UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE CIENCIAS

Complejidad Computacional 2024-1

TAREA 3

PROFESOR:

Oscar Hernández Constantino

AYUDANTE:

Malinali Gónzalez Lara

INTEGRANTES:

Pedro Mendez Jose Manuel - 315073120

9 de octubre de 2023

1. Ejercicio 1

Investiga un modelo de cómputo que sea equivalente a la máquina de Turing. Describe y ejemplifica un cómputo (ejecución de un algoritmo) en dicho modelo.

LAMBDA

Un modelo de cómputo que podría ser(es) equivalente a la Máquina de Turing es el cálculo lambda (también conocido como cálculo-λ). El cálculo lambda es un sistema formal en el campo de la teoría de la computación y la lógica matemática que se utiliza para representar y realizar cálculos con funciones(como en el paradigma de programación funcional).

Es un formalismo matemático desarrollado en los años 1930s creado por Alonzo Church, lo podemos ver como un lenguaje de programación minimalista, donde no tenemos números, caracteres o Booleanos, u otros tipos de datos que no sean funciones, en realidad, si podríamos tener algo parecido pero en realidad son funciones, es el más pequeño lenguaje universal de programación, consiste en en una regla de transformación simple (sustituir variables) y un esquema simple para definir funciones. Sin embargo pese a esto, puede representar cualquier Máquina de Turing. El cálculo lambda es universal porque porque cualquier función computable puede ser expresada y evaluada a través de él. Se puede decir que es equivalente a las máquinas Turing porque es capaz de evaluar y expresar cualquier función computable. Church había querido hacer un sistema formal completo para modelizar la matemática pero después separó el cálculo lambda y lo ideó para que estudiara la computabilidad.

3 puntos importantes(bases) del cálculo Lambda serían:

- Expresiones Lambda (representan Funciones): Las expresiones lambda se podría decir que son la base del cálculo lambda, estas se utilizan para representar funciones anónimas. Tienen la forma $\lambda x.M$, donde x es una variable y M es una expresión que puede contener variables y otras expresiones lambda.
- En el cálculo lambda las evaluaciones surgen al aplicar una función a un argumento utilizando la notación funcional. Por ejemplo:

  Sea $\lambda x.x + 1$ una expresión lambday y la evaluamos con 2, tendríamos: $(\lambda x.x + 1)2$, lo que daría como resultado 3, ya que x toma el valor de 2: $(2 + 1)$

- La reducción lambda es el proceso de simplificar las expresiones lambda aplicando funciones a argumentos. Puede considerarse como el equivalente de la ejecución de un programa en una Máquina de Turing.

Formalmente://
En el cálculo lambda, una expresión o término se define recursivamente a través de las siguientes reglas de formación:

- Toda variable es un término: x, y, z, u, v, w, $x1$, $x2$, $y9$, ...
- Si t es un término y x es una variable, entonces $(\lambda x.t)$ es un término (llamado una abstracción lambda).
- Si t y s son términos, entonces (ts) es un término (llamado una aplicación lambda).
- Nada más es un término.

Ejemplo de Cómputo en Cálculo Lambda:

Definición de la función AND en cálculo lambda: AND = $\lambda x.\lambda y.x$ if x = True, else False, AND es una función que toma dos argumentos x e y y devuelve x si x es True (verdadero) y False (falso) de lo contrario.

Evaluación de AND con valores booleanos:

- AND True True se evaluaría: AND $= \lambda x.\lambda y.x$ if $x =$ True, else False, aplicaremos la función AND a los argumentos True y True. Primero, se aplica a True como x, y la expresión se convierte en λy.True if True else False, déspues se evalua y como True teniendo así que la expresión se convierte en True if True else False, teniendo así que la expresión se evalúa como True.
- AND True False se evaluaría: Aplicamos la función AND a los argumentos True y False. en el cálculo lambda primero evaluamos a x como True teniendo así que la expresión se convierte en λy.True if True else False, despues evaluamos a y como False teniendo así que la expresión se convierte en True if False else False teniendo así que la expresión se evalúa como False.

2. Ejercicio 2

- Da una descripción general de un máquina de Turing que decide el siguiente lenguaje sobre el alfabeto $\Sigma = \{a, b\}$ y determina el tiempo de ejecución de dicha máquina.

$$L(a) = \{w \in \Sigma^* | w \textbf{ contiene el doble de a's que b's}\}$$

Descripción general de una máquina de Turing que decide $L(a)$.

1. Sea w la cadena de entrada para nuestra máquina digamos M
2. Si w es la cadena vacía ϵ, aceptamos, porque está cumple con que se tiene el doble de a's y b's.
3. Recorremos(desde el inicio) w hasta encontrar una b(la primera) sin marcar. Teniendo así dos casos:
   - 3.1 Si encontramos una b, la reemplazamos con una Y y regresamos al inicio de la cadena. (Y pasamos al paso que sigue)
   - 3.2 Si recorremos toda w, y no encontramos una b, nos vamos al paso final, el que se encargara de aceptar o rechazar w.
4. Desde el inicio de w buscaremos una a(la primera):
   - 3.1 Si encontramos una a, la reemplazamos con una X y regresamos al inicio de la cadena y pasando al siguiente paso que se encargara de buscar la segunda a.
   - 3.2 Si recorremos toda w, y no encontramos una a, rechazamos w.
5. Aquí volveremos a buscar la siguiente a en w igual que el paso anterior.
   - 3.1 Si encontramos una a, la reemplazamos con una X y regresamos al inicio de la cadena y volvemos a buscar una b, es decir, volvemos al paso 3(REPITIENDO HASTA TERMINAR LAS b'S DE w).
   - 3.2 Si recorremos toda w, y no encontramos una a, rechazamos w.
6. Como previamente mencionamos, este será el paso final, el que se encarga de aceptar o rechazar: recorremos toda la w, en teoría aquí ya no deberían de haber a's y b's o cualquier otro símbolo, si encontramos un símbolo que no sea X y Y, entonces rechazaremos.

   Pero si recorremos toda la cadena y nos encontramos con # ACEPTAMOS w.

Tiempo de ejecución.

La complejidad de esta maquina es de $0(n^2)$ donde n, es la longitud de la cadena w.

El proceso de reemplazo y busqueda de b's con Y y a's con X se realiza en un solo pase de la cadena, en el peor de los casos, sería recorrer toda w, por lo que tiene una complejidad de tiempo lineal en función de la longitud de la cadena w.

Y cómo lo anterior se podría decir que lo hacemos por cada elemento de w, entonces lo debemos multiplicar por n.

La verificación final también se realiza en un solo pase de la cadena, lo que también tiene una complejidad de tiempo lineal en función de la longitud de la cadena w, pero esto no influye tanto la complejidad, teniendo así que que el tiempo de ejecución esta dado por $0(n^2)$.

- Definición formal de la máquina anterior.

  Formalmente digamos que M esta definida $M = \{Q, \Sigma, \Gamma, \delta, q_0, q_a, q_r\}$ donde:

$$
\begin{aligned}
&Q \text{ (conjunto de estados)} : \{q_0, q_1, q_1', q_2, q_2', q_3, q_4, q_\text{a}, q_\text{r}\}\\
&\Sigma \text{ (alfabeto de entrada)} : \{a, b\}\\
&\Gamma \text{ (alfabeto de cinta)} : \{a, b, X, Y, \sqcup, \#\}\\
&\delta \text{ (función de transición)} : \text{ Dada en la tabla siguiente}
\end{aligned}
$$

Estados	$\sqcup$	a	b	X	Y	$\#$
q_0	$(q_1, \sqcup, R)$					
q_1		(q_1', a, R)	(q_2, Y, L)	(q_1, X, R)	(q_1, Y, R)	$(q_a, -, -)$
q_1'		(q_1', a, R)	(q_2, Y, L)	(q_1', X, R)	(q_1', Y, R)	$(q_r, -, -)$
q_2	$(q_3, \sqcup, R)$	(q_2, a, L)	(q_2, b, L)	(q_2, X, L)	(q_2, Y, L)	
q_2'	$(q_1, \sqcup, R)$	(q_2', a, L)	(q_2', b, L)	(q_2', X, L)	(q_2', Y, L)	
q_3		(q_4, X, R)	(q_3, b, R)	(q_3, X, R)	(q_3, Y, R)	$(q_r, -, -)$
q_4		(q_2', X, L)	(q_4, b, R)	(q_4, X, R)	(q_4, Y, R)	$(q_r, -, -)$

- Ejemplifica la ejecución de la máquina propuesta mostrando las configuraciones que se producen para las siguientes entradas: Para reducir la extensión, ocupamos xn: haciendo referencia a que usamos n movimientos análogos, además sabemos que w esta dada por: $w = \sqcup w\#$, donde $\sqcup$ es el carácter que denota el inicio de la cadena y $\#$ es el carácter que denota el fin de la cadena.

  - aaabbb:$= (q_0, \sqcup, aaabbb\#) = q_0 \sqcup aaaabb\# \implies (q_1, \sqcup, aaabbb\#) = \sqcup q_1 aaabb\# \implies (q_1', \sqcup a, aabbb\#) = \sqcup a q_1 abb\# \implies (q_1', \sqcup aa, abbb\#) = \sqcup aa q_1 bb\# \implies (q_1', \sqcup aaa, bbb\#) \implies (q_2, \sqcup aa, aYbb\#) \implies (q_2, \sqcup a, aaYbb\#) \implies (q_2, \sqcup, aaaYbb\#) \implies (q_3, \sqcup, aaaYbb\#) \implies (q_4, \sqcup X, aaYbb\#) \implies (q_2', \sqcup X, XaYbb\#) \implies (q_2', \sqcup, XXaYbb\#) \implies (q_1, \sqcup X, XaYbb\#) \implies (q_1, \sqcup XX, aYbb\#) \implies (q_1', \sqcup XXa, Ybb\#) \implies (q_1', \sqcup XXaY, bb\#) \implies (q_2, \sqcup XXa, YYb\#) \implies (q_2, \sqcup XX, aYYb\#) \implies (q_2, \sqcup X, XaYYb\#) \implies (q_2, \sqcup, XXaYYb\#) \implies (q_3, \sqcup X, XaYYb\#) \implies (q_3, \sqcup XX, aYYb\#) \implies (q_4, \sqcup XXX, YYb\#) \implies x2 : (q_4, \sqcup XXXYY, b\#) \implies (q_4, \sqcup XXXYYb, \#) \implies (q_r, \sqcup XXXYYb, \#)$
    $\therefore$ reject
  - abababa:$= (q_0, , \sqcup abababa\#) = q_0 \sqcup abababa\# \implies (q_1, \sqcup, abababa\#) = \sqcup q_1 abababa\# \implies (q_1', \sqcup a, bababa\#) \implies (q_2, \epsilon, \sqcup aYababa\#) \implies (q_3, \sqcup, aYababa\#) \implies (q_4, \sqcup X, Yababa\#) \implies (q_4, \sqcup XY, ababa\#) \implies (q_2', \sqcup X, YXbaba\#) \implies (q_2', \sqcup, XYXbaba\#) \implies (q_2', , \sqcup XYXbaba\#) \implies (q_1, \sqcup, XYXbaba\#) \implies x3 : (q_1, \sqcup XYX, baba\#) \implies (q_2, \sqcup XY, XYaba\#) \implies (q_2, \sqcup XYX, Yaba\#) = (q_2, \sqcup XY, XYaba\#) \implies x3 : (q_2, , \sqcup XYXYaba\#) \implies (q_3, \sqcup, XYXYaba\#) \implies X4 : (q_3, \sqcup XYXY, aba\#) \implies (q_4, \sqcup XYXYX, ba\#) \implies (q_4, \sqcup XYXYXb, a\#) \implies (q_2', \sqcup XYXY, XbX\#) = (q_2', \sqcup XYX, YXbX\#) \implies x3 : (q_2', \sqcup, XYXYXbX\#) \implies (q_1, \sqcup, XYXYXbX\#) \implies$
    $x5 : (q_1, \sqcup XYXYX, bX\#)$*(corremos la cinta hasta la primera b)* $\implies (q_2, \sqcup XYXY, XYX\#) \implies$
    $X5 : (q_2, , \sqcup XYXYXYX\#)$*(corremos al inicio de la cadena)* $\implies (q_3, \sqcup, XYXYXYX\#) \implies$
    $(q_3, \sqcup X, YXYXYX\#) \implies x6 : (q_3, \sqcup XYXYXYX, \#) \implies (q_r, \sqcup XYXYXYX, \#)$ $\therefore$ reject
  - aaaabb:$> q_0 \sqcup aaaabb \implies aq_1 aaabb \implies aaq_1' aabb \implies x2 : aaaaq_1' bb \implies aaaq_2 aYb \implies aaq_2 aaYb \implies aq_2 aaaYb \implies \sqcup q_2 aaaaYb \implies \sqcup q_3 aaaaYb \implies \sqcup X q_4 aaaYb \implies \sqcup q_2' XXaaYb \implies \sqcup q_1 XXaaYb \implies \sqcup X q_1 XaaYb \implies \sqcup XX q_1 aaYb \implies \sqcup XXaq_1' aYb \implies \sqcup XXaaq_1' Yb \implies \sqcup XXaaYq_1' b \implies \sqcup XXaaq_2 YY \implies \sqcup XXaq_2 aYY \implies \sqcup XXq_2 aaYY \implies x2 : \sqcup q_2 XXaaYY \implies \sqcup q_3 XXaaYY \implies \sqcup X q_3 XaaYY \implies \sqcup XX q_3 aaYY \implies \sqcup XXX q_4 aYY \implies$

$\sqcup XXq_2'XXYY \implies \sqcup Xq_2'XXXaYY \implies \sqcup q_2'XXXXYY \implies \sqcup Xq_1XXXYY \implies x3 :$
$\sqcup XXXXq_1YY \implies \sqcup XXXXYq_1Y\# \implies \sqcup XXXXYYq_1\# \implies \sqcup XXXXYYq_a\#.$
$\therefore$ accept.

- abb:-> $q_0 \sqcup abb \implies \sqcup q_1abb \implies \sqcup aq_1'bb \implies \sqcup q_2aYb \implies q_2 \sqcup aYb \implies \sqcup q_3aYb \implies \sqcup Xq_4Yb \implies \sqcup XYq_4b\# \implies \sqcup XYbq_4\# \implies \sqcup XYbq_r\# \therefore$ reject.

3. Ejercicio 3

Prueba que la colección de lenguajes decidibles es cerrada bajo las siguientes operaciones:

- Intersección: El conjunto de lenguajes decidibles es cerrado bajo la intersección, si para dos lenguaje decidibles, digamos L_1, L_2 su intersección $L_! \cup L_2$ támbien es decidible.

  Supongamos que tenemos dos lenguajes decidibles L_1 y L_2 con máquinas de Turing correspondientes M_1 y M_2 que los deciden. PD: $L_1 \cap L_2$ es decidible.

  Para hacerlo, podemos construir una nueva máquina de Turing, digamos M', que se encargara de simular simultáneamente las ejecuciones de M_1 y M_2 con la misma cadena-entrada w, tenido así:

  - Si ambos M_1 y M_2 aceptan w, entonces w pertenece a la intersección $L_1 \cap L_2$, por lo que M' aceptara a w.
  - Si M_1 o M_2 no aceptan w, entonces w no pertenece a la intersección $L_1 \cap L_2$, por lo que M' debe rechazar a w.

  Y de esta manera, tenemos que M' acepta cadenas que no están en L, por lo que M' acepta cadenas que se encuentran en $L_1 \cap L_2$, por lo que $L_1 \cap L_2$ es decidible.

  $\therefore$ Los lenguajes decidibles son cerrados bajo la intersección.

- Concatenación: El conjunto de lenguajes decidibles es cerrado bajo la concatenación, si para dos lenguaje decidibles, digamos L_1, L_2 su concatenación: $L_! \cdot L_2$ támbien es decidible.

  Supongamos que tenemos dos lenguajes decidibles, digamos L_1 y L_2 con sus máquinas de Turing correspondientes M_1 y M_2 que los deciden(por definición). PD: $L_1 \cdot L_2$ es decidible.

  Podemos construir una nueva máquina de Turing, digamos M que divide la entrada w en todas las posibles formas de partición y verifica si ambas partes de la partición pertenecen a L_1 y L_2 respectivamente.

  - Si M encuentra una partición que cumple esta condición, entonces w pertenece a $L_1 \cdot L_2$, por lo que M debe aceptar w.
  - Si ninguna partición satisface esto, entonces w no pertenece a la concatenación y M debe rechazar w.

  De esta manera, M decide $L_1 \cdot L_2$, lo que demuestra que la concatenación de lenguajes decidibles es decidible.

  $\therefore$ Los lenguajes decidibles son cerrados bajo la concatenación .

- Complemento: El conjunto de lenguajes decidibles es cerrado bajo complemento si para cada lenguaje decidible su complemento, digamos $l^{\complement}$ támbien es decidible.

  Sea L un lenguaje decidible, por definición entonces existe una MT, digamos M que lo decide, es decir, M acepta a w, si $w \in L$ y M rechaza a w, si $w \notin L$.

  Sea M', recibe como entrada a M codificada en binario y una cadena, digamos w, teniendo así la entrada de M', digamos $\alpha :=< M > \# < w >$

  Primero simularemos la ejecución de M con w, primero, si el primer elemento de w puede ser reconocido por el estado inicial de M, sí si: tendremos 2 casos:

- Si M acepta a w, ent $w \in L \implies w \notin L^{\complement}$ por lo que M' rechazara a w.
- Si M rechaza a w, ent $w \notin L \implies w \in L^{\complement}$ por lo que M' aceptará a w.

Teniendo así que M acepta cadenas que no están en L, por lo que M' acepta $L^{\complement}$, por lo que $L^{\complement}$ es decidible.

$\therefore$ Los lenguajes decidibles son cerrados bajo el complemento.

¿Para cuáles de las operaciones anteriores se mantiene la cerradura bajo la clase P?

La cerradura bajo la clase P se mantiene para las operaciones de intersección y complemento, pero no se mantiene para la operación de concatenación.

- No se mantiene para la operación de concatenación.

  Como ya hemos visto, la concatenación de dos lenguajes decidibles sigue siendo decidible, y la cerradura bajo la clase P se mantiene para la operación de concatenación. Ya que podemos construir una Máquina de Turing que concatene dos lenguajes decidibles en tiempo polinomial.

- Hemos visto que si tienes dos lenguajes decidibles L_1 y L_2, la concatenación es cerrada. Como todos los lenguajes que viven en P son decisibles, si tenemos dos lenguajes decidibles L_1 y L_2 que pertenecen a la clase P, la intersección $L_1 \cap L_2$ también será decidible y estará en P, porque podemos construir una Máquina de Turing que decida la intersección de manera eficiente utilizando las Máquinas de Turing que deciden L_1 y L_2 como subrutinas y verificando si una cadena pertenece a ambos lenguajes, en tiempo polinomial.

- El complemento bajo la cerradura de la clase P se mantiene, si tenemos un lenguaje decidible L que pertenece a la clase P, su complemento $L^{\complement}$ también será decidible y estará en P ya que también podemos construir un algoritmo en tiempo polinomial que acepte todas las cadenas que no están en L. Si L es decidible en tiempo polinomial por una Máquina de Turing, entonces su complemento L^c también es decidible en tiempo polinomial, como L es decidible en tiempo polinomial, entonces podemos utilizar la misma Máquina de Turing que decide L pero invertir sus respuestas (aceptar en lugar de rechazar y viceversa) para construir una Máquina de Turing que decida $L^{\complement}$ en tiempo polinomial.

4. Ejercicio 4

El décimo problema de Hilbert.

- Descripción del problema.

Dada una ecuación Diofántica (i.e ecuación algebraica o polinómica de varias variables con coeficientes en Z y soluciones en Z). ¿Existe un método que permita determinar, en un número finito de pasos, si una ecuación diofántica es resoluble en Z?

Este fue el planteamiento del decimo problema presentado por Hilbert, en el congreso internacional de Matemáticas en París, año 1900. El problema consiste en hallar un algoritmo de la forma:

$$(input) : D(x_1,\ldots,x_m) = 0$$

$$\Downarrow \qquad \dashrightarrow finitos \quad pasos$$

$$(output) : \begin{cases} Si \dashrightarrow D(x_1,\ldots,x_m) = 0 \quad tiene \quad solucion \quad en \quad los \quad enteros \\ No \dashrightarrow D(x_1,\ldots,x_m) = 0 \quad No \quad tiene \quad solucion \quad en \quad los \quad enteros \end{cases}$$

Donde $D(x_1, \ldots, x_m) = 0$ es una ecuación Diofántica de variables $x1, \ldots, x_m \in Z$.

Recordemos que una ecuación diofántica es una ecuación algebraica con coeficientes enteros y de la que se buscan soluciones enteras. Algunas de estas ecuaciones tienen solución y otras no. El décimo problema de Hilbert preguntaba si había algún procedimiento universal para, dada una ecuación diofántica cualquiera, responder simplemente con un sí o un no a la existencia de soluciones. **En pocas palabras, se trata de buscar un procedimiento efectivo para determinar si una ecuación diofántica tiene soluciones enteras o si no las tiene.**

Este problema está relacionado con la teoría de números y las ecuaciones diofánticas.

- Ejemplar(ex extras :D) Concreto:
  - Ejemplo con solución: Si $2x + 3y = 7$ es una ecuación diofántica. Así tendríamos: ¿La ecuación tiene una solución dónde x, y son números enteros? Aquí estamos buscando soluciones enteras para x e y. Puede resolverse de manera relativamente simple: si tomamos $x = 2$ y $y = 1$, obtenemos $2(2) + 3(1) = 4 + 3 = 7$, lo que satisface la ecuación. Por lo tanto, este ejemplo tiene solución con $x = 2$ e $y = 1$.
  - Ejemplo con solución: Supongamos que tenemos la ecuación diofántica $3x+5y = 16$. Así tendríamos: ¿La ecuación tiene una solución dónde x, y son números enteros? Estamos buscando soluciones enteras para x e y. Podemos encontrar una solución entera para esta ecuación al tomar $x = 2$ y $y = 2$. Si sustituimos estos valores en la ecuación, obtenemos: $3(2)+5(2) = 6+10 = 163(2)+5(2) = 6 + 10 = 16$ teniendo asi que la ecuación se satisface con $x = 2$ e $y = 2$, lo que significa que este ejemplo tiene una solución con números enteros.
  - Ejemplo sin solución: Ahora, consideremos la ecuación diofántica $2x+3y = 6$. Así tendríamos: ¿La ecuación tiene una solución dónde x, y son números enteros? Aquí, nuevamente estamos buscando soluciones enteras para x e y, pero en este caso, la ecuación no tiene solución. Puede demostrarse que no existen números enteros x e y que satisfagan esta ecuación. Esto se debe a que si x e y fueran enteros, $2x$ sería par y $3y$ sería múltiplo de 3, y la suma de un número par y un múltiplo de 3 nunca será igual a 6. Por lo tanto, este ejemplo no tiene solución con números enteros.
  - Ejemplo sin solución: Ahora, consideremos la ecuación diofántica $x^2+y^2+2 = 0$. Así tendríamos: ¿La ecuación tiene una solución dónde x, y son números enteros? Aquí, nuevamente estamos buscando soluciones enteras para x e y, pero en este caso, la ecuación no tiene solución.

    Para demostrar que esta ecuación no tiene solución con números enteros, observemos que el término x^2 y el término y^2 son siempre no negativos, lo que significa que su suma nunca puede ser igual a -2, que es un valor negativo. En otras palabras, no importa qué valores enteros seleccionemos para x e y, la suma de sus cuadrados siempre será no negativa o cero. Por lo tanto, no es posible encontrar valores enteros de x e y que hagan que $x^2 + y^2 + 2 = 0$ se cumpla, ya que la suma nunca será igual a -2.

- ¿Cuándo y por quién fue resuelto? Se podría decir que el décimo problema de Hilbert fue resuelto en un sentido negativo en 1970 por el matemático soviético Yuri Matiyasevich ya que este demostró que no existe un algoritmo general que pueda determinar si una ecuación diofántica polinómica tiene soluciones enteras. Esto se basó en los resultados previos de Gödel, Church y Turing, relacionados con la incompletitud y la computabilidad. Además el trabajo fue resultado del trabajo combinado con Martin Davis , Hilary Putnam y Julia Robinson que abarca 21 años, con Matiyasevich completando el teorema en 1970.

- ¿Cuál es la respuesta al problema planteado?

  La respuesta al décimo problema de Hilbert es fue negativa ya que no existe un algoritmo general que pueda determinar si una ecuación diofántica polinómica tiene soluciones enteras. Para algunos ejemplares de ecuaciones diofánticas, como el ejemplo con solución mencionado anteriormente ($2x + 3y = 7$), podemos encontrar soluciones enteras, pero para otros ejemplares, como el ejemplo sin solución ($2x+3y = 6$), no hay soluciones enteras. Demostrando que las limitaciones de la computación en términos de resolver ecuaciones diofánticas en general.

- Importancia o relevancia del problema: Podemos decir que el décimo problema de Hilbert es de gran importancia porque esté nos plantea una pregunta fundamental en la teoría de números y en la complejidad computacional, especialmente en la relación con las clases P y NP.

  La resolución(previamente mencionada) del décimo problema de Hilbert demostró la imposibilidad de encontrar un algoritmo general para determinar si una ecuación diofántica tiene soluciones enteras, resalta las limitaciones de la computación en términos de resolver este tipo de problemas.

  Finalmente el problema nos ilustra-enseña que la verificación de soluciones enteras es más fácil que encontrarlas, porque si se nos proporciona una solución podemos verificar en tiempo polinomial si es correcta, mientras que para encontrar una solución puede requerir un tiempo exponencial en algunos casos, y todo esto esta relacionado con la clasificación de problemas en P y NP, además aquí podemos resaltar la importancia de distinguir entre problemas de decisión (verificación de la solución) y problemas de búsqueda (encontrar la solución).

Bibliografía

- Garzon, J. C. (2020). Décimo problema de Hilbert (Lógica Matemática). Disponible en: https://www.academia.edu/43451748/D%C3%A9cimo_problema_de_Hilbert_L%C3%B3gica_Matem%C3%A1tica_
- Stadler, M. M. (2018, agosto 16). Julia Bowman Robinson y el décimo problema de Hilbert. Mujeres con ciencia. Disponible en: https://mujeresconciencia.com/2018/08/16/julia-bowman-robinson-y-el-decimo-problema-de-hilbert/
- Wikipedia contributors. (s/f). Décimo problema de Hilbert. Wikipedia, The Free Encyclopedia. Disponible en: https://es.wikipedia.org/w/index.php?title=D%C3%A9cimo_problema_de_Hilbert&oldid=133464742
- Décimo problema de Hilbert. (s/f). Hmn.Wiki. Recuperado el 8 de octubre de 2023, de https://hmn.wiki/es/Hilbert%27s_tenth_problem
- Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability: A guide to the theory of NP-completeness*. W.H. Freeman.
- Papadimitriou, C. H. (1993). *Computational Complexity*. Pearson.
- 02lambda. (s/f). Edu.ar. Recuperado el 8 de octubre de 2023, de https://cs.famaf.unc.edu.ar/~hoffmann/pd20/02.html
- Cálculo lambda. (s/f). Ecured.cu. Recuperado el 8 de octubre de 2023, de https://www.ecured.cu/C%C3%A1lculo_lambda
- Wikipedia contributors. (s/f-a). Cálculo lambda. Wikipedia, The Free Encyclopedia. Disponible en: https://es.wikipedia.org/w/index.php?title=C%C3%A1lculo_lambda&oldid=154143346