**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 permite 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 existen números enteros positivos x, y y z, tales que se cumpla la igualdad:

  x^n + y^n = z^n  \,

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 decir, 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.

1. **A partir de los resultados de Turing sobre el problema de la parada, demuestre el Teorema de Incompletitud de Gödel: Dado cualquier conjunto consistente y computable de axiomas, existe una sentencia verdadera sobre números enteros que no puede demostrarse a partir de dichos axiomas.**

**Consistente hace referencia a que no se puede derivar ninguna contradicción, mientras que computable quiere decir que existe un número finito de axiomas o bien, aunque haya un número infinito, existe un algoritmo que nos permite generarlos (si no incluyésemos este requisito, podríamos partir de un conjunto de axiomas formado por todas las sentencias verdaderas sobre números enteros; lo cual, de todas formas no sería demasiado útil en la práctica).**

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.

1. **A partir de los resultados de Turing sobre el problema de la parada, demuestre el Segundo Teorema de Incompletitud de Gödel:**

**Si una teoría F es consistente, entonces F no puede demostrar su propia consistencia. En otras palabras, las únicas teorías matemáticas que pueden demostrar su consistencia son precisamente aquéllas que no son consistentes.**

Por el ejercicio 2 sabemos que a partir de los resultados de Turing sobre el problema de la parada, podemos demostrar el primer teorema de Incompletitud de Gödel. Además, este primer teorema, nos permite demostrar el segundo teorema de Incompletitud de Gödel. Luego, apartir de los resultados de Turing sobre el problema de la parada, se puede demostrar el segundo teorema de Gödel. Veamos que efectivamente a partir del primer teorema se puede demostrar el segundo:

Sea una sentencia P indecidible. Asumamos que la consistencia del sistema se puede probar dentro del propio sistema. Si el sistema es consistente, entonces p no es demostrable. La prueba de esta implicación se formaliza en el propio sistema, luego la afirmación “P no es demostrable” o “no P(p)” se puede demostrar en el sistema. Esta última declaración equivale a P, de modo que P se puede declarar en el sistema. Llegamos a contradicción, luego el sistema tiene que ser inconsistente.

1. **Demuestre, por reducción al problema de la parada, que no es posible desarrollar un depurador automático que sea capaz de, en general:**

**Determinar si un programa incluye algún bucle infinito.**

**Determinar si un programa mostrará por pantalla un mensaje determinado. Determinar si un programa llegará a ejecutar una sentencia determinada. Determinar si una variable se inicializará siempre antes de ser utilizada.**

**Esto no quiere decir que no se puedan construir sistemas en los que sea posible realizar demostraciones sobre el comportamiento de un programa, s´olo que no existen soluciones universales a tales problemas. Por ejemplo, la m´aquina virtual de Java est´a dise˜nada para poder permitir determinados tipos de demostraciones y un compilador de Java es capaz de detectar muchos de estos errores en tiempo de compilaci´on. Busque informaci´on en Internet sobre el tema.**

1. 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 problema de la parada, y ya sabemos que es indecidible.
2. 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
3. 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.
4. 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. **Muestre que no es posible determinar, en general, si dos programas hacen lo mismo. Esto es, dados dos programas P y Q, demuestre que no es decidible el problema de determinar si ambos calculan la misma función: P(x) = Q(x) para todo x. Observe que esto implica que, en general, no se puede determinar si un programa original P y su versión optimizada P’ producida por un compilador calculan exactamente lo mismo.**

Un problema es indecidible cuando no existe un algoritmo que tome como entrada una instancia de ese problema y determine si la respuesta es sí o no.

Supongamos que tenemos un programa que siempre nos va a decir sí y un programa que nos dice sí en el caso de que realmente pare. Si el problema de saber si dos programas hacen lo mismo fuese decidible, entonces podríamos ver si estos dos programas dados son iguales o no. En ese caso tendríamos resuelto el problema de la parada, luego llegamos a contradicción.

1. **Demuestre que no existe ningún algoritmo que sea capaz de determinar si un problema matemático es decidible o no (esto es, un programa que sea capaz de indicarnos si una sentencia es decidible, sin llegar determinar si es cierta o falsa). En la práctica, esto quiere decir que, incluso aunque dispusiésemos de recursos ilimitados (CPU y memoria), siempre habrá cosas que no podremos demostrar formalmente con la ayuda de un sistema de demostración de teoremas. Pese a ello, los sistemas de verificación formal de programas siguen siendo útiles para demostrar las propiedades formales de determinados programas, siempre que impongamos las restricciones necesarias que nos permitan realizar tales demostraciones de forma eficiente (en ámbitos bien delimitados, no de forma universal).**

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.

1. **Muestre cómo se podría resolver el problema de la parada cuando imponemos una restricción sobre la cantidad de memoria que puede utilizar un programa. Obviamente, necesitará un sistema que disponga de muchísima más memoria de la que permitimos que utilice el programa...**

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 van repitiendo. Por cada configuración del programa que se de, comprobamos si ya se ha dado 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.