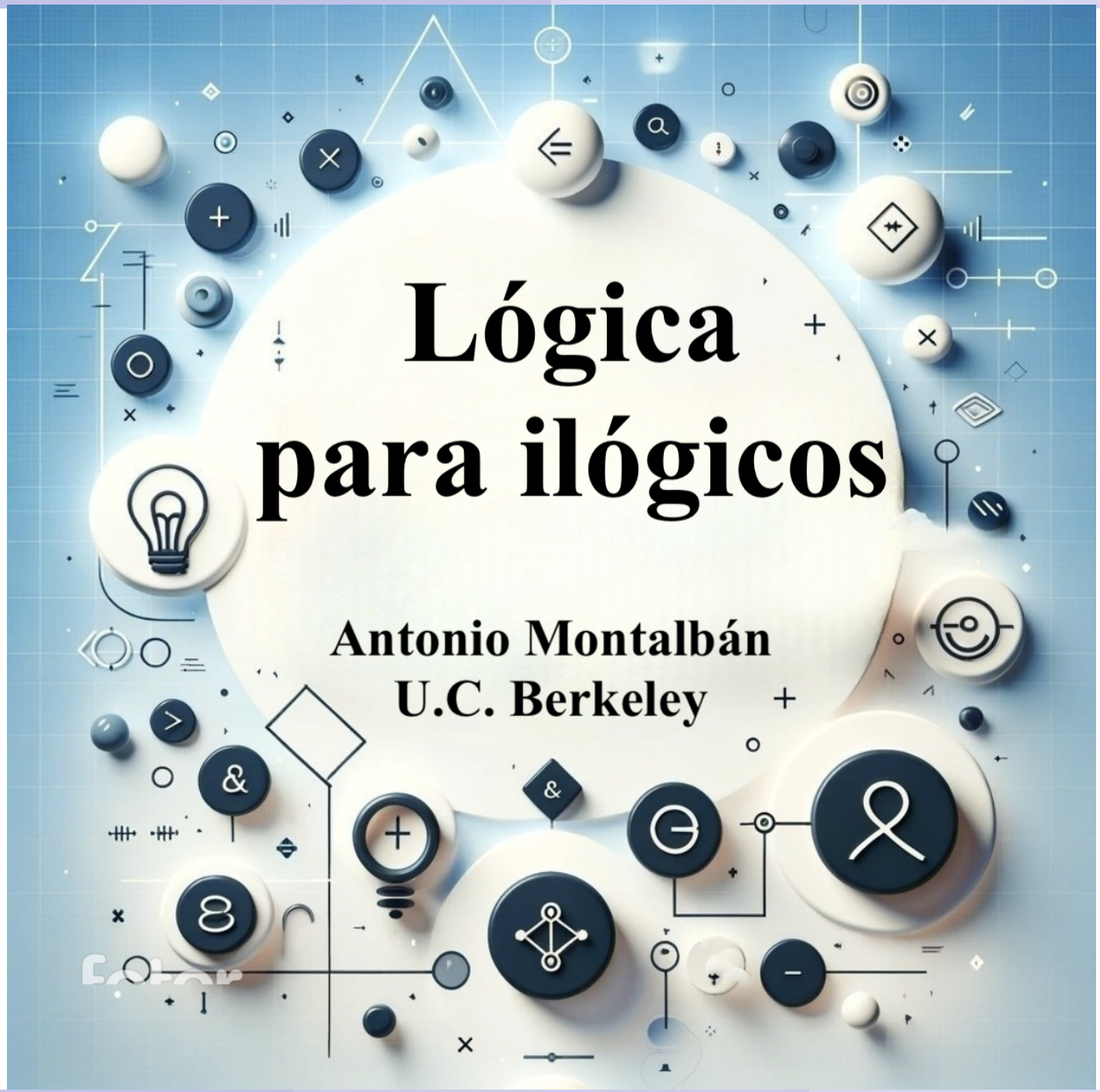

Lógica
para ilógicos
Antonio Montalbán
U.C. Berkeley

Paradojas

Paradojas

Paradoja del barbero, [Bernard Russell, 1901]:
Definamos $R = \{x | x \notin x\}$.

Paradojas

Paradoja del barbero, [Bernard Russell, 1901]:
Definamos $R = \{x | x \notin x\}$. Luego, tenemos que $R \in R \iff R \notin R$.

Paradojas

Paradoja del barbero, [Bernard Russell, 1901]:
Definamos $R = \{x | x \notin x\}$. Luego, tenemos que $R \in R \iff R \notin R$.

Se deduce que el conjunto de todos los conjuntos no existe. [Gregory Cantor]

Paradojas

Paradoja del barbero, [Bernard Russell, 1901]:
Definamos $R = \{x | x \notin x\}$. Luego, tenemos que $R \in R \iff R \notin R$.

Se deduce que el conjunto de todos los conjuntos no existe. [Gregory Cantor]

Paradoja de Berry, [Bernard Russell, 1908]:
Sea n el menor número natural que **no** puede ser definido en español en menos de 20 palabras.

Paradojas

Paradoja del barbero, [Bernard Russell, 1901]:
Definamos $R = \{x | x \notin x\}$. Luego, tenemos que $R \in R \iff R \notin R$.

Se deduce que el conjunto de todos los conjuntos no existe. [Gregory Cantor]

Paradoja de Berry, [Bernard Russell, 1908]:
Sea n el menor número natural que **no** puede ser definido en español en menos de 20 palabras. Acabamos de definirlo con 18 palabras.

Paradojas

Paradoja del barbero, [Bernard Russell, 1901]:
Definamos $R = \{x | x \notin x\}$. Luego, tenemos que $R \in R \iff R \notin R$.

Se deduce que el conjunto de todos los conjuntos no existe. [Gregory Cantor]

Paradoja de Berry, [Bernard Russell, 1908]:
Sea n el menor número natural que **no** puede ser definido en español en menos de 20 palabras. Acabamos de definirlo con 18 palabras.

Se deduce que el idioma natural no es suficientemente formal.

Paradojas

Paradoja del barbero, [Bernard Russell, 1901]:
Definamos $R = \{x | x \notin x\}$. Luego, tenemos que $R \in R \iff R \notin R$.

Se deduce que el conjunto de todos los conjuntos no existe. [Gregory Cantor]

Paradoja de Berry, [Bernard Russell, 1908]:
Sea n el menor número natural que **no** puede ser definido en español en menos de 20 palabras. Acabamos de definirlo con 18 palabras.

Se deduce que el idioma natural no es suficientemente formal.

Paradoja del mentiroso, [Mileto 400 BC]:
"Esta oración es falsa"

Paradojas

Paradoja del barbero, [Bernard Russell, 1901]:

Definamos $R = \{x | x \notin x\}$. Luego, tenemos que $R \in R \iff R \notin R$.

Se deduce que el conjunto de todos los conjuntos no existe. [Gregory Cantor]

Paradoja de Berry, [Bernard Russell, 1908]:

Sea n el menor número natural que **no** puede ser definido en español en menos de 20 palabras. Acabamos de definirlo con 18 palabras.

Se deduce que el idioma natural no es suficientemente formal.

Paradoja del mentiroso, [Mileto 400 BC]:

"Esta oración es falsa" Si es verdadera es falsa. Si es falsa es verdadera.

Programa de Hilbert, comienzos del siglo XX

Programa de Hilbert, comienzos del siglo XX

1. Desarrollar un lenguaje formal para la matemáticas.

Programa de Hilbert, comienzos del siglo XX

1. Desarrollar un lenguaje formal para la matemáticas.
2. Encontrar reglas que capturen el razonamiento matemático.

Programa de Hilbert, comienzos del siglo XX

1. Desarrollar un lenguaje formal para la matemáticas.
2. Encontrar reglas que capturen el razonamiento matemático.
3. Listar axiomas para toda la matemática.

Programa de Hilbert, comienzos del siglo XX

1. Desarrollar un lenguaje formal para la matemáticas.
2. Encontrar reglas que capturen el razonamiento matemático.
3. Listar axiomas para toda la matemática.
4. Escribir un algoritmo que decida si una oración es demostrable.

Programa de Hilbert, comienzos del siglo XX

1. Desarrollar un lenguaje formal para la matemáticas. ✓
2. Encontrar reglas que capturen el razonamiento matemático.
3. Listar axiomas para toda la matemática.
4. Escribir un algoritmo que decida si una oración es demostrable.

Bernard Russell

Programa de Hilbert, comienzos del siglo XX

1. Desarrollar un lenguaje formal para la matemáticas. ✓
2. Encontrar reglas que capturen el razonamiento matemático. ✓
3. Listar axiomas para toda la matemática. ✗
4. Escribir un algoritmo que decida si una oración es demostrable. ✗

Bernard Russell

Kurt Gödel (izquierda)

Sumario del mini-curso

Charla I: **Completitud**
del lenguaje matemático y las reglas de razonamiento.

Charla II: **Incompletitud**
de los axiomas para la matemática.

Charla III: **Indecidibilidad**
mediante algoritmos de qué es demostrable.

Primera Charla: Completitud

Lenguaje:

Lenguaje: Lógica de primer orden

Lenguaje: Lógica de primer orden

Para definir un *lenguaje* usamos

- un vocabulario,
- símbolos lógicos $\vee, \&, \rightarrow, \neg, \forall, \exists, (,)$,
- símbolos de variables $x, y, z, ...$,

Lenguaje: Lógica de primer orden

Para definir un *lenguaje* usamos

- un vocabulario,
- símbolos lógicos $\vee, \&, \rightarrow, \neg, \forall, \exists, (,)$,
- símbolos de variables $x, y, z, ...$,

y seguimos ciertas reglas para definir las *fórmulas bien formadas*.

Lenguaje: Lógica de primer orden

Para definir un *lenguaje* usamos

- un vocabulario,
- símbolos lógicos $\vee, \&, \rightarrow, \neg, \forall, \exists, (,)$,
- símbolos de variables $x, y, z, ...$,

y seguimos ciertas reglas para definir las *fórmulas bien formadas*.

Un *vocabulario* consiste en símbolos de constante, de función y de relación.

Lenguaje: Lógica de primer orden

Para definir un *lenguaje* usamos

- un vocabulario,
- símbolos lógicos $\vee, \&, \rightarrow, \neg, \forall, \exists, (,)$,
- símbolos de variables $x, y, z, ...$,

y seguimos ciertas reglas para definir las *fórmulas bien formadas*.

Un *vocabulario* consiste en símbolos de constante, de función y de relación.

Ej: $\{e, *, <\}$ es un vocabulario apto para hablar de grupos ordenados.

Lenguaje: Lógica de primer orden

Para definir un *lenguaje* usamos

- un vocabulario,
- símbolos lógicos $\vee, \&, \rightarrow, \neg, \forall, \exists, (,)$,
- símbolos de variables $x, y, z, ...$,

y seguimos ciertas reglas para definir las *fórmulas bien formadas*.

Un *vocabulario* consiste en símbolos de constante, de función y de relación.

Ej: $\{e, *, <\}$ es un vocabulario apto para hablar de grupos ordenados.

Ej: $\forall x, y(x < y \rightarrow \forall z(z * x < z * y))$ es una sentencia bien formada.

Lenguaje: Lógica de primer orden

Para definir un *lenguaje* usamos

- un vocabulario,
- símbolos lógicos $\vee, \&, \rightarrow, \neg, \forall, \exists, (,)$,
- símbolos de variables $x, y, z, ...$,

y seguimos ciertas reglas para definir las *fórmulas bien formadas*.

Un *vocabulario* consiste en símbolos de constante, de función y de relación.

Ej: $\{e, *, <\}$ es un vocabulario apto para hablar de grupos ordenados.

Ej: $\forall x, y(x < y \rightarrow \forall z(z * x < z * y))$ es una sentencia bien formada.

En *lenguajes de 1er orden*, cuantificación, $\forall$ y $\exists$, es sólo para elementos.

Lenguaje: Lógica de primer orden

Para definir un *lenguaje* usamos

- un vocabulario,
- símbolos lógicos $\vee, \&, \rightarrow, \neg, \forall, \exists, (,)$,
- símbolos de variables $x, y, z, ...$,

y seguimos ciertas reglas para definir las *fórmulas bien formadas*.

Un *vocabulario* consiste en símbolos de constante, de función y de relación.

Ej: $\{e, *, <\}$ es un vocabulario apto para hablar de grupos ordenados.
Ej: $\forall x, y(x < y \rightarrow \forall z(z * x < z * y))$ es una sentencia bien formada.

En *lenguajes de 1er orden*, cuantificación, $\forall$ y $\exists$, es sólo para elementos.

Ej: La sentencia anterior es cierta en $(\mathbb{Z} : 0, +, <)$ y falsa en $(\mathbb{R} : 1, \times, <)$.

Lenguaje:

Lenguaje: Lógica de primer orden

Para definir un *lenguaje* usamos

- un vocabulario,
- símbolos lógicos $\vee, \&, \rightarrow, \neg, \forall, \exists, (,)$,
- símbolos de variables $x, y, z, ...$,

y seguimos ciertas reglas para definir las *fórmulas bien formadas*.

Un *vocabulario* consiste en símbolos de constante, de función y de relación.

Ej: $\{e, *, <\}$ es un vocabulario apto para hablar de grupos ordenados.
Ej: $\forall x, y(x < y \rightarrow \forall z(z * x < z * y))$ es una sentencia bien formada.

En *lenguajes de 1er orden*, cuantificación, $\forall$ y $\exists$, es sólo para elementos.

Ej: La sentencia anterior es cierta en $(\mathbb{Z} : 0, +, <)$ y falsa en $(\mathbb{R} : 1, \times, <)$.

τ-estructuras

vocabulario $\tau = \{e, *, <\}$.

$(K : e, *, <)$ es un *grupo ordenado* si satisface AX1, AX2, AX3, AX4.

- AX1 es $\forall x, y, z(x * (y * z) = (x * y) * z)$
- AX2 es $\forall x(x * e = x)$
- AX3 es $\forall x\ \exists y\ (x * y = e \wedge y * x = e)$
- AX4 es $\forall x, y(x < y \rightarrow \forall z\ (z * x < z * y))$

τ-estructuras

vocabulario $\tau = \{e, *, <\}$.

$(K : e, *, <)$ es un *grupo ordenado* si satisface AX1, AX2, AX3, AX4.

- AX1 es $\forall x, y, z(x * (y * z) = (x * y) * z)$
- AX2 es $\forall x(x * e = x)$
- AX3 es $\forall x\ \exists y\ (x * y = e \wedge y * x = e)$
- AX4 es $\forall x, y(x < y \rightarrow \forall z\ (z * x < z * y))$

Ej: $(\mathbb{Z} : 0, +, <)$ es un grupo ordenado
Ej: $(\mathbb{R}^+ : 1, \times, <)$ es un grupo ordenado

τ-estructuras

vocabulario $\tau = \{e, *, <\}$.

$(K : e, *, <)$ es un *grupo ordenado* si satisface AX1, AX2, AX3, AX4.

- AX1 es $\forall x, y, z(x * (y * z) = (x * y) * z)$
- AX2 es $\forall x(x * e = x)$
- AX3 es $\forall x\ \exists y\ (x * y = e \wedge y * x = e)$
- AX4 es $\forall x, y(x < y \rightarrow \forall z\ (z * x < z * y))$

Ej: $(\mathbb{Z} : 0, +, <)$ es un grupo ordenado
Ej: $(\mathbb{R}^+ : 1, \times, <)$ es un grupo ordenado

En general:

Una τ-estructura es un conjunto de elementos
equipado con interpretaciones para los símbolos de τ.

τ-estructuras

vocabulario $\tau = \{e, *, <\}$.

$(K : e, *, <)$ es un *grupo ordenado* si satisface AX1, AX2, AX3, AX4.

- AX1 es $\forall x, y, z(x * (y * z) = (x * y) * z)$
- AX2 es $\forall x(x * e = x)$
- AX3 es $\forall x\ \exists y\ (x * y = e \wedge y * x = e)$
- AX4 es $\forall x, y(x < y \rightarrow \forall z\ (z * x < z * y))$

Ej: $(\mathbb{Z} : 0, +, <)$ es un grupo ordenado
Ej: $(\mathbb{R}^+ : 1, \times, <)$ es un grupo ordenado

En general:

Una τ-estructura es un conjunto de elementos
equipado con interpretaciones para los símbolos de τ.

Dada una τ-estructura $\mathcal{M}$ y una τ-sentencia φ,
se define por inducción la relación: φ es cierta en $\mathcal{M}$.

τ-estructuras

vocabulario $\tau = \{e, *, <\}$.

$(K : e, *, <)$ es un *grupo ordenado* si satisface AX1, AX2, AX3, AX4.

- AX1 es $\forall x, y, z(x * (y * z) = (x * y) * z)$
- AX2 es $\forall x(x * e = x)$
- AX3 es $\forall x\ \exists y\ (x * y = e \wedge y * x = e)$
- AX4 es $\forall x, y(x < y \rightarrow \forall z\ (z * x < z * y))$

Ej: $(\mathbb{Z} : 0, +, <)$ es un grupo ordenado
Ej: $(\mathbb{R}^+ : 1, \times, <)$ es un grupo ordenado $(\mathbb{R}^+ : 1, \times, <) \models \forall x(e * x = x)$

En general:

Una τ-estructura es un conjunto de elementos
equipado con interpretaciones para los símbolos de τ.

Dada una τ-estructura $\mathcal{M}$ y una τ-sentencia φ,
se define por inducción la relación: φ es cierta en $\mathcal{M}$. Denotado $\mathcal{M} \models \varphi$.

Lenguaje para todo la matemática

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

donde $0, 1, +, \times, <$ solo aplican a $\mathbb{N}$.

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

donde $0, 1, +, \times, <$ solo aplican a $\mathbb{N}$.

En esta estructura uno puede definir:

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

donde $0, 1, +, \times, <$ solo aplican a $\mathbb{N}$.

En esta estructura uno puede definir: $\mathbb{Z}$,

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

donde $0, 1, +, \times, <$ solo aplican a $\mathbb{N}$.

En esta estructura uno puede definir: $\mathbb{Z}$, $\mathbb{Q}$,

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

donde $0, 1, +, \times, <$ solo aplican a $\mathbb{N}$.

En esta estructura uno puede definir: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$,

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

donde $0, 1, +, \times, <$ solo aplican a $\mathbb{N}$.

En esta estructura uno puede definir: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $C[0, 1]$,

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

donde $0, 1, +, \times, <$ solo aplican a $\mathbb{N}$.

En esta estructura uno puede definir: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $C[0, 1]$, ...

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

donde $0, 1, +, \times, <$ solo aplican a $\mathbb{N}$.

En esta estructura uno puede definir: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $C[0, 1]$, ...

y toda la matemática

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

donde $0, 1, +, \times, <$ solo aplican a $\mathbb{N}$.

En esta estructura uno puede definir: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $C[0, 1]$, ...
y toda la matemática

Observación empírica: El lenguaje es completo.

Lenguaje para todo la matemática

Consideremos el vocabulario $\{0, 1, +, \times, <, \in\}$

y los elementos $M = \mathbb{N} \cup \mathcal{P}(\mathbb{N}) \cup \mathcal{P}(\mathcal{P}(\mathbb{N})) \cup \cdots$

donde $0, 1, +, \times, <$ solo aplican a $\mathbb{N}$.

En esta estructura uno puede definir: $\mathbb{Z}$, $\mathbb{Q}$, $\mathbb{R}$, $C[0, 1]$, ...
y toda la matemática

Observación empírica: El lenguaje es completo.
Todo enunciado matemático se puede escribir en el lenguaje formal.

Reglas del razonamiento

Reglas del razonamiento

Dado un conjunto de sentencias Γ y una sentencia φ, definimos $\Gamma \vdash \varphi$,
que se lee: φ es demostrable a partir de Γ

Reglas del razonamiento

Dado un conjunto de sentencias Γ y una sentencia φ, definimos $\Gamma \vdash \varphi$,
que se lee: φ es demostrable a partir de Γ

$$\Gamma \vdash \varphi \text{ if } \varphi \in \Gamma \quad (Assume_{\mathcal{L}}) \qquad \Gamma \vdash t = t \text{ for all } t \in Term_{\mathcal{L}} \quad (EqRefl)$$

$$\frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \varphi}\ (\wedge EL) \qquad \frac{\Gamma \vdash \varphi \wedge \psi}{\Gamma \vdash \psi}\ (\wedge ER) \qquad \frac{\Gamma \vdash \varphi \quad \Gamma \vdash \psi}{\Gamma \vdash \varphi \wedge \psi}\ (\wedge I)$$

$$\frac{\Gamma \vdash \varphi}{\Gamma \vdash \varphi \vee \psi}(\vee IL) \qquad \frac{\Gamma \vdash \psi}{\Gamma \vdash \varphi \vee \psi}(\vee IR) \qquad \frac{\Gamma \vdash \varphi \to \psi}{\Gamma \cup \{\varphi\} \vdash \psi}(\to E) \qquad \frac{\Gamma \cup \{\varphi\} \vdash \psi}{\Gamma \vdash \varphi \to \psi}(\to I)$$

$$\frac{\Gamma \cup \{\varphi\} \vdash \theta \quad \Gamma \cup \{\psi\} \vdash \theta}{\Gamma \cup \{\varphi \vee \psi\} \vdash \theta}\ (\vee PC) \qquad \frac{\Gamma \cup \{\psi\} \vdash \varphi \quad \Gamma \cup \{\neg\psi\} \vdash \varphi}{\Gamma \vdash \varphi}\ (\neg PC)$$

$$\frac{\Gamma \cup \{\neg\varphi\} \vdash \psi \quad \Gamma \cup \{\neg\varphi\} \vdash \neg\psi}{\Gamma \vdash \varphi}\ (Contr) \qquad \frac{\Gamma \vdash \varphi^t_x \quad \Gamma \vdash t = u}{\Gamma \vdash \varphi^u_x}\ (= Sub)$$

$$\frac{\Gamma \vdash \varphi^t_x}{\Gamma \vdash \exists x \varphi}\ (\exists I) \qquad \frac{\Gamma \cup \{\varphi^y_x\} \vdash \psi}{\Gamma \cup \{\exists x \varphi\} \vdash \psi} \text{ if } y \notin FreeVar(\Gamma \cup \{\exists x \varphi, \psi\})\ (\exists P)$$

$$\frac{\Gamma \vdash \forall x \varphi}{\Gamma \vdash \varphi^t_x}\ (\forall E) \qquad \frac{\Gamma \vdash \varphi^y_x}{\Gamma \vdash \forall x \varphi} \text{ if } y \notin FreeVar(\Gamma \cup \{\forall x \varphi\})\ (\forall I)$$

$$\frac{\Gamma \vdash \varphi}{\Gamma' \vdash \varphi} \text{ if } \Gamma' \supseteq \Gamma \quad (Super)$$

Teorema de Completitud — Gödel 1931

Teorema de Completitud — Gödel 1931

Dado un conjunto de sentencias Γ y una sentencia φ, los siguientes son equivalentes:

$\Gamma \models \varphi$ (Implicación semántica):
Todo estructura que satisface Γ, también satisface φ.

$\Gamma \vdash \varphi$ (Implicación sintáctica):
Hay una demostración formal de que Γ implica φ.

Teorema de Completitud — Gödel 1931

Dado un conjunto de sentencias Γ y una sentencia φ, los siguientes son equivalentes:

$\Gamma \models \varphi$ (Implicación semántica):
Todo estructura que satisface Γ, también satisface φ.

$\Gamma \vdash \varphi$ (Implicación sintáctica):
Hay una demostración formal de que Γ implica φ.

Se deduce que

las reglas anteriores son suficientes para toda demostración matemática.

Teorema de Completitud — Gödel 1931

Dado un conjunto de sentencias Γ y una sentencia φ, los siguientes son equivalentes:

$\Gamma \models \varphi$ (Implicación semántica):
Todo estructura que satisface Γ, también satisface φ.

$\Gamma \vdash \varphi$ (Implicación sintáctica):
Hay una demostración formal de que Γ implica φ.

Se deduce que

las reglas anteriores son suficientes para toda demostración matemática.

Completitud I: Para todo Γ y φ, $\Gamma \models \varphi \iff \Gamma \vdash \varphi$

Teorema de Completitud — Gödel 1931

Dado un conjunto de sentencias Γ y una sentencia φ, los siguientes son equivalentes:

$\Gamma \models \varphi$ (Implicación semántica):
Todo estructura que satisface Γ, también satisface φ.

$\Gamma \vdash \varphi$ (Implicación sintáctica):
Hay una demostración formal de que Γ implica φ.

Se deduce que

las reglas anteriores son suficientes para toda demostración matemática.

Completitud I: Para todo Γ y φ, $\Gamma \models \varphi \iff \Gamma \vdash \varphi$

Completitud II: Para todo Γ, Γ tiene un modelo $\iff$ Γ es consistente.

Teorema de Completitud

Dado un conjunto de sentencias Γ y una sentencia φ,

Completitud I: Para todo Γ y φ, $\Gamma \models \varphi \iff \Gamma \vdash \varphi$

Completitud II: Para todo Γ, Γ tiene un modelo $\iff$ Γ es consistente.

Definición: Γ es consistente si $\Gamma \nvdash \psi \wedge \neg\psi$ para ninguna ψ.

Teorema de Completitud

Dado un conjunto de sentencias Γ y una sentencia φ,

Completitud I: Para todo Γ y φ, $\quad \Gamma \models \varphi \iff \Gamma \vdash \varphi$

Completitud II: Para todo Γ, $\quad \Gamma$ tiene un modelo $\iff$ Γ es consistente.

Definición: Γ es consistente si $\Gamma \nvdash \psi \wedge \neg\psi$ para ninguna ψ.

Obs: Los siguientes son equivalentes:

- $\Gamma \models \varphi$ (I.e.: Todo estructura que satisface Γ, también satisface φ)
- Ninguna estructura satisface $\Gamma \cup \{\neg\varphi\}$.

Teorema de Completitud

Dado un conjunto de sentencias Γ y una sentencia φ,

Completitud I: Para todo Γ y φ, $\quad \Gamma \models \varphi \iff \Gamma \vdash \varphi$

Completitud II: Para todo Γ, $\quad \Gamma$ tiene un modelo $\iff$ Γ es consistente.

Definición: Γ es consistente si $\Gamma \nvdash \psi \wedge \neg\psi$ para ninguna ψ.

Obs: Los siguientes son equivalentes:

- $\Gamma \models \varphi$ (I.e.: Todo estructura que satisface Γ, también satisface φ)
- Ninguna estructura satisface $\Gamma \cup \{\neg\varphi\}$.

Obs: Los siguientes son equivalentes:

- $\Gamma \vdash \varphi$ (I.e.: Hay una demostración formal de que Γ implica φ)
- $\Gamma \cup \{\neg\varphi\}$ es inconsistente.

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Definición: El conjunto de $\mathcal{T}$ *términos cerrados* es tal que:

- contiene todos los símbolos de constante c_i de τ,y
- si $t_1, ..., t_n \in \mathcal{T}$ y $f \in \tau$ es un símbolo de función n-ario, $f(t_1, ..., t_n) \in \mathcal{T}$.

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Definición: El conjunto de $\mathcal{T}$ *términos cerrados* es tal que:

- contiene todos los símbolos de constante c_i de τ,y
- si $t_1, ..., t_n \in \mathcal{T}$ y $f \in \tau$ es un símbolo de función n-ario, $f(t_1, ..., t_n) \in \mathcal{T}$.

Ej: $\tau = \{e, c, *, <\}$, $\mathcal{T} = \{e, c,$

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Definición: El conjunto de $\mathcal{T}$ *términos cerrados* es tal que:

- contiene todos los símbolos de constante c_i de τ,y
- si $t_1, ..., t_n \in \mathcal{T}$ y $f \in \tau$ es un símbolo de función n-ario, $f(t_1, ..., t_n) \in \mathcal{T}$.

Ej: $\tau = \{e, c, *, <\}$, $\mathcal{T} = \{e, c, c * c,$

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Definición: El conjunto de $\mathcal{T}$ *términos cerrados* es tal que:

- contiene todos los símbolos de constante c_i de τ,y
- si $t_1, ..., t_n \in \mathcal{T}$ y $f \in \tau$ es un símbolo de función n-ario, $f(t_1, ..., t_n) \in \mathcal{T}$.

Ej: $\tau = \{e, c, *, <\}$, $\mathcal{T} = \{e, c, c * c, (c * c) * c,$

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Definición: El conjunto de $\mathcal{T}$ *términos cerrados* es tal que:

- contiene todos los símbolos de constante c_i de τ,y
- si $t_1, ..., t_n \in \mathcal{T}$ y $f \in \tau$ es un símbolo de función n-ario, $f(t_1, ..., t_n) \in \mathcal{T}$.

Ej: $\tau = \{e, c, *, <\}$, $\mathcal{T} = \{e, c, c * c, (c * c) * c, c * (c * e),$

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Definición: El conjunto de $\mathcal{T}$ *términos cerrados* es tal que:

- contiene todos los símbolos de constante c_i de τ,y
- si $t_1, ..., t_n \in \mathcal{T}$ y $f \in \tau$ es un símbolo de función n-ario, $f(t_1, ..., t_n) \in \mathcal{T}$.

Ej: $\tau = \{e, c, *, <\}$, $\mathcal{T} = \{e, c, c * c, (c * c) * c, c * (c * e), (c * c) * (c * e), ...\}$

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Definición: El conjunto de $\mathcal{T}$ *términos cerrados* es tal que:

- contiene todos los símbolos de constante c_i de τ,y
- si $t_1, ..., t_n \in \mathcal{T}$ y $f \in \tau$ es un símbolo de función n-ario, $f(t_1, ..., t_n) \in \mathcal{T}$.

Ej: $\tau = \{e, c, *, <\}$, $\mathcal{T} = \{e, c, c * c, (c * c) * c, c * (c * e), (c * c) * (c * e), ...\}$

Ej: $\tau = \{e, c, *, ^{-1}, <\}$, $\mathcal{T} = \{e, c, c * c^{-1}, (c * c)^{-1} * c, c * (c^{-1} * e), (c * c) * (c^{-1} * e), ...\}$

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Definición: El conjunto de $\mathcal{T}$ *términos cerrados* es tal que:

- contiene todos los símbolos de constante c_i de τ,y
- si $t_1, ..., t_n \in \mathcal{T}$ y $f \in \tau$ es un símbolo de función n-ario, $f(t_1, ..., t_n) \in \mathcal{T}$.

Ej: $\tau = \{e, c, *, <\}$, $\mathcal{T} = \{e, c, c*c, (c*c)*c, c*(c*e), (c*c)*(c*e), ...\}$

Ej: $\tau = \{e, c, *, ^{-1}, <\}$, $\mathcal{T} = \{e, c, c*c^{-1}, (c*c)^{-1}*c, c*(c^{-1}*e), (c*c)*(c^{-1}*e), ...\}$

Definición: El *modelo de términos de* Γ es $\mathcal{T}/\equiv$, donde $t \equiv s \iff \Gamma \vdash t = s$.

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Definición: El conjunto de $\mathcal{T}$ *términos cerrados* es tal que:

- contiene todos los símbolos de constante c_i de τ,y
- si $t_1, ..., t_n \in \mathcal{T}$ y $f \in \tau$ es un símbolo de función n-ario, $f(t_1, ..., t_n) \in \mathcal{T}$.

Ej: $\tau = \{e, c, *, <\}$, $\mathcal{T} = \{e, c, c*c, (c*c)*c, c*(c*e), (c*c)*(c*e), ...\}$

Ej: $\tau = \{e, c, *, ^{-1}, <\}$, $\mathcal{T} = \{e, c, c*c^{-1}, (c*c)^{-1}*c, c*(c^{-1}*e), (c*c)*(c^{-1}*e), ...\}$

Definición: El *modelo de términos de* Γ es $\mathcal{T}/\equiv$, donde $t \equiv s \iff \Gamma \vdash t = s$.

Teorema: Si Γ es (1) consistente, (2) completo, y (3) para toda formula $\varphi(\bar{x}, y)$ hay $f_\varphi \in \tau$ tal que $\forall \bar{x}(\exists y \varphi(\bar{x}, y) \to \varphi(\bar{x}, f_\varphi(\bar{x}))$ está en Γ, entonces el modelo de términos de Γ satisface Γ.

Demostración de Completitud / modelo de términos

Γ es un conjunto de τ sentencias y $\tau = \{c_0, c_1, ..., f_0, f_1, ..., R_0, R_1, ...\}$.

Definición: El conjunto de $\mathcal{T}$ *términos cerrados* es tal que:

- contiene todos los símbolos de constante c_i de τ,y
- si $t_1, ..., t_n \in \mathcal{T}$ y $f \in \tau$ es un símbolo de función n-ario, $f(t_1, ..., t_n) \in \mathcal{T}$.

Ej: $\tau = \{e, c, *, <\}$, $\mathcal{T} = \{e, c, c*c, (c*c)*c, c*(c*e), (c*c)*(c*e), ...\}$

Ej: $\tau = \{e, c, *, ^{-1}, <\}$, $\mathcal{T} = \{e, c, c*c^{-1}, (c*c)^{-1}*c, c*(c^{-1}*e), (c*c)*(c^{-1}*e), ...\}$

Definición: El *modelo de términos de* Γ es $\mathcal{T}/\equiv$, donde $t \equiv s \iff \Gamma \vdash t = s$.

Teorema: Si Γ es (1) consistente, (2) completo, y (3)
para toda formula $\varphi(\bar{x}, y)$ hay $f_\varphi \in \tau$ tal que $\forall \bar{x}(\exists y \varphi(\bar{x}, y) \to \varphi(\bar{x}, f_\varphi(\bar{x}))$ está en Γ,
entonces el modelo de términos de Γ satisface Γ.

Teorema: Si Γ es consistente,
existe $\tau' \supseteq \tau$ y un conjunto de τ'-sentencias $\Gamma' \supseteq \Gamma$ que satisface (1) (2) y (3).

Asistentes de pruebas

Un asistente de pruebas o un demostrador de teoremas interactivo, es un programa te ayuda a escribir pruebas formales.

Asistentes de pruebas

Un asistente de pruebas o un demostrador de teoremas interactivo, es un programa te ayuda a escribir pruebas formales.

Todo asistente de pruebas contiene:

- Verificador de pruebas formales

Asistentes de pruebas

Un asistente de pruebas o un demostrador de teoremas interactivo, es un programa te ayuda a escribir pruebas formales.

Todo asistente de pruebas contiene:

- Verificador de pruebas formales
- Interface para ingresar las pruebas

Asistentes de pruebas

Un asistente de pruebas o un demostrador de teoremas interactivo, es un programa te ayuda a escribir pruebas formales.

Todo asistente de pruebas contiene:

- Verificador de pruebas formales
- Interface para ingresar las pruebas
- Automatizador para pasos simples

Asistentes de pruebas

Un asistente de pruebas o un demostrador de teoremas interactivo, es un programa te ayuda a escribir pruebas formales.

Todo asistente de pruebas contiene:

- Verificador de pruebas formales
- Interface para ingresar las pruebas
- Automatizador para pasos simples

Asistentes de pruebas

Un asistente de pruebas o un demostrador de teoremas interactivo, es un programa te ayuda a escribir pruebas formales.

Todo asistente de pruebas contiene:

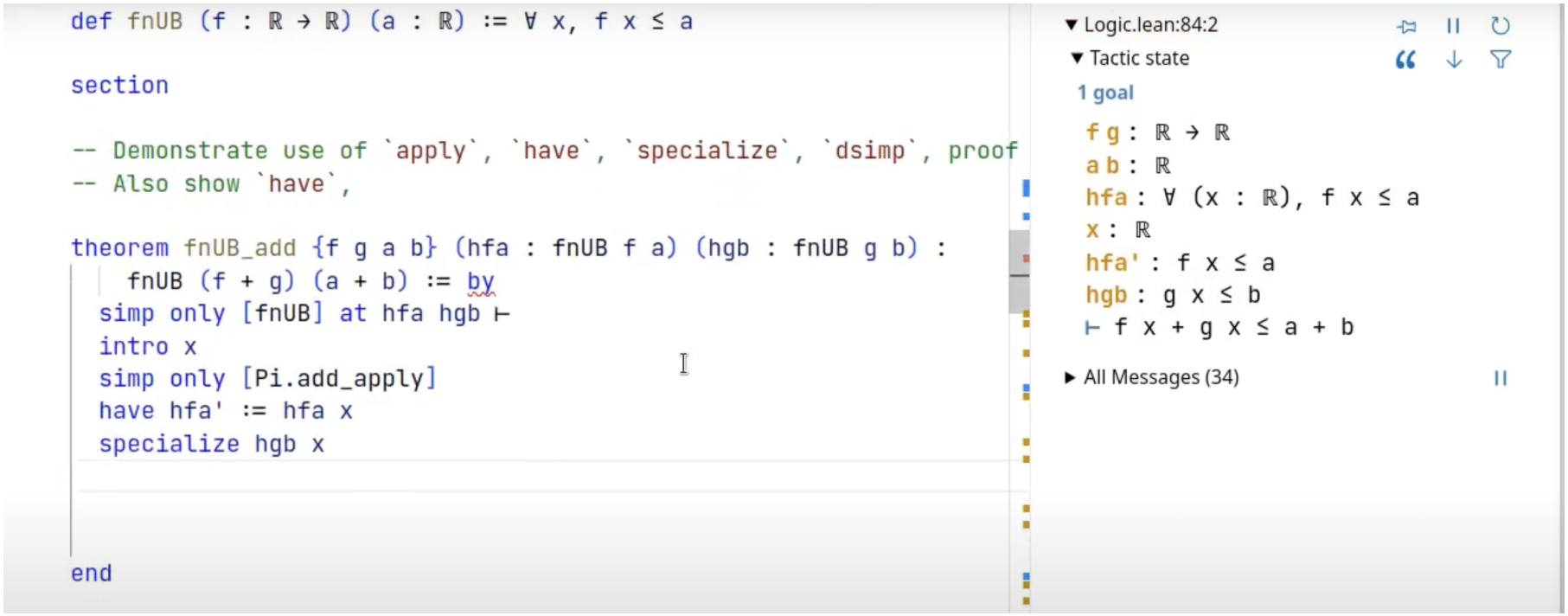
- Verificador de pruebas formales
- Interface para ingresar las pruebas
- Automatizador para pasos simples

Principales asistente de pruebas hoy en día:

- Coq, created in 1989
- Isabelle, created in 1986
- Lean, created in 2013

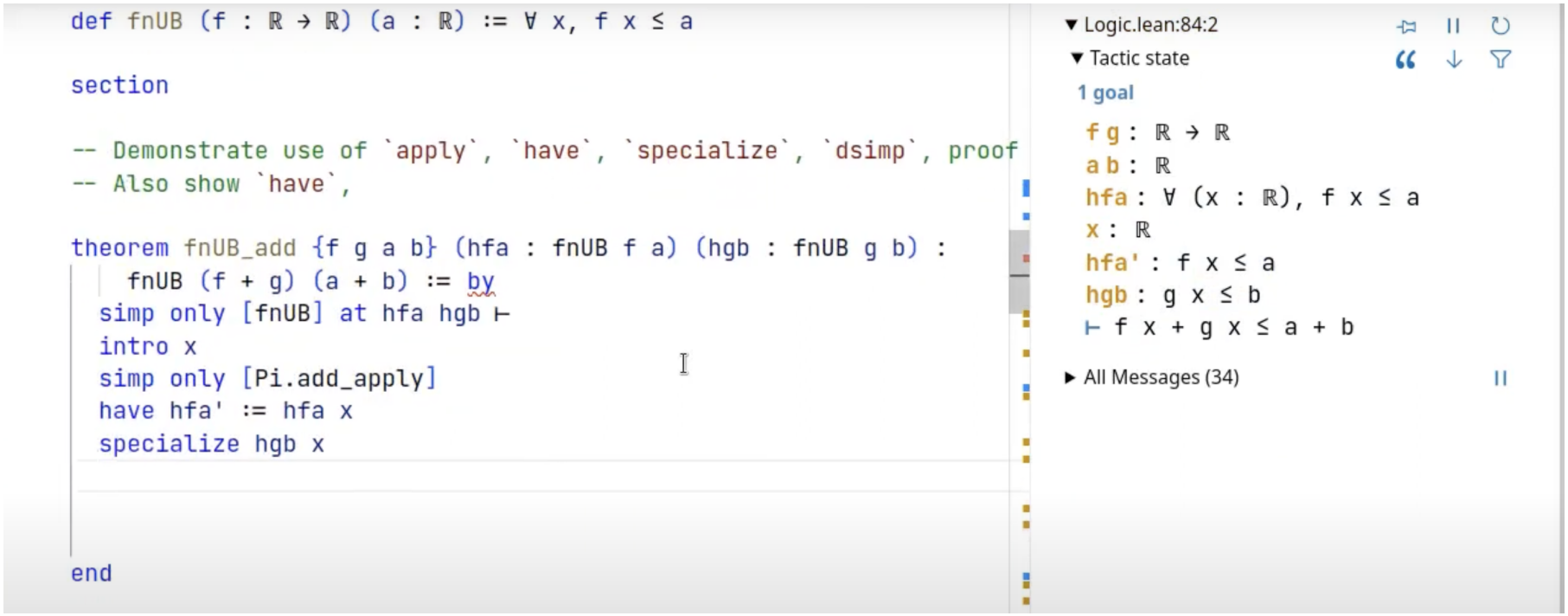
LEAN

Ejemplo de cómo se ve la interface:



LEAN

Ejemplo de cómo se ve la interface:



Showcase: En el 2022, se formalizó en LEAN la demostración del "liquid tensor experiment". Un teorema de Peter Scholze que nadie había podido verificar.

Pronósticos a futuro

En este momento, los siguiente crecen exponencialmente:

Pronósticos a futuro

En este momento, los siguiente crecen exponencialmente:

- Lo matemáticos que usan programas como LEAN.

Pronósticos a futuro

En este momento, los siguiente crecen exponencialmente:

- Lo matemáticos que usan programas como LEAN.
- La biblioteca de lemas y definiciones.

Pronósticos a futuro

En este momento, los siguiente crecen exponencialmente:

- Lo matemáticos que usan programas como LEAN.
- La biblioteca de lemas y definiciones.
- La capacidad de hacer pasos automáticamente.

Pronósticos a futuro

En este momento, los siguiente crecen exponencialmente:

- Lo matemáticos que usan programas como LEAN.
- La biblioteca de lemas y definiciones.
- La capacidad de hacer pasos automáticamente.

Pronósticos a futuro

En este momento, los siguiente crecen exponencialmente:

- Lo matemáticos que usan programas como LEAN.
- La biblioteca de lemas y definiciones.
- La capacidad de hacer pasos automáticamente.

En el futuro:

Pronósticos a futuro

En este momento, los siguiente crecen exponencialmente:

- Lo matemáticos que usan programas como LEAN.
- La biblioteca de lemas y definiciones.
- La capacidad de hacer pasos automáticamente.

En el futuro:

- Mayoría de los matemáticos sabrán usar asistentes de pruebas.

Pronósticos a futuro

En este momento, los siguiente crecen exponencialmente:

- Lo matemáticos que usan programas como LEAN.
- La biblioteca de lemas y definiciones.
- La capacidad de hacer pasos automáticamente.

En el futuro:

- Mayoría de los matemáticos sabrán usar asistentes de pruebas.
- Los resultados nuevos serán rápidamente formalizados.

Pronósticos a futuro

En este momento, los siguiente crecen exponencialmente:

- Lo matemáticos que usan programas como LEAN.
- La biblioteca de lemas y definiciones.
- La capacidad de hacer pasos automáticamente.

En el futuro:

- Mayoría de los matemáticos sabrán usar asistentes de pruebas.
- Los resultados nuevos serán rápidamente formalizados.
- Los asistentes de pruebas podrán ayudar a encontrar demostraciones.

Pronósticos a futuro

En este momento, los siguiente crecen exponencialmente:

- Lo matemáticos que usan programas como LEAN.
- La biblioteca de lemas y definiciones.
- La capacidad de hacer pasos automáticamente.

En el futuro:

- Mayoría de los matemáticos sabrán usar asistentes de pruebas.
- Los resultados nuevos serán rápidamente formalizados.
- Los asistentes de pruebas podrán ayudar a encontrar demostraciones.

La próxima charla

Teorema de incompletitud de Gödel:

Toda lista de axiomas para la matemáticas que sea medianamente razonable,
va a ser incompleto.

Segunda Charla: Incompletitud

Programa de Hilbert

Objetivo:

Encontrar axiomas para toda la matemática.

Programa de Hilbert

Objetivo:

Encontrar axiomas para toda la matemática.

Idealmente querríamos que los axiomas sean:

- Verdaderos
- Fáciles de reconocer
- Completos

Programa de Hilbert

Objetivo:

Encontrar axiomas para toda la matemática.

Idealmente querríamos que los axiomas sean:

- Verdaderos
- Fáciles de reconocer
- Completos es decir, que todo enunciado se pueda probar o refutar

Programa de Hilbert

Objetivo:

Encontrar axiomas para toda la matemática.

Idealmente querríamos que los axiomas sean:

- Verdaderos
- Fáciles de reconocer
- Completos es decir, que todo enunciado se pueda probar o refutar

Teorema: (de incompletitud de Gödel, 1931): Esto no es posible

Axiomas de la Aritmética de Peano (PA)

Vocabulario: $\{0, 1, +, \times\}$.

Axiomas de la Aritmética de Peano (PA)

Vocabulario: $\{0, 1, +, \times\}$.
Modelo a tener en mente: $(\mathbb{N} : 0, 1, +, \times)$.

Axiomas de la Aritmética de Peano (PA)

Vocabulario: $\{0, 1, +, \times\}$.
Modelo a tener en mente: $(\mathbb{N} : 0, 1, +, \times)$.

Axiomas *semi-anillo*:

- $0 \neq 1$
- $\forall x((x + 0) = x)$ — Cero neutro
- $\forall x, y(x + y = y + x)$ — Conmutatividad de la suma
- $\forall x, y, z((x + y) + z = x + (y + z))$ — Asociatividad de la suma
- $\forall x((x \times 1) = x)$ — Uno neutro
- $\forall x, y(x \times y = y \times x)$ — Conmutatividad del producto
- $\forall x, y, z((x \times y) \times z = x \times (y \times z))$ — Asociatividad del producto
- $\forall x((x \times 0) = 0)$ — Cero absobente
- $\forall x, y, z((x + y) \times z = (x \times z) + (y \times z))$ — Distributiva
- $\forall x(x = 0 \vee \exists y(x = y + 1))$ — Existencia predecesor

Para toda formula $\varphi(x)$ tenemos el axioma de inducción:

- $(\varphi(0) \wedge (\forall x(\varphi(x) \rightarrow \varphi(x + 1)))) \quad \rightarrow \forall x(\varphi(x))$

Axiomas de la Aritmética de Segundo orden (SOA)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.

Axiomas de la Aritmética de Segundo orden (SOA)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) : 0, 1, +, \times, \in)$.

Axiomas de la Aritmética de Segundo orden (SOA)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) : 0, 1, +, \times, \in)$.

Axiomas:

Axiomas de la Aritmética de Segundo orden (SOA)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) : 0, 1, +, \times, \in)$.

Axiomas:

- Axiomas de Peano para elementos del tipo número.

Axiomas de la Aritmética de Segundo orden (SOA)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) : 0, 1, +, \times, \in)$.

Axiomas:

- Axiomas de Peano para elementos del tipo número.
- Axioma de Comprensión:

  Para cada fórmula de primer orden $\varphi(x)$, tenemos el axioma:

$$\exists^{conj} A \; \forall^{num} x \; (x \in A \iff \varphi(x)).$$

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número más los axiomas de ZFC:

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número más los axiomas de ZFC:

Extensionalidad Dos conjuntos son iguales si contienen los mismos elementos.

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número más los axiomas de ZFC:

Extensionalidad Dos conjuntos son iguales si contienen los mismos elementos.

Vacío Hay un conjunto vacío.

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número más los axiomas de ZFC:

Extensionalidad Dos conjuntos son iguales si contienen los mismos elementos.

Vacío Hay un conjunto vacío.

Par Se puede construir el conjunto $\{a, b\}$.

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número más los axiomas de ZFC:

Extensionalidad Dos conjuntos son iguales si contienen los mismos elementos.

Vacío Hay un conjunto vacío.

Par Se puede construir el conjunto $\{a, b\}$.

Union Se puede tomar la union de todos los conjuntos en un conjunto.

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número más los axiomas de ZFC:

Extensionalidad Dos conjuntos son iguales si contienen los mismos elementos.

Vacío Hay un conjunto vacío.

Par Se puede construir el conjunto $\{a, b\}$.

Union Se puede tomar la union de todos los conjuntos en un conjunto.

Potencia Todo conjunto tiene un conjunto potencia.

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número más los axiomas de ZFC:

Extensionalidad Dos conjuntos son iguales si contienen los mismos elementos.

Vacío Hay un conjunto vacío.

Par Se puede construir el conjunto $\{a, b\}$.

Union Se puede tomar la union de todos los conjuntos en un conjunto.

Potencia Todo conjunto tiene un conjunto potencia.

Infinito Hay un conjunto infinito.

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número más los axiomas de ZFC:

Extensionalidad Dos conjuntos son iguales si contienen los mismos elementos.

Vacío Hay un conjunto vacío.

Par Se puede construir el conjunto $\{a, b\}$.

Union Se puede tomar la union de todos los conjuntos en un conjunto.

Potencia Todo conjunto tiene un conjunto potencia.

Infinito Hay un conjunto infinito.

Comprensión Para cada fórmula $\varphi(x)$, $\forall A \exists B\ \forall x\ (x \in B \iff x \in A\ \&\ \varphi(x))$.

Remplazo y una función dada una fórmula y su dominio.

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número más los axiomas de ZFC:

Extensionalidad Dos conjuntos son iguales si contienen los mismos elementos.

Vacío Hay un conjunto vacío.

Par Se puede construir el conjunto $\{a, b\}$.

Union Se puede tomar la union de todos los conjuntos en un conjunto.

Potencia Todo conjunto tiene un conjunto potencia.

Infinito Hay un conjunto infinito.

Comprensión Para cada fórmula $\varphi(x)$, $\forall A \exists B\ \forall x\ (x \in B \iff x \in A\ \&\ \varphi(x))$.

Remplazo y una función dada una fórmula y su dominio.

Regularidad No hay sequencias $a_0 \ni a_1 \ni a_2 \ni a_3 \ni \cdots$.

Axiomas de la teoría de conjuntos (ZFC)

Vocabulario: $\{0, 1, +, \times, \in\}$, con dos tipos de objetos: números y conjuntos.
Modelo a tener en mente: $(\mathbb{N}, \mathcal{P}(\mathbb{N}) \cup \mathcal{PP}(\mathbb{N}) \cup \mathcal{PPP}(\mathbb{N}) \cup \ldots : 0, 1, +, \times, \in)$.

Axiomas de Peano— para elementos del tipo número más los axiomas de ZFC:

Extensionalidad Dos conjuntos son iguales si contienen los mismos elementos.

Vacío Hay un conjunto vacío.

Par Se puede construir el conjunto $\{a, b\}$.

Union Se puede tomar la union de todos los conjuntos en un conjunto.

Potencia Todo conjunto tiene un conjunto potencia.

Infinito Hay un conjunto infinito.

Comprensión Para cada fórmula $\varphi(x)$, $\forall A \exists B\ \forall x\ (x \in B \iff x \in A\ \&\ \varphi(x))$.

Remplazo y una función dada una fórmula y su dominio.

Regularidad No hay sequencias $a_0 \ni a_1 \ni a_2 \ni a_3 \ni \cdots$.

Elección Este ya lo conocen.

Teoremas de incompletitud de Gödel, 1931

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula arithmética puede identificar las sentencias en Γ) ,

entonces:

- Γ es incompleto (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Teoremas de incompletitud de Gödel, 1931

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula arithmética puede identificar las sentencias en Γ) ,

entonces:

- Γ es incompleto (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Teorema: Más aún, $\Gamma \nvdash \mathit{Consistencia}(\Gamma)$.

Teoremas de incompletitud de Gödel, 1931

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente $(\Gamma \nvdash \psi \wedge \neg\psi)$,
- Γ es computable

(hay un algoritmo que puede identificar las sentencias en Γ),

entonces:

- Γ es incompleto (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Esta versión requiere ideas de Rosser de 1936.

Teorema: Más aún, $\Gamma \nvdash$ *Consistencia*(Γ).

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Si φ fuese falsa,

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Si φ fuese falsa, sería demostrable,

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Si φ fuese falsa, sería demostrable, y por lo tanto cierta.

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Si φ fuese falsa, sería demostrable, y por lo tanto cierta.

Por lo tanto:

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Si φ fuese falsa, sería demostrable, y por lo tanto cierta.

Por lo tanto:

φ es cierta,

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Si φ fuese falsa, sería demostrable, y por lo tanto cierta.

Por lo tanto:

φ es cierta, y no demostrable.

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Si φ fuese falsa, sería demostrable, y por lo tanto cierta.

Por lo tanto:

φ es cierta, y no demostrable.

Como φ es cierta, su negación tampoco es demostrable.

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Si φ fuese falsa, sería demostrable, y por lo tanto cierta.

Por lo tanto:

φ es cierta, y no demostrable.

Como φ es cierta, su negación tampoco es demostrable.

Necesitamos:

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Si φ fuese falsa, sería demostrable, y por lo tanto cierta.

Por lo tanto:

φ es cierta, y no demostrable.

Como φ es cierta, su negación tampoco es demostrable.

Necesitamos:

- expresar matemáticamente que una sentencia es *"demostrable"*.

Principales ideas de la demostración

Considere la oración φ: "Esta oración no es demostrable"

Si φ fuese falsa, sería demostrable, y por lo tanto cierta.

Por lo tanto:

φ es cierta, y no demostrable.

Como φ es cierta, su negación tampoco es demostrable.

Necesitamos:

- expresar matemáticamente que una sentencia es *"demostrable"*.
- escribir sentencias que se refieren a si mismas.

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \;,\; A \;,\; (\;,\;)\;,\; \neg$$

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \quad , \quad A \quad , \quad (\quad , \quad) \quad , \quad \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \quad , \quad A \quad , \quad (\quad , \quad) \quad , \quad \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \quad , \quad A \quad , \quad (\quad , \quad) \quad , \quad \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \quad , \quad A \quad , \quad (\quad , \quad) \quad , \quad \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

- $I(X)$
- $\neg I(X)$
- $IA(X)$
- $\neg IA(X)$

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \quad , \quad A \quad , \quad (\quad , \quad) \quad , \quad \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

- $I(X)$ *cierta* si X es imprimible
- $\neg I(X)$ *cierta* si X no es imprimible
- $IA(X)$ *cierta* si la auto-aplicación X es imprimible
- $\neg IA(X)$ *cierta* si la auto-aplicación X no es imprimible

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \ , \ A \ , \ (\ , \ ) \ , \ \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

- $I(X)$ *cierta* si X es imprimible
- $\neg I(X)$ *cierta* si X no es imprimible
- $IA(X)$ *cierta* si la auto-aplicación X es imprimible
- $\neg IA(X)$ *cierta* si la auto-aplicación X no es imprimible

Teorema: Si toda sentencia imprimible es cierta, hay una cierta no imprimible.

Demostración:

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \ , \ A \ , \ (\ , \ ) \ , \ \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

- $I(X)$ *cierta* si X es imprimible
- $\neg I(X)$ *cierta* si X no es imprimible
- $IA(X)$ *cierta* si la auto-aplicación X es imprimible
- $\neg IA(X)$ *cierta* si la auto-aplicación X no es imprimible

Teorema: Si toda sentencia imprimible es cierta, hay una cierta no imprimible.

Demostración: $\neg IA(\neg IA)$

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \ , \ A \ , \ (\ , \ ) \ , \ \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

- $I(X)$ *cierta* si X es imprimible
- $\neg I(X)$ *cierta* si X no es imprimible
- $IA(X)$ *cierta* si la auto-aplicación X es imprimible
- $\neg IA(X)$ *cierta* si la auto-aplicación X no es imprimible

Teorema: Si toda sentencia imprimible es cierta, hay una cierta no imprimible.

Demostración: $\neg IA(\neg IA)$ es cierta $\iff$ la auto-aplicación $\neg IA$ no es imprimible

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \ , \ A \ , \ (\ , \ ) \ , \ \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

- $I(X)$ *cierta* si X es imprimible
- $\neg I(X)$ *cierta* si X no es imprimible
- $IA(X)$ *cierta* si la auto-aplicación X es imprimible
- $\neg IA(X)$ *cierta* si la auto-aplicación X no es imprimible

Teorema: Si toda sentencia imprimible es cierta, hay una cierta no imprimible.

Demostración: $\neg IA(\neg IA)$ es cierta $\iff$ la auto-aplicación $\neg IA$ no es imprimible
$\iff$ $\neg IA(\neg IA)$ no es imprimible.

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \ , \ A \ , \ (\ , \ ) \ , \ \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

- $I(X)$ *cierta* si X es imprimible
- $\neg I(X)$ *cierta* si X no es imprimible
- $IA(X)$ *cierta* si la auto-aplicación X es imprimible
- $\neg IA(X)$ *cierta* si la auto-aplicación X no es imprimible

Teorema: Si toda sentencia imprimible es cierta, hay una cierta no imprimible.

Demostración: $\neg IA(\neg IA)$ es cierta $\iff$ la auto-aplicación $\neg IA$ no es imprimible
$\iff$ $\neg IA(\neg IA)$ no es imprimible.

Si $\neg IA(\neg IA)$ fuese falsa,

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \ , \ A \ , \ (\ , \ ) \ , \ \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

- $I(X)$ *cierta* si X es imprimible
- $\neg I(X)$ *cierta* si X no es imprimible
- $IA(X)$ *cierta* si la auto-aplicación X es imprimible
- $\neg IA(X)$ *cierta* si la auto-aplicación X no es imprimible

Teorema: Si toda sentencia imprimible es cierta, hay una cierta no imprimible.

Demostración: $\neg IA(\neg IA)$ es cierta $\iff$ la auto-aplicación $\neg IA$ no es imprimible
$\iff$ $\neg IA(\neg IA)$ no es imprimible.
Si $\neg IA(\neg IA)$ fuese falsa, sería imprimible,

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \ , \ A \ , \ (\ , \ ) \ , \ \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

- $I(X)$ *cierta* si X es imprimible
- $\neg I(X)$ *cierta* si X no es imprimible
- $IA(X)$ *cierta* si la auto-aplicación X es imprimible
- $\neg IA(X)$ *cierta* si la auto-aplicación X no es imprimible

Teorema: Si toda sentencia imprimible es cierta, hay una cierta no imprimible.

Demostración: $\neg IA(\neg IA)$ es cierta $\iff$ la auto-aplicación $\neg IA$ no es imprimible
$\iff$ $\neg IA(\neg IA)$ no es imprimible.
Si $\neg IA(\neg IA)$ fuese falsa, sería imprimible, y por lo tanto cierta.

Impresora de verdades

Consideremos una máquina que imprime secuencias usando los caracteres:

$$I \ , \ A \ , \ (\ , \ ) \ , \ \neg$$

Si la maquina imprime una sequencia, decimos que la sequence es *imprimible*.
Dada una sequencia X, la *auto-aplicación* de X es la sequencia $X(X)$.
Llamamos de *sentencia* a las secuencias de la siguiente forma

- $I(X)$ *cierta* si X es imprimible
- $\neg I(X)$ *cierta* si X no es imprimible
- $IA(X)$ *cierta* si la auto-aplicación X es imprimible
- $\neg IA(X)$ *cierta* si la auto-aplicación X no es imprimible

Teorema: Si toda sentencia imprimible es cierta, hay una cierta no imprimible.

Demostración: $\neg IA(\neg IA)$ es cierta $\iff$ la auto-aplicación $\neg IA$ no es imprimible
$\iff$ $\neg IA(\neg IA)$ no es imprimible.
Si $\neg IA(\neg IA)$ fuese falsa, sería imprimible, y por lo tanto cierta.

Observación: $IA(IA)$ es cierta $\iff$ es imprimible

Quines: Programas que se imprimen a si mismos

Observación: $IA(IA)$ es cierta $\iff$ es imprimible

Quines: Programas que se imprimen a si mismos

Observación: $IA(IA)$ es cierta $\iff$ es imprimible

En Python:

Quines: Programas que se imprimen a si mismos

Observación: $IA(IA)$ es cierta $\iff$ es imprimible

En Python:

```
quine = 'quine =%r \nprint quine%% quine'
print quine% quine
```

Quines: Programas que se imprimen a si mismos

Observación: $IA(IA)$ es cierta $\iff$ es imprimible

En Python:

```
quine = 'quine =%r \nprint quine%% quine'
print quine% quine
```

En español:

Quines: Programas que se imprimen a si mismos

Observación: $IA(IA)$ es cierta $\iff$ es imprimible

En Python:

```
quine = 'quine =%r \nprint quine%% quine'
print quine% quine
```

En español:

Imprime la siguiente frase dos veces:
Imprime la siguiente frase dos veces:

Quines: Programas que se imprimen a si mismos

Observación: $IA(IA)$ es cierta $\iff$ es imprimible

En Python:

```
quine = 'quine =%r \nprint quine%% quine'
print quine% quine
```

En español:

Imprime la siguiente frase dos veces:
Imprime la siguiente frase dos veces:

¿En código genético?

Aritmetización de la lógica

Asignemos un número a cada caracter:

$+$	$=$	0	ν	$\prime$	$\&$	$\neg$	$\forall$	$($	$)$
0	1	2	3	4	5	6	7	8	9

Aritmetización de la lógica

Asignemos un número a cada caracter:

$+$	$=$	0	v	$\prime$	$\&$	$\neg$	$\forall$	$($	$)$
0	1	2	3	4	5	6	7	8	9

y luego un número a cada secuencia de caracteres:

$$\forall v \neg \forall \neg v' (v' + v = 0)$$

$$7\ 3.\ 6\ 7\ 6.\ 3\ 4\ 8.\ 4\ 5\ 0.\ 3\ 1\ 9$$

Aritmetización de la lógica

Asignemos un número a cada caracter:

$+$	$=$	0	v	$\prime$	$\&$	$\neg$	$\forall$	$($	$)$
0	1	2	3	4	5	6	7	8	9

y luego un número a cada secuencia de caracteres:

$$\forall v \neg \forall \neg v' (v' + v = 0)$$

$$7\ 3.\ 6\ 7\ 6.\ 3\ 4\ 8.\ 4\ 5\ 0.\ 3\ 1\ 9$$

Ejemplo Sea $\texttt{And}(n, m, k)$: la formula que dice $\theta_k \equiv \theta_n \& \theta_m$

$$\exists i, a, b\ (10^{i-1} \leq m < 10^i\ \&\ k = n \times 10^{i+1} + 5 \times 10^i + m)$$

Aritmetización de la lógica

Asignemos un número a cada caracter:

$+$	$=$	0	v	$\prime$	$\&$	$\neg$	$\forall$	$($	$)$
0	1	2	3	4	5	6	7	8	9

y luego un número a cada secuencia de caracteres:

$$\forall v \neg \forall \neg v' (v' + v = 0)$$

$$7\ 3.\ 6\ 7\ 6.\ 3\ 4\ 8.\ 4\ 5\ 0.\ 3\ 1\ 9$$

Ejemplo Sea $\texttt{And}(n, m, k)$: la formula que dice $\theta_k \equiv \theta_n \& \theta_m$

$$\exists i, a, b\ (10^{i-1} \leq m < 10^i\ \&\ k = n \times 10^{i+1} + 5 \times 10^i + m)$$

Lema: Existe formulas $\varphi_{bf}(x)$ y $\varphi_p(x)$ de la aritmética tales que:

- $\varphi_{bf}(n) \iff n$ es el código de una formula bien formada

Aritmetización de la lógica

Asignemos un número a cada caracter:

$+$	$=$	0	v	$\prime$	$\&$	$\neg$	$\forall$	$($	$)$
0	1	2	3	4	5	6	7	8	9

y luego un número a cada secuencia de caracteres:

$$\forall v \neg \forall \neg v' (v' + v = 0)$$

$$7\ 3.\ 6\ 7\ 6.\ 3\ 4\ 8.\ 4\ 5\ 0.\ 3\ 1\ 9$$

Ejemplo Sea $\texttt{And}(n, m, k)$: la formula que dice $\theta_k \equiv \theta_n \& \theta_m$

$$\exists i, a, b\ (10^{i-1} \leq m < 10^i\ \&\ k = n \times 10^{i+1} + 5 \times 10^i + m)$$

Lema: Existe formulas $\varphi_{bf}(x)$ y $\varphi_p(x)$ de la aritmética tales que:

- $\varphi_{bf}(n) \iff n$ es el código de una formula bien formada
- $\varphi_p(m) \iff m = 2^{n_0} \cdot 3^{n_1} \cdot \ldots \cdot p_k^{n_k}$ y
  $n_0, \ldots, n_k$ son códigos de formulas en una demostración formal.

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times)$ $\iff$

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$
y toma los valores 0 o 1.

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$
y toma los valores 0 o 1.

- y es una variable libre

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$
y toma los valores 0 o 1.

- y es una variable libre
- x es una variable ligada

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$
y toma los valores 0 o 1.

- y es una variable libre
- x es una variable ligada

Escribimos $\varphi(y)$ para enfatizar que y es libre,

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$
y toma los valores 0 o 1.

- y es una variable libre
- x es una variable ligada

Escribimos $\varphi(y)$ para enfatizar que y es libre, y $\varphi(t)$ al remplazar y por t.

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$
y toma los valores 0 o 1.

- y es una variable libre
- x es una variable ligada

Escribimos $\varphi(y)$ para enfatizar que y es libre, y $\varphi(t)$ al remplazar y por t.

Las formulas que no tienen variables libres se llaman sentencias.

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$ y toma los valores 0 o 1.

- y es una variable libre
- x es una variable ligada

Escribimos $\varphi(y)$ para enfatizar que y es libre, y $\varphi(t)$ al remplazar y por t.

Las formulas que no tienen variables libres se llaman sentencias.

Sea $\theta_0, \theta_1, \theta_2, ...$ una enumeración de todas las sentencias.

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$
y toma los valores 0 o 1.

- y es una variable libre
- x es una variable ligada

Escribimos $\varphi(y)$ para enfatizar que y es libre, y $\varphi(t)$ al remplazar y por t.

Las formulas que no tienen variables libres se llaman sentencias.

Sea $\theta_0, \theta_1, \theta_2, ...$ una enumeración de todas las sentencias.
Sea $\theta_0^1(x), \theta_1^1(x), \theta_2^1(x), ...$ una enumeración de las formulas con una variable libre.

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$
y toma los valores 0 o 1.

- y es una variable libre
- x es una variable ligada

Escribimos $\varphi(y)$ para enfatizar que y es libre, y $\varphi(t)$ al remplazar y por t.

Las formulas que no tienen variables libres se llaman sentencias.

Sea $\theta_0, \theta_1, \theta_2, ...$ una enumeración de todas las sentencias.
Sea $\theta_0^1(x), \theta_1^1(x), \theta_2^1(x), ...$ una enumeración de las formulas con una variable libre.

Lema: Hay una fórmula $\texttt{Rem}(m, k, n)$ es cierta sii θ_n es $\theta_m^1(k)$.

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$ y toma los valores 0 o 1.

- y es una variable libre
- x es una variable ligada

Escribimos $\varphi(y)$ para enfatizar que y es libre, y $\varphi(t)$ al remplazar y por t.

Las formulas que no tienen variables libres se llaman sentencias.

Sea $\theta_0, \theta_1, \theta_2, ...$ una enumeración de todas las sentencias.
Sea $\theta_0^1(x), \theta_1^1(x), \theta_2^1(x), ...$ una enumeración de las formulas con una variable libre.

Lema: Hay una fórmula $\texttt{Rem}(m, k, n)$ es cierta sii θ_n es $\theta_m^1(k)$.

Lema: Si Γ es aritméticamente definible,

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$ y toma los valores 0 o 1.

- y es una variable libre
- x es una variable ligada

Escribimos $\varphi(y)$ para enfatizar que y es libre, y $\varphi(t)$ al remplazar y por t.

Las formulas que no tienen variables libres se llaman sentencias.

Sea $\theta_0, \theta_1, \theta_2, \ldots$ una enumeración de todas las sentencias.
Sea $\theta_0^1(x), \theta_1^1(x), \theta_2^1(x), \ldots$ una enumeración de las formulas con una variable libre.

Lema: Hay una fórmula $\mathtt{Rem}(m, k, n)$ es cierta sii θ_n es $\theta_m^1(k)$.

Lema: Si Γ es aritméticamente definible, (i.e. existe formula ψ tal que $\Gamma = \{\theta_k : \psi(k)\}$)

Enumeración de las sentencias

La formula φ dada por $\exists x(x + y = 1)$ es cierta en $(\mathbb{N} : 0, 1, +, \times) \iff$
y toma los valores 0 o 1.

- y es una variable libre
- x es una variable ligada

Escribimos $\varphi(y)$ para enfatizar que y es libre, y $\varphi(t)$ al remplazar y por t.

Las formulas que no tienen variables libres se llaman sentencias.

Sea $\theta_0, \theta_1, \theta_2, \ldots$ una enumeración de todas las sentencias.
Sea $\theta_0^1(x), \theta_1^1(x), \theta_2^1(x), \ldots$ una enumeración de las formulas con una variable libre.

Lema: Hay una fórmula $\mathtt{Rem}(m, k, n)$ es cierta sii θ_n es $\theta_m^1(k)$.

Lema: Si Γ es aritméticamente definible, (i.e. existe formula ψ tal que $\Gamma = \{\theta_k : \psi(k)\}$)
hay una fórmula $\mathtt{Pr}_\Gamma(p, n)$ que es cierta sii
p es el código de una prueba de θ_n a partir de Γ.

Teoremas de incompletitud de Gödel, 1931

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que $\theta_e \iff \varphi(e)$

Teoremas de incompletitud de Gödel, 1931

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que $\theta_e \iff \varphi(e)$

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula aritmética que puede identificar las sentencias en Γ)

tenemos que:

- Γ es incompleta (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Teoremas de incompletitud de Gödel, 1931

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que $\theta_e \iff \varphi(e)$

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula aritmética que puede identificar las sentencias en Γ)

tenemos que:

- Γ es incompleta (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Demostración:

recordemos que $\mathtt{Pr}_\Gamma(p, n)$ dice: p es el código de una prueba de θ_n a partir de Γ.

Teoremas de incompletitud de Gödel, 1931

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que $\theta_e \iff \varphi(e)$

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula aritmética que puede identificar las sentencias en Γ)

tenemos que:

- Γ es incompleta (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Demostración:

recordemos que $\mathtt{Pr}_\Gamma(p, n)$ dice: p es el código de una prueba de θ_n a partir de Γ.

Sea $\varphi(x)$ la formula $\neg\exists p\ \mathtt{Pr}_\Gamma(p, x)$: "La sentencia θ_x no es demostrable en Γ".

Teoremas de incompletitud de Gödel, 1931

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que $\theta_e \iff \varphi(e)$

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula aritmética que puede identificar las sentencias en Γ)

tenemos que:

- Γ es incompleta (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Demostración:

recordemos que $\mathtt{Pr}_\Gamma(p, n)$ dice: p es el código de una prueba de θ_n a partir de Γ.

Sea $\varphi(x)$ la formula $\neg\exists p\ \mathtt{Pr}_\Gamma(p, x)$: "La sentencia θ_x no es demostrable en Γ".

Sea e el punto fijo de φ:

Teoremas de incompletitud de Gödel, 1931

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que $\theta_e \iff \varphi(e)$

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula aritmética que puede identificar las sentencias en Γ)

tenemos que:

- Γ es incompleta (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Demostración:

recordemos que $\mathtt{Pr}_\Gamma(p, n)$ dice: p es el código de una prueba de θ_n a partir de Γ.

Sea $\varphi(x)$ la formula $\neg\exists p\ \mathtt{Pr}_\Gamma(p, x)$: "La sentencia θ_x no es demostrable en Γ".

Sea e el punto fijo de φ: $(\theta_e \iff \neg\exists p\ \mathtt{Pr}_\Gamma(p, e))$.

Teoremas de incompletitud de Gödel, 1931

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que $\theta_e \iff \varphi(e)$

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula aritmética que puede identificar las sentencias en Γ)

tenemos que:

- Γ es incompleta (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Demostración:

recordemos que $\mathtt{Pr}_\Gamma(p, n)$ dice: p es el código de una prueba de θ_n a partir de Γ.

Sea $\varphi(x)$ la formula $\neg\exists p\ \mathtt{Pr}_\Gamma(p, x)$: "La sentencia θ_x no es demostrable en Γ".

Sea e el punto fijo de φ: $(\theta_e \iff \neg\exists p\ \mathtt{Pr}_\Gamma(p, e))$.

θ_e es cierta sii θ_e no es demostrable en Γ

Teoremas de incompletitud de Gödel, 1931

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que $\theta_e \iff \varphi(e)$

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula aritmética que puede identificar las sentencias en Γ)

tenemos que:

- Γ es incompleta (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Demostración:

recordemos que $\mathtt{Pr}_\Gamma(p, n)$ dice: p es el código de una prueba de θ_n a partir de Γ.

Sea $\varphi(x)$ la formula $\neg\exists p\ \mathtt{Pr}_\Gamma(p, x)$: "La sentencia θ_x no es demostrable en Γ".

Sea e el punto fijo de φ: $(\theta_e \iff \neg\exists p\ \mathtt{Pr}_\Gamma(p, e))$.

θ_e es cierta sii θ_e no es demostrable en Γ

θ_e tiene que ser cierta, no demostrable,

Teoremas de incompletitud de Gödel, 1931

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que $\theta_e \iff \varphi(e)$

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula aritmética que puede identificar las sentencias en Γ)

tenemos que:

- Γ es incompleta (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Demostración:

recordemos que $\mathtt{Pr}_\Gamma(p, n)$ dice: p es el código de una prueba de θ_n a partir de Γ.

Sea $\varphi(x)$ la formula $\neg\exists p\ \mathtt{Pr}_\Gamma(p, x)$: "La sentencia θ_x no es demostrable en Γ".

Sea e el punto fijo de φ: $(\theta_e \iff \neg\exists p\ \mathtt{Pr}_\Gamma(p, e))$.

θ_e es cierta sii θ_e no es demostrable en Γ

θ_e tiene que ser cierta, no demostrable, y no refutable.

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$,

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$.

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Sea f el código de $\varphi(a(x))$,

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Sea f el código de $\varphi(a(x))$, i.e. $\theta^1_f(x) \iff \varphi(a(x))$. $FA(\cdot)$

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Sea f el código de $\varphi(a(x))$, i.e. $\theta^1_f(x) \iff \varphi(a(x))$. $FA(\cdot)$

$\theta^1_f(x)$ es $\exists y\ (\texttt{Rem}(x, x, y)\ \&\ \varphi(y))$

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Sea f el código de $\varphi(a(x))$, i.e. $\theta^1_f(x) \iff \varphi(a(x))$. $FA(\cdot)$

$\theta^1_f(x)$ es $\exists y\ (\texttt{Rem}(x, x, y)\ \&\ \varphi(y))$

Sea $e = a(f)$. $A(FA)$

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Sea f el código de $\varphi(a(x))$, i.e. $\theta^1_f(x) \iff \varphi(a(x))$. $FA(\cdot)$

$\theta^1_f(x)$ es $\exists y\ (\texttt{Rem}(x, x, y)\ \&\ \varphi(y))$

Sea $e = a(f)$. $A(FA)$

Concluímos que $\theta_e \iff$

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Sea f el código de $\varphi(a(x))$, i.e. $\theta^1_f(x) \iff \varphi(a(x))$. $FA(\cdot)$

$\theta^1_f(x)$ es $\exists y\ (\texttt{Rem}(x, x, y)\ \&\ \varphi(y))$

Sea $e = a(f)$. $A(FA)$

Concluímos que $\theta_e \iff \theta_{a(f)} \iff$

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Sea f el código de $\varphi(a(x))$, i.e. $\theta^1_f(x) \iff \varphi(a(x))$. $FA(\cdot)$

$\theta^1_f(x)$ es $\exists y\ (\texttt{Rem}(x, x, y)\ \&\ \varphi(y))$

Sea $e = a(f)$. $A(FA)$

Concluímos que $\theta_e \iff \theta_{a(f)} \iff \theta^1_f(f) \iff$

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Sea f el código de $\varphi(a(x))$, i.e. $\theta^1_f(x) \iff \varphi(a(x))$. $FA(\cdot)$

$\theta^1_f(x)$ es $\exists y\ (\texttt{Rem}(x, x, y)\ \&\ \varphi(y))$

Sea $e = a(f)$. $A(FA)$

Concluímos que $\theta_e \iff \theta_{a(f)} \iff \theta^1_f(f) \iff \varphi(a(f)) \iff$

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$\theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Sea f el código de $\varphi(a(x))$, i.e. $\theta^1_f(x) \iff \varphi(a(x))$. $FA(\cdot)$

$\theta^1_f(x)$ es $\exists y\ (\texttt{Rem}(x, x, y)\ \&\ \varphi(y))$

Sea $e = a(f)$. $A(FA)$

Concluímos que $\theta_e \iff \theta_{a(f)} \iff \theta^1_f(f) \iff \varphi(a(f)) \iff \varphi(e)$.

Autoreferencia

Lema: Para toda formula aritmética $\varphi(x)$, existe $e \in \mathbb{N}$, tal que

$$PA \vdash \theta_e \iff \varphi(e)$$

Demostración del teorema:

recordemos que $A(X) \to X(X)$, y que $A(FA) \to FA(FA)$.

recordemos también que $\texttt{Rem}(m, k, n)$ dice que θ_n es $\theta^1_m(k)$.

Sea $a(x)$ el código de $\theta^1_x(x)$, i.e. $\theta_{a(x)} \iff \theta^1_x(x)$. $A(\cdot)$

Sea f el código de $\varphi(a(x))$, i.e. $\theta^1_f(x) \iff \varphi(a(x))$. $FA(\cdot)$

$\theta^1_f(x)$ es $\exists y\ (\texttt{Rem}(x, x, y)\ \&\ \varphi(y))$

Sea $e = a(f)$. $A(FA)$

Concluímos que $\theta_e \iff \theta_{a(f)} \iff \theta^1_f(f) \iff \varphi(a(f)) \iff \varphi(e)$.

Segundo teorema de incompletitud

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente
- Γ es computable

tenemos que:

- $\Gamma \nvdash \mathit{Con}(\Gamma)$,

donde $\mathit{Con}(\Gamma)$ es la sentencia aritmética $\neg\exists p\ \mathtt{Pr}_\Gamma(p, k_0)$,

donde θ_{k_0} es "$0 = 1 \wedge 0 \neq 1$".

$\mathit{Con}(\Gamma)$ que dice que no existe prueba de una contradicción a partir de Γ.

Segundo teorema de incompletitud

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente
- Γ es computable

tenemos que:

- $\Gamma \nvdash \mathit{Con}(\Gamma)$,

donde $\mathit{Con}(\Gamma)$ es la sentencia aritmética $\neg\exists p\ \mathtt{Pr}_\Gamma(p, k_0)$,

donde θ_{k_0} es "$0 = 1 \wedge 0 \neq 1$".

$\mathit{Con}(\Gamma)$ que dice que no existe prueba de una contradicción a partir de Γ.

La demostración requiere formalizar el primer teorema de incompletitud:

Segundo teorema de incompletitud

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente
- Γ es computable

tenemos que:

- $\Gamma \nvdash \mathit{Con}(\Gamma)$,

donde $\mathit{Con}(\Gamma)$ es la sentencia aritmética $\neg\exists p\ \mathtt{Pr}_\Gamma(p, k_0)$,

donde θ_{k_0} es "$0 = 1 \wedge 0 \neq 1$".

$\mathit{Con}(\Gamma)$ que dice que no existe prueba de una contradicción a partir de Γ.

La demostración requiere formalizar el primer teorema de incompletitud:
O sea, probar que $\mathit{PA} \vdash \mathit{Con}(\Gamma) \rightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e_0)$ donde $\mathit{PA} \vdash \theta_e \leftrightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e)$.

Segundo teorema de incompletitud

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente
- Γ es computable

tenemos que:

- $\Gamma \nvdash Con(\Gamma)$,

donde $Con(\Gamma)$ es la sentencia aritmética $\neg\exists p\ \mathtt{Pr}_\Gamma(p, k_0)$,

donde θ_{k_0} es "$0 = 1 \wedge 0 \neq 1$".

$Con(\Gamma)$ que dice que no existe prueba de una contradicción a partir de Γ.

La demostración requiere formalizar el primer teorema de incompletitud:
O sea, probar que $PA \vdash Con(\Gamma) \rightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e_0)$ donde $PA \vdash \theta_e \leftrightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e)$.

Si $\Gamma \vdash Con(\Gamma) \Longrightarrow$

Segundo teorema de incompletitud

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente
- Γ es computable

tenemos que:

- $\Gamma \nvdash Con(\Gamma)$,

donde $Con(\Gamma)$ es la sentencia aritmética $\neg\exists p\ \mathtt{Pr}_\Gamma(p, k_0)$,

donde θ_{k_0} es "$0 = 1 \wedge 0 \neq 1$".

$Con(\Gamma)$ que dice que no existe prueba de una contradicción a partir de Γ.

La demostración requiere formalizar el primer teorema de incompletitud:
O sea, probar que $PA \vdash Con(\Gamma) \rightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e_0)$ donde $PA \vdash \theta_e \leftrightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e)$.

Si $\Gamma \vdash Con(\Gamma) \Longrightarrow \Gamma \vdash \theta_e \Longrightarrow$

Segundo teorema de incompletitud

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente
- Γ es computable

tenemos que:

- $\Gamma \nvdash \mathit{Con}(\Gamma)$,

donde $\mathit{Con}(\Gamma)$ es la sentencia aritmética $\neg\exists p\ \mathtt{Pr}_\Gamma(p, k_0)$,

donde θ_{k_0} es "$0 = 1 \wedge 0 \neq 1$".

$\mathit{Con}(\Gamma)$ que dice que no existe prueba de una contradicción a partir de Γ.

La demostración requiere formalizar el primer teorema de incompletitud:
O sea, probar que $\mathit{PA} \vdash \mathit{Con}(\Gamma) \rightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e_0)$ donde $\mathit{PA} \vdash \theta_e \leftrightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e)$.

Si $\Gamma \vdash \mathit{Con}(\Gamma) \Longrightarrow \Gamma \vdash \theta_e \Longrightarrow \mathit{PA} \vdash \exists p\ \mathtt{Pr}_\Gamma(p, e) \Longrightarrow$

Segundo teorema de incompletitud

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente
- Γ es computable

tenemos que:

- $\Gamma \nvdash \mathit{Con}(\Gamma)$,

donde $\mathit{Con}(\Gamma)$ es la sentencia aritmética $\neg\exists p\ \mathtt{Pr}_\Gamma(p, k_0)$,

donde θ_{k_0} es "$0 = 1 \wedge 0 \neq 1$".

$\mathit{Con}(\Gamma)$ que dice que no existe prueba de una contradicción a partir de Γ.

La demostración requiere formalizar el primer teorema de incompletitud:
O sea, probar que $\mathit{PA} \vdash \mathit{Con}(\Gamma) \rightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e_0)$ donde $\mathit{PA} \vdash \theta_e \leftrightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e)$.

Si $\Gamma \vdash \mathit{Con}(\Gamma) \Longrightarrow \Gamma \vdash \theta_e \Longrightarrow \mathit{PA} \vdash \exists p\ \mathtt{Pr}_\Gamma(p, e) \Longrightarrow \mathit{PA} \vdash \neg\theta_e \Longrightarrow$

Segundo teorema de incompletitud

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente
- Γ es computable

tenemos que:

- $\Gamma \nvdash Con(\Gamma)$,

donde $Con(\Gamma)$ es la sentencia aritmética $\neg\exists p\ \mathtt{Pr}_\Gamma(p, k_0)$,

donde θ_{k_0} es "$0 = 1 \wedge 0 \neq 1$".

$Con(\Gamma)$ que dice que no existe prueba de una contradicción a partir de Γ.

La demostración requiere formalizar el primer teorema de incompletitud:
O sea, probar que $PA \vdash Con(\Gamma) \rightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e_0)$ donde $PA \vdash \theta_e \leftrightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e)$.

Si $\Gamma \vdash Con(\Gamma) \Longrightarrow \Gamma \vdash \theta_e \Longrightarrow PA \vdash \exists p\ \mathtt{Pr}_\Gamma(p, e) \Longrightarrow PA \vdash \neg\theta_e \Longrightarrow \Gamma \vdash \neg\theta_e \Longrightarrow$

Segundo teorema de incompletitud

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente
- Γ es computable

tenemos que:

- $\Gamma \nvdash \mathit{Con}(\Gamma)$,

donde $\mathit{Con}(\Gamma)$ es la sentencia aritmética $\neg\exists p\ \mathtt{Pr}_\Gamma(p, k_0)$,

donde θ_{k_0} es "$0 = 1 \wedge 0 \neq 1$".

$\mathit{Con}(\Gamma)$ que dice que no existe prueba de una contradicción a partir de Γ.

La demostración requiere formalizar el primer teorema de incompletitud:
O sea, probar que $\mathit{PA} \vdash \mathit{Con}(\Gamma) \rightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e_0)$ donde $\mathit{PA} \vdash \theta_e \leftrightarrow \neg\exists p\ \mathtt{Pr}_\Gamma(p, e)$.

Si $\Gamma \vdash \mathit{Con}(\Gamma) \Longrightarrow \Gamma \vdash \theta_e \Longrightarrow \mathit{PA} \vdash \exists p\ \mathtt{Pr}_\Gamma(p, e) \Longrightarrow \mathit{PA} \vdash \neg\theta_e \Longrightarrow \Gamma \vdash \neg\theta_e \Longrightarrow$

Γ es inconsistente.

Jerarquía de teorías

Recordemos que

- PA es la lista de axiomas de la aritmética de Peano
- SOA es la lista de axiomas de la aritmética de segundo orden
- ZFC es la lista de axiomas de la teoría de conjuntos

Jerarquía de teorías

Recordemos que

- PA es la lista de axiomas de la aritmética de Peano
- SOA es la lista de axiomas de la aritmética de segundo orden
- ZFC es la lista de axiomas de la teoría de conjuntos

$$PA \vdash \quad \text{Con(ZFC)} \implies \text{Con(SOA)} \implies \text{Con(PA)}$$

Jerarquía de teorías

Recordemos que

- PA es la lista de axiomas de la aritmética de Peano
- SOA es la lista de axiomas de la aritmética de segundo orden
- ZFC es la lista de axiomas de la teoría de conjuntos

$$PA \vdash \quad \text{Con(ZFC)} \implies \text{Con(SOA)} \implies \text{Con(PA)}$$

SOA puede demostrar que existe un modelo de PA.

Jerarquía de teorías

Recordemos que

- PA es la lista de axiomas de la aritmética de Peano
- SOA es la lista de axiomas de la aritmética de segundo orden
- ZFC es la lista de axiomas de la teoría de conjuntos

$$PA \vdash \quad \text{Con(ZFC)} \implies \text{Con(SOA)} \implies \text{Con(PA)}$$

SOA puede demostrar que existe un modelo de PA.
Por lo tanto $Con(PA)$ es demostrable en SOA, pero no en PA.

Jerarquía de teorías

Recordemos que

- PA es la lista de axiomas de la aritmética de Peano
- SOA es la lista de axiomas de la aritmética de segundo orden
- ZFC es la lista de axiomas de la teoría de conjuntos

$$PA \vdash \quad \text{Con(ZFC)} \implies \text{Con(SOA)} \implies \text{Con(PA)}$$

SOA puede demostrar que existe un modelo de PA.
Por lo tanto $Con(PA)$ es demostrable en SOA, pero no en PA.

ZFC puede demostrar que existe un modelo de SOA.

Jerarquía de teorías

Recordemos que

- PA es la lista de axiomas de la aritmética de Peano
- SOA es la lista de axiomas de la aritmética de segundo orden
- ZFC es la lista de axiomas de la teoría de conjuntos

$$PA \vdash \quad \text{Con(ZFC)} \implies \text{Con(SOA)} \implies \text{Con(PA)}$$

SOA puede demostrar que existe un modelo de PA.
Por lo tanto $Con(PA)$ es demostrable en SOA, pero no en PA.

ZFC puede demostrar que existe un modelo de SOA.
Por lo tanto $Con(SOA)$ es demostrable en ZFC, pero no en SOA.

Jerarquía de teorías

Recordemos que

- PA es la lista de axiomas de la aritmética de Peano
- SOA es la lista de axiomas de la aritmética de segundo orden
- ZFC es la lista de axiomas de la teoría de conjuntos

$$PA \vdash \quad \text{Con(ZFC)} \implies \text{Con(SOA)} \implies \text{Con(PA)}$$

SOA puede demostrar que existe un modelo de PA.
Por lo tanto $Con(PA)$ es demostrable en SOA, pero no en PA.

ZFC puede demostrar que existe un modelo de SOA.
Por lo tanto $Con(SOA)$ es demostrable en ZFC, pero no en SOA.

Fenómeno empírico: Si Γ y Γ' con conjuntos de axiomas naturales,

$$Con(\Gamma) \implies Con(\Gamma') \quad \text{o} \quad Con(\Gamma') \implies Con(\Gamma)$$

Parte de mi investigación ha estado dedicada a entender este fenómeno.

La próxima charla

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es verdadero
- Γ es aritméticamente definible

(hay una formula arithmética puede identificar las sentencias en Γ) ,

tenemos que:

- Γ es incompleto (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

La próxima charla

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente $(\Gamma \nvdash 0 \neq 1 \wedge 0 = 1)$,
- Γ es computable

(hay un algoritmo que puede identificar las sentencias en Γ),

tenemos que:

- Γ es incompleto (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Esta versión requiere ideas de Rosser de 1936.

La próxima charla

Teorema: Si Γ es una lista de axiomas tales que:

- Γ contiene a la aritmética de Peano,
- Γ es consistente $(\Gamma \nvdash 0 \neq 1 \wedge 0 = 1)$,
- Γ es computable

(hay un algoritmo que puede identificar las sentencias en Γ),

tenemos que:

- Γ es incompleto (hay una sentencia φ tal que $\Gamma \nvdash \varphi$ y $\Gamma \nvdash \neg\varphi$).

Esta versión requiere ideas de Rosser de 1936.

Teorema

No existe algoritmo que decida si una sentencia es demostrable o no.

Tercera Charla: Indecidibilidad

Programa de Hilbert

Objetivo:

Encontrar un algoritmo que decida si un enunciado es demostrable o no.

Programa de Hilbert

Objetivo:

Encontrar un algoritmo que decida si un enunciado es demostrable o no.

Teorema: Esto no es posible.

Programa de Hilbert

Objetivo:

Encontrar un algoritmo que decida si un enunciado es demostrable o no.

Teorema: Esto no es posible.

Más aún:

Los conjuntos

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

son *computablemente inseparables*.

Programa de Hilbert

Objetivo:
Encontrar un algoritmo que decida si un enunciado es demostrable o no.

Teorema: Esto no es posible.

Más aún:
Los conjuntos

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

son *computablemente inseparables*.

Usaremos esto para demostrar
la versión completa del primer teorema de incompletitud de Gödel.

Conjuntos definibles

Consideremos $\mathcal{M} = (\mathbb{N}, ... : 0, 1, +, \times, ...)$

Conjuntos definibles

Consideremos $\mathcal{M} = (\mathbb{N}, \ldots : 0, 1, +, \times, \ldots)$

Def: Decimos que $A \subseteq \mathbb{N}$ es definible en $\mathcal{M}$ si hay una formula $\varphi(x)$ tal que $A = \{n \in \mathbb{N} : \mathcal{M} \models \varphi(n)\}$

Conjuntos definibles

Consideremos $\mathcal{M} = (\mathbb{N}, \ldots : 0, 1, +, \times, \ldots)$

Def: Decimos que $A \subseteq \mathbb{N}$ es definible en $\mathcal{M}$ si hay una formula $\varphi(x)$ tal que $A = \{n \in \mathbb{N} : \mathcal{M} \models \varphi(n)\}$

Teorema: ("Indefinibilidad de la verdad" de Tarski) $Th_M = \{n \in \mathbb{N} : \mathcal{M} \models \theta_n\}$ no es definible en $\mathcal{M}$.

Conjuntos definibles

Consideremos $\mathcal{M} = (\mathbb{N}, \ldots : 0, 1, +, \times, \ldots)$

Def: Decimos que $A \subseteq \mathbb{N}$ es definible en $\mathcal{M}$ si hay una formula $\varphi(x)$ tal que $A = \{n \in \mathbb{N} : \mathcal{M} \models \varphi(n)\}$

Teorema: ("Indefinibilidad de la verdad" de Tarski) $Th_M = \{n \in \mathbb{N} : \mathcal{M} \models \theta_n\}$ no es definible en $\mathcal{M}$.

En particular,

- el conjunto de sentencias verdaderas en $(\mathbb{N} : 0, 1, +, \times)$ no es definible en $(\mathbb{N} : 0, 1, +, \times)$.

Conjuntos definibles

Consideremos $\mathcal{M} = (\mathbb{N}, \ldots : 0, 1, +, \times, \ldots)$

Def: Decimos que $A \subseteq \mathbb{N}$ es definible en $\mathcal{M}$ si hay una formula $\varphi(x)$ tal que $A = \{n \in \mathbb{N} : \mathcal{M} \models \varphi(n)\}$

Teorema: ("Indefinibilidad de la verdad" de Tarski) $Th_M = \{n \in \mathbb{N} : \mathcal{M} \models \theta_n\}$ no es definible en $\mathcal{M}$.

En particular,

- el conjunto de sentencias verdaderas en $(\mathbb{N} : 0, 1, +, \times)$ no es definible en $(\mathbb{N} : 0, 1, +, \times)$.
- El conjunto de sentencias verdaderas de la teoría de conjuntos

Conjuntos definibles

Consideremos $\mathcal{M} = (\mathbb{N}, \ldots : 0, 1, +, \times, \ldots)$

Def: Decimos que $A \subseteq \mathbb{N}$ es definible en $\mathcal{M}$ si hay una formula $\varphi(x)$ tal que $A = \{n \in \mathbb{N} : \mathcal{M} \models \varphi(n)\}$

Teorema: ("Indefinibilidad de la verdad" de Tarski) $Th_M = \{n \in \mathbb{N} : \mathcal{M} \models \theta_n\}$ no es definible en $\mathcal{M}$.

En particular,

- el conjunto de sentencias verdaderas en $(\mathbb{N} : 0, 1, +, \times)$ no es definible en $(\mathbb{N} : 0, 1, +, \times)$.
- El conjunto de sentencias verdaderas de la teoría de conjuntos es un poco más problemático...

Conjuntos definibles

Consideremos $\mathcal{M} = (\mathbb{N}, \ldots : 0, 1, +, \times, \ldots)$

Def: Decimos que $A \subseteq \mathbb{N}$ es definible en $\mathcal{M}$ si hay una formula $\varphi(x)$ tal que $A = \{n \in \mathbb{N} : \mathcal{M} \models \varphi(n)\}$

Teorema: ("Indefinibilidad de la verdad" de Tarski) $Th_M = \{n \in \mathbb{N} : \mathcal{M} \models \theta_n\}$ no es definible en $\mathcal{M}$.

En particular,

- el conjunto de sentencias verdaderas en $(\mathbb{N} : 0, 1, +, \times)$ no es definible en $(\mathbb{N} : 0, 1, +, \times)$.
- El conjunto de sentencias verdaderas de la teoría de conjuntos es un poco más problemático...

Observación: Si Γ es definible, también lo es $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$.

Conjuntos definibles

Consideremos $\mathcal{M} = (\mathbb{N}, \ldots : 0, 1, +, \times, \ldots)$

Def: Decimos que $A \subseteq \mathbb{N}$ es definible en $\mathcal{M}$ si hay una formula $\varphi(x)$ tal que $A = \{n \in \mathbb{N} : \mathcal{M} \models \varphi(n)\}$

Teorema: ("Indefinibilidad de la verdad" de Tarski) $Th_M = \{n \in \mathbb{N} : \mathcal{M} \models \theta_n\}$ no es definible en $\mathcal{M}$.

En particular,

- el conjunto de sentencias verdaderas en $(\mathbb{N} : 0, 1, +, \times)$ no es definible en $(\mathbb{N} : 0, 1, +, \times)$.
- El conjunto de sentencias verdaderas de la teoría de conjuntos es un poco más problemático...

Observación: Si Γ es definible, también lo es $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$.

Por lo tanto $\mathcal{P}_{PA} \neq Th_{\mathbb{N}}$.

Qué es un algoritmo?

Qué es un algoritmo?

Definición: Una función $f : \mathbb{N} \to \mathbb{N}$ es *computable*
si existe un programa que, con entrada n, devuelve $f(n)$.

Qué es un algoritmo?

Definición: Una función $f : \mathbb{N} \to \mathbb{N}$ es *computable* si existe un programa que, con entrada n, devuelve $f(n)$.

Un conjunto $A \subseteq \omega$ es *computable* si su función característica $\chi_A : \mathbb{N} \to \{0, 1\}$ es computable.

Qué es un algoritmo?

Definición: Una función $f : \mathbb{N} \to \mathbb{N}$ es *computable* si existe un programa que, con entrada n, devuelve $f(n)$.

Un conjunto $A \subseteq \omega$ es *computable* si su función característica $\chi_A : \mathbb{N} \to \{0, 1\}$ es computable.

Tesis de Church-Turing [1930's]: Esta definición es independiente del lenguaje, y captura exactamente la noción de algoritmo.

Qué es un algoritmo?

Definición: Una función $f : \mathbb{N} \to \mathbb{N}$ es *computable* si existe un programa que, con entrada n, devuelve $f(n)$.

Un conjunto $A \subseteq \omega$ es *computable* si su función característica $\chi_A : \mathbb{N} \to \{0, 1\}$ es computable.

Tesis de Church-Turing [1930's]: Esta definición es independiente del lenguaje, y captura exactamente la noción de algoritmo.

Ejemplos: Los siguientes son computables:

- El conjunto de números primos.
- La sequencia de los dígitos de π.
- El conjunto de los programas escritos correctamente.

Qué es un algoritmo?

Definición: Una función $f : \mathbb{N} \to \mathbb{N}$ es *computable* si existe un programa que, con entrada n, devuelve $f(n)$.

Un conjunto $A \subseteq \omega$ es *computable* si su función característica $\chi_A : \mathbb{N} \to \{0, 1\}$ es computable.

Tesis de Church-Turing [1930's]: Esta definición es independiente del lenguaje, y captura exactamente la noción de algoritmo.

Ejemplos: Los siguientes son computables:

- El conjunto de números primos.
- La sequencia de los dígitos de π.
- El conjunto de los programas escritos correctamente.

Pregunta: ¿Porque restringirnos solamente a $\mathbb{N}$?

Qué es un algoritmo?

Definición: Una función $f : \mathbb{N} \to \mathbb{N}$ es *computable* si existe un programa que, con entrada n, devuelve $f(n)$.

Un conjunto $A \subseteq \omega$ es *computable* si su función característica $\chi_A : \mathbb{N} \to \{0, 1\}$ es computable.

Tesis de Church-Turing [1930's]: Esta definición es independiente del lenguaje, y captura exactamente la noción de algoritmo.

Ejemplos: Los siguientes son computables:

- El conjunto de números primos.
- La sequencia de los dígitos de π.
- El conjunto de los programas escritos correctamente.

Pregunta: ¿Porque restringirnos solamente a $\mathbb{N}$?
Todo objeto finito puede ser codificado con un número.

Qué es un algoritmo?

Definición: Una función $f : \mathbb{N} \to \mathbb{N}$ es *computable* si existe un programa que, con entrada n, devuelve $f(n)$.

Un conjunto $A \subseteq \omega$ es *computable* si su función característica $\chi_A : \mathbb{N} \to \{0, 1\}$ es computable.

Tesis de Church-Turing [1930's]: Esta definición es independiente del lenguaje, y captura exactamente la noción de algoritmo.

Ejemplos: Los siguientes son computables:

- El conjunto de números primos.
- La sequencia de los dígitos de π.
- El conjunto de los programas escritos correctamente.

Pregunta: ¿Porque restringirnos solamente a $\mathbb{N}$?

Todo objeto finito puede ser codificado con un número.

Ej: $\langle a_1, ..., a_n \rangle$ puede ser codificada como $2^{a_1} \cdot 3^{a_2} \cdot 5^{a_3} \cdot ... \cdot p_n^{a_n}$.

Ejemplos de conjuntos no computable

El problema de la palabra: Consideren los grupos definidos a partir de una cantidad finita de generadores y de relaciones entre los generadores.

Ejemplos de conjuntos no computable

El problema de la palabra: Consideren los grupos definidos a partir de una cantidad finita de generadores y de relaciones entre los generadores.
El conjunto de pares (generadores, relaciones), de groupos **no-triviales no** es computable.

Ejemplos de conjuntos no computable

El problema de la palabra: Consideren los grupos definidos a partir de una cantidad finita de generadores y de relaciones entre los generadores.
El conjunto de pares (generadores, relaciones), de groupos **no-triviales no** es computable.

Variedades simplemente conexas: El conjunto de triangulaciones de variedades **simplemente conexas no** es computable.

Ejemplos de conjuntos no computable

El problema de la palabra: Consideren los grupos definidos a partir de una cantidad finita de generadores y de relaciones entre los generadores.
El conjunto de pares (generadores, relaciones), de groupos **no-triviales no** es computable.

Variedades simplemente conexas: El conjunto de triangulaciones de variedades **simplemente conexas no** es computable.

El décimo problema de Hilbert: El conjunto de polinomios en $\mathbb{Z}$ con varias variables y con **raíces en** $\mathbb{Z}$ **no** es computable.

Ejemplos de conjuntos no computable

El problema de la palabra: Consideren los grupos definidos a partir de una cantidad finita de generadores y de relaciones entre los generadores.
El conjunto de pares (generadores, relaciones), de groupos **no-triviales no** es computable.

Variedades simplemente conexas: El conjunto de triangulaciones de variedades **simplemente conexas no** es computable.

El décimo problema de Hilbert: El conjunto de polinomios en $\mathbb{Z}$ con varias variables y con **raíces en** $\mathbb{Z}$ **no** es computable.

El problema de la detención: (al que llamaremos K)
El conjunto de programas que **paran no** es computable.

Teorías decidibles

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble,

Teorías decidibles

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble, es decir $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Teorías decidibles

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble, es decir $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Pr: Dado n buscar, una por una, una demostración de $\Gamma \vdash \theta_n$ o $\Gamma \vdash \neg\theta_n$.

Teorías decidibles

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble, es decir $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Pr: Dado n buscar, una por una, una demostración de $\Gamma \vdash \theta_n$ o $\Gamma \vdash \neg\theta_n$.

Las siguientes listas de axiomas son consistentes, decidibles y completas:

Teorías decidibles

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble, es decir $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Pr: Dado n buscar, una por una, una demostración de $\Gamma \vdash \theta_n$ o $\Gamma \vdash \neg\theta_n$.

Las siguientes listas de axiomas son consistentes, decidibles y completas:

- Axiomas de cuerpo algebraicamente cerrados de característica p.

Teorías decidibles

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble, es decir $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Pr: Dado n buscar, una por una, una demostración de $\Gamma \vdash \theta_n$ o $\Gamma \vdash \neg\theta_n$.

Las siguientes listas de axiomas son consistentes, decidibles y completas:

- Axiomas de cuerpo algebraicamente cerrados de característica p.
- Axiomas de cuerpo real cerrado ordenado.

Teorías decidibles

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble, es decir $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Pr: Dado n buscar, una por una, una demostración de $\Gamma \vdash \theta_n$ o $\Gamma \vdash \neg\theta_n$.

Las siguientes listas de axiomas son consistentes, decidibles y completas:

- Axiomas de cuerpo algebraicamente cerrados de característica p.
- Axiomas de cuerpo real cerrado ordenado.
- Aritmética de Presburger, $Th(\mathbb{N} : 0, 1, +)$

Teorías decidibles

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble, es decir $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Pr: Dado n buscar, una por una, una demostración de $\Gamma \vdash \theta_n$ o $\Gamma \vdash \neg\theta_n$.

Las siguientes listas de axiomas son consistentes, decidibles y completas:

- Axiomas de cuerpo algebraicamente cerrados de característica p.
- Axiomas de cuerpo real cerrado ordenado.
- Aritmética de Presburger, $Th(\mathbb{N} : 0, 1, +)$

Teorías decidibles

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble, es decir $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Pr: Dado n buscar, una por una, una demostración de $\Gamma \vdash \theta_n$ o $\Gamma \vdash \neg\theta_n$.

Las siguientes listas de axiomas son consistentes, decidibles y completas:

- Axiomas de cuerpo algebraicamente cerrados de característica p.
- Axiomas de cuerpo real cerrado ordenado.
- Aritmética de Presburger, $Th(\mathbb{N} : 0, 1, +)$

La Teoría de Modelos es un area de la lógica que estudia listas de axiomas con este tipo de limitaciones y busca formas de explotarlas.

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n) \downarrow$ denota que P_e con entrada n eventualmente se detiene.

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow= m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba:

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba: Asumiendo que sí, definimos el siguiente programa:

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba: Asumiendo que sí, definimos el siguiente programa:

```
Con entrada n:
```

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba: Asumiendo que sí, definimos el siguiente programa:

- Chequear si $(n,n) \in K$

Con entrada n:

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba: Asumiendo que sí, definimos el siguiente programa:

```
                              - Chequear si (n, n) ∈ K
Con entrada n:                - Si no, detenerse.
```

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba: Asumiendo que sí, definimos el siguiente programa:

Con entrada n:
- Chequear si $(n,n) \in K$
- Si **no**, detenerse.
- Si **sí**, entrar en loop infinito.

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba: Asumiendo que sí, definimos el siguiente programa:

```
                              - Chequear si (n, n) ∈ K
Con entrada n:                - Si no, detenerse.
                              - Si sí, entrar en loop infinito.
```

Sea e_0 tal que P_{e_0} es este programa.

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba: Asumiendo que sí, definimos el siguiente programa:

```
                              - Chequear si (n, n) ∈ K
Con entrada n:                - Si no, detenerse.
                              - Si sí, entrar en loop infinito.
```

Sea e_0 tal que P_{e_0} es este programa. Luego

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba: Asumiendo que sí, definimos el siguiente programa:

Con entrada n:
- Chequear si $(n,n) \in K$
- Si **no**, detenerse.
- Si **sí**, entrar en loop infinito.

Sea e_0 tal que P_{e_0} es este programa. Luego
$P_{e_0}(e_0)$ se detiene $\iff$

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba: Asumiendo que sí, definimos el siguiente programa:

Con entrada n:
- Chequear si $(n,n) \in K$
- Si **no**, detenerse.
- Si **sí**, entrar en loop infinito.

Sea e_0 tal que P_{e_0} es este programa. Luego
$P_{e_0}(e_0)$ se detiene $\iff$ $(e_0,e_0) \notin K$ $\iff$

El problema de la dentención

Consideremos una enumeración de todos los programas P_0, P_1,....

- $P_e(n)\downarrow$ denota que P_e con entrada n eventualmente se detiene.
- $P_e(n)\downarrow = m$ denota que al detenerse, devuelve m.

Teorema: [Turing 36] Existe un programa $U(e,x)$ tal que $U(e,x) \cong P_e(x)$.

Definición: El problema de la dentención es $K = \{(e,n) \in \mathbb{N}^2 : P_e(n)\downarrow\}$.

Teorema: [Turing 36] K no es computable.

Prueba: Asumiendo que sí, definimos el siguiente programa:

Con entrada n:
- Chequear si $(n,n) \in K$
- Si **no**, detenerse.
- Si **sí**, entrar en loop infinito.

Sea e_0 tal que P_{e_0} es este programa. Luego
$P_{e_0}(e_0)$ se detiene $\iff$ $(e_0,e_0) \notin K$ $\iff$ $P_{e_0}(e_0)$ no se detiene.

Arithmetiazión de la computación

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n)$$

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n)$$

Pr: $\texttt{Prog}(e, n)$ dice:

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

- cada s_i codifica un estado de memoria y l_i es la proxima linea de P_e a correr,

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

- cada s_i codifica un estado de memoria y l_i es la proxima linea de P_e a correr,
- s_0 es la configuración inicial con entrada n, y $l_0 = 1$

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n) \iff PA \vdash \texttt{Prog}(p, n)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

- cada s_i codifica un estado de memoria y l_i es la proxima linea de P_e a correr,
- s_0 es la configuración inicial con entrada n, y $l_0 = 1$
- ejecutando la linea l_i de P_e, uno pasa de s_i a s_{i+1}

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n) \iff PA \vdash \texttt{Prog}(p, n)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

- cada s_i codifica un estado de memoria y l_i es la proxima linea de P_e a correr,
- s_0 es la configuración inicial con entrada n, y $l_0 = 1$
- ejecutando la linea l_i de P_e, uno pasa de s_i a s_{i+1}
- la linea l_k de P_e contiene la instrucción de detenerse.

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n) \iff PA \vdash \texttt{Prog}(p, n)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

- cada s_i codifica un estado de memoria y l_i es la proxima linea de P_e a correr,
- s_0 es la configuración inicial con entrada n, y $l_0 = 1$
- ejecutando la linea l_i de P_e, uno pasa de s_i a s_{i+1}
- la linea l_k de P_e contiene la instrucción de detenerse.

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n) \iff PA \vdash \texttt{Prog}(p, n)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

- cada s_i codifica un estado de memoria y l_i es la proxima linea de P_e a correr,
- s_0 es la configuración inicial con entrada n, y $l_0 = 1$
- ejecutando la linea l_i de P_e, uno pasa de s_i a s_{i+1}
- la linea l_k de P_e contiene la instrucción de detenerse.

Corolario: $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ no es computable.

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n) \iff PA \vdash \texttt{Prog}(p, n)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

- cada s_i codifica un estado de memoria y l_i es la proxima linea de P_e a correr,
- s_0 es la configuración inicial con entrada n, y $l_0 = 1$
- ejecutando la linea l_i de P_e, uno pasa de s_i a s_{i+1}
- la linea l_k de P_e contiene la instrucción de detenerse.

Corolario: $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ no es computable.

Pr: Si $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ fuese computable, también lo sería K:

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n) \iff PA \vdash \texttt{Prog}(p, n)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

- cada s_i codifica un estado de memoria y l_i es la proxima linea de P_e a correr,
- s_0 es la configuración inicial con entrada n, y $l_0 = 1$
- ejecutando la linea l_i de P_e, uno pasa de s_i a s_{i+1}
- la linea l_k de P_e contiene la instrucción de detenerse.

Corolario: $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ no es computable.

Pr: Si $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ fuese computable, también lo sería K:
Para decidir si $(e, n) \in K$,

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n)$ de la aritmética tal que

$$P_e(n)\downarrow \iff \texttt{Prog}(e, n) \iff PA \vdash \texttt{Prog}(p, n)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

- cada s_i codifica un estado de memoria y l_i es la proxima linea de P_e a correr,
- s_0 es la configuración inicial con entrada n, y $l_0 = 1$
- ejecutando la linea l_i de P_e, uno pasa de s_i a s_{i+1}
- la linea l_k de P_e contiene la instrucción de detenerse.

Corolario: $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ no es computable.

Pr: Si $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ fuese computable, también lo sería K:
Para decidir si $(e, n) \in K$,

buscar m tal que θ_m es $\texttt{Prog}(e, n)$ y preguntar si $m \in \mathcal{P}$.

Arithmetiazión de la computación

Teorema: Existe una formula $\texttt{Prog}(e, n, m)$ de la aritmética tal que

$$P_e(n)\downarrow = m \iff \texttt{Prog}(e, n, m) \iff PA \vdash \texttt{Prog}(p, n, m)$$

Pr: $\texttt{Prog}(e, n)$ dice: $\exists\langle(s_0, l_0), (s_1, l_1), (s_2, l_2), ...(s_k, l_k)\rangle$, tal que,

- cada s_i codifica un estado de memoria y l_i es la proxima linea de P_e a correr,
- s_0 es la configuración inicial con entrada n, y $l_0 = 1$
- ejecutando la linea l_i de P_e, uno pasa de s_i a s_{i+1}
- la linea l_k de P_e contiene la instrucción de detenerse.

Corolario: $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ no es computable.

Pr: Si $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ fuese computable, también lo sería K:
Para decidir si $(e, n) \in K$,

buscar m tal que θ_m es $\texttt{Prog}(e, n)$ y preguntar si $m \in \mathcal{P}$.

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C.

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii
existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii
existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:
$e \in C \Rightarrow$

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:
$e \in C \Rightarrow P_e(e) \downarrow = 1 \Rightarrow$

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:
$e \in C \Rightarrow P_e(e) \downarrow = 1 \Rightarrow e \in K_1 \Rightarrow$

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:
$e \in C \Rightarrow P_e(e) \downarrow = 1 \Rightarrow e \in K_1 \Rightarrow e \notin C$.

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii
existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:
$e \in C \Rightarrow P_e(e) \downarrow = 1 \Rightarrow e \in K_1 \Rightarrow e \notin C$.
$e \notin C \Rightarrow$

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii
existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:
$e \in C \Rightarrow P_e(e) \downarrow = 1 \Rightarrow e \in K_1 \Rightarrow e \notin C$.
$e \notin C \Rightarrow P_e(e) \downarrow = 0 \Rightarrow$

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii
existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:
$e \in C \Rightarrow P_e(e) \downarrow = 1 \Rightarrow e \in K_1 \Rightarrow e \notin C$.
$e \notin C \Rightarrow P_e(e) \downarrow = 0 \Rightarrow e \in K_0 \Rightarrow$

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii
existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:
$e \in C \Rightarrow P_e(e) \downarrow = 1 \Rightarrow e \in K_1 \Rightarrow e \notin C$.
$e \notin C \Rightarrow P_e(e) \downarrow = 0 \Rightarrow e \in K_0 \Rightarrow e \in C$.

Conjuntos computablemente inseparables

Definición: $A, B \subseteq \mathbb{N}$ son computablemente separables sii existe un conjunto computable C tal que $A \subseteq C$ y $B \subseteq \mathbb{N} \setminus C$.

Lema: Los siguientes son computablemente **in**separables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Pr: Supongamos que C es computable y los separa: $K_0 \subseteq C \subseteq \mathbb{N} \setminus K_1$
Sea P_e el programa que calcula la función característica de C. Luego:
$e \in C \Rightarrow P_e(e) \downarrow = 1 \Rightarrow e \in K_1 \Rightarrow e \notin C$.
$e \notin C \Rightarrow P_e(e) \downarrow = 0 \Rightarrow e \in K_0 \Rightarrow e \in C$.

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$.

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$. Afirmación: C separa K_0 de K_1.

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$. Afirmación: C separa K_0 de K_1.
$e \in K_0 \Rightarrow$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$. Afirmación: C separa K_0 de K_1.
$e \in K_0 \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{P} \Rightarrow$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$. Afirmación: C separa K_0 de K_1.
$e \in K_0 \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{P} \Rightarrow \texttt{Prog}(e, e, 0) \in D \Rightarrow$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$. Afirmación: C separa K_0 de K_1.
$e \in K_0 \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{P} \Rightarrow \texttt{Prog}(e, e, 0) \in D \Rightarrow e \in C$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$. Afirmación: C separa K_0 de K_1.
$e \in K_0 \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{P} \Rightarrow \texttt{Prog}(e, e, 0) \in D \Rightarrow e \in C$
$e \in K_1 \Rightarrow$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$. Afirmación: C separa K_0 de K_1.
$e \in K_0 \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{P} \Rightarrow \texttt{Prog}(e, e, 0) \in D \Rightarrow e \in C$
$e \in K_1 \Rightarrow \texttt{Prog}(e, e, 1) \in \mathcal{P} \Rightarrow$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$. Afirmación: C separa K_0 de K_1.
$e \in K_0 \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{P} \Rightarrow \texttt{Prog}(e, e, 0) \in D \Rightarrow e \in C$
$e \in K_1 \Rightarrow \texttt{Prog}(e, e, 1) \in \mathcal{P} \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{R} \Rightarrow$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$. Afirmación: C separa K_0 de K_1.
$e \in K_0 \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{P} \Rightarrow \texttt{Prog}(e, e, 0) \in D \Rightarrow e \in C$
$e \in K_1 \Rightarrow \texttt{Prog}(e, e, 1) \in \mathcal{P} \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{R} \Rightarrow \texttt{Prog}(e, e, 0) \notin D \Rightarrow$

Conjuntos computablemente inseparables

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Pr: Sea $\texttt{Prog}(e, n, m) \iff P_e(n) \downarrow = m \iff PA \vdash \texttt{Prog}(e, n, m)$.
Más aún, $PA \vdash \texttt{Prog}(e, n, 1) \to \neg\texttt{Prog}(e, n, 0)$.
Supongamos que D separa $\mathcal{P}$ de $\mathcal{R}$.
Sea $C = \{e \in \mathbb{N} : \texttt{Prog}(e, e, 0) \in D\}$. Afirmación: C separa K_0 de K_1.
$e \in K_0 \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{P} \Rightarrow \texttt{Prog}(e, e, 0) \in D \Rightarrow e \in C$
$e \in K_1 \Rightarrow \texttt{Prog}(e, e, 1) \in \mathcal{P} \Rightarrow \texttt{Prog}(e, e, 0) \in \mathcal{R} \Rightarrow \texttt{Prog}(e, e, 0) \notin D \Rightarrow e \notin C$

Incomplititud – versión completa

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Incomplititud – versión completa

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Incomplititud – versión completa

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble.

Incomplititud – versión completa

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble. I.e.: $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Incomplititud – versión completa

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble. I.e.: $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Incomplititud – versión completa

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble. I.e.: $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Teorema: Si $\Gamma \supseteq PA$ es consistente y computable, entonces Γ es incompleto.

Incomplititud – versión completa

Lema: Los siguientes son computablemente inseparables:

- $K_0 = \{e \in \mathbb{N} : P_e(e) \downarrow = 0\}$
- $K_1 = \{e \in \mathbb{N} : P_e(e) \downarrow = 1\}$

Lema: Los siguientes son computablemente inseparables:

- $\mathcal{P} = \{n \in \mathbb{N} : PA \vdash \theta_n\}$ y
- $\mathcal{R} = \{n \in \mathbb{N} : PA \vdash \neg\theta_n\}$

Lema: Si $\Gamma \supseteq PA$ es consistente, computable, y completo,
Γ es decidble. I.e.: $\mathcal{P}_\Gamma = \{n \in \mathbb{N} : \Gamma \vdash \theta_n\}$ es computable

Teorema: Si $\Gamma \supseteq PA$ es consistente y computable, entonces Γ es incompleto.

Pr: Si Γ fuese completo, $\mathcal{P}_\Gamma$ sería separador computable de $\mathcal{P}$ y $\mathcal{R}$.

10mo problema de Hilbert

[Hilbert 1900]

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.
Más aún, existe un polinomio $p(y, z, x_1, ..x_k) \in \mathbb{Z}[y, z, x_1, ...]$ tal que

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.
Más aún, existe un polinomio $p(y, z, x_1, ..x_k) \in \mathbb{Z}[y, z, x_1, ...]$ tal que para todo $e, n \in \mathbb{N}$, $P_e(n) \downarrow \iff p(e, n, x_1, ..x_k)$ tiene raíces enteras.

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si
un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.
Más aún, existe un polinomio $p(y, z, x_1, ..x_k) \in \mathbb{Z}[y, z, x_1, ...]$ tal que
para todo $e, n \in \mathbb{N}$, $P_e(n)\downarrow \iff p(e, n, x_1, ..x_k)$ tiene raíces enteras.

Corolario: Para toda lista de axiomas $\Gamma \supseteq PA$ computable y consistente,
existe un polinomio $q(\bar{x}) \in \mathbb{Z}[x_1, ...]$ que

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si
un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.
Más aún, existe un polinomio $p(y, z, x_1, ..x_k) \in \mathbb{Z}[y, z, x_1, ...]$ tal que
para todo $e, n \in \mathbb{N}$, $P_e(n)\downarrow \iff p(e, n, x_1, ..x_k)$ tiene raíces enteras.

Corolario: Para toda lista de axiomas $\Gamma \supseteq PA$ computable y consistente,
existe un polinomio $q(\bar{x}) \in \mathbb{Z}[x_1, ...]$ que
no tiene raíces enteras, pero que Γ no puede demostrarlo.

Pr: Consideremos el programa P_e que

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si
un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.
Más aún, existe un polinomio $p(y, z, x_1, ..x_k) \in \mathbb{Z}[y, z, x_1, ...]$ tal que
para todo $e, n \in \mathbb{N}$, $P_e(n)\downarrow \iff p(e, n, x_1, ..x_k)$ tiene raíces enteras.

Corolario: Para toda lista de axiomas $\Gamma \supseteq PA$ computable y consistente,
existe un polinomio $q(\bar{x}) \in \mathbb{Z}[x_1, ...]$ que
no tiene raíces enteras, pero que Γ no puede demostrarlo.

Pr: Consideremos el programa P_e que
`busca demostraciones de` $\Gamma \vdash \psi \& \neg\psi$,

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si
un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.
Más aún, existe un polinomio $p(y, z, x_1, ..x_k) \in \mathbb{Z}[y, z, x_1, ...]$ tal que
para todo $e, n \in \mathbb{N}$, $P_e(n)\downarrow \iff p(e, n, x_1, ..x_k)$ tiene raíces enteras.

Corolario: Para toda lista de axiomas $\Gamma \supseteq PA$ computable y consistente,
existe un polinomio $q(\bar{x}) \in \mathbb{Z}[x_1, ...]$ que
no tiene raíces enteras, pero que Γ no puede demostrarlo.

Pr: Consideremos el programa P_e que
`busca demostraciones de` $\Gamma \vdash \psi \& \neg\psi$`, y si encuentra una se detiene.`

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si
un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.
Más aún, existe un polinomio $p(y, z, x_1, ..x_k) \in \mathbb{Z}[y, z, x_1, ...]$ tal que
para todo $e, n \in \mathbb{N}$, $P_e(n)\downarrow \iff p(e, n, x_1, ..x_k)$ tiene raíces enteras.

Corolario: Para toda lista de axiomas $\Gamma \supseteq PA$ computable y consistente,
existe un polinomio $q(\bar{x}) \in \mathbb{Z}[x_1, ...]$ que
no tiene raíces enteras, pero que Γ no puede demostrarlo.

Pr: Consideremos el programa P_e que
`busca demostraciones de` $\Gamma \vdash \psi \& \neg\psi$`, y si encuentra una se detiene.`
Luego, $\Gamma \vdash Con(\Gamma) \iff P_e\uparrow$

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si
un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.
Más aún, existe un polinomio $p(y, z, x_1, ..x_k) \in \mathbb{Z}[y, z, x_1, ...]$ tal que
para todo $e, n \in \mathbb{N}$, $P_e(n)\downarrow \iff p(e, n, x_1, ..x_k)$ tiene raíces enteras.

Corolario: Para toda lista de axiomas $\Gamma \supseteq PA$ computable y consistente,
existe un polinomio $q(\bar{x}) \in \mathbb{Z}[x_1, ...]$ que
no tiene raíces enteras, pero que Γ no puede demostrarlo.

Pr: Consideremos el programa P_e que
`busca demostraciones de` $\Gamma \vdash \psi \& \neg\psi$, `y si encuentra una se detiene.`
Luego, $\Gamma \vdash Con(\Gamma) \iff P_e\uparrow \iff \forall \bar{x} \in \mathbb{Z}^* \; p(e, \bar{x}) \neq 0$.

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si
un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.
Más aún, existe un polinomio $p(y, z, x_1, ..x_k) \in \mathbb{Z}[y, z, x_1, ...]$ tal que
para todo $e, n \in \mathbb{N}$, $P_e(n)\downarrow \iff p(e, n, x_1, ..x_k)$ tiene raíces enteras.

Corolario: Para toda lista de axiomas $\Gamma \supseteq PA$ computable y consistente,
existe un polinomio $q(\bar{x}) \in \mathbb{Z}[x_1, ...]$ que
no tiene raíces enteras, pero que Γ no puede demostrarlo.

Pr: Consideremos el programa P_e que
`busca demostraciones de` $\Gamma \vdash \psi \& \neg\psi$, `y si encuentra una se detiene.`
Luego, $\Gamma \vdash Con(\Gamma) \iff P_e\uparrow \iff \forall \bar{x} \in \mathbb{Z}^* \; p(e, \bar{x}) \neq 0$.
Como $\Gamma \nvdash Con(\Gamma)$, tenemos que

10mo problema de Hilbert

[Hilbert 1900] ¿Existe un algoritmo que decida si
un polinomio $p(x_1, ...x_k) \in \mathbb{Z}[x_1, ...]$ tiene raíces enteras?

Teorema: [Matiyasevich, Putnam, Davis, Robinson 1970] No.
Más aún, existe un polinomio $p(y, z, x_1, ..x_k) \in \mathbb{Z}[y, z, x_1, ...]$ tal que
para todo $e, n \in \mathbb{N}$, $P_e(n)\downarrow \iff p(e, n, x_1, ..x_k)$ tiene raíces enteras.

Corolario: Para toda lista de axiomas $\Gamma \supseteq PA$ computable y consistente,
existe un polinomio $q(\bar{x}) \in \mathbb{Z}[x_1, ...]$ que
no tiene raíces enteras, pero que Γ no puede demostrarlo.

Pr: Consideremos el programa P_e que
`busca demostraciones de` $\Gamma \vdash \psi \& \neg\psi$, `y si encuentra una se detiene.`
Luego, $\Gamma \vdash Con(\Gamma) \iff P_e\uparrow \iff \forall \bar{x} \in \mathbb{Z}^* \; p(e, \bar{x}) \neq 0$.
Como $\Gamma \nvdash Con(\Gamma)$, tenemos que $\Gamma \nvdash \forall \bar{x} \in \mathbb{Z}^* \; p(e, \bar{x}) \neq 0$.

Áreas de la Lógica

- Teoría de Modelos
- Teoría de Conjuntos
- Teoría de la Computabilidad
- Teoría de pruebas
- Lógica en ciencias de la computación
- Lógicas no clásicas
- Lógica filosófica
- ...