Máquinas de Turing
Jerarquía de la Computabilidad.

April 4, 2019

Agustin Vanzato
Federico Gasquez

1. **Responder brevemente los siguientes incisos:**

   (a) **¿Qué es un problema (computacional) de decisión? ¿Es el tipo de problema más general que se puede formular?**
   Un problema de decisión es un problema en donde las respuestas posibles son "sí" o "no".
   Se entiende que este conjunto de problemas no es el conjunto más general dado que existen aquellos problemas que no pueden ser resueltos algorítmicamente, siendo estos aquellos que son no deterministicos para ciertos subconjuntos de un dominio dado, así como tambien aquellos problemas que ni siquiera son computables.
   De igual forma existen problemas que pueden ser resueltos algoritmicamente y son más generales que los problemas de decisión como por ejemplo los problemas de búsqueda.

   (b) **¿Qué cadenas integran el lenguaje aceptado por una MT?**
   El conjunto de las cadenas aceptadas por la MT M es el lenguaje aceptado o reconocido por M, y se denota con L(M). Considerando la visión de MT M calculadora, sólo cuando M se detiene en un estado $q \in F$ debe tenerse en cuenta el contenido final de la cinta, es decir la cadena de salida (o simplemente la salida).

   (c) **En la clase teórica 1 se hace referencia al problema de satisfactibilidad de las fórmulas booleanas (se da como ejemplo la fórmula $\varphi = (x_1 \vee x_2) \wedge (x_3 \vee x_4)$ y la asignación A = (V, F, V, V)).**
   Formular las tres formas del problema, teniendo en cuenta las tres visiones de MT consideradas: calculadora, aceptadora o reconocedora y generadora.

Para la MT reconocedora, recibe una fórmula y si es satisfactoria dice SI, caso contrario dice NO.
Para la MT calculadora, recibe una fórmula y calcula la asignación de verdad que la satisface.
Para la MT generadora, la máquina genera todas las fórmulas satisfacibles.

(d) **¿Qué postula la Tesis de Church-Turing?**

la conjetura conocida como Tesis de Church-Turing, indica que todo lo computable puede ser llevado a cabo por una máquina de Turing

(e) **¿Cuándo dos MT son equivalentes? ¿Cuándo dos modelos de MT son equivalentes?**

Dos máquinas de turing son equivalentes cuando reconocen el mismo lenguaje. Dos modelos de máquina de turing son equivalentes cuando poseen el mismo poder de cómputo, es decir a partir de una se puede simular la otra y viceversa.

(f) **¿En qué difiere un lenguaje recursivo de un lenguaje recursivamente numerable no recursivo?**

Un lenguaje es recursivamente numerable si y sólo si existe una MT que lo reconoce. Es decir, si L es el conjunto de todos los lenguajes (cada uno integrado por cadenas finitas de símbolos pertenecientes a un alfabeto universal $\sum$), sólo los lenguajes recursivamente numerables de L son reconocibles por una MT (por esto es que a los problemas de decisión asociados se los conoce como computables). La clase de los lenguajes recursivamente numerables se denomina RE (por recursively enumerable languages). El nombre se debe a que las cadenas de estos lenguajes se pueden enumerar. De esta manera, dado L $\in$ RE, si M es una MT tal que L(M) = L, se cumple para toda cadena w de $\sum^*$ que:

- Si w $\in$ L, entonces M a partir de w se detiene en su estado q_A .
- Si w $\notin$ L, entonces M a partir de w se detiene en su estado q R o no se detiene.

Se define que un lenguaje es recursivo si y sólo si existe una MT M que lo reconoce y que se detiene cualquiera sea su entrada. La clase de los lenguajes recursivos se denomina R. A los problemas de decisión asociados se los conoce como decidibles, porque las MT que los resuelven pueden justamente decidir, cualquiera sea la instancia, si es positiva o negativa. Ahora, dado L $\in$ R, si M es una MT tal que L(M) = L, se cumple para toda cadena w de $\sum^*$ que:

- Si w $\in$ L, entonces M a partir de w se detiene en su estado q A .

- Si w $\notin$ L, entonces M a partir de w se detiene en su estado q R .

Luego, sea $L \in RE - R$, diremos que L es un lenguaje recursivamente numerable no recursivo, es decir, que si existe una MT Ml que lo acepta, la misma no para siempre.

(g) **¿En qué difiere un lenguaje recursivamente numerable de uno que no lo es?**

Sea L un lenguaje computable que no es recursivamente numerable, es decir $L \in CORE - R$, entonces no existe una MT M_l tal que L(M_l)=L. Sin embargo, sabemos que existe una MT M_l talque L(M_l)=$L * c$.
Luego, si L fuese un lenguaje no computable, sabemos que existe una MT que lo reconozca.
Por lo tanto, sabemos que sea L un que no es recursivamente numerable, es decir $L \notin RE$, entonces no existe una MT M_l tal que L(M_l)=L.

(h) **Probar que** $R \subseteq RE \subseteq \mathcal{L}$.

Asumiendo un alfabeto universal de símbolos: $\sum$ ={a 1, a2, a3, ...}. $\sum^*$ es el conjunto de todas las cadenas finitas formadas con símbolos de $\sum$. L es el conjunto de todos los lenguajes formados con cadenas de $\sum^*$: L = P($\sum^*$), es decir que L es el conjunto de partes de $\sum^*$.
Un lenguaje L es recursivamente numerable L $\in$ R E si y sólo si existe una MT M_l que lo acepta, es decir L(M_l)=L . Por lo tanto, para toda cadena w de $\sum^*$:

- Si w $\in$ L , entonces M_l a partir de w para en su estado q_A.
- Si w *notin* L , entonces M_l a partir de w para en su estado q_R o no para.

Un lenguaje L es recursivo, L $\in$ R , si y sólo si existe una MT M_l que lo acepta y para siempre (también se puede decir directamente que lo decide). Por lo tanto, para toda cadena w de $\sum^*$:

- Si w $\in$ L , entonces M_l a partir de w para en su estado q_A .
- Si w $\notin$ L , entonces M_l a partir de w para en su estado q_R .

Luego, por definición se cumple $R \subseteq RE \subseteq L$.

(i) **¿Cuándo un lenguaje está en la clase CO-RE? ¿Puede un lenguaje estar al mismo tiempo en la clase RE y en la clase CO-RE? ¿Para todo lenguaje de la clase CO-RE existe una MT que lo acepta?**

Un lenguaje está en CO-RE si su lenguaje complemento está en RE. Si, en ese caso, dicho lenguaje está en R. No, solo si esta en R.

(j) **Justificar por qué los lenguajes $\sum$ y $\emptyset$ son recursivos.**
Sabemos por definición de $\sum^*$ que el mismo es un lenguaje infinito de cadenas finitas formado a partir de un alfabeto $\sum$, también finito por definición. Además, sabemos que $\sum^*$ es un conjunto recursivamente enumerable dado que existe un MT que compute su orden canónico. Luego, se entiende que el lenguaje $\sum^*$ pertenece a RE lo cual nos dice que siempre existe una MT que al menos lo acepte. Por otro lado, sabemos que dado un input $w \in \sum^*$ el mismo será finito y bastará con una cantidad finita de pasos dados por una MT para recorrerlo y, a su vez, determinar si el mismo es valido o no.
Finalmente se deduce que dicha MT parará, validando el input. Por lo tanto, $\sum^*$ es un lenguaje recursivo. La construcción de una MT similar a la anterior permitiría a su vez validar cualquier input perteneciente al $\emptyset$. Luego, el mismo será recursivo.

(k) **Si $L \subseteq \sum^*$, ¿se cumple que L $\in$ R?**
No necesariamente. Si esto sucediera, todos los lenguajes pertenecerían a R. Por ejemplo, HP no pertenece a R y esta incluido en $\sum^*$

(l) **Justificar por qué un lenguaje finito es recursivo.**
Sea L un lenguaje finito, sabemos que L es un conjunto recursivamente enumerable dado que existe un MT que lo compute. Luego, se entiende que el lenguaje L pertenecea RE lo cual nos dice que siempre existe una MT que al menos lo acepte. Por otro lado, sabemos que dado un input w $\in$ L el mismo será finito y bastará con una cantidad finita de pasos dados por una MT para recorrerlo y, a su vez, determinar si el mismo es valido o no. Finalmente se deduce que dicha MT parará, validando el input. Por lo tanto, L es un lenguaje recursivo.

(m) **Justificar por qué si L1 $\in$ CO-RE y L2 $\in$ CO-RE, entonces (L1 $\wedge$ L2) $\in$ CO-RE**
Sea L1 $\in$ CO-RE y L2 $\in$ CO-RE, entonces sabemos que $L_1^c, L_2^c RE$. De esto sabemos que para ambos lenguajes existe una MT que los acepta. Luego, se puede construir una MT M que reconozca su union, es decir:

i. $L_1^c, L_2^c RE$
ii. $(L_1^c \cup L_2^c) \in$ RE, por propiedad de union de lenguajes
iii. $L_3 \in$ RE, $L_3 = (L_1^c \cup L_2^c)$
iv. $L_3^c \in CO - RE$
v. $(L_1^c \cup L_2^c)^c \in$ CO-RE
vi. $(L_1 \cup L_2) \in$ CO-RE, por ley de morgan

vii. $(L_1 \cup L_2) \in$ **CO-RE**

2. **Dado el alfabeto $\sum$ = {a, b, c}:**

   (a) **Obtener el lenguaje $\sum^*$ y el conjunto de partes del subconjunto de $\sum^*$ con cadenas de a lo sumo dos símbolos. ¿Cuál es el cardinal (o tamaño) de este último conjunto?**
   $\sum = \{a, b, c\}$
   $\sum^*$ es todos los strings compuestos por combinaciones de a , b , y c .
   P(A) = { w $\in \sum^*$ / w no tenga más de 2 caracteres }
   P (A) son los conjuntos con todos los subconjuntos posibles de A
   |A|= 13 , |P(A)|= $2^1 3$

   (b) **Dado el lenguaje $L = a^n b^n c^n | n \geq 0$, obtener la intersección $\sum^* \cap$ L, la unión $\sum^* \cup$ L, el complemento de L respecto de $\sum^*$, y la concatenación $\sum^*$. L.**
   $L \cap \sum^* = L \Leftrightarrow L \subseteq \sum^*$
   $L \cup \sum^* = \sum^*$
   L^C con respecto a $\sum^*$ = conjunto de strings tal que no cumplen la forma $a^n b^n c^n$ con n ≥ 0 y pertenecientes a $\sum^*$
   $\sum$. L = { aw, bw, cw |w $\in$ L}

3. **Construir una MT (puede tener varias cintas) que acepte de la manera más eficiente posible el lenguaje $L = \{a^n b^n c^n | n \geq 0\}$. Plantear primero la idea general.**

   Ir avanzando en el string de entrada mientras sean "a", y agregar en la 2nda cinta un marcador por cada a. Una vez que se encuentra un carácter que no sea una "a" en la 1er cinta, seguir avanzando si son "b", y retroceder en la cinta 2. Cuando terminan las "b" en la 1er cinta, se tiene que estar al principio de la 2nda nuevamente. Avanzar nuevamente en las 2 cintas mientras que sean "c" en la 1era hasta que ambas encuentren blanco.
   $M = (Q, \sum, \Gamma, \delta, q_0, q_A, q_R)$
   $Q = \{q_1, q_2, q_3, q_0, q_A, q_R\}$
   $\sum = \{a, b\}$
   $\Gamma = \{a, b, x\}$
   $\delta(q_0, a, B) = (q_1, (a, R), (x, R))$
   $\delta(q_1, b, B) = (q_2, (b, S), (B, I))$
   $\delta(q_2, b, x) = (q_2, (b, R), (x, I))$
   $\delta(q_2, c, B) = (q_3, (c, S), (B, R))$
   $\delta(q_3, c, x) = (q_3, (c, R), (x, R))$
   $\delta(q_3, B, B) = (q_A, (B, S), (B, S))$

Esta máquina recorre sólo una vez el input dado, siendo así la más eficiente para la tarea dada.

4. **Explicar (informal pero claramente) cómo simular una MT por otra que en un paso no pueda simultáneamente modificar un símbolo y moverse.**

   Una MT normal tiene el movimiento Static, por lo que simular una máquina que no puede simultáneamente moverse y escribir un carácter se hace simplemente usando un movimiento S cuando se quiere modificar el carácter, y luego moverse sin modificar el carácter (porque ya estaba actualizado). Al poder simular el movimiento Static ya las máquinas serían equivalentes ya que comparten todos los otros movimientos y deltas posibles.

5. **Explicar (informal pero claramente) cómo simular una MT por otra que no tenga el movimiento S (es decir el no movimiento).**

   Para simular una MT por otra que no tenga el movimiento S, el mismo se podria simular haciendo que cuando modifica y se mueva(por ejemplo hacia la derecha), en el siguiente paso haga un movimiento hacia atras(siguiendo el ejemplo anterior, seria hacia la izquierda).

6. **Sea USAT el lenguaje de las fórmulas booleanas satisfactibles con exactamente una asignación de valores de verdad. P.ej.** $x_1 \wedge x_2$ **pertenece a USAT, mientras que** $(x_1 \wedge x_2) \vee x_3$ **no. Indicar, justificando la respuesta, si la siguiente MTN acepta USAT:**

   (a) **Si la fórmula de entrada no es correcta sintácticamente, rechaza.**

   (b) **Genera no determinísticamente una asignación A, y si A no satisface la fórmula, rechaza.**

   (c) **Genera no determinísticamente una asignación A' $\neq$ A. Si A' no satisface la fórmula, acepta, y si A' la satisface, rechaza.**

   Ayuda: Considerar p.ej. el caso en que la fórmula tiene dos asignaciones que la satisfacen.

   Acepta USAT, en el paso 1 se asegura que sea una fórmula sintácticamente correcta, en el paso 2 se asegura que sea satisfacible, pues al generar no deterministicamente las asignaciones A las genera a todas. En el paso 3 se asegura que no hay 2 asignaciones que den verdadero, por lo tanto la MTN acepta USAT.

7. **Considerando el Lema 4 estudiado en la Clase Teórica 2** ($R = RE \cap CO - RE$)**:**

(a) **Construir la MT M.**

(b) **Probar la correctitud de la construcción.**

8. **Sean L_1 y L_2 dos lenguajes recursivamente numerables de números naturales representados en notación unaria (por ejemplo, el número 5 se representa con 11111). Probar que también es recursivamente numerable el lenguaje L = {x | x es un número natural representado en notación unaria, y existen y, z, tales que y + z = x, con y $\in L_1$, z $\in L_2$}.**
Ayuda: la prueba es similar a la de la clausura de RE con respecto a la concatenación.

$L_1 \in RE \wedge L_2 \in RE$, entonces $\exists MT_1$ y MT_2 tal que acepten a L_1 y L_2 respectivamente. Por definición de L_1 y L_2 , ambos lenguajes están formados por cadenas de números naturales unarios (es decir que son representados por cadenas de "1").
$L = \{x/xesunnaturalunario \wedge \exists y, ztalquex = y + z\}$
Para que L ∈ RE tiene que ocurrir que $\exists MT_L$ que acepte a L .
Para cualquier entrada w el string se puede dividir en 2 partes (sin importar el largo de w , siempre serán cantidades finitas de combinaciones), lo que dejaría a w separado en 2 strings, y correr la MT_1 sobre la 1er parte y la MT_2 sobre la 2nda. Como L_1 y L_2 pertenecen a RE dichas máquinas podrían no parar nunca, por lo que se debe hacer hasta k pasos con todas las combinaciones posibles, haciendo correr las máquinas k pasos para cada división posible del string, luego k+1 pasos y así siguiendo para asegurar encontrar la combinación correcta, buscando el caso en donde ambas acepten su parte. De esta manera la MT_L aceptara solo si ambos aceptan, y en cualquier otro caso rechazará.

9. **Dada una MT M1 con $\sum$ = {0,1}**

(a) **Construir una MT M_2 que determine si L(M_1) tiene al menos una cadena.**

(b) **¿Se puede construir además una MT M_3 para determinar si L(M_1) tiene a lo sumo una cadena? Justificar.**

Ayuda para la parte (1): Si L(M1) tiene al menos una cadena, entonces existe al menos una cadena w de unos y ceros, de tamaño n, tal que M1 a partir de w acepta en k pasos. Teniendo en cuenta esto, pensar cómo M2 podría simular M1 considerando todas las cadenas de unos y ceros hasta encontrar eventualmente una que M1 acepte (¡cuidándose de los casos en que M1 entre en loop!)