IIC2213 — Lógica para Ciencia de la Computación

Notas Semana 1

1. Máquinas de Turing

Las máquinas de Turing son la herramienta más popular para entender qué problemas pueden ser resueltos por computadores. Luego de ver un poco de intuición, vamos a definir formalmente las nociones de máquinas de turing, lenguaje de una máquina, y vamos a plantear la hipótesis de que las máquinas corresponden exactamente a los computadores. Pero comencemos primero con un poco de notación.

1.1. Problemas como lenguajes

Para poder hablar sobre los problemas que resuelven las máquinas de Turing, tenemos que encontrar primero una noción general de *problema*. Si no hacemos esto, nos vamos a perder simplemente en comparar un problema con otro: qué es más difícil, calcular el camino más corto en un mapa, saber si una imagen tiene un gato o ganarle al campeón mundial de ajedrez?

En este cursos usamos el símbolo $\mathbf{A}$ para describir conjuntos de símbolos o caracteres, los que formarán un alfabeto. Por ejemplo, $\mathbf{A} = \{a, b, c\}$ es un alfabeto. Una palabra w sobre un alfabeto $\mathbf{A}$ es una secuencia finita de caracteres; por ejemplo $w = abba$ es una palabra del alfabeto $\{a, b, c\}$. Por convención, admitimos también una palabra de largo 0 (es decir, que no tiene caracteres), y la denotamos como ε. Finalmente, un lenguaje L sobre un alfabeto $\mathbf{A}$ es un conjunto de palabras sobre $\mathbf{A}$. Por ejemplo, $L = \{a^i b^i \mid i \in \mathbb{N}\}$ es un lenguaje, que describe el conjunto de las palabras ab, $aabb$, $aaabbb$ y así sucesivamente. En la definición, a^i es un atajo para hablar de una secuencia de a's de largo i.

Un lenguaje importante: cuando hablamos de $\mathbf{A}^*$ nos referimos al lenguaje $\mathbf{A}^* = \{w \mid w$ es una palabra sobre $\mathbf{A}\}$ de todas las palabras posibles que podemos formar con caracteres de $\mathbf{A}$ (incluyendo la palabra vacía ε). De esta forma, podemos definir el lenguaje de todos los palíndromos sobre $\mathbf{A}$ como $L_{\text{pal}} = \{w \in \mathbf{A}^* \mid w$ es un palíndromo $\}$, y luego $aabbaa$ y $abaaba$ pertenecen a L_{pal}, pero abb no pertenece.

A cada lenguaje L sobre un alfabeto $\mathbf{A}$ le podemos asociar el siguiente problema de decisión: Dado una palabra $w \in \mathbf{A}^*$, ¿es cierto que w pertenece a L?

Todos los problemas de decisión en este curso van a corresponder a este tipo de problemas: saber si una palabra dada pertenece a un lenguaje (la razón de por qué hacemos esto va a quedar clara más adelante). Por mientras, veamos que esta definición es bastante general, ya que podemos codificar muchos problemas como problemas de decision.

Por ejemplo, imaginemos el problema de saber si un entero n es primo. Para formalizar este problema como un problema de decisión de lenguajes, hacemos $\mathbf{A} = \{0, 1\}$ y $L_{\text{prim}} = \{w \in \mathbf{A}^* \mid w \text{ es la codificación binaria de un número primo}\}$. De esta forma, saber si un número n es primo es equivalente a preguntarnos si la codificación de n en binario pertenece a L_{prim}.

O el camino más corto: Si tenemos un mapa con ciudades $c_1, c_2, c_3, \ldots$, podemos razonar sobre el camino más corto entre una ciudad c y otra c' usando $\mathbf{A} = \{c_1, c_2, \ldots\}$. Así, cada palabra sobre $\mathbf{A}$ representa un camino entre c y c' y el problema de saber si una secuencia w es el camino más corto equivale a saber si w pertenece al lenguaje $L_{\text{shortest}} = \{w \in \mathbf{A}^* \mid w$ es el camino más corto entre c y $c'\}$.

1.2. Máquinas de Turing - Intuición

Para entender cómo funciona una máquina de turing, imaginemos que queremos diseñar un algoritmo que sea capaz de *resolver* el problema del lenguaje L_{pal}, es decir, saber si una palabra es un palíndromo. Fijemos $\mathbf{A} = \{0, 1\}$.

Entonces, un posible algoritmo para saber si una palabra es un palíndromo es el siguiente (usaremos como ejemplo $w = 010010$.

1. Leer el primer símbolo de una palabra, guardarlo en memoria y borrar ese símbolo. Es decir, quedamos con 10010 y guardamos un 0.

2. Avanzar hasta el último símbolo de una palabra, verificar que corresponde al símbolo que teníamos en memoria (si no, no es un palíndromo!) y borrar ese símbolo. Nos quedamos con 1001.

3. Leer el ultimo símbolo de la nueva palabra (un 1), guardarlo en memoria, borrarlo, avanzar hacia atrás

4. Verificar que el primer símbolo de la nueva palabra corresponde a lo guardado, borrarlo, repetir desde el inicio

5. Si la palabra se acaba al final de la instrucción (2) o (4), entonces sabemos que la palabra es un palíndromo

nota: Puedes ver este algoritmo funcionar en una Máquina de Turing, acá:

- Ingresa a `https://turingmachinesimulator.com/`
- Hace click en Examples $->$ binary palindrome
- Pon en input la palabra 010010 y aprieta Load
- Aprieta play y ve como funciona!

Las máquinas de Turing nos proveen una abstracción para especificar los algoritmos de manera formal. Se cree y se asume que esta abstracción es suficiente para formalizar todo lo que puede hacer un computador (esta es la hipótesis de Church-Turing).

Como te habrás dado cuenta, las máquinas funcionan con cuatro componentes principales:

- Una cinta de trabajo, infinita en ambas direcciones, dividida en celdas. En cada celda podemos escribir un símbolo de $\mathbf{A}$, y también hay un símbolo especial para denotar que la celda está vacía (en nuestro caso usaremos el B).
- Una cabeza lectora, que apunta a una celda determinada de la cinta.
- Un conjunto de estados: en cada instante la máquina se encuentra en un estado determinado.
- Un conjunto de instrucciones que determinan las acciones de la máquina.

El funcionamiento de las máquinas:

- Comenzamos con la cinta de trabajo con la palabra input w escrita y el resto de las celdas vacías. La cabeza lectora apuntando al primer símbolo de la palabra w y la máquina en un estado inicial q_0
- En cada paso, la máquina observa el símbolo que está leyendo la cabeza, y el estado de la máquina.
- De acuerdo al símbolo leído y al estado, el conjunto de instrucciones determina qué hacer: la máquina escribe un símbolo en la celda apuntada por la cabeza, pasa a un nuevo estado y se mueve a la izquierda o a la derecha.
- La máquina se detiene cuando no hay una instrucción definida para el símbolo leído y el estado actual. Se dice que la máquina *acepta* a la palabra w si el estado en el que se detuvo la máquina es uno de los estados designados como *finales*.

1.3. Máquinas de Turing - definiciones formales

Definición. Una máquina de turing determinista es una tupla $M = (Q, \mathbf{A}, B, q_0, F, \delta)$, donde

- Q es un conjunto de estados
- $\mathbf{A}$ es un alfabeto
- $q_0 \in Q$ es un estado inicial
- $F \subseteq Q$ son los estados finales

- δ es la función de transición, una función parcial definida como

$$\delta : Q \times (\mathbf{A} \cup \{B\}) \to Q \times (\mathbf{A} \cup \{B\}) \times \{\leftarrow, \rightarrow\}$$

Ejercicio. La siguiente es una máquina que acepta todas las palabras binarias de largo par. Definimos $M = (Q, \mathbf{A}, B, q_0, F, \delta)$, con $Q = \{q_0, q_1, q_p, q_i\}$, $\mathbf{A} = \{0, 1\}$, $F = \{q_p\}$ y donde la función de transición se define como:

$$\begin{array}{llll} \delta(q_0, B) & \to & (q_p, B, \to) & \delta(q_1, 0) \to (q_0, 0, \to) \\ \delta(q_0, 0) & \to & (q_1, 0, \to) & \delta(q_1, 1) \to (q_0, 1, \to) \\ \delta(q_0, 1) & \to & (q_1, 1, \to) & \delta(q_1, B) \to (q_i, B, \to) \end{array}$$

para este ejercicio, comprueba cómo opera la máquina, paso por paso, con inputs $w_1 = 0010$, $w_2 = 010$ y $w_3 = \varepsilon$ (usamos ε para representar la palabra que no tiene caracteres)

Definición. El lenguaje aceptado por una máquina de turing $M = (Q, \mathbf{A}, B, q_0, F, \delta)$, escrito como $L(M)$, corresponde a todas las palabras $w \in \mathbf{A}^*$ tal que M acepta a la palabra w cuando comienza con la palabra input w escrita y el resto de las celdas vacías, y con la cinta lectora apuntando al primer símbolo de w.

1.4. Algoritmos = máquinas de turing

Definición. Sea L un lenguaje sobre un alfabeto $\mathbf{A}$. Decimos que una máquina de Turing M con alfabeto $\mathbf{A}$ puede *resolver* el problema de decisión asociado a L si el lenguaje $L(M)$ corresponde a L.

Hipótesis de Church-Turing. Un problema de decisión puede ser resuelto por un procedimiento mecánico si y solo si existe una máquina de turing que lo resuelve, y que además siempre se detiene en todas las entradas.

Aunque no hemos podido demostrar esto, todos lo creemos cierto. ¿Por qué pedimos además que la máquina se detenga en todas las entradas?

2. Ejercicios

2.1. Diseñando máquinas

Diseña máquinas que acepten los siguientes lenguajes sobre $\mathbf{A} = \{0, 1, \#\}$. Asegúrate que funcionen, usando varios inputs. Para el último ejercicio, si ya estás familiarizado, puedes ahorrarte una descripción completa de la máquina, e intentar en cambio una descripción más de alto nivel.

1. Una máquina de turing que acepte todas las palabras en $\mathbf{A}^*$ que no tienen el símbolo $\#$.

2. Una máquina de turing que acepte todas las palabras en $\mathbf{A}^*$ cuyo largo es un múltiplo de tres.

3. Una máquina de turing que acepte todas las palabras en $\mathbf{A}^*$ en los que la cantidad de 0's es igual a la cantidad de $1'$s.

4. Una máquina que acepte solo los strings de la forma $u\#v$, con $u, v \in \{0,1\}^*$ y donde $n(v)$ sea el sucesor de $n(u)$, donde $n(s)$ para un string s en binario es el número representado por s.

5. Una máquina que *compute* el sucesor: que reciba un string u binario y que al final de la ejecución la máquina tenga el string dado por $n(u) + 1$ (en binario). ¿Qué lenguaje acepta esta máquina?

6. Una máquina que acepte los strings de la forma $u\#v\#w$, con $u, v, w \in \mathbf{A}^*$ y donde $n(u) + n(v) = n(w)$. Nota que puedes escribir esta máquina directamente, o usar la máquina anterior como subrutina! ¿Puedes explicar cómo harías la subrutina?

2.2. Máquinas como subrutinas

Asume que tienes todas las máquinas de los ejercicios anteriores. Cómo podrías diseñar una máquina que acepta a los strings de la forma $u\#v$, con $u, v \in \{0,1\}^*$ y donde $n(u) = n(v) + 1$?

Cómo podrías diseñar una máquina que acepta a los strings de la forma $u\#v\#w$, con $u, v, w \in \{0,1\}^*$ y donde $n(u) = n(v) + n(w)$?

3. Problema

Definición. Una máquina de turing con cinta para un solo lado es una tupla $M = (Q, \mathbf{A}, B, \vdash, q_0, F, \delta)$, donde

- Q es un conjunto de estados
- $\mathbf{A}$ es un alfabeto que no contiene ni a B ni a $\vdash$
- $q_0 \in Q$ es un estado inicial
- $F \subseteq Q$ son los estados finales
- δ es la función de transición, una función parcial definida como

$$\delta : Q \times (\mathbf{A} \cup \{B, \vdash\}) \to Q \times (\mathbf{A} \cup \{B\}) \times \{\leftarrow, \rightarrow\},$$

en donde además $\delta(q, \vdash)$ está siempre en $Q \times (\vdash) \times \{\rightarrow\}$, lo que quiere decir que cada vez que la máquina lee $\vdash$, tiene que volver a escribir $\vdash$ y además moverse a la derecha.

El lenguaje aceptado por una máquina de turing con cinta para un solo lado $M = (Q, \mathbf{A}, B, \vdash, q_0, F, \delta)$, escrito como $L(M)$, corresponde a todas las palabras $w \in \mathbf{A}^*$ tal que M acepta a la palabra $w = a_1 \cdots a_n$ cuando comienza con la palabra $\vdash a_1 \cdots a_n$ escrita y el resto de las celdas vacías, y con la cinta lectora apuntando al símbolo a_1.

Demuestra que para cada máquina de Turing M existe una máquina de Turing con cinta para un solo lado M' tal que $L(M) = L(M')$, y que para cada máquina de Turing con cinta para un solo lado M existe una máquina de Turing M' tal que $L(M) = L(M')$.