

Autómatas y Lenguajes Formales 2016-1

Maestría en Ciencia e Ingeniería de la Computación UNAM

Tema 14: Máquinas de Turing y computabilidad

Dr. Favio Ezequiel Miranda Perea

favio@ciencias.unam.mx

Facultad de Ciencias UNAM

3 de febrero de 2020



Lenguajes recursivos

- Un lenguaje L es **recursivo** si es reconocido por una máquina de Turing que siempre se detiene, es decir, si existe una máquina de Turing M que se detiene con todas las cadenas de entrada y $L = L(M)$.
- Los lenguajes recursivos también se conocen como Turing-decidibles.
- El adjetivo “recursivo” se debe a la siguiente propiedad de estos lenguajes:
Un lenguaje L es recursivo si y sólo si su función característica χ_L es recursiva.
- Esto implica que existe un algoritmo para decidir si una cadena pertenece a L .



Lenguajes recursivamente enumerables

- Un lenguaje L es **recursivamente enumerable** si es reconocido por una máquina de Turing, es decir, si existe una máquina de Turing M tal que $L = L(M)$.
- Los lenguajes recursivamente enumerables también se conocen como Turing-reconocibles o semidecidibles
- El adjetivo “recursivamente” se debe a la siguiente propiedad de estos lenguajes:
Un lenguaje L es recursivamente enumerable si y sólo si L es la imagen de una función recursiva total.
- Esto implica que existe un procedimiento para decidir si una cadena pertenece a L .



Lenguajes recursivos

Enumeración

- Si un lenguaje L es recursivo entonces existe un proceso de enumeración para L .
- Es decir, existe un proceso que genera todas las cadenas de L .
- Una máquina que enumera a L se construye componiendo una máquina M que genere a todas las cadenas del alfabeto de entrada, con una máquina N que reconozca a L .
- El proceso de enumeración consiste en repetir los siguientes pasos:
 - ▶ M genera a w
 - ▶ N verifica si $w \in L$, en caso positivo se imprime w , en otro caso se ignora a w .
- Este procedimiento funciona pues N siempre se detiene.



Lenguajes recursivamente enumerables

Enumeración

- Un lenguaje L es recursivamente enumerable si y sólo si existe un proceso de enumeración para L .
- El proceso de enumeración para un lenguaje recursivo no funciona aquí. Se debe combinar la ejecución de M y N de otra manera.
- El proceso de enumeración consiste en repetir los siguientes pasos:
 - ▶ M genera a la i -ésima cadena w_i
 - ▶ N ejecuta un paso (transición) en w_i , dos pasos en w_{i-1} , tres pasos en w_{i-2} , y así sucesivamente.
 - ▶ Si en algún momento N acepta a w_i , se imprime dicha cadena.
- Este procedimiento funciona pues N va procesando en tiempo finito fragmentos de cada cadena w_i



¿Qué es un algoritmo?

Introducción

- ¿Qué es un algoritmo?
- Podemos reconocer cuando vemos un algoritmo.
- ¿Podemos dar una definición precisa del concepto de algoritmo?
- ¿Por qué es importante tener una definición precisa (matemática) de algoritmo?



¿Qué es un algoritmo?

Introducción

- Un algoritmo es una colección de instrucciones simples para realizar una tarea o problema particular (procedimientos o recetas)
- Si se tiene un algoritmo para un problema dado P significa tener una manera para calcular efectivamente o resolver P .
- Un algoritmo es un proceso **potencialmente** realizable
 - ▶ Las operaciones del proceso se pueden realizar inequívocamente.
 - ▶ El número de operaciones o pasos del proceso es finito.



Existencia de algoritmos

Introducción

- Décimo problema de Hilbert: Hallar un proceso de acuerdo al cual pueda determinarse en un número finito de pasos (un algoritmo) si un polinomio dado tiene una raíz entera.
- Se creía que todo problema P tenía una solución algorítmica.
- Más aún se pensaba en la existencia de un algoritmo universal U que pudiera resolver todos los problemas matemáticos.



Existencia de algoritmos

Introducción

- Los intentos por hallar el algoritmo universal U fallaron.
- Tal vez U no existía.
- ¿Cómo probar la no existencia de U ?
- Era necesario definir el concepto de algoritmo de una manera precisa y hallar un formalismo para poder probar propiedades de los mismos.
- Un formalismo de algoritmos debería ser preciso y libre de ambigüedades, simple y general.



Formalización del concepto de algoritmo

Introducción

- Máquinas de Turing (Alan Turing, Cambridge 1936)
- Cálculo Lambda (Alonzo Church, Princeton 1936)
- Sistemas de Post (Emile Post)
- Funciones μ -recursivas (Gödel, Herbrand, Kleene)
- Lógica Combinatoria (Curry, Schönfinkel)
- Máquinas de registro (Sheperdson, Sturgis)



Equivalencia de los formalismos

Introducción

- Sorprendentemente todos los formalismos han resultado equivalentes.
- Un problema tiene solución en un formalismo si y sólo si tiene solución en cualquiera de los otros.
- Tal afirmación es un teorema, o una serie de teoremas rigurosamente demostrados.
- Esta coincidencia nos lleva a conjeturar que existe una única noción de computabilidad.



La Tesis de Church-Turing

Introducción

- La tesis de Church-Turing afirma que la noción intuitiva de algoritmo es capturada de manera exacta por la noción matemática de máquina de Turing.
- Tesis de Church-Turing:
Un problema es soluble algorítmicamente si y sólo si es soluble mediante una máquina de Turing
- Es decir, las máquinas de Turing implementan a cualquier algoritmo.
- Equivalentemente, una función es computable si y sólo si es soluble mediante una máquina de Turing.



La Tesis de Church-Turing

Introducción

Un problema es soluble algorítmicamente si y sólo si es soluble mediante una máquina de Turing

- La afirmación es una tesis indemostrable pues la noción de algoritmo es intuitiva.
- Por otro lado la tesis es refutable y se destruiría mostrando un algoritmo que no pudiera ser implementado en una máquina de Turing.



Tesis de Church-Turing

Evidencias a favor

- Existen fuertes evidencias a favor de la tesis.
- Intuitivamente cualquier algoritmo detallado para el cálculo manual puede programarse en una MT.
- La equivalencia con otros formalismos mas modernos.
- Existen demasiados ejemplos a favor y por supuesto ningún contraejemplo.
- La comunidad tanto en matemáticas como en ciencias de la computación acepta ampliamente la tesis.



Máquinas de Turing vs. Computadoras

- Una computadora es capaz de interpretar algoritmos arbitrarios y obtener la misma respuesta que cada algoritmo particular.
- Entonces una computadora es una máquina útil para propósitos generales.
- Las MT en cambio son diseñadas para propósitos particulares.
- Conclusión: el poder computacional de las MT no puede ser equiparable al de las computadoras actuales.
- Las computadoras son programables, las MT no.



La Maquina Universal de Turing

Introducción

- ¿Será factible pensar en la existencia de una MT que se comporte de la misma forma que una computadora real?
- Es decir, una MT que sea útil para propósitos multiples.
- Dicha máquina sería capaz de programar y ejecutar máquinas de Turing.



La Máquina Universal de Turing

Introducción

- Tal máquina existe y se conoce como máquina universal de Turing (MUT).
- La MUT recibe como entrada una descripción de una MT, M y una cadena w y simula el comportamiento de M sobre w .
- Los datos de entrada M y w deben ser codificados de manera adecuada.



Codificación de MT

Convenciones

- Se fija un alfabeto de entrada Σ .
- Se asume $Q = \{q_0, \dots, q_n\}$ siendo q_0 el estado inicial y q_1 el **único** estado final.
- El alfabeto de cinta es de la forma

$$\Gamma = \{s_1, s_2, \dots, s_p\}$$

siendo $s_1 = \perp$.



Alfabeto de Cinta

Codificación de MT

$$\Gamma = \{s_1, s_2, \dots, s_p\}$$

- El símbolo s_i se codifica como 1^i .
- Las cadenas Γ^* se codifican separando cada símbolo con 0.
- Por ejemplo, si $\Gamma = \{\perp, a, b\}$ entonces codificamos
 $\perp := 1, a := 11, b := 111$.
- La palabra $bab\perp aa$ se codifica como:

01110110111010110110



Alfabeto de Cinta

Codificación de MT

$$\Gamma = \{s_1, s_2, \dots, s_p\}$$

- En general, si $w = s_{n_1} s_{n_2} \dots s_{n_k}$ la codificación de w es:

$$01^{n_1} 01^{n_2} 0 \dots 01^{n_k} 0$$



Estados

Codificación de MT

$$Q = \{q_0, q_1, \dots, q_n\}$$

- El estado q_i se codifica, análogamente a los símbolos, mediante cadenas de unos pero con un símbolo más que el índice del estado

q_i se codifica como 1^{i+1}

- Con la convención tomada, el estado inicial q_0 se codifica con 1 y el estado final q_1 con 11.



Direcciones de desplazamiento

Codificación de MT

$$\{\rightarrow, \leftarrow, -\}$$

- \rightarrow se codifica con 1.
- \leftarrow se codifica con 11.
- $-$ se codifica con 111.



Transiciones

Codificación de MT

$$\delta(q_i, s_k) = (q_j, b_\ell, D)$$

- Los estados, símbolos y dirección de desplazamiento se codifican de la manera indicada anteriormente.
- La transición se codifica escribiendo en orden los códigos respectivos separados por ceros:

$$01^{i+1}01^k01^{j+1}01^\ell01^n0$$

donde $n = 1, 2, 3$.

- Por ejemplo $\delta(q_2, s_3) = (q_0, s_5, \leftarrow)$ se codifica como

$$01^301^30101^501^20$$



Máquinas de Turing

Codificación de MT

- Una MT queda completamente determinada mediante su función de transición δ .
- La MT se codifica mediante la sucesión de los códigos de sus transiciones sin separaciones. Es decir, si C_1, \dots, C_k son los códigos de todas las transiciones de M . Entonces M se codifica mediante

$$C_1 C_2 \dots C_k$$

- Obsérvese que no hay ambigüedad pues cada transición tiene exactamente seis ceros, además dos ceros consecutivos indican que inicia otra transición.



Codificación de MT

Ejemplo

La máquina M dada por:

$$\delta(q_0, a) = (q_2, b, \rightarrow) \quad \delta(q_2, b) = (q_3, c, \rightarrow)$$

$$\delta(q_3, a) = (q_1, c, \rightarrow) \quad \delta(q_1, b) = (q_3, \perp, -)$$

se codifica mediante $a := 11$, $b := 111$, $c := 1111$ como sigue:

0101101110111010011101110111101111010

01111011011011110100110111011110101110



Codificación de MT

Observaciones

- La codificación de una MT no es única, puesto que el orden de las transiciones no importa y un orden distinto genera una codificación distinta.
- De hecho si M tiene n transiciones, existen $n!$ codificaciones distintas para M .
- El proceso de codificación puede revertirse, no es difícil definir un algoritmo que decida si una secuencia binaria representa un código válido para MT y en tal caso lo decodifique.



Enumerabilidad de las MT

- En conclusión toda MT puede representarse como una cadena binaria.
- No todas las cadenas binarias representan MT válidas, por ejemplo, las cadenas que empiezan o terminan con 1 o las que tienen más de dos ceros consecutivos.
- Cada cadena binaria, representa por otra parte un número natural y viceversa. Es decir, hay tantas cadenas binarias como números naturales.
- De lo anterior se concluye que hay sólo un número numerable de MT.



Enumerabilidad de las MT

- Las cadenas binarias pueden enumerarse en orden lexicográfico con $0 < 1$:

0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, ...

- Cada MT figura varias veces en esta lista.
- Por motivos prácticos si una cadena no codifica directamente a una MT entonces acordamos que codifica a la máquina sin transiciones que acepta el lenguaje vacío.
- De esta manera cada cadena binaria codifica a una MT.



Existencia de funciones no computables

Enumerabilidad de MT

- Dado que las cadenas binarias son tantas como los números naturales y cada cadena codifica a una MT concluimos que sólo hay un número infinito numerable de MT.
- Por otro lado si consideramos las funciones $f : \mathbb{N} \rightarrow \mathbb{N}$ es bien sabido que son un número **no** numerable (tantas como números reales)
- De donde se concluye que existen funciones, que **no** pueden calcularse mediante una MT.
- Lo cual bajo la tesis de Church-Turing equivale a que existen funciones que no pueden ser calculadas mediante una computadora.



El lenguaje diagonal \mathcal{L}_D

- Consideraremos la enumeración de las máquinas de Turing M_1, M_2, \dots así como la enumeración de todas las cadenas de Σ^* , digamos $w_1, w_2, \dots, w_n, \dots$
- Podemos entonces dar como entrada la i -ésima palabra w_i a la i -ésima máquina M_i .
- El lenguaje diagonal se define como:

$$\mathcal{L}_D = \{w_i \mid w_i \text{ no es aceptada por } M_i\}$$

- Es decir \mathcal{L}_D contiene a la i -ésima cadena si y sólo si ésta no es aceptada por la i -ésima máquina.



\mathcal{L}_D no es recursivamente enumerable

- Si \mathcal{L}_D fuera R.E. sería aceptado por una MT, digamos la k -ésima máquina M_k .
- En tal caso $\mathcal{L}_D = L(M_k)$.
- Podemos preguntarnos entonces si $w_k \in \mathcal{L}_D$.
 - ▶ $w_k \in \mathcal{L}_D \Rightarrow w_k$ no es aceptada por $M_k \Rightarrow w_k \notin L(M_k) = \mathcal{L}_D$.
 - ▶ $w_k \notin \mathcal{L}_D \Rightarrow w_k \notin L(M_k) \Rightarrow w_k$ es aceptada por $M_k \Rightarrow w_k \in L(M_k) = \mathcal{L}_D$.
- Por lo que se tendría

$$w_k \in \mathcal{L}_D \text{ si y sólo si } w_k \notin \mathcal{L}_D$$

- Lo cual es absurdo.



El lenguaje universal $\mathcal{L}_{\mathcal{U}}$

Definición

- El lenguaje aceptado por la máquina universal \mathcal{U} se conoce como lenguaje universal, denotado $\mathcal{L}_{\mathcal{U}}$.

$$\mathcal{L}_{\mathcal{U}} = \{\langle M \rangle 0 \langle w \rangle \mid M \text{ acepta a } w\}$$

- Por definición $\mathcal{L}_{\mathcal{U}}$ es un lenguaje recursivamente enumerable.



El lenguaje universal $\mathcal{L}_{\mathcal{U}}$ no es recursivo

Supongamos lo contrario y sea M una máquina de Turing que siempre se detiene y tal que $\mathcal{L}_{\mathcal{U}} = L(M)$.

Veamos que a partir de M podemos construir una máquina de Turing M' que acepte al lenguaje diagonal \mathcal{L}_D , lo cual es absurdo.

M' funciona como sigue para la entrada $w \in \Sigma^*$:

- Enumerar las cadenas de Σ^* hasta encontrar $k \in \mathbb{N}$ tal que $w = w_k$.
- Llamar a M con entrada $\langle M_k \rangle 0 \langle w_k \rangle$.
- Como M siempre se detiene entonces decide si M_k acepta a w_k .
- En tal caso forzamos a que M' acepte también a w_k .
- Esto implica que $L(M') = \mathcal{L}_D$. Un absurdo.



Preguntas y Problemas

Introducción

- Preguntas:
 - ▶ ¿Qué x real cumple $2x^2 - 3x + 5 = 0$?
 - ▶ Dadas las ciudades a, b, c, d ¿Cuál es la forma óptima de visitarlas sin pasar dos veces por la misma ciudad?
- Una problema es una clase de preguntas:
 - ▶ ¿Cuáles son las soluciones de $ax^2 + bx + c = 0$?
 - ▶ ¿Dados n vértices en un grafo existirá un camino hamiltoniano?
- Cada caso particular de un problema es un ejemplo, ejemplar o instancia de éste.



Tipos de Problemas

Introducción

- La mayoría de los problemas de interés en ciencias de la computación son de dos tipos.
- Problemas de cómputo: obtener el valor de una función en un argumento dado.
 - ▶ Por ejemplo obtener la raíz cuadrada de un número dado x con exactitud de milésimas.
- Problemas de decisión: estos problemas tienen como respuesta si o no.
 - ▶ Por ejemplo, decidir la existencia de un camino óptimo en costos para recorrer varias ciudades.
- Si bien los problemas de decisión también podrían considerarse problemas de cómputo, es útil hacer la distinción.



Algoritmos y problemas

- Un algoritmo para un problema consiste de una serie de instrucciones capaces de responder cualquier instancia del problema dado.
- Un problema P se dice soluble si existe un algoritmo para P . En otro caso se dice insoluble.
- Si un problema de decisión P es soluble entonces decimos que P es decidible. En caso contrario el problema se dice indecidible.
- Un proceso o algoritmo para un problema de decisión se conoce como un proceso o algoritmo de decisión.



Algoritmos y funciones

- Cualquier algoritmo puede verse como una función:

entrada $x \rightarrow$ algoritmo \rightarrow salida y

- La salida está en función de la entrada

$$f(x) = y$$

- Una función es computable si existe un algoritmo que la calcule.
- La teoría de la computabilidad se encarga esencialmente a contestar la pregunta **¿Qué funciones son computables?**
- Lo cual equivale entonces a responder que problemas son solubles.



Problemas no computables

Introducción

- ¿Por qué nos interesa averiguar qué problemas no son computables mediante un algoritmo o proceso?
- Tales problemas son aquellos que **no** podemos resolver.
- Son resultados fundamentales y debemos conocerlos para tener una visión general de las ciencias de la computación.
- Debemos conocerlos para evitar intentar resolverlos.
- Debemos entender que tales problemas son insolubles independientemente del desarrollo futuro del hardware.



Recursividad y decidibilidad

Problemas insolubles

- Considérese una propiedad \mathcal{P} acerca de máquinas de Turing.
- \mathcal{P} genera el problema de decisión siguiente:
 ¿Satisface la máquina M la propiedad \mathcal{P} ?
- Asumiendo la tesis de Church-Turing, tal problema de decisión será decidable (soluble) si y sólo si el lenguaje
$$L = \{\langle M \rangle \mid \langle M \rangle \text{ es el código de una MT que satisface } \mathcal{P}\}$$
es recursivo.



El problema de la detención

Halting problem

Dada una máquina M y una cadena w ¿Se detendrá M al procesar w ?

- El problema sería soluble si pudieramos hallar una máquina H tal que al recibir como entrada a cualquier cadena $\langle M \rangle 0 \langle w \rangle$ se detuviera si y sólo si M se detiene al procesar w .
- Podría pensarse que una máquina universal puede hacer el trabajo.



El problema de la detención

Halting problem

- El problema de la detención resulta indecidible, es decir, no existe tal máquina H .
- Además es quizas el problema indecidible más relevante en la teoría de la computabilidad.
- Una consecuencia inmediata de su indecidibilidad es que no puede existir un programa que verifique si cualquier programa dado se cicla.



El problema universal

Problemas no computables

Dada una máquina de Turing cualquiera M y una cadena w
¿Acepta M a w ?

- El problema universal equivale a que el lenguaje universal

$$\mathcal{L}_{\mathcal{U}} = \{\langle M \rangle 0 \langle w \rangle \mid M \text{ acepta a } w\}$$

sea recursivo.

- Ya demostramos que $\mathcal{L}_{\mathcal{U}}$ no es recursivo.
- Por lo tanto el problema universal es indecidible.



Reducibilidad de problemas

- La reducibilidad de problemas es una técnica de gran utilidad para probar la indecidibilidad de un problema dado a partir de la indecidibilidad de un problema conocido.
- Dados dos problemas $\mathcal{P}_1, \mathcal{P}_2$ decimos que \mathcal{P}_1 se reduce a \mathcal{P}_2 si un algoritmo para decidir \mathcal{P}_2 puede emplearse para decidir \mathcal{P}_1 .



Reducibilidad de problemas

- Formalmente decimos que \mathcal{P}_1 se reduce a \mathcal{P}_2 , denotado $\mathcal{P}_1 \prec \mathcal{P}_2$ si existe una MT M tal que:
 - M recibe como entrada una instancia I_1 de \mathcal{P}_1
 - M devuelve como salida una instancia I_2 de \mathcal{P}_2 .
 - M decide a I_1 de la misma manera que I_2 .
 - Es decir, M responde con sí a I_1 si y sólo si responde con sí a I_2 .
 - De esta manera se tiene que \mathcal{P}_1 es decidible si y sólo si \mathcal{P}_2 es decidible.



- El problema universal puede reducirse al problema de la detención
- Supongamos que existe una máquina H que decide el problema de la detención.
- En tal caso H decide también al problema universal.
- Al recibir una entrada $\langle M \rangle 0\langle w \rangle$, H decide si M se detiene o no con entrada w .
- Si M no se detiene con w entonces M no acepta a w .
- Si M se detiene con w entonces M procesa a w y decide si la acepta o no.
- De manera que el problema universal se decide, lo cual es absurdo.



Otros problemas indecidibles

Reducibilidad

- Detención con cinta en blanco: ¿ se detiene la máquina M al iniciar con la cinta en blanco?
- Impresión de un símbolo: Dada M y $s \in \Sigma$ ¿Escribirá M en algún momento a s sobre la cinta?
- Dada una gramática libre de contexto G ¿ G es ambigua?
- Determinar si dos GLC son equivalentes.
- Cualquier propiedad no trivial acerca de MT (Teorema de Rice)

