Irresolubilidad

Departamento de Informática
Universidad de Oviedo

Contenidos

Parte I

¿Qué es la irresolubilidad?

Un poco de Historia

- En 1928, David Hilbert y Wilhelm Ackerman propusieron el Entscheidungsproblem ("problema de la decisión", en alemán) para la lógica de primer orden

Entscheidungsproblem

Encuéntrese un **algoritmo** para determinar si una **fórmula de lógica de primer orden dada** es válida o no.

- Para nosotros, un algoritmo como ese sería extremadamente útil

¿Qué pedían Hilbert y Ackerman?

- Tenemos un algoritmo (resolución general) que responde "sí" siempre que la fórmula dada es válida y, en algunas ocasiones, también responde "no" cuando la fórmula no es válida
- Hilbert y Ackerman querían un algoritmo que respondiera correctamente para **cualquier** fórmula, no solamente en casos particulares.
- El problema se podría resolver de dos maneras distintas:
  - Encontrando un algoritmo y demostrando que resuelve correctamente todos los casos
  - Demostrando que **dicho algoritmo no existe**
- Finalmente, se demostró que, de hecho, no existe un método algorítmico para resolver el problema (volveremos sobre ello) pero... ¿**cómo podríamos demostrar que un algoritmo no existe**?

DEMOSTRAR: NO EXISTE UN ALGORITMO PARA UN PROBLEMA

- Para demostrar que no existe un algoritmo para resolver un determinado problema, típicamente se procede de la siguiente manera:
  - Fijamos un modelo de computación (p. ej. Programas While).
  - Demostramos que el problema no se puede resolver empleando dicho modelo, utilizando técnicas como:
    - Demostración por contradicción (¿Recuerdas?)
    - Diagonalización
    - Reducción
    - ...
- Entonces, invocamos la **tesis de Church-Turing** (o simplemente consideramos que todos los modelos de computación conocidos son equivalentes)
- En esta situación, decimos que el problema es **irresoluble** o **indecidible** (algorítmicamente)

Parte II

Problemas de decisión

¿Qué es un problema de decisión?

- Acabamos de decir que para demostrar que un problema es irresoluble, trabajamos con un modelo de computación
- Pero los modelos de computación se refieren a **funciones matemáticas**, no a problemas
- Relacionamos ambos conceptos mediante lo que denominamos **problemas de decisión**

Problema de decisión

Un **problema de decisión** es una pregunta con respuesta sí/no que depende de algunos parámetros de entrada

Ejemplos de problemas de decisión

En Informática, Lógica y Matemáticas hay muchos ejemplos de problemas de decisión

- Decidir si un número natural dado es primo o no
- Decidir si una fórmula dada de lógica proposicional es una tautología o no
- Decidir si una fórmula dada de lógica de predicados es una tautología o no
- Decidir si un programa dado es correcto sintácticamente o no
- Decidir si una gramática libre de contexto dada es ambigua o no
- Decidir si un programa dado para (termina) o no
- Decidir si un programa dado es un virus o no

PROBLEMAS DE DECISIÓN Y FUNCIONES COMPUTABLES

- Podemos transformar problemas de decisión generales en funciones sobre enteros utilizando **codificación**
- Si codificamos ecuaciones, fórmulas, gramáticas, programas... mediante números naturales, entonces todos esos problemas se pueden ver como funciones sobre los números naturales que devuelven 0 (falso) o 1 (verdadero)
- La función asociada a un problema de decisión se denomina su **función característica**

COMPUTABILIDAD Y RESOLUBILIDAD

Demostrar que un problema de decisión es resoluble (resp. irresoluble) es equivalente a demostrar que su función característica es computable (resp. no computable)

Parte III

El problema de la parada

¿Qué es el problema de la parada?

- El problema de la parada es la piedra angular en el estudio de la irresolubilidad: es un problema irresoluble y puede utilizarse para demostrar que muchos otros problemas son irresolubles
- Se puede formular para cualquier modelo de computación, pero aquí nos centraremos en su versión para los Programas While

El problema de la parada para Programas While

Decidir si un Programa While dado devuelve un valor (para) con una entrada dada.

El problema de la parada es irresoluble

- Para demostrar que el problema de la parada es irresoluble adaptamos la demostración clásica de Turing (1936)
- Es un ejemplo de demostracción por contradicción
- También está profundamente ligada a la paradoja del mentiroso ("*Estoy mintiendo*") y a las demostraciones de Gödel de sus teoremas de incompletitud
- Puedes ver una versión animada de la demostración en el siguiente vídeo (en inglés): https://www.youtube.com/watch?v=92WHN-pAFCs
- O, si prefieres la poesía, puedes leer la demostración en verso (en inglés): http://www.lel.ed.ac.uk/ gpullum/loopsnoop.html

La demostración de irresolubilidad (I)

- Empezamos suponiendo que existe una macro $H(p, a)$ que decide el problema de la parada para Programas While
- Es decir:
  - Si el programa codificado por p para con la entrada a entonces $H(p, a)$ devuelve 1
  - Si no, $H(p, a)$ devuelve 0
- A partir de este supuesto, vamos a llegar a una contradicción

La demostración de irresolubilidad (II)

- Ahora, podemos construir el siguiente programa

```
begin
    X2 := H(X1,X1);
    if X2 = 0 then
       X1:= 0;
   else
       while X1 = X1 do
           X1:=succ(X1);
end
```

- Dado que *H* es una macro, este es un programa While válido y, por lo tanto, tendrá un código *c*

LA DEMOSTRACIÓN DE IRRESOLUBILIDAD (III)

- Ahora, piensa qué pasaría si la entrada del programa fuera su propio código c
- ¿Pararía o no?
- Podemos distinguir dos casos:
  - Si $H(c, c) = 0$, entonces el programa pararía y retornaría 0. Pero entonces $H(c, c)$ debería haber sido 1!!!!!
  - Si $H(c, c) = 1$, entonces el programa entraría en un bucle infinito. Pero entonces $H(c, c)$ debería haber sido 0!!!!!
- Hemos alcanzado una contradicción en los dos casos, por lo que es imposible que la macro H exista
- Concluimos que el problema de la parada es irresoluble

Parte IV

Reducción y el teorema de Rice

¿Qué es reducción?

- Ahora que sabemos que un problema concreto es irresoluble, muchos otros caerán también, como piezas de un dominó...
- Para ello, emplearemos el método de **reducción**

Reducción

Decimos que un problema de decisión P se reduce a otro problema de decisión P' si a partir de un algoritmo que resuelva P' podemos **construir** un algoritmo que resuelva P.

Uso del método de reducción

- Nótese que si podemos reducir P a P', significa que P es *más fácil* de resolver que P'. Esto se denota por $P \leq P'$.
- Nótese también que si P se puede reducir a P' pero sabemos que P es irresoluble, entonces P' (que es más difícil) debe ser necesariamente irresoluble también
- Esto es sencillo de demostrar:
    - Sabemos que P es irresoluble y suponemos que P se puede reducir a P'
    - Suponemos además que existe un algoritmo que resuelve P'
    - Pero, entonces, dado que $P \leq P'$, a partir del algoritmo que resuelve P' podemos **construir** un algoritmo para P
    - Esto es una contradicción, ya que sabemos P es irresoluble
- Por tanto, si podemos reducir el problema de la parada a otros problemas sabremos inmediatamente que dichos otros problemas también son irresolubles

El problema de la totalidad

- El problema de la totalidad es otro problema indecidible que está relacionado con el problema de la parada

El problema de la totalidad

Decidir si un programa While dado devuelve un valor (para) con **todas** las entradas.

- De hecho, el problema de la totalidad es el problema de decisión sobre si la función semántica asociada a un programa es total o no

El poblema de la totalidad es irresoluble (I)

- Utilizaremos el método de reducción para demostrar que el problema de la totalidad es irresoluble
- Entonces, suponemos que tenemos acceso a una macro $T(c)$ que devuelve 1 cuando el programa codificado por c para con todas las entradas y 0 en caso contrario (es decir, hay *alguna* entrada con la que no para)
- Ahora, utilizando T intentamos resolver el problema de la parada (para demostrar que problema de la parada $\leq$ problema de la totalidad)

El problema de la totalidad es irresoluble (II)

- Dado el código c de un programa y una entrada k, construimos el siguiente programa (donde U es la macro de universalidad)

```
begin
    X1:=U(c,k);
end
```

- Este programa tendrá un código d
- Nótese que el programa no depende de su entrada (tanto c como k son constantes), así que, o bien para con todas las entradas o con ninguna
- De hecho, parará con todas las entradas si y solo si el programa c para con la entrada k

El problema de la totalidad es irresoluble (III)

- Pero podemos comprobar si el programa para con todas las entradas calculando $T(d)$
- Por tanto, para decidir si el programa c para con la entrada k, podemos proceder del siguiente modo:
  - A partir de c y k calcular el código d (de hecho, una aplicación del teorema de Parametrización!)
  - Calcular $T(d)$
  - Si $T(d)$ es 1, entonces el programa c para con la entrada k; en otro caso, no para
- Hemos demostrado que el problema de la parada se reduce al problema de la totalidad
- En consecuencia, el problema de la totalidad también es indecidible

Propiedades semánticas (I)

- El método de reducción se puede aplicar para demostrar que muchos problemas de decisión sobre programas son irresolubles
- Para ver cómo se puede hacer para una clase muy extensa de problemas, necesitamos definir el concepto de propiedad semántica

Propiedades semáticas

Fíjese una aridad k. Una **propiedad semántica** (para la aridad k) de un programa es una propiedad que solo depende de la función semática (de aridad k) calculada por dicho programa.

- Para simplicar, en lo que sigue fijaremos la aridad a 1

Propiedades semánticas (II)

- Algunos ejemplos de propiedades semánticas:
  - El programa para con todas las entradas
  - El programa devuelve 0 para al menos una entrada
  - El programa siempre devuelve el mismo valor
  - ...
- Al contrario, las siguientes **no** son propiedades semánticas:
  - El programa tiene un número par de líneas
  - El programa utiliza la variable X5
  - El programa no tiene instrucciones *pred*
  - ...

Propiedades semánticas (III)

- Para nosotros, el hecho más importante sobre propiedades semánticas de programas es el siguiente:

Propiedades semánticas y programas

Si S es una propiedad semántica y P_1 y P_2 son programas que tienen la misma función semántica, entonces o bien S se cumple para tanto P_1 como P_2 o no se cumple para ninguno.

- También diremos que una propiedad semántica es **no trivial** si se cumple para al menos un programa y no se cumple para al menos otro

Demostración: Propiedad NO Semántica (I)

- Veamos cómo demostrar que una propiedad no es semántica

Ejercicio

Demuestra que la siguiente propiedad no es una propiedad semántica: El programa tiene un número par de líneas.

- Basándonos en lo que acabamos de ver, podemos afirmar que si hay un programa P_1 que cumple la propiedad y otro programa P_2 con la misma función semántica que P_1 que no la cumple, entonces no es una propiedad semántica.

DEMOSTRACIÓN: PROPIEDAD NO SEMÁNTICA (II)

- Este programa P_1 tiene un número par de líneas:

```
begin
    X2:=0;
    X1:=0;
end
```

- Este programa P_2 **no** tiene un número par de líneas:

```
begin
    X1:=0;
end
```

- Sin embargo ambos computan la misma función semántica $\varphi(x) = 0$
- Por tanto, “tener un número par de líneas” **no** es una propiedad semántica

Teorema de Rice

- El teorema de Rice permite demostrar, fácilmente, que un amplio número de preguntas sobre programas son indecidibles
- Con nuestras definiciones, se puede enunciar como sigue:

Teorema de Rice

Si S es una propiedad semántica no trivial, entonces el problema de decidir si S se cumple para un programa P dado es indecidible.

Demostración del teorema de Rice (I)

- Para demostrar el teorema de Rice, emplearemos el método de reducción. En concreto, vamos a reducir el problema de la parada al problema de decidir si una propiedad semántica S se cumple para un programa P dado.
- Dependiendo de la propiedad semántica S, podemos considerar dos casos: o bien S se cumple para un programa N que no para con ninguna entrada o bien S no se cumple para dicho programa
- Consideraremos que S no se cumple para N (la demostración para el otro caso es análoga).
- También cosideraremos un programa Q para el que S se cumple (tiene que haber uno, ya que S es no trivial).

Demostración del teorema de Rice (II)

- Utilizaremos la macro de universalidad U
- Y, por supuesto, supondremos que tenemos una macro R que resuelve el problema de decidir si la propiedad S se cumple para un programa dado. Es decir, si c es el código de un programa entonces $R(c) = 1$ si S se cumple para dicho programa y $R(c) = 0$ si no.
- Ahora, veremos que podríamos resolver el problema de la parada si tuviéramos acceso a R

Demostración del teorema de Rice (III)

- Dado un código c de un programa y una entrada k, construimos el siguiente programa (donde U es la macro de universalidad)

```
begin
    X2:=U(c,k);
    X2:=0;
    Q
end
```

- Este programa tendrá un código d

Demostración del Teorema de Rice (IV)

- Nótese que el programa anterior calcula o bien la función totalmente indefinida o la misma función semática que Q
- Lo que es más importante, la función que calcula (de entre las dos) **se puede determinar a partir de únicamente** c **y** k
- De hecho, si el programa c para con la entrada k, el programa calcula la misma función semántica que Q y tendrá la propiedad S
- En el otro caso, si c no para con k, el programa no parará y no tendrá la propiedad S

Demostración del teorema de Rice (V)

- Pero podemos comprobar si S se cumple para el programa calculando $R(d)$
- Por tanto, para decidir si el programa c para con la entrada k, podemos proceder del siguiente modo:
  - A partir de c y k calcular el código d (teorema de Parametrización, de nuevo!!!)
  - Calcular $R(d)$
  - Si $R(d)$ es 1, entonces el programa c para con la entrada k; si no, no para
- Hemos demostrado que el problema de la parada se reduce al problema de decidir si S se cumple para un programa dado
- En consecuencia, el problema de decidir si S se cumple para un programa dado es indecidible

Parte V

Problemas irresolubles en la vida real

Computabilidad y computadores reales

- En la Computabilidad trabajamos con modelos simples, versiones abstractas de computadores reales
- Sin embargo, sus propiedades esenciales son idénticas: máquinas que utilizan instrucciones (simples) sobre datos discretos, reciben entradas, hacen cálculos paso a paso y devuelven resultados
- Un computador real podría ser simulado, aunque muy lentamente, por una Máquina de Turing!!!

El problema de la parada y el teorema de Rice

- El estudio que hemos llevado a cabo sobre problemas irresolubles se puede adaptar fácilmente a computadores reales
- Tan solo hemos utilizado las ideas de codificación y de la existencia de una macro de universalidad
- Pero codificación y universalidad son elementos frecuentes en computadores reales (código binario, emuladores, máquinas virtuales, intérpretes...)
- Nuestra demostración de la irresolubilidad del problema de la parada puede aplicarse a la mayoría de los lenguajes modernos de programación con solo unos pocos cambios sintácticos
- Y, por tanto, empleando reducción también podemos demostrar algo similar al teorema de Rice
- Pero esta no es la única aplicación de estos métodos...

El antivirus perfecto

- Detectar *malware* y prevenir sus consecuencias es una tarea muy importante en la informática moderna
- ¿Pero, cuáles son las propiedades deseables de un antivirus *perfecto* ?
  - No debería decir que un programa que es un virus es inofensivo (no falsos negativos)
  - No debería decir que un programa que no es un virus es peligroso (no falsos positivos)
  - Debería parar siempre con cualquier entrada
  - **Debería no ser dañino en sí mismo**
- Desafortunadamente... un antivirus así es matemáticamente imposible!

El antivirus perfecto es imposible (1)

- Utilizaremos un enfoque parecido al que utilizamos para demostrar que el problema de la parada es indecidible (adaptado de la demostración original de Cohen, publicada en 1987)
- Entonces, suponemos que tenemos un procedimiento *AV* que implementa el antivirus perfecto (devuelve *Verdadero* si y solo si el programa a analizar es un virus)
- Consideramos el siguiente “programa”:

if AV(`ruta-al-programa`) then
    imprimir(“Tenga usted un buen día!”)
else
    borrar todos los archivos e insultar al usuario

El antivirus perfecto es imposible(2)

- Ahora, considera qué pasaría si sustituyéramos la ruta a este programa *dentro de sí mismo* (creamos un nuevo programa que comprueba si él mismo es un virus)
- Si el procedimiento *AV* determinase que este nuevo programa es un virus entonces se comportaría inofensivamente (nótese que es importante que *AV* sea inofensivo). Esto es una contradicción.
- Pero si *AV* decide que el programa NO es un virus entonces hará algo muy maligno. De nuevo una contradicción!
- Por tanto, concluimos que el antivirus perfecto es imposible matemáticamente (tenemos que conformarnos con heurísticos!)

Problemas irresolubles en análisis sintáctico

- Quizás recuerdes de Teoría de Autómatas que algunos problemas sobre Gramáticas Libres de Contexto (GLCs) son indedicibles. Por ejemplo:
  - Determinar si una GLC es ambigua
  - Determinar si dos GLCs generan exactamente el mismo lenguaje
  - Determiniar si una GLC genera todas las posibles cadenas sobre su alfabeto
  - Determinar si una GLC genera un lenguaje regular
  - ...
- Se puede demostrar que todos estos problemas son indecidibles mediante reducciones del conocido como Problema de Correspondencia de Post (véase el libro de Hopcroft, Ullman & Motwani)

Lógica y Computabilidad

- Quizás recuerdes (**deberías**) que determinar si una fórmula proposicional es una tautología es un problema decidible (se pueden utilizar tablas de verdad, p. ej.)
- Sin embargo, determinar si una fórmula de lógica de **predicados** es una tautología es ¡¡¡**indecidible**!!!
- Esto es, de nuevo, consecuencia de la indecibilidad del problema de la parada
- Dado el código p de un programa While y una entrada a, podemos escribir una fórmula de lógica de predicados $H_{p,a}$ que es válida si y solo si p para con entrada a.
- Hemos reducido el problema de la parada al problema de decidir la validez de fórmulas en lógica de predicados y, por tanto, ¡este último problema también es irresoluble!