
LÓGICA Y PROGRAMACIÓN

PROBLEMAS DE λ -CALCULUS Y LÓGICA COMBINATORIA



**UNIVERSIDAD
DE GRANADA**

UNIVERSIDAD DE GRANADA
LÓGICA Y PROGRAMACIÓN

CURSO
5TO CURSO

DOCENTE
FRANCISCO MIGUEL GARCÍA OLMEDO

AUTORES
ALEJANDRO EGEA LÓPEZ
NICOLÁS RAMÍREZ RODILES

A FECHA DE
20 DE ENERO DE 2026

Índice general

1. Problemas de λ-Calculus y Lógica Combinatoria	3
Ejercicio 1	3
1.1.1. Introducción	3
1.1.2. Name-carrying expressions y árbol sintáctico	4
1.1.3. De nombres a índices: notación de de Bruijn	5
1.1.4. Sintaxis de expresiones sin nombres	5
1.1.5. Sustitución en notación de de Bruijn	6
1.1.6. Reducción β	6
1.1.7. Reducción η	6
1.1.8. Algunos comentarios sobre la reducción múltiple	7
1.1.9. Conclusión	7
Ejercicio 2	8
Ejercicio 3	9
Ejercicio 4	10
Ejercicio 5	12
Ejercicio 6	14
Ejercicio 7	15
Ejercicio 8	16
Ejercicio 9	17
1.9.1. Introducción	17
1.9.2. Términos y combinadores de CL	17
1.9.3. Sustitución y sustitución débil	18
1.9.4. Eliminación de la abstracción λ	18
1.9.5. Conclusión	19
2. Unidad Temática: Lógica de Primer Orden – Los Teoremas de Incompletitud de Gödel	20
2.1. Introducción	20
2.2. Consistencia, coherencia y completitud	20
2.2.1. Consistencia	20
2.2.2. Coherencia	21
2.2.3. Completitud	21
2.3. Axiomática de Peano	21
2.3.1. Axiomas de Peano	21
2.3.2. Aritmética y propiedades de AP	22
2.4. Numeración de Gödel	23
2.5. Teoremas de Incompletitud de Gödel	25
2.5.1. Lema de la diagonalización	25
2.5.2. Primer Teorema de Incompletitud	25
2.5.3. Segundo Teorema de Incompletitud	26

2.6. Conclusión	26
3. Ejercicios de Programación en Haskell	27
Ejercicio 2.	27
Ejercicio 3.	28

Capítulo 1

Problemas de λ -Calculus y Lógica Combinatoria

Ejercicio 1.

Exponga y desarrolle justificadamente el tema de la “Notación de de Bruijn”.

1.1.1. Introducción

La notación tradicional del cálculo λ usa nombres para las variables ligadas (x, y, z, \dots). Esta práctica introduce tres dificultades fundamentales:

1. **Captura accidental:** al sustituir una expresión por una variable libre, puede producirse captura si surgen colisiones de nombre con variables ligadas en el contexto.
2. **Conversión- α obligatoria:** para evitar capturas debe renombrarse sistemáticamente las variables ligadas.
3. **Complejidad en meta-demostraciones:** propiedades como la confluencia o la prueba de Church-Rosser se complican debido a la constante necesidad de renombrar.

La propuesta de N. G. de Bruijn consiste en *eliminar los nombres de variables ligadas* y reemplazarlos por **índices naturales** que indican cuántos ligadores separan la ocurrencia de su λ correspondiente. De este modo:

- desaparece por completo la conversión- α ,
- no hay colisiones de nombres,
- las definiciones de sustitución, reducción β y η se vuelven puramente estructurales,
- las demostraciones metateóricas se simplifican sustancialmente.

El propio de Bruijn establece tres criterios para evaluar una notación:

- (i) legibilidad humana,

- (ii) claridad en discusión metalenguística,
- (iii) utilidad para implementaciones automáticas.

La notación de índices sacrifica parcialmente el punto (i), pero destaca en (ii) y (iii).

1.1.2. Name-carrying expressions y árbol sintáctico

Antes de eliminar nombres, representamos aplicaciones mediante un símbolo especial:

$$A((M), (N)),$$

que corresponde a la aplicación usual MN . Esto permite un análisis uniforme de la estructura.

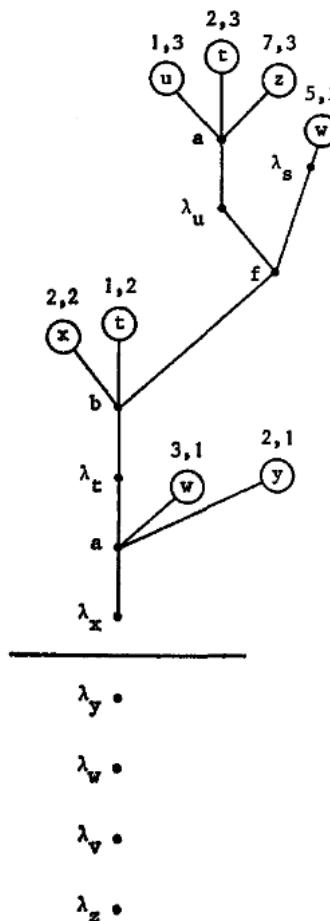
Consideremos ahora la siguiente expresión λ -cálculo (tomada del ejemplo del artículo):

$$\boxed{\lambda x a(\lambda b(x, t, f(\lambda u a(u, t, z), \lambda s w)), w, y)}$$

En forma estructurada “name-carrying”, cada rama del árbol lleva etiquetas a, b, f según la construcción.

El artículo presenta el siguiente **árbol anotado** con *reference depth* y *level*. Los pares (d, ℓ) representan:

- $d = \text{reference depth}$: número de ligadores λ desde la ocurrencia hasta su ligador,
- $\ell = \text{level}$: número total de λ hasta la raíz.



Lectura precisa del árbol

Tomemos algunos nodos para ejemplificar:

- El nodo x tiene etiqueta $(2, 2)$:
 - reference depth = 2: está bajo dos λ antes de llegar a su ligador original λ_x ,
 - level = 2: desde la ocurrencia hasta la raíz hay dos λ .
- El nodo t bajo $b(x, t)$ tiene $(1, 2)$:
 - depth = 1: está inmediatamente bajo λ_t ,
 - level = 2: hay dos ligadores superiores en total.
- El nodo v etiquetado $(3, 1)$ indica que:
 - depth = 3: su ligador está tres λ más arriba,
 - level = 1: solo un λ separa esa rama de la raíz.
- Lo mismo ocurre para u, t, z, w en la rama derecha bajo f .

Este árbol es la base para convertir la expresión a notación sin nombres.

1.1.3. De nombres a índices: notación de de Bruijn

Para cada ocurrencia ligada, sustituimos su nombre por su *reference depth*.

- Las variables libres se conservan en una lista ordenada (x_1, x_2, \dots) ,
- Los *level* sirven como información auxiliar para verificar la corrección formal, pero se omiten en la expresión final.

Ejemplos simples

$$\lambda x. x \longmapsto \lambda. 1$$

$$\lambda x. \lambda y. (x y) \longmapsto \lambda. \lambda. (2 1)$$

$$\lambda x. \lambda y. \lambda z. x \longmapsto \lambda. \lambda. \lambda. 3$$

1.1.4. Sintaxis de expresiones sin nombres

Usamos una gramática claramente indentada:

```

<NF>      ::= <Const>
           | <Index>
           | <NFList>
           | lambda.<NF>

<Const>    ::= a | b | c | ...
               ; constantes simbólicas

<Index>    ::= 1 | 2 | 3 | ...
               ; variable ligada por índice

<NFList>   ::= A(<NF>, <NF>)
               | <NF> <NF>
               ; aplicación
               ; concatenación de expresiones
  
```

Esta sintaxis es extremadamente regular: ya no aparecen nombres ligados, sólo índices.

1.1.5. Sustitución en notación de de Bruijn

La sustitución se define como:

$$S(Z_1, Z_2, \dots; Q),$$

donde Z_i sustituye a la variable libre cuyo índice es i .

Casos fundamentales:

- Si Q es una constante, permanece igual.

- Si $Q = k$ es un índice, entonces:

$$S(\dots, Z_k; k) = Z_k,$$

con ajuste de índices si la sustitución entra bajo una λ .

- Si $Q = A((Q_1), (Q_2))$, entonces:

$$S(Z; A((Q_1), (Q_2))) = A((S(Z; Q_1)), (S(Z; Q_2))).$$

- Si $Q = \lambda. R$, entonces:

$$S(Z_1, Z_2, \dots; \lambda. R) = \lambda. S(Z'_1, Z'_2, \dots; R),$$

donde cada Z'_i es la versión de Z_i con sus índices incrementados en uno (para preservar referencias correctas).

1.1.6. Reducción β

La regla de contracción β se expresa elegantemente:

$$A((\lambda. Q), (r)) \longrightarrow S(r, 2, 3, \dots; Q),$$

es decir: sustituimos el índice 1 por r .

Ejemplos

$$(\lambda x. x) a \rightsquigarrow \lambda. 1 a \longrightarrow a.$$

$$(\lambda x. \lambda y. x) y \rightsquigarrow A((\lambda. \lambda. 2), (y)) \longrightarrow \lambda. 1.$$

En ningún momento aparece conversión- α .

1.1.7. Reducción η

La regla extensional:

$$\lambda. A((Q), 1) \longrightarrow Q$$

siempre que Q no contenga ninguna referencia al índice 1 ligado por la λ .

Ejemplo:

$$\lambda x. f x \mapsto \lambda. A((f), 1) \longrightarrow f.$$

1.1.8. Algunos comentarios sobre la reducción múltiple

De Bruijn desarrolla una teoría de *reducción múltiple* β_U , donde un conjunto U de símbolos de aplicación se reduce simultáneamente.

Ideas centrales:

- se define la noción de *expresión U-correcta*,
- la sustitución conserva U -corrección (Teorema 10.1),
- la reducción múltiple y la sustitución poseen reglas claras de commutación y composición (Teorema 11.1),
- reducciones múltiples para conjuntos distintos U, V commutan entre sí (Teorema 11.2),
- todo ello conduce a una demostración pulcra del **teorema de Church-Rosser** en la notación sin nombres.

En la notación de índices no existe la conversión- α ni conflictos de nombres, por lo que la demostración se vuelve más transparente: la confluencia se logra por propiedades puramente estructurales.

1.1.9. Conclusión

La notación de de Bruijn resuelve de raíz los problemas de captura y renombrado al sustituir nombres ligados por índices estructurales. La sustitución y la reducción se vuelven definiciones limpias, algebraicas y mecanizables.

La aplicación a la reducción múltiple y a la prueba de Church-Rosser muestra que la notación no sólo simplifica cálculos locales, sino que también clarifica la teoría global del cálculo λ .

Ejercicio 2.

Demuestre que para todo λ -término N , $\lambda x.x K N \neq \lambda x.x S N$. (Nota: Recuérdese que $S \equiv \lambda xyz.xz(yz)$ y $K \equiv \lambda xy.x$).

Solución. Supongamos, por reducción al absurdo, que $\lambda x.x K N = \lambda x.x S N$ y vamos a llegar a la contradicción de que $K = S$, lo cual es imposible porque $S \neq K$ por clase.

En primer lugar, aplicamos la “regla 3” con $Z = K$, de modo que

$$\lambda x.x K N = \lambda x.x S N \Rightarrow (\lambda x.x K N)K = (\lambda x.x S N)K$$

Ahora bien, aplicando “ β -conversión” y “sustitución 1” obtenemos que

$$(\lambda x.x K N)K = KKN \quad \text{y} \quad (\lambda x.x S N)K = KSN$$

tal que, por transitividad, obtenemos

$$KKN = KSN$$

Finalmente, basta con aplicar la propia definición de K , concluyendo por transitividad que

$$KKN = K \quad \text{y} \quad KSN = S$$

$$\Rightarrow K = S$$

lo cual es una contradicción.

Ejercicio 3.

Dibuje razonadamente el grafo $G_\beta(WWW)$, donde $W \equiv \lambda xy.xyy$.

Solución. Primero, es interesante recordar la definición del grafo de β -conversión de un λ -término tal que

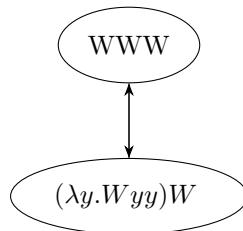
El grafo de β -conversión de un λ -término T (o grafo de reducción de un λ -término T), que denotaremos por $G_\beta(T)$, verifica:

1. Un λ -término M es un nodo de $G_\beta(T)$ si $\lambda \vdash T = M$.
2. Si M_1, M_2 son nodos distintos de $G_\beta(T)$, entonces $M_1 \not\equiv M_2$ (en particular, admite $M_1 =_\beta M_2$).
3. $n \geq 1$ aristas unen nodo M_1 a nodo M_2 (pudiendo ser $M_1 \equiv M_2$) si, y solo si, $\lambda \vdash M_1 = M_2$

Comencemos por reducir el λ -término (WWW):

$$\begin{aligned}
 WWW &\equiv (\lambda xy.xyy)WW \\
 &= (\lambda y.xyy[x := W])W \quad \text{por } \beta\text{-conversión} \\
 &= (\lambda y.Wyy)W \quad \text{por sustitución (4)} \\
 &= (Wyy[y := W]) \quad \text{por } \beta\text{-conversión} \\
 &= WWW \quad \text{por sustitución (4)}
 \end{aligned}$$

En efecto, tenemos dos λ -términos no iguales (en el sentido de \equiv). Y $G_\beta(WWW)$ es



Ejercicio 4.

Encuentre razonadamente un λ -término M tal que $G_\beta(M)$ sea exactamente:

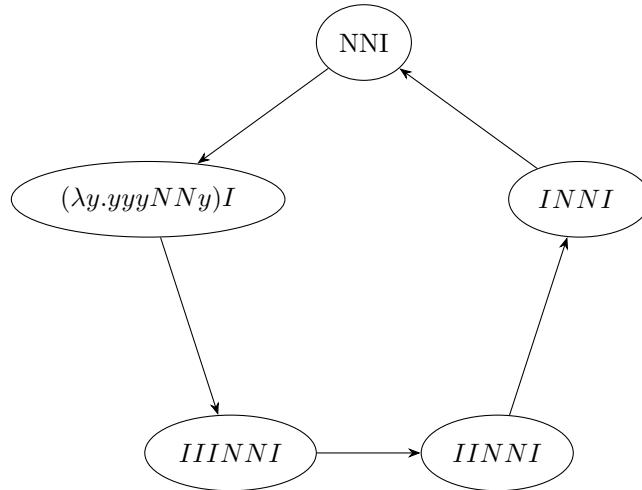
Solución. Sea el candidato $M \equiv NNI$ donde

$$N \equiv \lambda xy.yyyxxy \quad \text{e} \quad I \equiv \lambda x.x$$

y veamos que $G_\beta(M)$ es el grafo del enunciado. Sabiendo que $II \equiv (\lambda x.x)I = I$, tenemos que

$$\begin{aligned} NNI &\equiv (\lambda xy.yyyxxy)NI \\ &= (\lambda y.yyyNNy)I \\ &= IIINNI \\ &= IINNI \\ &= INNI \\ &= NNI \quad (\text{¡hemos cerrado el ciclo!}) \end{aligned}$$

En efecto, tenemos cinco λ -términos no iguales (en el sentido de \equiv). Y $G_\beta(M)$ es



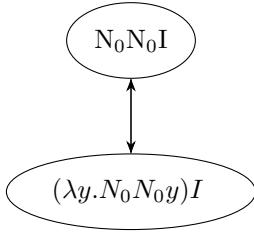
Vamos a explorar un poco más este λ -término que hemos propuesto. Presentamos las siguientes definiciones:

$$N_k = \lambda xy. \underbrace{y \cdots y}_{k \text{ veces}} xxy \quad \text{y} \quad C_{k+2} = N_k N_k I \quad \text{para } k \in \mathbb{N}$$

Así pues, estas definiciones nos conducen a plantear el siguiente lema.

$G_\beta(C_{k+2})$ es un grafo de reducción cíclico de $k + 2$ nodos para todo $k \in \mathbb{N}$.

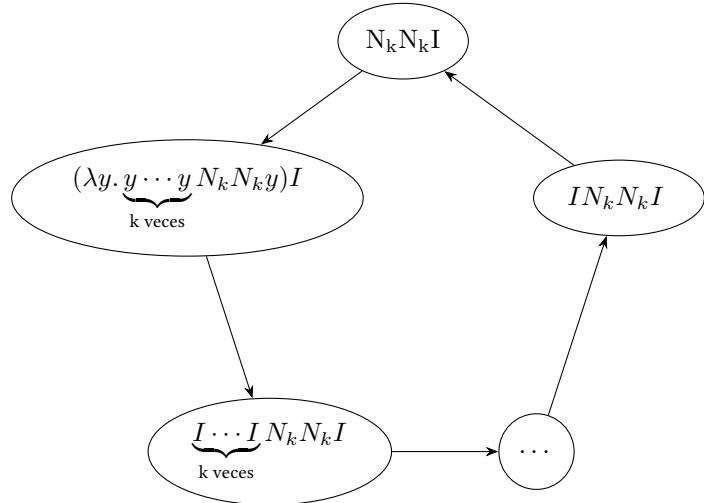
Por una parte, veamos el caso base $k = 0$ tal que $N_0 = \lambda xy.xxy$ y por ende, $C_2 = N_0 N_0 I$. Tras la primera parte del ejercicio, es inmediato comprobar que $(\lambda y.N_0 N_0 y)I = N_0 N_0 I$ se trata de $G_\beta(C_2)$ cíclico de 2 nodos. Cabe destacar que el caso base constituye un caso *degenerado*, puesto que *un ciclo de dos nodos no es más que el segmento que los une*.



Por otra parte, veamos el caso para un $k > 0$ arbitrario tal que $N_k = \lambda xy. \underbrace{y \cdots y}_{k \text{ veces}} xxy$, es decir

$$\begin{aligned}
 C_{k+2} &= N_k N_k I = (\lambda xy. \underbrace{y \cdots y}_{k \text{ veces}} xxy) N_k I = \\
 &\quad = (\lambda y. \underbrace{y \cdots y}_{k \text{ veces}} N_k N_k y) I \\
 &\quad = \underbrace{I \cdots I}_{k \text{ veces}} N_k N_k I \\
 &\quad = \underbrace{I \cdots I}_{k-1 \text{ veces}} N_k N_k I \\
 &\quad = \dots \\
 &\quad = I N_k N_k I \\
 &\quad = N_k N_k I
 \end{aligned}$$

En conclusión, es inmediato observar que hay $k + 2$ λ -términos no iguales (en el sentido de \equiv) y así podemos hacer que $G_\beta(C_{k+2})$ sea un policiclo de cualquier número **finito** de nodos mayor o igual que 2.



Ejercicio 5.

Sea el λ -término:

$$G \equiv \lambda yx.x(yx) \quad \text{y} \quad M \equiv (\lambda xy.y(xxy))(\lambda xy.y(xxy)).$$

1. Demuestre que M es un punto fijo de G .
2. Demuestre que si el combinador N es un punto fijo de G , entonces N es un operador de punto fijo.
3. Demuestre que M es un combinador de punto fijo.
4. Demuestre que si M es un combinador de punto fijo, entonces $M = GM$.

Solución.

(1) Si aplicamos el Teorema del Punto Fijo al λ -término G , sean y tal que $y \notin FV(G)$, $W = \lambda z.G(zz)$ y $X = WW$, entonces obtenemos que $X = M$ es punto fijo de G . En efecto,

$$\begin{aligned} X &\equiv \\ &\equiv WW \\ &= (\lambda yx.x(yyx))(\lambda yx.x(yyx)) \quad \text{por } (\dagger) \\ &\equiv (\lambda xy.y(xxy))(\lambda xy.y(xxy)) \quad \text{por } \alpha\text{-congruencia} \end{aligned}$$

donde (\dagger) resulta por

$$\begin{aligned} W &\equiv \\ &\equiv \lambda y.G(yy) \\ &\equiv \lambda y.(\lambda yx.x(yx))(yy) \\ &= \lambda y.((\lambda x.x(yx))[y := yy]) \quad \text{por } \beta\text{-conversión} \\ &\equiv \lambda y.\lambda x.x(yyx) \quad \text{por sustitución (1)} \\ &= \lambda yx.x(yyx) \end{aligned}$$

(2) Supongamos que el combinador N es un punto fijo de G , es decir, $N = GN$. Por esto,

$$\begin{aligned} N &= \\ &= GN \\ &\equiv (\lambda yx.x(yx))N \\ &= \lambda x.((x(yx))[y := N]) \quad \text{por } \beta\text{-conversión} \\ &\equiv \lambda x.x(Nx) \quad \text{por sustitución 1} \end{aligned}$$

Y concluimos que, dado F un λ -término,

$$\begin{aligned} NF &= \\ &= (\lambda x.x(Nx))F \\ &= (x(Nx))[x := F] \quad \text{por } \beta\text{-conversión} \\ &= F(NF) \quad \text{por sustitución 1} \end{aligned}$$

que es precisamente lo que se quería demostrar.

(3) Vamos a presentar una forma inmediata de resolver el ejercicio y otra más didáctica.

Por una parte, M es claramente un combinador porque ninguna variable ocurre libre y, además, es punto fijo de G por el apartado (1). Tenemos que verifica las hipótesis del apartado (2) y por tanto, M es un operador (o combinador) de punto fijo.

Por otra parte, si denotamos por $W := \lambda xy.y(xxy)$ tenemos que

$$\begin{aligned} M &= \\ &\equiv WW \\ &\equiv (\lambda xy.y(xxy))W \\ &= \lambda y.((y(xxy))[x := W]) \quad \text{por } \beta\text{-conversión} \\ &\equiv \lambda y.y(WWy) \quad \text{por sustitución 1} \\ &\equiv \lambda y.y(My) \quad \text{denotamos por } (\dagger) \end{aligned}$$

Y llegamos a que, dado F un λ -término,

$$\begin{aligned} MF &= \\ &= (\lambda y.y(My))F \quad \text{por } (\dagger) \\ &= (y(My))[y := F] \quad \text{por } \beta\text{-conversión} \\ &\equiv F(MF) \quad \text{por sustitución 1} \end{aligned}$$

es decir, que M es un combinador de punto fijo.

(4) Supongamos que N es un combinador de punto fijo, es decir, para todo F λ -término $NF = F(NF)$. Tomando el G del enunciado tenemos que

$$\begin{aligned} GM &= \\ &\equiv (\lambda yx.x(yx))N \\ &= \lambda x.((x(yx))[y := N]) \quad \text{por } \beta\text{-conversión} \\ &\equiv \lambda x.x(Nx) \quad \text{por sustitución 1} \end{aligned}$$

Ahora bien, si tomamos $T = GN$ llegamos a que

$$\begin{aligned} TF &= \\ &= GNF \quad \text{por regla 3} \\ &= (\lambda x.x(Nx))F \\ &= (x(Nx))[x := F] \quad \text{por } \beta\text{-conversión} \\ &= F(NF) \quad \text{por sustitución 1} \end{aligned}$$

Finalmente, aplicamos la hipótesis obteniendo que $TF = F(NF) = NF$, y de nuevo, por la regla 3, tenemos que $T = N$.

Como partíamos de que $T = GN$, efectivamente llegamos a que $N = GN$, es decir, N es punto fijo de G .

Ejercicio 6.

Considere el combinador:

$$Y \equiv \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$$

y demuestre que $GY = Y$.

Solución. Probemos que Y es un combinador de punto fijo para que, aplicando el Ejercicio 5 (d), concluyamos que $Y = GY$. Es decir, debemos probar que para cada F λ -término se tiene que $YF = F(YF)$, que es inmediato por β -conversión tal que

$$\begin{aligned} YF &\equiv \\ &\equiv (\lambda y.(\lambda x.y(xx))(\lambda x.y(xx)))F \\ &= ((\lambda x.y(xx))(\lambda x.y(xx)))[y := F] \quad \text{por } \beta\text{-conversión} \\ &\equiv (\lambda x.y(xx))[y := F](\lambda x.y(xx))[y := F] \quad \text{por sustitución 6} \\ &\equiv (\lambda x.F(xx))(\lambda x.F(xx)) \quad \text{por sustitución 1} \\ &\equiv \omega\omega \end{aligned}$$

Y ahora bastaría continuar reduciendo $YF = \omega\omega$ tal que

$$\begin{aligned} \omega\omega &= \\ &= F(xx)[x := \omega] \quad \text{por } \beta\text{-conversión} \\ &\equiv F(\omega\omega) \quad \text{por sustitución 1} \\ &\equiv F(YF) \quad \text{por el cálculo previo} \end{aligned}$$

Por lo tanto, hemos probado que Y es combinador de punto fijo, con lo cual afirmamos que $Y = GY$.

Ejercicio 7.

Considere la sucesión de combinadores $\{Y^n\}_n$ definida para todo número natural n como sigue:

$$Y^n = \begin{cases} Y, & \text{si } n = 0, \\ Y^{n-1}G, & \text{si } n > 0. \end{cases}$$

Demuestre que para todo $n \geq 0$, Y^n es un combinador de punto fijo.

Solución. Razonamos por inducción sobre $n \geq 0$. Por una parte, el caso base $[n = 0]$ es $Y^0 = Y$, que es claramente un combinador de punto fijo por el Ejercicio 6.

Por otra parte, veamos el caso inductivo $[S(k) \Rightarrow S(k + 1) \forall k > 0]$ donde la hipótesis de inducción es $S(k) = «Y^k \text{ es un combinador de punto fijo}»$. Queremos demostrar que $S(k + 1)$, y procedemos de la siguiente manera

$$Y^{k+1} \stackrel{\text{def.}}{=} Y^k G \stackrel{H.I.}{=} G(Y^k G) \stackrel{\text{def.}}{=} G(Y^{k+1})$$

En efecto, como $Y^{k+1} = G(Y^{k+1})$, entonces Y^{k+1} es un combinador de punto fijo como se quería demostrar, completando la inducción.

Ejercicio 8.

Encuentre razonadamente el CL-término $(\lambda xy.xyy)_{CL}$.

Solución. En primer lugar, escribimos el λ -término como un CL-término de tipo ‘[x].M’ (que usaremos solo para permitir la conversión $\lambda \rightarrow CL$, pero que no es un CL-término en sí mismo):

$$(\lambda xy.xyy)_{CL} \rightarrow [xy].xyy,$$

y, por definición de abstracción de múltiples variables, obtenemos que

$$[xy].xyy \equiv [x]([y].xyy)$$

de modo que ya podemos proceder a aplicar las reglas de abstracción (a), (b), (c), (f) (las que se requieran). A modo de recordatorio:

Definition 2.18 (Abstraction). Para todo término de CL llamémoslo M y toda variable x , se define por inducción un término de CL llamado $[x].M$ del siguiente modo:

- (a) $[x].M \equiv KM$ si $x \notin FV(M)$.
- (b) $[x].x \equiv I$.
- (c) $[x].Ux \equiv U$ si $x \notin FV(U)$.
- (f) $[x].UV \equiv S([x].U)([x].V)$ si no se aplica ninguna de las reglas anteriores.

*Cabe destacar que la referencia prescinde de las reglas (d) y (e); y que en $[x].UV$ usando la regla (f), V es siempre la parte derecha de la aplicación más externa del término.

Vamos a operar según las reglas (*las reglas se enuncian en orden de aplicación):

$$\begin{aligned} [x].([y].xyy) &\equiv \\ &\equiv [x].(S([y].xy)([y].y)) \quad \text{por regla (f)} \\ &\equiv [x].((Sx)I) \quad \text{por regla (c) y (b)} \\ &\equiv S([x].(Sx))([x].I) \quad \text{por regla (f)} \\ &\equiv S(S)(KI) \quad \text{por regla (c) y (a)} \\ &\equiv SSKI \end{aligned}$$

Y por último, podemos aplicar las propias definiciones de los combinadores S, K, I para simplificar la expresión tal que

$$\begin{aligned} SSKI &\equiv \\ &\equiv SI(KI) \end{aligned}$$

Sin embargo, esta reducción no produce una expresión conceptualmente más simple. Por ello, aceptamos como solución final el combinador

$SSKI$

Ejercicio 9.

Esquematice la relación entre el sistema λ y la lógica combinatoria.

1.9.1. Introducción

El sistema λ y la lógica combinatoria son dos sistemas equivalentes en poder expresivo, pero con distintos enfoques. El λ -cálculo usa variables, abstracción ($\lambda x.M$), sustitución y α -conversión mientras que la lógica combinatoria trabaja únicamente con aplicación y combinadores (S, K, I , etc.). Para profundizar más en cómo se relacionan, se hará una definición formal de la lógica combinatoria para después relacionarlo con el sistema λ .

1.9.2. Términos y combinadores de CL

Los sistemas de combinadores realizan la misma funcionalidad que el λ -cálculo sin el uso de variables ligadas. Un combinador es un término cerrado de la lógica combinatoria, es decir, un término que no contiene variables y que se construye únicamente mediante aplicación y constantes combinatorias. Hay un número infinito de combinadores de los cuales tres se denominan combinadores básicos:

- **I**, el operador identidad: $I(f) = f$
- **K**, combinador que forma funciones constantes: $(K(a))(x) = a$
- **S**, el operador de composición: $(S(f, g))(x) = f(x, g(x))$

El alfabeto de los términos de la lógica combinatoria es el compuesto por los siguientes símbolos:

- símbolos de variable v_0, v_{00}, v_{000}
- constantes atómicas, incluyendo los combinadores básicos.

El conjunto de términos de la lógica combinatoria se define inductivamente como sigue:

- a) Toda variable y toda constante atómica es un término;
- b) si X e Y son términos, entonces (XY) también lo es.

Cualquier sucesión finita de elementos del alfabeto de términos es un término de la lógica combinatoria. Por ejemplo:

$$((S(KS))K), \quad ((S(Kv_0))((SK)K))$$

son ambos términos de la lógica combinatoria, siendo el primero además un combinador.

Conociendo los términos podemos definir la relación *X ocurre en Y*, o *X es un subtérmino de Y*:

- a) X ocurre en X
- b) si X ocurre en U o en V , entonces X ocurre en (UV)

Es importante destacar que todas las variables que ocurran en un término Y pertenecen a $FV(Y)$ puesto que no hay ningún λ que las capture.

1.9.3. Sustitución y sustitución débil

Al igual que en el λ -cálculo, la lógica combinatoria también define la sustitución de términos de forma recursiva. $[U/X]Y$ se define recursivamente como el resultado de sustituir el término U por cada ocurrencia de x en Y de la siguiente forma:

- a) $[U/x]x \equiv U,$
- b) $[U/x]a \equiv a$ si $a \not\equiv x$
- c) $[U/x](VW) \equiv ([U/x]V[U/x]W).$

La lógica combinatoria carece de β reducción, por lo que la computación se define mediante reglas de reducción para **I**, **K** y **S**. Cualquier término **IX**, **KYZ** o **SXYZ** se denomina un *redex débil*. *Contraer* un redex débil significa reducir los términos de la siguiente forma:

- **IX** $\rightarrow X,$
- **KXY** $\rightarrow X,$
- **SXYZ** $\rightarrow XZ(YZ).$

Si mediante una de estas reglas U pasa a U' , se dice que U se *contrae débilmente a* U' expresado notacionalmente como

$$U \rightarrow_{1w} U'$$

Si U' se obtiene a partir de una sucesión finita de reducciones débiles, se dice que *reduce débilmente a* U' , o

$$U \rightarrow_w U'$$

1.9.4. Eliminación de la abstracción λ

Dado que la lógica combinatoria carece de abstracción λ , es necesario introducir un mecanismo que permita representar funciones con parámetros. Esto se logra mediante la construcción meta-teórica $[x].M$ que traduce la abstracción del cálculo lambda a un término de lógica combinatoria con la propiedad de que

$$([x].M)N \rightarrow_w [N/x]M.$$

El término teórico $[x].M$, que no es un CL-término en sí mismo, actuará de forma equivalente a $\lambda x.M$ y se creará como una combinación de los términos **I**, **K** y **S** de la siguiente forma:

- a) $[x].M \equiv \mathbf{KM}$ si $x \notin FV(M);$
- b) $[x].x \equiv \mathbf{I};$
- c) $[x].(Ux) \equiv U$ si $x \notin FV(U);$
- d) $[x].(UV) \equiv \mathbf{S}([x].U)([x].V)$ si no se aplican (a) ni (c).

Por ejemplo:

$$\begin{aligned} [x].(xy) &\equiv \mathbf{S}(([x].x)([x].y) && \text{por regla (f)} \\ &\equiv \mathbf{SI}(\mathbf{Ky}) && \text{por (b) y (a).} \end{aligned}$$

Así, la propiedad fundamental de la abstracción, esto es,

$$([x].M)N \rightarrow_w [N/x]M,$$

muestra que el término combinatorio $[x].M$ reproduce exactamente el comportamiento de la abstracción $\lambda x.M$ bajo aplicación, siendo la reducción débil el análogo de la β -reducción del cálculo lambda.

1.9.5. Conclusión

En consecuencia, toda abstracción del cálculo λ puede eliminarse sistemáticamente mediante la construcción $[x].M$, obteniendo un término de la lógica combinatoria formado únicamente por aplicación y combinadores básicos. Este procedimiento permite traducir cualquier término del cálculo lambda a un término combinatorio que preserva su comportamiento computacional.

Por tanto, la lógica combinatoria constituye una reformulación del cálculo λ sin variables ligadas ni abstracción explícita, pero con el mismo poder expresivo. Ambos sistemas describen la misma clase de funciones computables, diferenciándose únicamente en el enfoque sintáctico utilizado para representar la noción de función.

Capítulo 2

Unidad Temática: Lógica de Primer Orden – Los Teoremas de Incompletitud de Gödel

2.1. Introducción

Una objeción natural al estudiar los Teoremas de Incompletitud de Gödel es que la Aritmética de Peano, a partir de ahora **AP**, quizás no sea la axiomática más adecuada para describir el modelo estándar de los números naturales \mathbb{N} , y que los resultados obtenidos dependan esencialmente de esta elección.

Sin embargo, el interés de los teoremas no radica en las particularidades de **AP**, sino en el hecho de que cualquier sistema formal consistente y suficientemente expresivo para formalizar la aritmética elemental está sujeto a limitaciones análogas.

En este trabajo se adopta la Aritmética de Peano como marco formal por su claridad y potencia expresiva, introduciendo las nociones de consistencia, coherencia y completitud necesarias para formular una versión de los Teoremas de Incompletitud. Aunque la exposición se centra en **AP**, los resultados se extienden a otras teorías aritméticas que satisfacen condiciones técnicas similares, lo que pone de manifiesto el carácter general de los fenómenos de incompletitud.

2.2. Consistencia, coherencia y completitud

2.2.1. Consistencia

Sea Φ un conjunto de fórmulas de un lenguaje lógico L . Llamaremos *contradicción* a toda fórmula de la forma $\varphi \wedge \neg\varphi$, es decir, a una afirmación que sostiene simultáneamente una proposición y su negación.

Proposición (Ex falso quodlibet). Dado un conjunto de fórmulas Φ y φ una fórmula cualquier, son equivalentes:

1. $\Phi \vdash \varphi \wedge \neg\varphi$.
2. Para cada fórmula ψ , se tiene $\Phi \vdash \psi$.

Este principio expresa que, en presencia de una contradicción, el sistema deductivo colapsa. Esto es, el sistema pierde su capacidad para distinguir entre enunciados verdaderos y falsos. Por ello, diremos que Φ es *inconsistente* si demuestra alguna contradicción, y *consistente* en caso contrario.

2.2.2. Coherencia

La relación entre demostrabilidad sintáctica y verdad semántica se recoge en el siguiente resultado:

Teorema (Teorema de la Coherencia). Sea Φ un conjunto de fórmulas y \mathcal{M} un modelo de Φ tal que $\mathcal{M} \models \Phi$. Entonces, para toda fórmula φ se tiene:

$$\Phi \vdash \varphi \Rightarrow \mathcal{M} \models \varphi.$$

Este teorema garantiza que si disponemos de un modelo en el que todas las hipótesis son verdaderas, entonces toda conclusión obtenida mediante demostración formal a partir de ellas también es verdadera en dicho modelo. En otras palabras, ninguna fórmula falsa en un modelo puede ser demostrada a partir de hipótesis que son verdaderas en ese mismo modelo, lo que asegura la fiabilidad semántica del sistema deductivo.

2.2.3. Completitud

Una teoría T se dice *completa* si, para toda fórmula cerrada φ , se cumple que

$$T \vdash \varphi \text{ o bien } T \vdash \neg\varphi.$$

es decir, si toda sentencia formulable en el lenguaje de la teoría es decidible dentro de ella mediante demostración formal.

El siguiente resultado clásico conecta consistencia, modelos y demostrabilidad:

Teorema (Teorema de Completitud). Sea Φ un conjunto de fórmulas. Si Φ es consistente, entonces tiene un modelo M . Además, sea σ una sentencia, si para cada modelo M de Φ se verifica que $M \models \sigma$, entonces $\Phi \vdash \sigma$.

Este resultado puede interpretarse como la afirmación de que toda verdad semántica válida en todos los modelos de una teoría consistente es, en efecto, demostrable a partir de sus axiomas. De este modo, el teorema de completitud garantiza que el sistema deductivo es suficientemente potente para capturar todas las consecuencias semánticas universales de la teoría.

No hay contradicción con los Teoremas de Incompletitud de Gödel porque el Teorema de Completitud se refiere a consecuencias válidas en *todos los modelos* de una teoría, mientras que la incompletitud afirma la existencia de sentencias verdaderas en el modelo estándar que no lo son en todos los modelos. Por tanto, dichas sentencias no son consecuencias semánticas universales y la completitud no exige que sean demostrables.

2.3. Axiomática de Peano

Los teoremas de incompletitud de Gödel exigen una aritmética con suma, multiplicación y cuantificación sobre números para poder codificar fórmulas y demostraciones. Esto es exactamente lo que proporciona la axiomática de Peano, publicada en 1889 por Giuseppe Peano.

2.3.1. Axiomas de Peano

En esta memoria se adopta la axiomatización de Peano en lógica de primer orden con símbolos:

- 0) Símbolo de constante que llamaremos 0.
- s) Símbolo de una función de aridad 1 que llamaremos sucesor.
- +) Símbolo de una función de aridad 2 que llamaremos suma.

.) Símbolo de una función de aridad 2 que llamaremos multiplicación.

La teoría **AP** está formada por los siguientes axiomas:

- AP0: $\neg\exists x, (s(x) = 0)$
- AP1: $\forall x\forall y, (s(x) = s(y) \rightarrow x = y)$
- AP2: $\forall x, (x + 0 = x)$
- AP3: $\forall x\forall y, (x + s(y) = s(x + y))$
- AP4: $\forall x, (x \cdot 0 = 0)$
- AP5: $\forall x\forall y, (x \cdot s(y) = (x \cdot y) + x)$

2.3.2. Aritmética y propiedades de AP

A partir de los axiomas de Peano, podemos definir nuevas constantes, funciones y relaciones. Interpretaremos las fórmulas aritméticas de la teoría de Peano en el sentido usual sobre los números naturales \mathbb{N} . Por ejemplo, podemos definir la relación de orden como:

$$\begin{aligned} x \leq y &\iff \exists r (x + r = y), \\ x < y &\iff x \leq y \wedge x \neq y, \\ x \geq y &\iff y \leq x, \\ x > y &\iff y < x. \end{aligned}$$

Teorema (Propiedades básicas del orden en **AP**). Se verifican en la teoría **AP** las siguientes propiedades del orden:

- a) $\mathbf{AP} \vdash \forall x (x \leq x)$
- b) $\mathbf{AP} \vdash \forall x\forall y ((x \leq y \wedge y \leq x) \rightarrow x = y)$
- c) $\mathbf{AP} \vdash \forall x\forall y\forall z ((x \leq y \wedge y \leq z) \rightarrow (x \leq z))$
- d) $\mathbf{AP} \vdash \forall x (0 \leq x)$
- e) $\mathbf{AP} \vdash \forall x\forall y (x \leq y \vee y \leq x)$
- f) $\mathbf{AP} \vdash \forall x\forall y (x < y \vee x = y \vee y < x)$
- g) $\mathbf{AP} \vdash \forall x\forall y\forall z (x \leq y \rightarrow x + z \leq y + z)$

Además de las propiedades básicas del orden, la teoría **AP** permite formalizar y demostrar propiedades elementales de la suma y el producto que reflejan su comportamiento usual en los números naturales. En particular, dentro de **AP** se pueden demostrar, entre otras, las siguientes propiedades, cuya demostración es de poco interés en esta memoria por lo que hemos decidido omitirlas:

- Asociatividad de la suma:

$$\mathbf{AP} \vdash \forall x\forall y\forall z (x + (y + z) = (x + y) + z),$$

- Propiedad cancelativa de la suma:

$$\mathbf{AP} \vdash \forall x\forall y\forall z (x + y = x + z \rightarrow y = z)$$

- Propiedades de las fórmulas de desigualdad:

$$\begin{aligned}\mathbf{AP} &\vdash \forall x \forall y (x < sy \leftrightarrow x \leq y) \\ \mathbf{AP} &\vdash \forall x \forall y (x < y \leftrightarrow sx \leq y)\end{aligned}$$

- Propiedades de la divisibilidad:

$$\begin{aligned}\mathbf{AP} &\vdash \forall x \forall y \forall z \leq y (x \mid y \wedge x \mid z \rightarrow x \mid y \pm z) \\ \mathbf{AP} &\vdash \forall x \forall y \forall z (x \mid y \rightarrow x \mid yz)\end{aligned}$$

En la demostración de los teoremas también está implícito el concepto de coprimalidad:

$$\begin{aligned}\mathbf{AP} &\vdash \forall x \forall y \forall k (k \mid x \wedge \text{coprimos}(x, y) \rightarrow \text{coprimos}(k, y)) \\ \mathbf{AP} &\vdash \forall k \forall y \neq 0 \forall j (k \mid y \rightarrow \text{coprimos}(1 + (j + k)y, 1 + jy)) \\ \mathbf{AP} &\vdash \forall x \forall y (\text{coprimos}(x, y) \leftrightarrow (x \neq 0 \wedge y \neq 0 \wedge \forall z ((z \mid x \wedge z \mid y) \rightarrow z = 1)))\end{aligned}$$

La codificación aritmética utilizada en la numeración de Gödel se basa en la construcción de múltiplos comunes:

$$\mathbf{AP} \vdash \forall m \exists y \forall k ((k \neq 0 \wedge k \leq m) \rightarrow k \mid y)$$

Finalmente, veamos como podemos definir funciones en **AP** mediante fórmulas. Diremos que una función

$$F(x_1, \dots, x_n) = y \iff \varphi(x_1, \dots, x_n, y)$$

es *definible en AP* si existe una fórmula φ tal que, para cada elección de los argumentos x_1, \dots, x_n , existe un único valor y que satisface $\varphi(x_1, \dots, x_n, y)$. En este caso diremos que φ es funcional.

Este concepto permite construir el máximo de un dominio finito de una función definible en **AP** como expresa el siguiente resultado.

Sea F una función definible en **AP**. Entonces:

$$\mathbf{AP} \vdash \forall k > 0 \exists! x (\exists i < k (F(i) = x) \wedge \forall j < k (F(j) \leq x))$$

Las propiedades anteriores garantizan que la teoría **AP** reproduce internamente el comportamiento aritmético habitual de los números naturales. Este hecho es esencial para poder definir, dentro de la propia teoría, funciones y relaciones aritméticas para codificar fórmulas, secuencias finitas y demostraciones, lo cual será la base de la numeración de Gödel.

2.4. Numeración de Gödel

Hasta ahora hemos considerado, por un lado, la sintaxis de la teoría de Peano y, por otro, su interpretación semántica sobre los números naturales. El paso fundamental para demostrar los teoremas de incompletitud consiste en poder hablar de la propia teoría aritmética dentro del lenguaje de la aritmética. A este proceso se le denomina *gödelización*.

La idea general es asignar números naturales a símbolos, fórmulas y demostraciones, de forma que las expresiones formales puedan traducirse en códigos numéricos.

Primer problema: exponentes variables. Aunque para cada exponente fijo k puede escribirse una fórmula que exprese $x = p^k$ para un primo p , la teoría de Peano no permite escribir directamente una fórmula de la forma

$$\exists p \exists k (x = p^k),$$

ya que esto requeriría cuantificar sobre exponentes arbitrarios y disponer de una noción interna de potenciación con exponente variable. La aritmética de Peano no contiene primitivamente una función de potencia, por lo que no puede expresar directamente este tipo de propiedades.

Segundo problema: factorizaciones de longitud arbitraria. De manera análoga, aunque es posible expresar propiedades como “ x es primo” o “ x es producto de dos primos”, no puede escribirse una fórmula que exprese que todo número mayor que uno se descompone como producto de una cantidad arbitraria (finita pero indeterminada) de números primos.

Tercer problema: sucesiones finitas de números naturales. El obstáculo más general es que la aritmética de Peano no puede hablar directamente de sucesiones finitas arbitrarias de números naturales. Sin embargo, para poder codificar fórmulas y demostraciones es esencial poder representar listas finitas de símbolos, números o fórmulas dentro del propio lenguaje aritmético.

Solución. La solución de Gödel consiste en representar cualquier fórmula finita de números naturales mediante un único número natural. Para ello, se define una función inyectiva para identificar cada fórmula de nuestra teoría:

$$f : \mathbb{N}^3 \rightarrow \mathbb{N},$$

$$(p, q, s) \mapsto 2^p 3^q 5^s.$$

Si aplicamos recursivamente esta función, podemos codificar todas las expresiones utilizando las siguientes reglas:

Símbolo	$\#\zeta$	τ	$\#\tau$	φ	$\#\varphi$
0	2^0	v_n	$2 \cdot n + 3$	$\tau_0 = \tau_1$	$2^{\#=} \cdot 3^{\#\tau_0} \cdot 5^{\#\tau_1}$
s	2^1	st	$2^{\#s} \cdot 3^{\#t}$	$\neg\psi$	$2^{\#\neg} \cdot 3^{\#\psi}$
$+$	2^2	$t_0 + t_1$	$2^{\#t_0} \cdot 3^{\#+} \cdot 5^{\#t_1}$	$\psi_0 \wedge \psi_1$	$2^{\#\wedge} \cdot 3^{\#\psi_0} \cdot 5^{\#\psi_1}$
\cdot	2^3	$t_0 \cdot t_1$	$2^{\#t_0} \cdot 3^{\cdot} \cdot 5^{\#t_1}$	$\psi_0 \vee \psi_1$	$2^{\#\vee} \cdot 3^{\#\psi_0} \cdot 5^{\#\psi_1}$
$=$	2^4			$\psi_0 \rightarrow \psi_1$	$2^{\#\rightarrow} \cdot 3^{\#\psi_0} \cdot 5^{\#\psi_1}$
\neg	2^5			$\exists x \psi$	$2^{\#\exists} \cdot 3^{\#x} \cdot 5^{\#\psi}$
\wedge	2^6			$\forall x \psi$	$2^{\#\forall} \cdot 3^{\#x} \cdot 5^{\#\psi}$
\vee	2^7				
\rightarrow	2^8				
\exists	2^9				
\forall	2^{10}				

En esta tabla, τ y φ se utilizan como *meta-variables*. El símbolo τ denota un *término* del lenguaje (por ejemplo, variables, constantes y expresiones formadas mediante los símbolos funcionales), mientras que φ (y ψ) denota una *fórmula*, es decir, una expresión con valor de verdad.

Por ejemplo, dada una fórmula

$$\varphi \equiv v_o = v_1 \wedge v_2,$$

su número de Gödel es

$$\#\varphi \equiv 2^{16} \cdot 3^3 \cdot 5^{2^5 \cdot 3^5 \cdot 5^7}$$

De este modo, una sucesión finita de números queda representada por un único número natural, y las propiedades de dicha sucesión pueden traducirse en propiedades aritméticas de su código.

Usaremos la notación

$$\Gamma \zeta \vdash : \equiv \# \zeta,$$

para denotar el número de Gödel del símbolo, término o fórmula ζ .

Con esta notación, se definen las siguientes funciones aritméticas, que construyen números de Gödel de expresiones compuestas:

$$\begin{aligned} \text{suc}(n) &: \equiv 2^{\Gamma s \vdash} \cdot 3^n, \\ \text{sum}(n, m) &: \equiv 2^{\Gamma + \vdash} \cdot 3^n \cdot 5^m, \\ \text{mult}(n, m) &: \equiv 2^{\Gamma \cdot \vdash} \cdot 3^n \cdot 5^m, \\ \text{ig}(t, t') &: \equiv 2^{\Gamma = \vdash} \cdot 3^t \cdot 5^{t'}, \\ \text{not}(f) &: \equiv 2^{\Gamma \neg \vdash} \cdot 3^f, \\ \text{and}(f, f') &: \equiv 2^{\Gamma \wedge \vdash} \cdot 3^f \cdot 5^{f'}, \\ \text{or}(f, f') &: \equiv 2^{\Gamma \vee \vdash} \cdot 3^f \cdot 5^{f'}, \\ \text{imp}(f, f') &: \equiv 2^{\Gamma \rightarrow \vdash} \cdot 3^f \cdot 5^{f'}, \\ \text{ex}(v, f) &: \equiv 2^{\Gamma \exists \vdash} \cdot 3^v \cdot 5^f, \\ \text{pt}(v, f) &: \equiv 2^{\Gamma \forall \vdash} \cdot 3^v \cdot 5^f. \end{aligned}$$

2.5. Teoremas de Incompletitud de Gödel

2.5.1. Lema de la diagonalización

Teorema (Lema de la diagonalización). Dada una $\varphi(v)$ una fórmula de **AP** con una única variable libre que no está ligada en φ , existe una única sentencia de **AP** llamémosla σ_φ tal que: . Existe una sentencia σ tal que:

$$\sigma_\varphi \leftrightarrow_{\text{AP}} \varphi(\Gamma \sigma_\varphi \vdash).$$

Este lema permite construir sentencias autorreferenciales mediante su propio número de Gödel, constituyendo la herramienta fundamental de los Teoremas de Incompletitud de Gödel.

2.5.2. Primer Teorema de Incompletitud

Teorema (Primer Teorema de Incompletitud). La **AP** es incompleta.

Demostración. Tomando $\varphi(v_0) := \neg \text{dem}(v_0)$ en el lema de diagonalización sabemos que existe σ tal que:

$$\sigma \leftrightarrow_{\text{AP}} \neg \text{dem}(\Gamma \sigma \vdash)$$

Ahora, si asumimos, por reducción al absurdo, que **AP** es completa, tenemos dos posibilidades:

- Si **AP** $\vdash \sigma$, esto implica que **AP** $\vdash \neg \text{dem}(\Gamma \sigma \vdash)$. Además, se puede probar que también se verifica **AP** $\vdash \text{dem}(\Gamma \sigma \vdash)$. Es decir, **AP** $\vdash \text{dem}(\Gamma \sigma \vdash) \wedge \neg \text{dem}(\Gamma \sigma \vdash)$.
- Si **AP** $\vdash \neg \sigma$, esto implica que **AP** $\vdash \neg (\neg \text{dem}(\Gamma \sigma \vdash))$, es decir, **AP** $\vdash \text{dem}(\Gamma \sigma \vdash)$ y $\mathbb{N} \models \text{dem}(\Gamma \sigma \vdash)$. Por tanto, tenemos **AP** $\vdash \sigma$, con lo cual **AP** $\sigma \wedge \neg \sigma$

En conclusión, si **AP** es consistente, entonces **AP** $\nvDash \sigma$ y **AP** $\nvDash \neg \sigma$. Esto es, **AP** es incompleta. \square

En efecto, este teorema establece que ningún sistema formal consistente y lo suficiente expresivo como **AP** puede demostrar todas las verdades aritméticas. La incompletitud surge al construir una sentencia σ que, mediante el lema de diagonalización, afirma su propia no demostrabilidad. Esto implica que, si **AP** fuese capaz de decidir σ , entonces generaría una contradicción, mostrando que la consistencia implica la existencia de sentencias indecibiles.

2.5.3. Segundo Teorema de Incompletitud

Teorema (Segundo Teorema de Incompletitud). La **AP** no puede probar su propia consistencia.

Demostración. Por el lema de diagonalización, existe σ tal que

$$\sigma \Leftrightarrow_{\text{AP}} \neg \text{dem}(\Gamma \sigma^\neg)$$

Asumimos que $\text{AP} \vdash \neg \text{dem}(\Gamma \sigma \wedge \neg \sigma^\neg)$, es decir, que **AP** puede demostrar que no existe una demostración de una contradicción. Para llegar a una contradicción, veamos por un resultado previo que

$$\begin{aligned} (\dagger) \quad \text{dem}(\Gamma \sigma \wedge \neg \sigma^\neg) &\Leftrightarrow_{\text{AP}} \text{dem}(\Gamma \sigma^\neg) \wedge \neg \text{dem}(\Gamma \neg \sigma^\neg) \\ &\Leftrightarrow_{\text{AP}} \text{dem}(\Gamma \sigma^\neg) \wedge \text{dem}(\Gamma \text{dem}(\Gamma \sigma^\neg)^\neg) \end{aligned}$$

Usando una condición de derivabilidad de Hilbert-Bernays-Löb y la Tautología de «Introducción de la Conjunción bajo Antecedente Común» obtenemos

$$\text{AP} \vdash \text{dem}(\Gamma \sigma^\neg) \rightarrow (\text{dem}(\Gamma \sigma^\neg) \wedge \text{dem}(\Gamma \text{dem}(\Gamma \sigma^\neg)^\neg))$$

que junto al Axioma «Eliminación de la Conjunción» demuestra

$$\text{dem}(\Gamma \sigma^\neg) \Leftrightarrow_{\text{AP}} \text{dem}(\Gamma \sigma^\neg) \wedge \text{dem}(\Gamma \text{dem}(\Gamma \sigma^\neg)^\neg)$$

De esta forma, volviendo a añadir la negación a (\dagger) , obtenemos

$$\neg \text{dem}(\Gamma \sigma \wedge \neg \sigma^\neg) \Leftrightarrow_{\text{AP}} \neg \text{dem}(\Gamma \sigma^\neg) \Leftrightarrow_{\text{AP}} \sigma$$

donde, recordamos, la última equivalencia se da por el Lema de Diagonalización. Por lo tanto, lo que hemos asumido implica que $\text{AP} \vdash \sigma$. Sin embargo, hemos demostrado en el Primer Teorema de Incompletitud que $\text{AP} \not\vdash \sigma$, lo cual es una contradicción. \square

Intuitivamente, este teorema se desprende un hecho todavía más fuerte. A saber, aunque **AP** sea consistente, el propio sistema no puede demostrar que sea consistente sin generar contradicciones. La demostración utiliza una sentencia autorreferencial construida por el Lema de Diagonalización y muestra que, si **AP** pudiera probar su consistencia, también podría probar una sentencia que es indecidible por el Primer Teorema de Incompletitud.

Aunque el Segundo Teorema de Incompletitud afirma que **AP** no puede probar su propia consistencia, la consistencia de **AP** sí *puede demostrarse desde un punto de vista externo*. Tal es el caso de usar el modelo estándar de \mathbb{N} donde cada axioma de **AP** es verdadero, y si **AP** no fuera consistente entonces algún axioma sería falso en \mathbb{N} ; o bien podríamos formalizar **AP** dentro de la teoría de conjuntos ZFC (Zermelo-Fraenkel con Axioma de Elección, la utiliza en la matemática moderna).

2.6. Conclusión

En conclusión, los Teoremas de Incompletitud de Gödel muestran que cualquier sistema consistente y suficientemente expresivo, como la Aritmética de Peano, es incompleto y no puede demostrar su propia consistencia. En efecto, estas limitaciones son inherentes a la formalización de la aritmética y se extienden a otras teorías aritméticas con propiedades similares.

Capítulo 3

Ejercicios de Programación en Haskell

Ejercicio 2.

Listing 3.1: Cálculo del área de un triángulo según la Fórmula de Herón

```
1 import Data.List (sort)
2
3 heron :: (Ord a, Fractional a) => a -> a -> a -> Maybe a
4 heron a b c
5   | not $ isTriangle a b c = Nothing
6   | otherwise               = Just . newton f df $ x0
7   where
8     s = (a+b+c)/2
9     t = s*(s-a)*(s-b)*(s-c)
10    f x = x*x-t
11    df x = 2*x
12    x0 = max 1 $ t/2
13
14 isTriangle :: (Ord a, Num a) => a -> a -> a -> Bool
15 isTriangle a b c = x>0 && x+y>z
16   where [x,y,z] = sort [a,b,c]
17
18 newton :: (Ord a, Fractional a) => (a -> a) -> (a -> a) -> a -> a
19 newton f df x0 = until (isCloseEnough 1e-12) nextIteration x0
20   where
21     isCloseEnough eps x = abs (f x) < eps
22     nextIteration x = x - f x / df x
```

Ejercicio 3.

Listing 3.2: Criptosistema de Vigenère

```

1 import Data.Char (isLetter, toUpper, ord, chr)
2 import Data.List (group, sort, nub, elemIndices, sortOn, maximumBy)
3 import Control.Applicative (ZipList(..))
4
5 class Monoid g => Group g where
6   invert :: g -> g
7
8 ----- Datos y Algebras -----
9
10 data Mode = Encrypt | Decrypt
11
12 newtype Shift = Shift Int deriving (Eq, Show)
13
14 instance Semigroup Shift where
15   (Shift a) <*> (Shift b) = Shift ((a + b) `mod` 26)
16
17 instance Monoid Shift where
18   mempty = Shift 0
19
20 instance Group Shift where
21   invert (Shift n) = Shift (-n `mod` 26)
22
23 ----- Logica de transformacion -----
24
25 applyMode :: Mode -> Shift -> Shift
26 applyMode Encrypt = id
27 applyMode Decrypt = invert
28
29 normalize :: String -> String
30 normalize str = toUpper <$> filter isLetter str
31
32 shiftChar :: Char -> Shift -> Char
33 shiftChar c (Shift n) = chr (ord 'A' + (ord c - ord 'A' + n) `mod` 26)
34
35 ----- Algoritmo de Vigenere -----
36
37 vigenereTransform :: Mode -> String -> String -> String
38 vigenereTransform mode key str = getZipList $
39   shiftChar <$> ZipList (str)
40   <*> ZipList (cycle shifts)
41   where
42     shifts = map toShift $ key
43     toShift c = applyMode mode $ Shift (ord c - ord 'A')
44
45 encrypt :: String -> String -> String
46 encrypt key str = vigenereTransform Encrypt cKey cStr
47   where
48     cKey = normalize key
49     cStr = normalize str
50
51 decrypt :: String -> String -> String
52 decrypt = vigenereTransform Decrypt
53
54 ----- Metodo de Kasiski -----
55
56 factorize :: Int -> [Int]
57 factorize n = sort . concat $ [ [x, n `div` x] | x <- [1..limit], n `mod` x == 0 ]
58   where
59     limit = floor . sqrt . fromIntegral $ n
60

```

```

61  possibleLengths :: String -> [Int]
62  possibleLengths str = concatMap (take 1) . sortOn (negate . length) . group $
63      ↪ sortedFactors
64  where
65      trigrams = zip3 str (drop 1 str) (drop 2 str)
66      dists = do
67          (x:_)<- filter ((>1) . length) $ group $ sort trigrams
68          let indices = elemIndices x trigrams
69          zipWith (-) (drop 1 indices) indices
70      sortedFactors = sort . filter (\k -> k > 1 && k <= 20) . concatMap factorize $ dists
71  ----- Indice de Coincidencia -----
72
73  buildSubstrings :: Int -> String -> [String]
74  buildSubstrings k msg = [ [ msg !! j | j <- [i, i+k .. length msg - 1] ] | i <- [0 ..
75      ↪ k-1] ]
76
77  frequencies :: String -> [Int]
78  frequencies s = map length . group . sort $ s
79
80  indexOfCoincidence :: String -> Double
81  indexOfCoincidence s
82      | n <= 1     = 0
83      | otherwise = fromIntegral acc / fromIntegral (n * (n - 1))
84  where
85      freqs = frequencies s
86      acc   = sum [ f * (f - 1) | f <- freqs ]
87      n     = sum freqs
88
89  indexOfCoincidenceTest :: Int -> String -> [Double]
90  indexOfCoincidenceTest k message = map indexOfCoincidence substrings
91  where
92      substrings = buildSubstrings k (message)
93
94  textoCifrado :: String
95  textoCifrado =
96      "UECWKDVLOTTVACKTPVGZQMDAMRNPDUXLBUICAMRHOECKBHSQQLVIWO\
97      \FFEAILPNTESMLDRUURIFAEQTPXADWIABLACCRPBHSRZIVQWOFROGTT\
98      \NNXEVIVIBPDTTGAHVIACLAYKGJIEQHGECMESNNOCTHSGGNVWTQHKBPR\
99      \HMVUOYWLIAFIRIGDBOEBQLIGWARQHNLOISQKEPEIDVXXNETPAXNZGDX\
100     \WWEYQCTIGONNGJVHSQGEATHSYGSDVVVOAQCXLHSPQMDMETRTMDUXTEQQ\
101     \JMFAEEAAIMEZREGIMUECICBXRVQRSMENNWTXTNSRNPZHMVRVDYNECG\
102     \SPMEAVENTXKEQKCTTHSPCMQQHSQGTXMFPBGLWQZRBOEIZHQHGRTOBSG\
103     \TATTZRFNOSMLEDWESIWRNAPBFOFHEGIXLFVOGUZLNUSRCRAZGZRTTA\
104     \YFEHKHMQNTZLENPUCBAYCICUBNRPCXIWEYCSIMFPRUTPLXSYCBGCC\
105     \UYCQJMIKEGTUBRHVATTLERKACBXQHGPDEANNTJZTDRNSDTFEVPDXK\
106     \TMVNAIQMUQNOHKKOAQMKTBFOSUTUXPRTMXBNPCLRCEAEIOIAWGGVVUS\
107     \GIOEWLIQFOZKSPVMEBLOLHDVCYSMGOPJEFCXMRLUIGDXNCCRPMLECEWT\
108     \PZMOQQSAWLPHPTDAWEYJOGQSOAVERCTNQQEAVTUGKLJAXMRTGTIEAFW\
109     \PTZYIPKESMEAFCGJILSBPLDABNFVRJUXNGQSWIUGWAAMLDRNPNPDGXN\
110     \PTTGLUHUOBMXSPQNDKBDBTEECLECGRDPTYBVRDATQHKQJMKEFROCLXN\
111     \FKNSCWNNAHXTRGKCJTTRRUEMQZEAEIIPAWEYPAJBBLHUEHMVUNFRPV\
112     \EDWEKMHRREOGZBDBROCGGANIUYIBNZQVXTGORUUUCUTNBOEIZHEFWNB\
113     \GOZTGWXNRHERBPHGSIWXNPQMJVBCNEIDVVOAGLPONAPWYPXKEFKOC\
114     \MQTRTIIDZBNQKCPLTNOBXMGNLNRRDNNNQKDPLTLSUTAXMNPTXMGZKA\
115     \EIKAGQ"
116  ----- Mutual Index Of Coincidence -----
117
118  spanishFreq :: [(Char, Double)]
119  spanishFreq =
120      [ ('A', 0.1253), ('B', 0.0142), ('C', 0.0468), ('D', 0.0586),
121      , ('E', 0.1368), ('F', 0.0069), ('G', 0.0101), ('H', 0.0070),
122      , ('I', 0.0625), ('J', 0.0044), ('K', 0.0002), ('L', 0.0497) ]

```

```

123 , ('M',0.0315),('N',0.0671),('O',0.0868),('P',0.0251)
124 , ('Q',0.0088),('R',0.0687),('S',0.0798),('T',0.0463)
125 , ('U',0.0393),('V',0.0090),('W',0.0001),('X',0.0022)
126 , ('Y',0.0090),('Z',0.0052)
127 ]
128
129 relativeFreq :: Char -> String -> Double
130 relativeFreq c s = fromIntegral (length (filter (== c) s)) / fromIntegral (length s)
131
132 mutualIndex :: String -> Double
133 mutualIndex s = sum [ p * relativeFreq c s | (c,p) <- spanishFreq ]
134
135 micForShift :: String -> Shift -> Double
136 micForShift s b = mutualIndex (map (`shiftChar` b) s)
137
138 bestShift :: String -> Shift
139 bestShift s = snd $ maximumBy cmp results
140   where
141     results = [ (micForShift s b, b) | b <- map Shift [0..25] ]
142     cmp (x,_) (y,_) = compare x y
143
144 shiftToKeyChar :: Shift -> Char
145 shiftToKeyChar b = shiftChar 'A' (invert b)
146
147 mutualIndexTest :: Int -> String -> String
148 mutualIndexTest k msg = map (shiftToKeyChar . bestShift) substrings
149   where
150     substrings = buildSubstrings k (msg)
151
152 ----- Attack -----
153
154 attackVigenere :: String -> (String, String)
155 attackVigenere cipherText = (clave, textoDescifrado)
156   where
157     normText = normalize cipherText
158     candidates = possibleLengths normText
159     bestK = maximumBy compareIndexOfCoincidence candidates
160     clave = mutualIndexTest bestK normText
161     textoDescifrado = decrypt clave normText
162     compareIndexOfCoincidence k1 k2 = compare (avgIC k1) (avgIC k2)
163   where
164     avgIC k = let result = indexOfCoincidenceTest k normText
165           in sum result / fromIntegral (length result)

```

Bibliografía

- [1] De Bruijn, N. G. (1972). *Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church–Rosser theorem*. *Indagationes Mathematicae (Proceedings)*, 75(5), 381–392. [https://doi.org/10.1016/1385-7258\(72\)90034-0](https://doi.org/10.1016/1385-7258(72)90034-0)
- [2] Venturini Zilli, M. (1984). *Reduction graphs in the lambda calculus*. *Theoretical Computer Science*, 29(3), 251–275. [https://doi.org/10.1016/0304-3975\(84\)90002-1](https://doi.org/10.1016/0304-3975(84)90002-1)
- [3] Hindley, J. R., & Seldin, J. P. (2008). *Lambda-calculus and combinators: An introduction* (2nd ed.). Cambridge University Press. (Secciones relevantes: Section 2C, *Abstraction in CL*, pp. 26–29; Chapter 9, *Correspondence between λ and CL*, pp. 92–106.)
- [4] García López, J. (2023). *Los teoremas de incompletitud de Gödel (Gödel's Incompleteness Theorems)*. Trabajo de Fin de Grado, Grado en Matemáticas, Universidad de Cantabria. <https://hdl.handle.net/10902/29859>.