IIC2213 — Lógica para Ciencia de la Computación

Notas Semana 2

1. Hacia adentro en la hipótesis de Church-Turing

Esta semana vamos a precisar algunas definiciones, y profundizar sobre la diferencia de poder de máquinas cuando exigimos o no exigimos que estas siempre paren.

1.1. Loops infinitos en máquinas

Ejercicio. Diseña una máquina de turing M cuyo lenguaje sean los strings de largo par sobre el alfabeto $\mathbf{A} = \{0, 1\}$, pero tal que tal que existe una palabra w para la que M nunca se detiene con input w.

El hecho de que no todas las máquinas se detengan con todos los input no pone en una posición incómoda: Si queremos saber si una máquina M acepta a una palabra w, no sabemos a priori cuanto esperar! Sabemos que si M acepta a w entonces eventualmente se va a detener en un estado final, pero no sabemos en qué minuto se detiene. Por otro lado, si M no acepta a w podría ser que M se quedara pegada, en un loop infinito, y no tendríamos cómo saber!

De aquí surge uno de los problemas más importantes en ciencia de la computación. Le llamamos el problema de la parada, pero bien podría ser el problema del loop infinito.

Problem:	HALTING (PARADA)
Input:	Máquina de turing M, palabra w
Output:	¿Es verdad que M se detiene con input w?

Qué pasa con estas máquinas? En general, poder darse el lujo de no parar significa que podemos usar estas máquinas para buscar respuestas en un espacio potencialmente infinito de respuestas. Y, la próxima semana, vamos a ver que efectivamente nos da poder adicional. Por ahora, un ejemplo:

El décimo problema de Hilbert consiste en saber si un polinomio, con coeficientes enteros de cualquier grado y con cualquier cantidad de variables, puede evaluar a cero cuando las variables solo toman valores enteros. Este problema puede resolverse con una máquina de Turing que no se detiene en algunas entradas. La pregunta de si acaso puede resolverse con una máquina de Turing que si se detiene en todas las entradas estuvo abierta 70 años, entre 1900 y 1970, y hoy sabemos que no es posible.

1.2. Dos clases de lenguajes

Definición. Un lenguaje L sobre un alfabeto $\mathbf{A}$ es *recursivamente enumerable* si existe una máquina de turing M tal que

- $L(M) = L$, es decir, el lenguaje de la máquina es precisamente L.

Definición. Un lenguaje L sobre un alfabeto $\mathbf{A}$ es *decidible* si existe una máquina de turing M tal que

- $L(M) = L$, es decir, el lenguaje de la máquina es precisamente L.
- M para en todas las entradas.

Si L no es decidible, entonces decimos que L es *indecidible*.

De acuerdo a estas definiciones, el lenguaje de polinomios descrito arriba es recursivamente enumerable, pero no es decidible.

Ejercicio. A pesar de que ya conocemos uno, podemos demostrar rápidamente la existencia de lenguajes indecidibles por un argumento de conteo. Sea $\mathbf{A}$ un alfabeto. ¿Cuál es la cárdinalidad del conjunto de todas las máquinas de turing sobre $\mathbf{A}$? ¿Cuál es la cardinalidad del conjunto de todos los lenguajes posibles? ¿Qué puedes concluir al respecto?

Máquinas que computan, funciones computables

Sea $\mathbf{A}$ un alfabeto, y $f : \mathbf{A}^* \to \mathbf{A}^*$ una función que transforma algunas palabras en otras. Decimos que f es *computable* si existe una máquina de turing M con un único estado final tal que para cada $w \in \mathbf{A}^*$ la máquina se detiene en el estado final, y donde la cinta de trabajo consiste en una secuencia infinita de blancos, seguidos de $f(w)$, seguidos de otra secuencia infinita de blancos.

Ejercicio. Sea $f : \{0,1\}^* \to \{0,1\}^*$ la función que invierte 0's y 1's en un string. Por ejemplo, $f(001) = 110$ y $f(11011) = 00100$. Muestre que f es computable. Muestra también que la función $f(x) = x + 1$ es computable (asumiendo que x viene en binario). ¿Cómo podríamos definir una máquina de turing que compute la suma de dos números binarios? ¿La multiplicación? Nuevamente, piensa que podemos usar máquinas dentro de otras máquinas, como si fueran sub-rutinas.

Ejercicio. ¿Puedes pensar en una función que no sea computable?

Hipótesis más precisa

Hipótesis de Church-Turing, de verdad. Una función $f : \mathbb{N} \to \mathbb{N}$ se puede calcular con un procedimiento mecánico si y solo si f es computable.

2. Problema

Máquinas de turing universales

Queremos mostrar también que es posible diseñar una máquina de turing *universal*: que pueda recibir cualquier otra máquina y ejecutar lo que esa máquina quiere ejecutar. Esto sería, intuitivamente, equivalente a un computador que ejecuta cualquier pedazo de código.

Codificación de una máquina. Para poder pasar las máquinas como input, vamos a ver como codificar todas las máquinas como palabras sobre el alfabeto $\mathbf{A} = \{0, 1\}$. La idea, en corto, es usar los símbolos 0 para separar las distintas componentes de las máquinas, y codificar esas componentes en unario, usando los símbolos 1.

Supongamos que la máquina $M = (Q, \mathbf{A}, B, q_1, \delta, F)$ es tal que $Q = \{q_1, \ldots, q_n\}$ y $F = \{q_{i_1}, \ldots, q_{i_m}\}$, con cada i_j un número entre 1 y n.

Definimos la palabra $C(M) = w_Q 00 w_F 00 w_\delta$, donde

- w_Q es $10110 \cdots 01^n$: una palabra que es la concatenación de un uno, dos unos, hasta llegar a n unos, separados por un símbolo 0. Intuitivamente, esto codifica la cantidad de estados en Q.
- w_F es la palabra $1^{i_1}01^{i_2}0 \ldots 1^{i_m}$, donde cada 1^{i_j} es una palabra con i_j unos.
- w_δ consiste en la concatenación de todas las instrucciones de la forma $\delta(q, s) = \{q', s', d\}$, con $d \in \{\leftarrow, \rightarrow\}$. Cada concatenación está separada por 00, y cada componente está separada por un 0. Para representar al estado q_i se usa la palabra 1^i, para representar los símbolos 0, 1 y B usamos 1, 11 y 111, respectivamente, y usamos 1 para $\leftarrow$ y 11 para $\rightarrow$.

  Por ejemplo, si $\delta(q_2, 0) = (q_4, 1, \rightarrow)$, esa instrucción se codifica con la palabra

  $$110101111011011$$

Ejercicio. Repasa la codificación con la máquina para ver palabras de largo par que vimos en clases.

Máquina Universal. Una máquina universal $\mathcal{U}$ es una máquina que:

- Si su input es de la forma $C(M)0000w$, para $w \in \mathbf{A}^*$, entonces $\mathcal{U}$ acepta al input si y solo si M acepta a w.
- Si su input no es de la forma $C(M)0000w$, entonces $\mathcal{U}$ no acepta al input.

Ejercicio. ¿Puedes pensar en como se construiría esta máquina?

Re-visitando el problema de la parada

Consideremos el lenguaje

$$H = \{w \in \{0,1\}^* \mid$$
$$w = u0000v, \text{ existe una máquina } M \text{ tal que } u = C(M) \text{ y } M \text{ se detiene con entrada } v\}$$

Dada una máquina M y una palabra v, vemos que el problema de HALTING es equivalente a decidir si $C(M)0000v$ pertenece a H. Este es el marco de trabajo en el que vamos a apoyarnos para mostrar que HALTING es indecidible.

Ejercicio. Muestra que H es recursivamente enumerable. Para eso, puedes apoyarte en la noción de máquina de turing universal.