Teoría de la Computación

Clase 26: Lenguajes decidibles

Mauro Artigiani

29 Octubre 2021

Universidad del Rosario, Bogotá

Un poco de historia

Los problemas de Hilbert

En el congreso internacional de matemáticos de Paris de 1900, David Hilbert propuso una lista de veintitrés problemas.

Los problemas de Hilbert

En el congreso internacional de matemáticos de Paris de 1900, David Hilbert propuso una lista de veintitrés problemas. Entro ellos estaban problemas de todas áreas de las matemáticas. Unos, cómo la hipótesis de Riemann, siguen abiertos. Otros han sido solucionado en parte o completamente.

Los problemas de Hilbert

En particular nos interesan dos:

II *"Probar que los axiomas de la aritmética son consistentes (esto es, que la aritmética es un sistema formal que no supone una contradicción)."*

Los problemas de Hilbert

En particular nos interesan dos:

II *"Probar que los axiomas de la aritmética son consistentes (esto es, que la aritmética es un sistema formal que no supone una contradicción)."*
Gödel ha demostrado en 1931 que esto es imposible sin asumir nada más. Pero Gentzen ha demostrado que sí se puede asumiendo algo razonable.

Los problemas de Hilbert

En particular nos interesan dos:

II *"Probar que los axiomas de la aritmética son consistentes (esto es, que la aritmética es un sistema formal que no supone una contradicción)."*
Gödel ha demostrado en 1931 que esto es imposible sin asumir nada más. Pero Gentzen ha demostrado que sí se puede asumiendo algo razonable.

X *"Encontrar un algoritmo que determine si una ecuación polinómica dada con coeficientes enteros tiene solución entera."*

Los problemas de Hilbert

En particular nos interesan dos:

II *"Probar que los axiomas de la aritmética son consistentes (esto es, que la aritmética es un sistema formal que no supone una contradicción)."*
Gödel ha demostrado en 1931 que esto es imposible sin asumir nada más. Pero Gentzen ha demostrado que sí se puede asumiendo algo razonable.

X *"Encontrar un algoritmo que determine si una ecuación polinómica dada con coeficientes enteros tiene solución entera."*
La solución de este problema es otra vez negativa (Matiyasevich, 1970), y nos lleva a una pregunta fundamental: ¿Qué es un algoritmo?

¿Qué es un algoritmo?

¿Qué es un algoritmo?

La RAE dice en su diccionario (`https://dle.rae.es/`):

algoritmo

*Quizá del lat. tardío *algobarismus, y este abrev. del ár. clás. hisābu lgubār 'cálculo mediante cifras arábigas'.*

Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.

¿Qué es un algoritmo?

Nótese que Hilbert no pidió *si* era posible encontrar un algoritmo. De hecho, él, como todos los matemáticos de su época, estaba convencido que todo problema se puede resolver y, por ende, para todo problema existe un algoritmo que lo resuelve.

¿Qué es un algoritmo?

Nótese que Hilbert no pidió *si* era posible encontrar un algoritmo. De hecho, él, como todos los matemáticos de su época, estaba convencido que todo problema se puede resolver y, por ende, para todo problema existe un algoritmo que lo resuelve.

Para demostrar que no existe un algoritmo para un dado problema, tenemos que dar una definición más precisa de algoritmo.

La tesis de Church y Turing

En 1936, independientemente Church y Turing dieron dos definiciones de algoritmo, que después demostraron ser equivalentes.

La tesis de Church y Turing

En 1936, independientemente Church y Turing dieron dos definiciones de algoritmo, que después demostraron ser equivalentes.

Tesis de Church-Turing

La noción intuitiva de algoritmo es *igual* a los algoritmos para máquinas de Turing.

La tesis de Church y Turing

En 1936, independientemente Church y Turing dieron dos definiciones de algoritmo, que después demostraron ser equivalentes.

Tesis de Church-Turing

La noción intuitiva de algoritmo es *igual* a los algoritmos para máquinas de Turing.

Siendo una *tesis*, esto no se puede demostrar.

La tesis de Church y Turing

La tesis de Church-Turing se puede representar en la siguiente manera:

$$\text{algoritmos de } \texttt{TM} \subseteq \text{algoritmos} \subseteq \text{algoritmos de } \texttt{TM}$$

La tesis de Church y Turing

La tesis de Church-Turing se puede representar en la siguiente manera:

$$\text{algoritmos de } \texttt{TM} \subseteq \text{algoritmos} \subseteq \text{algoritmos de } \texttt{TM}$$

Aparentemente, un algoritmo en el sentido intuitivo del termino es un objeto bastante diferente a un algoritmo de una `TM`.

La tesis de Church y Turing

La tesis de Church-Turing se puede representar en la siguiente manera:

$$\text{algoritmos de } \texttt{TM} \subseteq \text{algoritmos} \subseteq \text{algoritmos de } \texttt{TM}$$

Aparentemente, un algoritmo en el sentido intuitivo del termino es un objeto bastante diferente a un algoritmo de una `TM`. De hecho, la continencia se demuestra entre los lenguajes relativos:

$$\text{lenguajes Turing-decidibles} \subseteq \text{lenguajes calculables} \subseteq \text{lenguajes Turing-decidibles}$$

El décimo problema y los lenguajes

Dado un polinomio p a coeficientes enteros, podemos *codificarlo* en una secuencia de 0 y 1, que llamaremos $\langle p \rangle$.

El décimo problema y los lenguajes

Dado un polinomio p a coeficientes enteros, podemos *codificarlo* en una secuencia de 0 y 1, que llamaremos $\langle p \rangle$.

Según la tesis de Church-Turing, encontrar un algoritmo como nos pide el décimo problema de Hilbert significa preguntarse si el lenguaje

$$D = \{p, p \text{ polinomio entero con una raíz entera}\}.$$

es Turing-decidible o no.

Recordatorio

Un lenguaje $L \subseteq \{0,1\}^*$ es Turing-reconocible si existe una TM M tal que $w \in L$ si y solo si M, al procesar w, termina en q_{accept}.

Recordatorio

Un lenguaje $L \subseteq \{0,1\}^*$ es Turing-reconocible si existe una TM M tal que $w \in L$ si y solo si M, al procesar w, termina en q_{accept}.

L es Turing-decidible si existe una TM M tal que:

1. M se detiene en toda entrada;
2. $w \in L$ si y solo si M, al procesar w, termina en q_{accept}.

Un algoritmo para el décimo problema

Para encontrar si un polinomio tiene una raíz entera hay un algoritmo sencillo: lo evaluamos en todos enteros, revisando si es 0 o no.

Un algoritmo para el décimo problema

Para encontrar si un polinomio tiene una raíz entera hay un algoritmo sencillo: lo evaluamos en todos enteros, revisando si es 0 o no.

1: **procedure** evaluar_raices_enteras($\langle P \rangle$)
2: $i \leftarrow 0$
3: Encontrar n_i, el i-ésimo número entero
4: Evaluar $P(n_i)$
5: **if** $P(n_i == 0)$ **then**
6: Aceptar
7: **else**
8: $i \leftarrow i + 1$
9: Volver al paso 3

Un algoritmo para el décimo problema

Problema: si el polinomio no tiene raíces, la máquina de Turing *nunca se para*. Es decir, D no es decidible por la `TM` que realiza este algoritmo.

Un algoritmo para el décimo problema

Problema: si el polinomio no tiene raíces, la máquina de Turing *nunca se para*. Es decir, D no es decidible por la `TM` que realiza este algoritmo. De hecho, Yuri Matiyasevich, continuando el trabajo de Martin Davis, Hilary Putnam, y Julia Robinson, ha demostrado que no existe ninguna `TM` que decida D, lo que implica que no existe un algoritmo como el que pedía Hilbert.

Un algoritmo para el décimo problema

Problema: si el polinomio no tiene raíces, la máquina de Turing *nunca se para*. Es decir, D no es decidible por la `TM` que realiza este algoritmo. De hecho, Yuri Matiyasevich, continuando el trabajo de Martin Davis, Hilary Putnam, y Julia Robinson, ha demostrado que no existe ninguna `TM` que decida D, lo que implica que no existe un algoritmo como el que pedía Hilbert.

De otro lado, buscar raíces de un polinomio entero en una sola indeterminada *sí* es decidible.

Lenguajes decidibles

¿Cómo podemos describir una máquina de Turing?

Cuando estábamos hablando de autómatas, siempre los hemos descritos en todos los detalles: estados, transiciones y respectivo diagrama de estados. Este nivel de descripción es el más bajo y se llama *descripción formal*.

¿Cómo podemos describir una máquina de Turing?

Cuando estábamos hablando de autómatas, siempre los hemos descritos en todos los detalles: estados, transiciones y respectivo diagrama de estados. Este nivel de descripción es el más bajo y se llama *descripción formal*.

Cuando hemos empezado a hablar de máquinas de Turing hemos empezado a dejar unos detalles afuera de la descripción de una `TM`. Nos hemos contentado describiendo en español cómo una `TM` se movía mientras calculaba una palabra de un dado lenguaje. Este nivel intermedio se llama *descripción de implementación*.

¿Cómo podemos describir una máquina de Turing?

Cuando estábamos hablando de autómatas, siempre los hemos descritos en todos los detalles: estados, transiciones y respectivo diagrama de estados. Este nivel de descripción es el más bajo y se llama *descripción formal*.

Cuando hemos empezado a hablar de máquinas de Turing hemos empezado a dejar unos detalles afuera de la descripción de una `TM`. Nos hemos contentado describiendo en español cómo una `TM` se movía mientras calculaba una palabra de un dado lenguaje. Este nivel intermedio se llama *descripción de implementación*.

Finalmente, hoy para reconocer D hemos dado en español una descripción de un algoritmo, suficiente precisa para saber qué hacer, pero sin explicar exactamente cómo funciona una `TM` que lo realiza. Este nivel, se llama *descripción de alto nivel*.

Ejemplo 1: el lenguaje de un DFA

Sea

$$A_{\mathtt{DFA}} = \{\langle B, w\rangle : B \text{ es un } \mathtt{DFA} \text{ que acepta } w\} \subseteq \{0,1\}^*$$

Ejemplo 1: el lenguaje de un DFA

Sea

$$A_{\mathtt{DFA}} = \{\langle B, w\rangle : B \text{ es un } \mathtt{DFA} \text{ que acepta } w\} \subseteq \{0,1\}^*$$

Con $\langle B, w\rangle$ queremos decir que hemos codificado el $\mathtt{DFA}$ M y el input w en $\{0,1\}^*$, en manera parecida a cuando hemos construido una máquina de Turing universal.

Ejemplo 1: el lenguaje de un DFA

Sea

$$A_{\mathtt{DFA}} = \{\langle B, w\rangle : B \text{ es un } \mathtt{DFA} \text{ que acepta } w\} \subseteq \{0,1\}^*$$

Con $\langle B, w\rangle$ queremos decir que hemos codificado el DFA M y el input w en $\{0,1\}^*$, en manera parecida a cuando hemos construido una máquina de Turing universal.

Mostramos que $A_{\mathtt{DFA}}$ es Turing-decidible. Sea M una TM que ejecuta el siguiente algoritmo:

1: **procedure** evaluar_cadena_dfa($\langle B, w\rangle$)
2: Simular el comportamiento de B leyendo w
3: **if** B termina en un estado de aceptación **then**
4: Aceptar
5: **else**
6: Rechazar

Ejemplo 2: el lenguaje de un NFA

En manera similar podemos demostrar que

$$A_{\mathtt{NFA}} = \{\langle B, w\rangle : B \text{ es un } \mathtt{NFA} \text{ que acepta la cadena } w\}$$

es Turing-decidible.

Ejemplo 2: el lenguaje de un NFA

En manera similar podemos demostrar que

$$A_{\mathtt{NFA}} = \{\langle B, w\rangle : B \text{ es un } \mathtt{NFA} \text{ que acepta la cadena } w\}$$

es Turing-decidible.

Podemos explotar la conexión entre NFAs y DFAs para construir una TM que reconoce $A_{\mathtt{NFA}}$ y reducir este problema al problema, que ya hemos resuelto, de reconocer $A_{\mathtt{DFA}}$:

Ejemplo 2: el lenguaje de un NFA

En manera similar podemos demostrar que

$$A_{\mathtt{NFA}} = \{\langle B, w\rangle : B \text{ es un } \mathtt{NFA} \text{ que acepta la cadena } w\}$$

es Turing-decidible.

Podemos explotar la conexión entre NFAs y DFAs para construir una TM que reconoce $A_{\mathtt{NFA}}$ y reducir este problema al problema, que ya hemos resuelto, de reconocer $A_{\mathtt{DFA}}$:

```
1: procedure evaluar_cadena_nfa(⟨B, w⟩)
2:     Convertir B en un DFA C equivalente
3:     Correr evaluar_cadena_dfa(⟨C, w⟩)
4:     if se obtiene Aceptar then
5:         Aceptar
6:     else
7:         Rechazar
```

Ejemplo 3: reconocer si un lenguaje regular es vacío

Sea

$$E_{\texttt{DFA}} = \{\langle A \rangle : A \text{ es un } \texttt{DFA} \text{ tal que } L(A) = \emptyset\}$$

Ejemplo 3: reconocer si un lenguaje regular es vacío

Sea

$$E_{\texttt{DFA}} = \{\langle A \rangle : A \text{ es un } \texttt{DFA} \text{ tal que } L(A) = \emptyset\}$$

Mostramos que $E_{\texttt{DFA}}$ es Turing-decidible.

Ejemplo 3: reconocer si un lenguaje regular es vacío

Sea

$$E_{\texttt{DFA}} = \{\langle A \rangle : A \text{ es un } \texttt{DFA} \text{ tal que } L(A) = \emptyset\}$$

Mostramos que $E_{\texttt{DFA}}$ es Turing-decidible.

1: **procedure** evaluar_dfa_vacio($\langle A \rangle$)
2: Marcados $\leftarrow \{q_0\}$
3: **if** Marcados contiene algún estado de aceptación **then**
4: Rechazar
5: **else**
6: aux $\leftarrow$ Marcados $\cup \{q' : \delta(a, q) = q'$ para algún a y algún $p \in$ Marcados$\}$
7: **if** aux == Marcados **then**
8: Aceptar
9: **else**
10: Marcados $\leftarrow$ aux
11: Volver al paso 3

Ejemplo 4: reconocer si dos lenguajes regulares son iguales

Sea

$$EQ_{\texttt{DFA}} = \{\langle A, B\rangle : \ A \text{ y } B \text{ son } \texttt{DFA} \text{ tales que } L(A) = L(B)\}$$

Ejemplo 4: reconocer si dos lenguajes regulares son iguales

Sea

$$EQ_{\texttt{DFA}} = \{\langle A, B\rangle : \ A \text{ y } B \text{ son } \texttt{DFA} \text{ tales que } L(A) = L(B)\}$$

Para mostrar que este lenguaje es Turing-decidible, aprovechamos del ejemplo que acabamos de ver utilizando $X = Y \Longleftrightarrow X \triangle Y = \emptyset$.

Ejemplo 4: reconocer si dos lenguajes regulares son iguales

Sea

$$EQ_{\texttt{DFA}} = \{\langle A, B\rangle : \ A \text{ y } B \text{ son } \texttt{DFA} \text{ tales que } L(A) = L(B)\}$$

Para mostrar que este lenguaje es Turing-decidible, aprovechamos del ejemplo que acabamos de ver utilizando $X = Y \Longleftrightarrow X \triangle Y = \emptyset$.

1: **procedure** evaluar_equivalencia_dfa($\langle A, B\rangle$)
2: Construir C un DFA tal que $L(C) = L(A) \triangle L(B)$
3: Correr evaluar_dfa_vacio(C)
4: **if** se obtiene Aceptar **then**
5: Aceptar
6: **else**
7: Rechazar

Resumen

Resumen

Hoy aprendimos:

- Un poco de la historia alrededor de los veintitrés problemas de Hilbert y su relevancia para la Teoría de la Computación;
- Qué dice la tesis de Church-Turing;
- Unos ejemplos de lenguajes decidibles;
- Como reducir un problema de lenguaje decidible a otro.