anteriores siguien valiendo si los predicados son recursivos primitivos.

Demostración.
Todo es una consecuencia inmediata de las siguientes identidades, y el hecho de que la suma, el producto y la resta truncada son recursivas primitivas:

$$\begin{aligned} C_{\neg P} &= 1 - C_P \\ C_{P \wedge Q} &= C_P C_Q \\ C_{P \vee Q} &= C_P + C_Q - C_P C_Q \\ C_{P \Rightarrow Q} &= (1 - C_P) + C_Q C_P \end{aligned}$$

□

Proposición.
Sean $f_1, \ldots, f_k : \mathbb{N}^n \longrightarrow \mathbb{N}$ k funciones recursivas primitivas, y sean $P_1, \ldots, P_k$ k predicados recursivos primitivos tales que $P_i \wedge P_j = \emptyset$ si $i \neq j$ y $\bigvee_{i=1}^{k} P_i = \mathbb{N}^n$. Entonces la siguiente función $g : \mathbb{N}^n \longrightarrow \mathbb{N}$ es recursiva primitiva:

$$g(x_1, \ldots, x_n) = \begin{cases} f_1(x_1, \ldots, x_n) & \text{si } P_1(x_1, \ldots, x_n) \text{ es verdadero} \\ \vdots & \\ f_k(x_1, \ldots, x_n) & \text{si } P_k(x_1, \ldots, x_n) \text{ es verdadero} \end{cases}$$

Demostración. Tan solo basta observar que $g = \sum_{i=1}^{k} C_{P_i} f_i$ □

Ejemplo.
Para n entero positivo, sea $r_b(n)$ el resto de dividir a n por b.
Tenemos que $r_b(0) = 0$, y

$$\begin{aligned} r_b(n+1) &= \begin{cases} r_b(n) + 1 & \text{si } r_b(n) + 1 < b \\ 0 & \text{si } r_b(n) + 1 \geq b \end{cases} \\ &= H(n, r_b(n)) \end{aligned}$$

con $H(x, y) = \begin{cases} y + 1 & \text{si } y + 1 < b \\ 0 & \text{si } y + 1 \geq b \end{cases}$ (recursiva primitiva)
Luego, r_b es recursiva primitiva. ▲

Definición (existencial y universal acotados)**.**
Sea $P = P(x_1, \ldots, x_n)$ un predicado de n variables. Se define el predicado existencial acotado $Q(x_1, \ldots, x_{n-1}, Y) = \exists_{t \leq Y} P(x_1, \ldots, x_{n-1}, t)$ en la forma obvia:

$Q(x_1, \ldots, x_{n-1}, Y)$ es verdadero $\iff \exists\, t \leq Y$ tal que $P(x_1, \ldots, x_{n-1}, t)$ es verdadero

Se define el universal acotado $\forall_{t \leq Y} P(x_1, \ldots, x_{n-1}, t) := \neg\Big(\exists_{t \leq Y} \neg(P(x_1, \ldots, x_{n-1}, t))\Big)$.

Definición (Sumas y productos acotados)**.**
Sea $f : \mathbb{N}^n \longrightarrow \mathbb{N}$ una función, con $n \geq 2$. Se definen la suma y producto acotados $\mathcal{S}_f, \mathcal{P}_f : \mathbb{N}^n \longrightarrow \mathbb{N}$ asociados a f como

$$\mathcal{S}_f(x_1, \ldots, x_{n-1}, Y) = \sum_{t=0}^{Y} f(x_1, \ldots, x_{n-1}, t)$$

$$\mathcal{P}_f(x_1, \ldots, x_{n-1}, Y) = \prod_{t=0}^{Y} f(x_1, \ldots, x_{n-1}, t)$$

Proposición.
Si f es recursiva primitiva, entonces $\mathcal{S}_f$ y $\mathcal{P}_f$ son recursivas primitivas.

Demostración.
Hacemos recursión en Y: tenemos que

$$\mathcal{S}_f(x_1, \ldots, x_{n-1}, 0) = f(x_1, \ldots, x_{n-1}, 0)$$

donde el miembro derecho es recursivo primitivo, y

$$\begin{aligned} \mathcal{S}_f(x_1, \ldots, x_{n-1}, Y+1) &= \mathcal{S}_f(x_1, \ldots, x_{n-1}, Y) + f(x_1, \ldots, x_{n-1}, Y+1) \\ &= \mathcal{S}_f(x_1, \ldots, x_{n-1}, Y) + f(x_1, \ldots, x_{n-1}, \text{suc}(Y)) \\ &= H(x_1, \ldots, x_{n-1}, Y, S_f(x_1, \ldots, x_{n-1}, Y)) \end{aligned}$$

con $H(x_1, \ldots, x_{n-1}, Y, Z) = Z + f(x_1, \ldots, x_{n-1}, \text{suc}(Y))$, que es recursiva primitiva. La prueba para el producto acotado es esencialmente idéntica. □

Corolario.
Si $P = P(x_1, \ldots, x_n)$ es un predicado de n-variables recursivo primitivo, entonces $\exists_{t \leq Y} P(x_1, \ldots, x_{n-1}, t)$ y $\forall_{t \leq Y} P(x_1, \ldots, x_{n-1}, t)$ son recursivos primitivos.

Demostración.
Basta probarlo para el universal acotado.
Pues bien, llamando $Q(x_1, \ldots, x_{n-1}, Y) = \forall_{t \leq Y} P(x_1, \ldots, x_{n-1}, Y)$, tan solo basta observar que

$$C_Q(x_1, \ldots, x_{n-1}, Y) = \prod_{t=0}^{Y} C_P(x_1, \ldots, x_{n-1}, t) = \mathcal{P}_{C_P}(x_1, \ldots, x_{n-1}, Y)$$

y usar la proposición precedente. □

Ejemplos
1) Sea el predicado $\text{div}(x, y) = x$ divide a y. Tenemos que

$$x | y \iff \exists\, t \leq y \text{ tal que } y = xt$$

, esto es, $\text{div}(x,y) \equiv \underbrace{\exists_{t\leq y} \underbrace{\text{eq}(x,yt)}_{\text{rec prim}}}_{\text{rec prim}}$. Luego, div es recursivo primitivo.

2) Sea la función $\tau : \mathbb{N} \longrightarrow \mathbb{N}$ definida por $\tau(0) = 0$ y para $n > 0$,

$$\tau(n) = \text{número de divisores positivos de } n$$

Tenemos que $\tau(n) = \sum_{x=1}^{n} \text{div}(x,n)$ (que es una suma acotada asociada a div evaluada en (n,n)). Luego, τ es recursiva primitiva.

3) Sea el predicado $\text{prime}(n) = n$ es primo .
Observando que $\text{prime}(n) = \text{eq}(\tau(n), 2)$, deducimos que el predicado prime es recursivo primitivo.

MINIMIZACIÓN
Sea $P(x_1, \ldots, x_k, x_{k+1})$ un predicado de $k+1$ variables. Se llama operador minimización (de P) a la siguiente función parcial de k variables:

$$\min_t P(x_1, \ldots, x_k, t) = \min\{t \in \mathbb{N} : P(x_1, \ldots, x_k, t) \text{ es verdadero } \}$$

Ejemplos
1) Sea el predicado $P(x,y) : x < y$. Dado $x \in \mathbb{N}$, tenemos que

$$\min_t P(x,t) = \min_t x < t = x + 1 = \text{suc}(x)$$

2) Sea el predicado $Q(x,y) : x > y$. Su operador de minimización es

$$\min_t Q(x,t) = \min_t x > t = \begin{cases} 0 & x > 0 \\ \uparrow & x = 0 \end{cases}$$

Proposición.
Si $P(x_1, \ldots, x_{k+1})$ es un predicado computable de $k+1$ variables, entonces la función parcial de k variables $f(x_1, \ldots, x_k) = \min_t P(x_1, \ldots, x_k, t)$ es parcialmente computable.

Demostración.
El siguiente programa computa a f:

[C] **if** $P(X_1, ..., X_k, Z)$, go to [A]
$Z \leftarrow Z + 1$
go to [C]
[A] $Y \leftarrow Z$

□

MINIMIZACIÓN ACOTADA
Sea $P = P(x_1, \ldots, x_{k+1})$ un predicado de $k+1$ variables. Se llama minimización acotada de P a la siguiente función de $k+1$-variables:

$$\min_{t \leq x_{k+1}} P(x_1, \ldots, x_k, t)$$

En caso de que no existe $t \leq x_{k+1}$ tal que $P(x_1, \ldots, x_k, t)$ es verdadero, me devuelve el valor 0.

Ejemplo.
Si $Q(x, y) : x > y$, entonces $\min_{t \leq y} Q(x, y) = \min_{t \leq y} x > t = 0$ para todo $x, y \in \mathbb{N}$. ▲

Proposición.
Si $P = P(x_1, \ldots, x_{k+1})$ es un predicado primitivo recursivo, entonces el predicado $G(x_1, \ldots, x_k, y) = \min_{t \leq y} P(x_1, \ldots, x_k, t)$ es primitivo recursivo.

Demostración.
En efecto, tenemos que $G(x_1, \ldots, x_k, 0) = 0$ y

$$\begin{aligned} G(x_1, \ldots, x_k, y+1) &= \min_{t \leq y} P(x_1, \ldots, x_k, t) \\ &= \begin{cases} G(x_1, \ldots, x_k, y) & \text{si } (\neg P(x_1, \ldots, x_k, y)) \vee (\exists_{t \leq y} P(x_1, \ldots, x_k, t)) \text{ es verdadero} \\ y+1 & \text{en caso contrario} \end{cases} \\ &= H(x_1, \ldots, x_k, y, G(x_1, \ldots, x_k, y)) \end{aligned}$$

donde H viene dada por

$$H(x_1, \ldots, x_k, y, z) = \begin{cases} z & \text{si } (\neg P(x_1, \ldots, x_k, y)) \vee (\exists_{t \leq y} P(x_1, \ldots, x_k, t)) \text{ es verdadero} \\ y+1 & \text{en caso contrario} \end{cases}$$

Como P es recursivo primitivo, H también lo es, y por lo tanto G es recursiva primitiva.
□

Veamos algunos ejemplos de aplicación:

Ejemplos
1) Sea la función $f(n) = [\sqrt{n}]$ (la parte entera de la raíz cuadrada de n).
Tenemos que

$$f(n) = \min_{t \leq n} \{t + 1 > \sqrt{n}\} = \min_{t \leq n} \{(t+1)^2 > n\}$$

Luego, f es recursiva primitiva.

2) Sea $(p(n))_{n \in \mathbb{N}}$ la secuencia creciente de todos los números primos positivos.
Tenemos que $p(0) = 2$.

Ahora, si $p(j)|p(n)!+1$, entonces $j > n$, y en particular $p(n+1) \leq p(j) \leq p(n)!+1$. Por consiguiente,

$$p(n+1) = \min_{t \leq p(n)!+1} (\text{prime}(t) \wedge t > p(n)) = H(n, p(n))$$

con $H(x,y) = \min_{t \leq y!+1} (\text{prime}(t) \wedge t > y)$, que es recursiva primitiva.
Luego, la función $(p(n))_{n\in\mathbb{N}}$ es recursiva primitiva.

10 Números reales computables

Sea $x \in \mathbb{R}_{\geq 0}$, $x = b, a_1 a_2 \ldots$ (su desarrollo decimal). Se define $\text{dig}_x : \mathbb{N} \longrightarrow \mathbb{N}$ por $\text{dig}_x(0) = b$ y $\text{dig}_x(n) = a_n$ para $n \geq 1$. ¿Es dig_x una función computable? En general no lo es, ya que el conjunto de funciones computables es numerable, pero $\mathbb{R}_{\geq 0}$ no lo es.
Observemos que si $b = 0$, entonces $\text{dig}_x(n) = r_{10}(\lfloor 10^n x \rfloor)$.

Definición (números computables)**.**
Un número real $x \geq 0$ es computable si la función dig_x definida anteriormente es computable.

Ejercicio:
Probar que la definición de número computable en realidad no depende del sistema numérico.

Ejemplos
1) Todo número racional es computable.

2) Dados $N, m \in \mathbb{N}$ con $m \geq 1$, el número $\sqrt[m]{N}$ es computable. En efecto, si $h(n) = [10^n \sqrt[m]{N}]$, tenemos que

$$\begin{aligned} h(n) &= \min\{t \in \mathbb{N} : t+1 > 10^n \sqrt[m]{N}\} = \min\{t \in \mathbb{N} : (t+1)^m > 10^{nm} N\} \\ &= \min\{t \leq 10^n N : (t+1)^m > 10^{nm} N\} \end{aligned}$$

, de modo que h es recursiva primitiva.

11 Codificación

Definición (La función par)**.**
Llamaremos FUNCIÓN PAR a la función $<,>: \mathbb{N}^2 \longrightarrow \mathbb{N}$ definida como

$$< x, y >= 2^x(2y+1) - 1$$

Vemos de inmediato que la función par es biyectiva y recursiva primitiva. Su inversa $<,>^{-1}: \mathbb{N} \longrightarrow \mathbb{N}^2$ es de la forma $<,>^{-1}(n) = (\ell(n), r(n))$, con $\ell, r : \mathbb{N} \longrightarrow \mathbb{N}$. Explícitamente, ℓ y r vienen dadas por:

$$\ell(n) = \max\{t \in \mathbb{N} : 2^t | n+1\}$$
$$r(n) = \frac{\frac{n+1}{2^{\ell(n)}} - 1}{2}$$

De aquí, vemos que ℓ y r son recursivas primitivas.

Definición.
Si $p_1, p_2, \ldots, p_n, \ldots$ es una enumeración de los números primos, definimos para cada $k \geq 1$, una función de k variables $[\ldots] : \mathbb{N}^k \longrightarrow \mathbb{N}$ como sigue: dado $(a_1, \ldots, a_k) \in \mathbb{N}^k$, se define

$$[a_1, a_2, \ldots, a_k] = p_1^{a_1} \ldots p_k^{a_k} - 1$$

(Notar que es inyectiva y recursiva primitiva)

Si P es un programa en el lenguaje $\mathscr{S}$ (esto es, una secuencia finita de instrucciones $I_1, I_2, \ldots, I_k$ de $\mathscr{S}$), le asociaremos un número natural que llamamos el código de P, y denotamos $\#P$. Antes de esto, primero requerimos codificar las instrucciones.

Codificación de las instrucciones

Si I es una instrucción del lenguaje $\mathscr{S}$, le asignaremos un número natural $\#I$, llamado el código de I. Como convención, a la lista de instrucciones básicas del lenguaje $\mathscr{S}$ agregaremos el macro $V \leftarrow V$.

Enumeraremos las variables y las etiquetas del lenguaje $\mathscr{S}$ del siguiente modo:

$$Y, x_1, z_1, x_2, z_2, \ldots$$
$$A_1, B_1, C_1, D_1, A_2, B_2, C_2, D_2, \ldots$$

donde las x_i serán las que se usan como variables de entrada, y la Y como variable de salida.
Si V es una variable (de las de arriba), $\#V$ denotará la ubicación de la variable V en la enumeración anterior (empezando desde 1). Ídem, si E es una etiqueta, $\#E$ denotará la ubicación de la etiqueta E.
Si I es una instrucción básica del lenguaje $\mathscr{S}$, definimos

$$\#I =< a, < b, c >>$$

donde $a, b, c \in \mathbb{N}$ son definidos como sigue:

- $a = \begin{cases} \#E & \text{si } I \text{ tiene una etiqueta } E \\ 0 & \text{si no tiene etiqueta} \end{cases}$.

• $c = \#V - 1$, donde V es la única variable que aparece en I.

• $b = \begin{cases} 0 & \text{si } I \text{ es } V \leftarrow V \\ 1 & \text{si } I \text{ es } V \leftarrow V + 1 \\ 2 & \text{si } I \text{ es } V \leftarrow V - 1 \\ \#C + 2 & \text{si } I \text{ es if } V \neq 0, \text{ go to C} \end{cases}$.

Proposición. La función que a cada instrucción I le asigna su código $\#I$ es una biyección entre el conjunto de instrucciones del lenguaje $\mathscr{S}$ y los números naturales.

Demostración.
Si I, J instrucciones tales que $\#I = \#J$, esto es,

$$< a_I, < b_I, c_I >>=< a_J, < b_J, c_J >>$$

por inyectividad de $<,>$, se tiene que $a_I = a_J$, $b_I = b_J$ y $c_I = c_J$, esto es, $I = J$.

Ahora, dado $n \in \mathbb{N}$, veamos cuál es la instrucción I tal que $n = \#I$, esto es,

$$a_I = \ell(n) \quad \text{y} \quad < b_I, c_I >= r(n)$$

equivalentemente,

$$a_I = \ell(n),\ b_I = \ell(r(n)),\ c_I = r(r(n))$$

Si $\ell(n) = 0$, entonces I no debe tener etiqueta. Si $\ell(n) > 0$, I tiene como etiqueta a aquella de ubicación $\ell(n)$.
La variable V que aparece en I es aquella de ubicación $\#V = r(r(n)) + 1$.
Si $\ell(r(n)) = 0$, entonces I es del tipo $V \leftarrow V$.
Si $\ell(r(n)) = 1$, es del tipo $V \leftarrow V + 1$
Si $\ell(r(n)) = 2$, es del tipo $V \leftarrow V - 1$
Si $\ell(r(n)) \geq 3$, es del tipo if $V \neq 0$, go to C, con C la etiqueta que está en la posición $\ell(r(n)) - 2$. □

Definición. Si $\mathscr{P} = \{I_1, \ldots, I_k\}$ es un programa del lenguaje $\mathscr{S}$, se define el código de $\mathscr{P}$ como

$$\#\mathscr{P} = [\#I_1, \ldots, \#I_k] = p_1^{\#I_1} \ldots p_k^{\#I_k} - 1$$

Ejemplo. Consideremos los siguientes programas $\mathscr{P}_1$ y $\mathscr{P}_2$:

$$\begin{aligned} \mathscr{P}_1 &: \text{ if } X_1 \neq 0, \text{ go to } A_1 \\ \mathscr{P}_2 &: \text{ if } X_1 \neq 0, \text{ go to } A_1 \\ & \quad\ Y \leftarrow Y \end{aligned}$$

Se tiene que $\#P_1 = 2^{\#I_1} - 1$ y $\#P_2 = 2^{\#I_1} 3^0 - 1$, así que $\#P_1 = \#P_2$. ▲

Convención: Si P es un programa con más de una instrucción, no permitiremos que la última instrucción sea la de código nulo.

Bajo la convención anterior, se tiene el siguiente resultado:

Proposición. La función que asigna a cada programa $\mathscr{P}$ su código $\#\mathscr{P}$ es una biyección entre el conjunto de programas del lenguaje $\mathscr{S}$ y los naturales.

12 El problema de la parada o Halting Problem

Ejemplo. Sea la función parcial $f(x) = \begin{cases} 0 & \text{si } x = 0 \\ \uparrow & \text{si } x \neq 0 \end{cases}$.
El siguiente programa computa a f:

$$[\text{A}] \text{ if } X \neq 0, \text{ go to [A]}$$

▲

El problema que consideramos aquí es ver si es posible determinar algorítmicamente si un programa se detiene o no con una entrada dada. En otros términos, ¿Es cierto que la siguiente función Halt : $\mathbb{N} \longrightarrow \mathbb{N}$ es computable?:

$$\text{Halt}(x, y) := \begin{cases} 1 & \text{si el programa de código } y \text{ se detiene con entrada } x \\ 0 & \text{si no} \end{cases}$$

(decimos que un programa se detiene con entrada $x \in \mathbb{N}$ si se detiene cuando le asignamos el valor x a la variable X_1 y 0 a otra variable de entrada que aparezca en dicho programa)

Teorema. Halt no es una función computable.

Demostración.
Supongamos por absurdo que Halt(x, y) es computable. Entonces Halt(x, x) es computable. Consideremos el siguiente programa $\mathscr{P}$:

$$[\text{A}] \text{ if Halt}(X, X) = 1, \text{ go to [A]}$$

Llamemos $n = \#\mathscr{P}$. Tenemos que

$$\begin{aligned} \text{Halt}(n, n) &= \begin{cases} 1 & \text{si el programa } \mathscr{P} \text{ se detiene con entrada } n \\ 0 & \text{si no} \end{cases} \\ &= \begin{cases} 1 & \text{si Halt}(n, n) = 0 \\ 0 & \text{si no} \end{cases} \end{aligned}$$

, lo cual es absurdo. Luego, Halt no es una función computable. □

Corolario. Existe un número real no computable

Demostración. Desde luego, ya vimos esto por un argumento de numerabilidad, pero ahora queremos probarlo exhibiendo un real no computable: el número $x = 0, a_1 a_2 \dots a_n \dots$, con $a_i = \text{Halt}(i, i)$, es no computable, dado que $\text{Halt}|_\Delta$ es no computable por la prueba del teorema anterior. □

13 Programas Universales

Definimos una función parcial $\phi^{(n)} : \mathbb{N}^{n+1} \longrightarrow \mathbb{N}$ como sigue: dados $x_1, \ldots, x_n, y \in \mathbb{N}$, se define $\phi^{(n)}(x_1, \ldots, x_n, y) = \psi_P^{(n)}(x_1, \ldots, x_n)$, donde P es el programa de código y, y $\psi_P^{(n)}$ es la función parcial de n variables que computa P.
Una observación inmediata es que si f es una función parcialmente computable de n-variables, entonces $f(x_1, \ldots, x_n) = \phi^{(n)}(x_1, \ldots, x_n, k)$, siendo $k = \#P$, donde P es un programa que computa f.

Teorema. Para cada $n \in \mathbb{N}$, $\phi^{(n)}(x_1, \ldots, x_{n+1})$ es una función parcialmente computable de $n + 1$ variables.

Demostración. Consideremos las funciónes $Lt, ()_i : \mathbb{N} \longrightarrow \mathbb{N}$ definidas como

$$Lt(n) = \begin{cases} 0 & \text{si } n = 0 \text{ o } n = 1 \\ \max\{k \geq 1 : p_k | n\} & \text{si } n \geq 2 \end{cases}$$

$$(n)_i = \max\{t \in \mathbb{N} : p_i^t | n\}$$

Estas funciones son recursivas primitivas.
El siguiente programa computa a $\phi^{(n)}$:

$z \leftarrow X_{n+1} + 1$
$s \leftarrow \prod_{i=1}^{n} p_{2i}^{X_i}$
$k \leftarrow 1$
[C] if $(k = Lt(z) + 1 \vee k = 0)$, go to F
$U \leftarrow r((z)_k)$
$P \leftarrow p_{r(U)+1}$
if $\ell(U) = 0$, go to N
if $\ell(U) = 1$, go to A
if $\neg(P|s)$, go to N
if $\ell(U) = 2$, go to M
$K \leftarrow \min\{i \leq Lt(z) : Lt((z)_n) + 2 = \ell(U)\}$
go to C
[M] $s \leftarrow \lfloor \frac{s}{P} \rfloor$
[A] $s \leftarrow s.P$
[N] $K \leftarrow K + 1$
go to C
[F] $Y \leftarrow (s)_1$

□

Ejemplo. Sea f una función parcialmente computable. ¿Existe g total y computable que extiende a f?.
Tomemos la función parcialmente computable $f(x) = \phi^{(1)}(x,x)+1$. Si consideremos el programa:

$$Z \leftarrow Z + 1$$
$$[A] \text{ if } Z \neq 0, \text{ go to } [A]$$

, se tiene que f está indefinida en su código (lo cual muestra que f no es total).
Supongamos que existe $g : \mathbb{N} \longrightarrow \mathbb{N}$ que extiende a f. Sea P un programa que computa a g, y sea $z_0 = \#P$, de modo que $g = \phi^{(1)}(-, z_0)$. En particular, como g es total, tenemos que $\phi^{(1)}(z_0, z_0)\downarrow$, equivalentemente, $f(z_0)\downarrow$. Como g extiende a f, tenemos entonces que $\underbrace{f(z_0)}_{\phi^{(1)}(z_0,z_0)+1} = \underbrace{g(z_0)}_{\phi^{(1)}(z_0,z_0)}$, o sea, $1 = 0$, lo cual es absurdo. ▲

Definición.
Si $n \in \mathbb{N}$, se define el siguiente predicado de $n+2$ variables $\text{STP}^{(n)}(x_1, \ldots, x_n, y, t)$: este predicado es verdadero sii el programa de código y y entradas $x_1, \ldots, x_n$ se detiene en a lo sumo t pasos.

Teorema. El predicado $\text{STP}^{(n)}$ es recursivo primitivo.

Demostración. La prueba se divide en tres pasos:

1) Consideremos las siguientes funciones de 2 variables, donde i representa la i-ésima instrucción del programa de código Y:

$$\text{label}(i,y) = \ell((y+1)_i)$$
$$\text{var}(i,y) = r(r((y+1)_i)) + 1$$
$$\text{inst}(i,y) = \ell(r((y+1)_i))$$
$$\text{label}^1(i,y) = \ell(r((y+1)_i)) \dot{-} 2$$

2) A partir de las funciones definidas en el paso anterior, definimos los siguientes predicados recursivos primitivos de 2 variables x, y, donde y es el código de un programa, y x representa una descripción instantánea:

$$\text{skip}(x,y) : \Big((\text{inst}(\ell(x),y) = 0 \wedge \ell(x) \leq Lt(y+1)) \vee (\text{inst}(\ell(x),y) \geq 2 \wedge \neg p_{\text{var}(\ell(x),y)} | r(x))\Big)$$
$$\text{incr}(x,y) : \text{inst}(\ell(x),y) = 1$$
$$\text{decr}(x,y) : \text{inst}(\ell(x),y) = 2 \wedge p_{\text{var}(\ell(x),y)} | r(x)$$
$$\text{branch}(x,y) : \text{inst}(\ell(x),y) > 2 \wedge \exists_{i \leq Lt(y+1)} \text{label}(i,y) = \text{label}^1(\ell(x),y)$$

3) Definimos la siguiente función $\text{suc}(x,y)$:

$$\text{suc}(x,y) = \begin{cases} < \ell(x)+1, r(x) > & \text{si skip}(x,y) \\ < \ell(x)+1, p_{\text{var}(\ell(x),y)} . r(x) > & \text{si incr}(x,y) \\ < \ell(x)+1, \left\lfloor \dfrac{r(x)}{p_{\text{var}(\ell(x),y)}} \right\rfloor > & \text{si decr}(x,y) \\ < \min\limits_{i \leq Lt(y+1)} \text{label}(i,y) = \text{label}^1(\ell(x),y), r(x) > & \text{si branch}(x,y) \\ < Lt(y+1)+1, r(x) > & \text{en otro caso} \end{cases}$$

Definimos a partir de la función $\text{suc}(x, y)$ los sucesivos snapshots del siguiente modo:

$$\text{snap}^{(n)}(x_1, \ldots, x_n, y, 0) =< 1, \prod_{i=1}^{n} p_{2i}^{x_i} >$$

$$\text{snap}^{(n)}(x_1, \ldots, x_n, y, i+1) = \text{suc}(\text{snap}^{(n)}(x_1, \ldots, x_n, y, i), y)$$

Definimos el predicado $\text{term}(x, y) : \ell(x) > Lt(y+1)$.
Se tiene que $\text{STP}^{(n)}(x_1, \ldots, x_n, y, t)$ es verdadero sii $\text{term}(\text{snap}^{(n)}(x_1, \ldots, x_n, y, t), y)$ es verdadero, lo cual concluye la demostración. □

El siguiente ejemplo muestra que un existencial o universal no acotado sobre un predicado recursivo primitivo no es en general computable:

Ejemplo.
Sea el predicado recursivo primitivo $P(x, t) = \text{STP}^{(1)}(x, x, t)$. Tenemos que

$$\exists\, t\ P(x,t) = \begin{cases} 1 & \text{si Halt}(x,x) = 1 \\ 0 & \text{si Halt}(x,x) = 0 \end{cases} = \underbrace{\text{Halt}(x,x)}_{\text{no computable}}$$

▲

Teorema (de la forma normal)**.**
Si $f(x_1, \ldots, x_n)$ es una función parcialmente computable de n variables, entonces existe un predicado $\mathcal{P} = \mathcal{P}(x_1, \ldots, x_n, z)$ recursivo primitivo tal que

$$f(x_1, \ldots, x_n) = \ell(\min_z \mathcal{P}(x_1, \ldots, x_n, z))$$

Demostración. Sea $\mathscr{P}$ un programa que computa a f, $y_0 = \#\mathscr{P}$, y sea $\mathcal{P}$ el siguiente predicado de $n+1$ variables:

$$\mathcal{P}(x_1, \ldots, x_n, z) = \text{STP}^{(n)}(x_1, \ldots, x_n, y_0, r(z)) \wedge \Big(r(\text{snap}^{(n)}(x_1, \ldots, x_n, y_0, r(z)))_1 = \ell(z) \Big)$$

Supongamos primero que $\ell(\min_z \mathcal{P}(x_1, \ldots, x_n, z))$ está definido, esto es, existe un z tal que $\mathcal{P}(x_1, \ldots, x_n, z)$ es verdadero. Entonces $\text{STP}^{(n)}(x_1, \ldots, x_n, y_0, r(z))$ se cumple y $\ell(z) = r(\text{snap}^{(n)}(x_1, \ldots, x_n, y_0, r(z)))_1 = f(x_1, \ldots, x_n)$.
Si $\ell(\min_z \mathcal{P}(x_1, \ldots, x_n, z))$ está indefinido, ello significa $\text{stp}^{(n)}(x_1, \ldots, x_n, y_0, t) \uparrow$ para todo t, es decir, $f(x_1, \ldots, x_n) \uparrow$. □

Definición. Una función se denomina RECURSIVA PARCIAL si se obtiene en un número finito de pasos a partir de las funciones iniciales usando las operaciones de composición, recursión y/o minimización no acotada.

Corolario. La clase de funciones parcialmente computables coincide con la clase de funciones recursivas parciales.

Damos el siguiente ejemplo solo para ilustrar el resultado del corolario previo:

Ejemplo. Sea la función parcialmente computable $f(x) = \begin{cases} 1 & x \neq 0 \\ \uparrow & x = 0 \end{cases}$.

El siguiente programa computa a f:

```
[C]  if X ≠ 0, go to [A]
       go to [C]
     [A]  Y ← Y + 1
```

Veamos si hay un predicado $\mathcal{P} = \mathcal{P}(x, y)$ tal que $f(x) = 1 + \min_y \mathcal{P}(x, y)$.
Si $x = 0$, no debe existir y tal que $\mathcal{P}(0, y)$ es verdadero, o sea, $\mathcal{P}(0, y)$ debe ser falso para todo y.
Podemos tomar $\mathcal{P}(x, y) : y < x$. ▲

El siguiente teorema será utilizado en la demostración del teorema de Rice:

Teorema (Teorema del Parámetro)**.** Dados $m, n \in \mathbb{N}$, existe una función primitiva recursiva S_m^n que verifica lo siguiente:

$$\phi^{(m+n)}(x_1, \ldots, x_n, \mu_1, \ldots, \mu_m, y) = \phi^{(n)}(x_1, \ldots, x_n, S_m^n(\mu_1, \ldots, \mu_m, y))$$

Demostración.
Procedemos por inducción en m:
Si $m = 1$, debemos encontrar una función recursiva primitiva $S_1^n(\mu, y)$ tal que

$$\phi^{(n+1)}(x_1, \ldots, x_n, \mu, y) = \phi^{(n)}(x_1, \ldots, x_n, S_1^n(\mu, y))$$

Consideramos el siguiente programa:

$$\mu \text{ veces } \begin{cases} X_{n+1} \leftarrow X_{n+1} + 1 \\ X_{n+1} \leftarrow X_{n+1} + 1 \\ \vdots \\ X_{n+1} \leftarrow X_{n+1} + 1 \end{cases}$$

Sea $t = \#I$, donde $I : X_{n+1} \leftarrow X_{n+1} + 1$. Defino

$$S_1^n(\mu, y) = \Big[\prod_{j=1}^{\mu} p_j^t \prod_{j=1}^{Lt(y+1)} p_{n+j}^{(y+1)_j} \Big] - 1$$

Supongamos que el resultado es válida para m y consideremos el caso $m+1$. Tenemos que

$$\begin{aligned} \phi^{(m+1+n)}(x_1, \ldots, x_n, \mu_1, \ldots, \mu_m, \mu_{m+1}, y) &= \phi^{(m+n)}(x_1, \ldots, x_n, \mu_1, \ldots, \mu_m, S_1^{n+m}(\mu_{m+1}, y)) \\ &\underbrace{=}_{HI} \phi^{(n)}(x_1, \ldots, x_n, S_m^n(\mu_1, \ldots, \mu_m, S_1^{n+m}(\mu_{m+1}, y))) \end{aligned}$$

con S_1^{n+m} y S_m^n recursivas primitivas. □

14 Conjuntos recursivos y recursivamente numerables

Definición (Conjunto recursivo)**.**
Un conjunto $B \subseteq \mathbb{N}$ se dice RECURSIVO si su función característica C_B es computable.

Ejemplo.
El conjunto $B = \{x \in \mathbb{N} : \phi^{(1)}(x,x) \downarrow\}$ no es recursivo, pues $C_B(x) = \text{Halt}(x,x)$. ▲

Supongamos que $B \subseteq \mathbb{N}$ es recursivo. Sea P el siguiente programa

$$\text{[A] if } C_B(x) = 0, \text{ go to [A]}$$

Tenemos que P se detiene con entrada x si y solo si $x \in B$. Concretamente, P computa la función: $f(x) = \begin{cases} 0 & x \in B \\ \uparrow & x \notin B \end{cases}$
Se tiene evidentemente que $B = \{x \in \mathbb{N} : f(x) \downarrow\}$.
Así, probamos que todo conjunto recursivo es el dominio de definición de una función parcialmente computable.

Proposición.

(a) $\emptyset$ y $\mathbb{N}$ son conjuntos recursivos.

(b) $\{x\}$ es recursivo $\forall\, x \in \mathbb{N}$.

(c) Si $B \subseteq \mathbb{N}$, entonces B es recursivo sii B^c es recursivo.

(d) Si $B, C \subseteq \mathbb{N}$ son recursivos, entonces $B \cup C$ y $B \cap C$ son recursivos.

Demostración. Es inmediata. □

Corolario.

(a) Todo subconjunto finito de $\mathbb{N}$ es recursivo.

(b) $\{x \in \mathbb{N} : \phi^{(1)}(x,x) \downarrow\}$ es infinito.

Escribiendo $B = \bigcup_{x \in B} \{x\}$ para un conjunto no recursivo B, vemos que la unión arbitraria de conjuntos recursivos no es en general recursivo.

Definición (Conjunto recursivamente numerable)**.**
Un conjunto $B \subseteq \mathbb{N}$ se dice RECURSIVAMENTE NUMERABLE si es el dominio de una función parcialmente computable. Es decir, existe una función g de una variable parcialmente computable tal que $B = \{x \in \mathbb{N} : g(x) \downarrow\}$.

Ejemplo.
El conjunto $B = \{x \in \mathbb{N} : \phi^{(1)}(x,x) \downarrow\}$ es recursivamente numerable, pues es el dominio de la función parcialmente computable $\phi^{(1)}(x,x)$. Pero no es recursivo, pues $C_B(x) = \text{Halt}(x,x)$ (no computable). ▲

Proposición. Dado $B \subseteq \mathbb{N}$, son equivalentes:

(1) B es recursivo.

(2) B y B^c son recursivamente numerables.

Demostración.
(1) $\Rightarrow$ (2): lo probamos al comienzo de esta sección.
(2) $\Rightarrow$ (1): supongamos que B, B^c son recursivamente numerables, i.e, existen f, g parcialmente computables tq $B = \{x \in \mathbb{N} : f(x)\downarrow\}$ y $B^c = \{x \in \mathbb{N} : g(x)\downarrow\}$.
Sean $\mathscr{P}, \mathscr{Q}$ programas con códigos y_0, z_0 que computan a f, g respectivamente.
El siguiente programa computa a la función característica de B (y por ende, B es recursivo):

[D] if $\text{STP}^{(1)}(x, y_0, t)$, go to C
if $\text{STP}^{(1)}(x, z_0, t)$, go to E
$t \leftarrow t + 1$
go to D
[C] $Y \leftarrow Y + 1$

donde recordamos que la etiqueta [E] denota exit. □

Teorema. Dado $B \subseteq \mathbb{N}$ no vacío, son equivalentes:

(1) B es recursivamente numerable.

(2) Existe un predicado recursivo primitivo $P(x, t)$ tal que $B = \{x \in \mathbb{N} : \exists\, t\ P(x, t)\}$.

(3) Existe una función $f : \mathbb{N} \longrightarrow \mathbb{N}$ recursiva primitiva tal que $\text{Im}(f) = B$.

(4) Existe una función total y computable $f : \mathbb{N} \longrightarrow \mathbb{N}$ tal que $\text{Im}(f) = B$.

(5) Existe una función parcialmente computable $f : \mathbb{N} \longrightarrow \mathbb{N}$ tal que $\text{Im}(f) = B$.

Demostración.
(1) $\Rightarrow$ (2): Sea f parcialmente computable tal que $B = \{x \in \mathbb{N} : f(x)\downarrow\}$.
Sea n_0 el código de un programa que computa a f y consideremos el predicado recursivo primitivo $P(x, t) = \text{STP}^{(1)}(x, n_0, t)$. Se tiene que $x \in B$ sii $f(x)\downarrow$ sii existe $\exists\, t\ P(x, t)$.

(2) $\Rightarrow$ (3): Sea $P = P(x, t)$ un tal predicado recursivo primitivo. Fijemos $b_0 \in B$ y sea la función (recursiva primitiva) $f : \mathbb{N} \longrightarrow \mathbb{N}$ definida por

$$f(x) = \begin{cases} \ell(x) & P(\ell(x), r(x)) \\ b_0 & \text{si no} \end{cases}$$

Es claro que $\text{Im}(f) \subseteq B$. Para ver la otra contención, sea $b \in B$, i.e, existe t tal que $P(b, t)$ es verdadero. Tomo $x =< b, t >$, de modo que $\ell(x) = b$ y $r(x) = t$, y así $P(\ell(x), r(x))$ es verdadero, y luego $f(x) = \ell(x) = b$.

(3) $\Rightarrow$ (4) y (4) $\Rightarrow$ (5) son evidentes.

(5) $\Rightarrow$ (1): Supongamos primero que f es total, y consideremos el siguiente programa:

[C] if $X = f(Z)$, go to E
$Z \leftarrow Z + 1$
go to C

Este programa computa la función parcial $h(x) = \begin{cases} 0 & \text{si } x \in B \\ \uparrow & \text{si } x \notin B \end{cases}$.
Tenemos que $\{x \in \mathbb{N} : h(x) \downarrow\} = B$, así que B es recursivamente numerable.

Supongamos ahora que f es parcialmente computable, y sea $C = \{x \in \mathbb{N} : f(x) \downarrow\}$ (es no vacío, pues B es no vacío). Por definición, C es recursivamente numerable. Como ya probamos que (1) implica (4), existe una función total computable $g : \mathbb{N} \longrightarrow \mathbb{N}$ tal que $\text{Im}(g) = C = \text{dom}(f)$. En particular, $f \circ g$ es total computable y $\text{Im}(f \circ g) = \text{Im}(f) = B$. Por el caso anterior, B es recursivamente numerable. □

Aplicación: El conjunto $\{x \in \mathbb{N} : \psi_x^{(1)}$ es total $\}$ no es recursivamente numerable.

Demostración.
Llamemos C a tal conjunto, y supongamos por absurdo que es recursivamente numerable. Por el teorema previo, existe $g : \mathbb{N} \longrightarrow \mathbb{N}$ total y computable tal que $\text{Im}(g) = C$. Sea la función $h(x) = \phi^{(1)}(x, g(x)) + 1$ (es total y computable). Sea n_0 el código de un programa que computa a h. Como es total, tenemos que $y_0 \in C = \text{Im}(g)$, de modo que existe $z_0 \in \mathbb{N}$ tal que $g(z_0) = y_0$. Como $h(x) = \phi^{(1)}(x, y_0) = \phi^{(1)}(x, g(z_0))$, en particular, con $x = z_0$, tenemos que

$$\underbrace{h(z_0)}_{\phi^{(1)}(z_0,g(z_0))+1} = \phi^{(1)}(z_0, g(z_0))$$

o sea, $0 = 1$, lo cual es absurdo. □

Teorema de Rice

Lema. Sea $f : \mathbb{N} \longrightarrow \mathbb{N}$ una función total y computable y $B \subseteq \mathbb{N}$.

(1) Si B es recursivo, entonces $f^{-1}(B)$ es recursivo.

(2) Si B es recursivamente numerable, entonces $f^{-1}(B)$ es recursivamente numerable

Demostración.
(1) Basta observar que $C_{f^{-1}(B)} = C_B \circ f$.
(2) Supongamos que B recursivamente numerable, esto es, existe $g : \mathbb{N} \longrightarrow \mathbb{N}$ parcialmente computable tal que $B = \{x \in \mathbb{N} : g(x) \downarrow\}$.
Tenemos que $f^{-1}(B) = \{x \in \mathbb{N} : (g \circ f)(x) \downarrow\}$, con $g \circ f$ parcialmente computable, así que $f^{-1}(B)$ es recursivamente numerable. □

PROBLEMA
Sea $f : \mathbb{N} \longrightarrow \mathbb{N}$ una función total tal que $f^{-1}(B)$ es recursivo para todo $B \subseteq \mathbb{N}$ recursivo. ¿Es cierto que f es computable?

Teorema (Teorema de Rice)**.**
Sea Γ un subconjunto propio no vacío del conjunto de funciones parcialmente computables de 1 variable. Entonces $R(\Gamma) := \{x \in \mathbb{N} : \psi_x^{(1)} \in \Gamma\}$ no es recursivo.

Demostración.
Sea $K = \{x \in \mathbb{N} : \phi^{(1)}(x, x) \downarrow\}$. Sabemos que K no es recursivo (ya que $C_K(x) = \text{Halt}(x, x)$). Construiremos una función $g : \mathbb{N} \longrightarrow \mathbb{N}$ total y computable tal que $g^{-1}(R(\Gamma)) = K$ (y como consecuencia, se tendrá que $R(\Gamma)$ no es recursivo)

Tomemos una $f \in \Gamma$, y asumamos primero que la función vacía no está en Γ. Definimos $H : \mathbb{N}^2 \longrightarrow \mathbb{N}$ como sigue

$$H(x_1, x_2) = \begin{cases} f(x_1) & \phi^{(1)}(x_2, x_2) \downarrow \\ \uparrow & \phi^{(1)}(x_2, x_2) \uparrow \end{cases}$$

H es parcialmente computable. En efecto, el siguiente programa computa a H:

$$z \leftarrow \phi^{(1)}(x_2, x_2)$$
$$Y \leftarrow f(x_1)$$

Sea y_0 el código de este programa, de modo que $H(x_1, x_2) = \phi^{(2)}(x_1, x_2, y_0)$.
Por el teorema del parámetro, existe una función recursiva primitiva S de dos variables tal que $\phi^{(2)}(x_1, x_2, y_0) = \phi^{(1)}(x_1, S(x_2, y_0))$. Llamemos $g(x) = S(x, y_0)$ (es total y computable). Veamos que $K = g^{-1}(R(\Gamma))$:

$$x \in K \Rightarrow \phi(x, x) \downarrow \Rightarrow H(-, x) = f \Rightarrow \phi^{(1)}(-, g(x)) = f \Rightarrow \psi^{(1)}_{g(x)} = \underbrace{f}_{\in \Gamma} \Rightarrow x \in g^{-1}(R(\Gamma))$$

de modo que $K \subseteq g^{-1}(R(\Gamma))$.
Recíprocamente, sea $x \in g^{-1}(R(\Gamma))$, esto es, $\psi^{(1)}_{g(x)} \in \Gamma$, o sea, existe g (no vacía, porque estamos asumiendo que la función vacía no está en Γ) tal que $\psi^{(1)}_{g(x)} = g$. Tomemos $x_1 \in \mathbb{N}$ tal que $g(x_1) \downarrow$. Entonces $\underbrace{\phi^{(1)}(x_1, g(x))}_{=H(x_1, x)} \downarrow$, que implica que $\phi^{(1)}(x, x) \downarrow$, esto es, $x \in K$. Luego, $g^{-1}(R(\Gamma)) \subseteq K$.

Hemos probado así que $K = g^{-1}(R(\Gamma))$.

Ahora, si la función vacía está en Γ, entonces no está en Γ^c, así que por el caso anterior, tenemos que $R(\Gamma^c) = (R(\Gamma))^c$ no es recursivo, equivalentemente, $R(\Gamma)$ no es recursivo. □

Ejemplos
1) Sea $B = \{x \in \mathbb{N} : \phi^{(1)}(x, x) = 1\}$. Este conjunto no es recursivo: en efecto, considerando $\Gamma = \{\psi^{(1)}_x : x \in \mathbb{N}, \psi_x(x) = 1\}$, tenemos que

$$B = \{x \in \mathbb{N} : \psi^{(1)}_x \in \Gamma\} = R(\Gamma)$$

2) Sea la función $f(x) = \text{Halt}(0, x)$. Esta función no es computable, equivalentemente, $\{x \in \mathbb{N} : \text{Halt}(0, x) = 1\}$ no es recursivo. En efecto, tenemos que

$$\{x \in \mathbb{N} : \text{Halt}(0, x) = 1\} = \{x \in \mathbb{N} : \psi_x(0) \downarrow\} = R(\Gamma)$$

con Γ el conjunto de funciones parcialmente computables de una variable, que están definidas en 0.

(3) La función $f(x) = \begin{cases} 1 & 1 \in \text{Im}(\psi^{(1)}_x) \\ 0 & \text{si no} \end{cases}$ no es computable. En efecto, f es la función característica del conjunto $B = \{x \in \mathbb{N} : 1 \in \text{Im}(\psi^{(1)}_x)\}$. Pues bien, considerando Γ el conjunto de funciones parcialmente computables de una variable cuya imagen contiene al 1, tenemos que $B = R(\Gamma)$.