1.

Serafín Moral
Cristina Zuheros Montes

Práctica 1:

1.

Muestre que, si pudiésemos resolver el problema de la parada, también podríamos resolver muchos problemas matemáticos no resueltos.
Ilustre cómo se podría demostrar la conjetura de Goldbach si tuviésemos un algoritmo que resuelva el problema de la parada.

Conjetura de Goldbach: Todo número par mayor que 2 puede descomponerse como la suma de dos números primos.

Supongamos que existe un algoritmo M que resuelve el problema de la parada, es decir, supongamos que existe un programa que nos pe rmite saber si cualquier programa finaliza o no (queda en ciclo infinito).
En este caso podríamos resolver muchos problemas matemáticos no resueltos como puede ser el último teorema de Fermat y la Conjetura de Goldbach. Veámoslo:

✦

Último teorema de Fermat:
Teorema:
Si n es un número entero mayor que 2, entonces no e
xisten números enteros positivos x, y y z,
tales que se cumpla la igualdad:
Demostración (suponiendo cierto el problema de la parada):
La idea es ir probando para cada cuádrupla (x,y,z,n), si se satisface la ecuación o no. Podemos diseñar un programa que se encargue de esto, es dec
ir, de probar todas las cuádruplas posibles de números, y que se detenga en el caso en el que la ecuación se satisfaga. Usando el
algoritmo M se podría saber si este programa acaba
o no. En el caso de que no pare, tendríamos probada la afirmación de Fermat.

✦

Conjetura de Goldbach:
Conjetura: Todo número par mayor que 2 puede descomponerse como la suma de dos números primos.
Demostración (suponiendo cierto el problema de la parada):
Podemos diseñar un programa que, para cada número par a partir del número 6 (el número 4 es trivialmente 2+2), trate de descomponerlo en la suma de dos números primos impares. Y que se detenga en el caso en el que uno de esos números pares no se pueda descomponer como la suma de dos números primos. Usando el algoritmo M se podría saber si este programa acaba o no. En el caso de que pare, tendríamos un contraejemplo de la Conjetura de Goldbach.

2.

Las máquinas de Turing podemos verlas como números enteros. Cada paso en una máquina de Turing podemos verlo como una operación sobre números enteros. Así, el hecho de si una máquina

de Turing para o no puede verse como un problema de aritmética. Si tuviese un algoritmo para saber si un teorema es verdadero entonces tendría un algoritmo para saber si un programa para o no, por lo que no puedo tener un algoritmo que me diga si un teorema es verdadero o es falso.

Si todo teorema cierto tiene demostración y su contrario no (al ser el conjunto de axiomas consistente) entonces, ordenando por orden alfabético las demostraciones, para cada una de ellas comprobamos si es de un teorema P, o de su opuesto (no)P. El algoritmo que recorre las demostraciones y comprueban si son de un teorema o de su opuesto acaba, por lo que tendríamos un algoritmo que nos permite saber si un teorema es cierto o no, lo que nos lleva a contradicción.

3.

Sea la sentencia X(P) = El programa P no termina para la entrada P.
y sea Q(P) el programa que busca en todas las demostraciones una de X(P) y termina si encuentra una.
Si hay consistencia X(Q) no tiene demostración, ya que en caso contrario, Q(Q) encontraría dicha demostración y terminaría.
Si pudiésemos demostrar la consistencia, podríamos demostrar que X(Q) no tiene demostración.
Ahora bien, que X(Q) no tenga demostración es equivalente a X(Q), de donde concluimos que X(Q) tiene demostración y que X(Q) no tiene demostración, lo que es contradictorio-

4.

a)

Supongamos que queremos ver si un programa P termina para unos datos. Entonces consideramos un segundo programa que ejecutará el primero para esos datos y si P termina no entrará en un bucle infinito y si P no termina si entrará en un bucle infinito. Por consiguiente, si pudiésemos saber si un programa entra en un bucle infinito o no podríamos resolver el problea de la parada, y ya sabemos que es indecidible.

b)

Supongamos que queremos ver si un programa P termina para unos datos. Consideramos un segundo programa que ejecutará el primero para esos datos y después imprimirá el mensaje "He terminado". El segundo programa escribirá dicho mensaje cuando y sólamente cuando P termine. Entonces si fuese decidible este problema también lo sería el de la parada, cosa que sabemos que no sucede

c)

Supongamos que queremos ver si un programa P termina para unos datos. Consideramos otro programa Q que ejecutará el primero para esos datos y después ejecutará la sentencia X. Que este programa ejecute la sentencia X es equivalente a que P termine. Entonces si pudiésemos saber si el programa Q llega a ejecutar la sentencia X tendríamos una solución a si el programa P para o no, con lo que llegamos a una contradicción, pues el problema de la parada es indecidible.

d)

Supongamos que queremos ver si un programa P termina para unos datos. Consideramos un segundo programa que inicializará todas las variables, ejecutará el programa P, y después usará una

variable v que no ha sido inicializada. Por tanto, el segundo programa utilizará la variable v sin inicalizar si y sólo si, P termina, por lo que si pudiésemos determinar si un programa usará una variabe sin unicializar pudiésemos resolver el problema de la parada, lo que es contradictorio.

5.

Consideramos un programa P que tenga como entrada unos datos. Otro programa que ejecute el programa P y después imprima “Sí” y un tercer programa que imprima siempre “Sí” (y sólo haga eso). Si pudiésemos determinar si el segundo y el tercer programa hacen lo mismo podríamos saber si el programa P para para esos datos, cosa que no puede ocurrir al ser el problema de la parada indecidible.

6.

Como el troyano sólo puede capturar pasando desapercibido el login y el password de otro osuario sólo cuando al otro usuario le pide el login y el password, como por el ejercicio anterior no se puede determinar si dos programas hacen lo mismo, no es posible que un troyano pase desapercibido.

7.

Supongamos que existe un algoritmo T capaz de determinar si un problema es decidible o no. El problema de la parada para un caso particular no sabemos si es decidible o no. Hay que pasar este programa al programa T que hemos supuesto, el cuál nos tendría que decir si es decidible o no, llegando a una contradicción. Luego no puede existir dicho algoritmo T.

8.

Si la cantidad de memoria que permitimos que utilice un programa está limitada, entonces si el programa cicla indefinidamente las configuraciones por las que pasa un programa se tienen que repetir. Podemos comprobar si un programa cicla, comprobando para cada configuración si ya había salido anteriormente. En caso afirmativo se tiene que el programa cicla indefinidamente. Como la cantidad de memoria está limitada, el número de configuraciones que se pueden dar es finito. Por lo tanto, el programa para si, y solamente si, no se repiten configuraciones.