IIC2213 — Lógica para Ciencia de la Computación

Notas Semana 3

1. Lenguajes indecidibles, reducciones

En esta clase nos vamos a adentrar más en poder mostrar qué lenguajes son indecidibles y cuáles no. Estas propiedades son fundamentales al tomarlas en conjunto con la hipótesis de Church-Turing: nos dicen qué cosas pueden o no pueden hacer los computadores.

Lo primero que vamos a hacer es mostrar la existencia de dos lenguajes indecidibles. Sabemos por el argumento de conteo que no puede ser que todos los lenguajes sean decidibles (hay una cantidad enumerable de máquinas de turing, pero una cantidad no-numerable de lenguajes). Sin embargo, la demostración de abajo es *constructiva* (construye los lenguajes), y por lo tanto más valiosa, por que esos lenguajes van a ser nuestro punto de partida. Otra gracia es que los lenguajes de abajo son recursivamente enumerables, y por tanto esto también muestra la existencia de lenguajes RE que no son decidibles.

1.1. Diagonalización

Consideremos el lenguaje

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

En palabras, D es el lenguaje de todas las codificaciones de máquinas tales que la máquina codificada se detiene con el input dado por su propia codificación.

Teorema (diagonalización). El lenguaje D es indecidible.

Este teorema se demuestra usando la técnica de diagonalización de Cantor. Podemos también usar la misma técnica para demostrar que el siguiente lenguaje también es indecidible:

$$DA = \{w \in \{0,1\}^* \mid \text{existe una máquina } M \text{ tal que } w = C(M) \text{ y } M \text{ acepta a} w\}$$

1.2. Reducciones

Queremos demostrar que H es indecidible, pero hasta ahora solo sabemos que D es indecidible. Sería un poco incómodo tener que encontrar un argumento parecido a diagonalización cada vez que queremos demostrar que un lenguaje es indecidible. En vez de eso, vamos a usar el concepto de *Reducción de Turing*, que nos va a ayudar a capturar la idea de que

cierto lenguaje L_2 es al menos tan difícil como un lenguaje L_1. Recuerda que las funciones computables son todas las que pueden ser expresadas por máquinas de Turing.

Definición. Dados lenguajes L_1 y L_2 sobre un alfabeto $\mathbf{A}$, decimos que L_1 es Turing-reducible a L_2, o igualmente que existe una reducción (de Turing) de L_1 a L_2, si existe una función computable f tal que para todo $w \in \mathbf{A}^*$,

$$w \in L_1 \text{ si y solo si } f(w) \in L_2.$$

Como prometimos, el concepto de reducción captura la noción de que L_2 es al menos tan difícil como L_1, o que, equivalentemente, L_1 no es más difícil que L_2. Esto lo podemos formalizar así:

Teorema[1.2] . Si L_1 es reducible a L_2, entonces:

- Si L_2 es decidible, entonces L_1 es decidible.
- Si L_1 es indecidible, entonces L_2 es indecidible

Demostración. Como L_1 es reducible a L_2 entonces existe una función computable f y una máquina M_r que, al recibir como input una palabra $w \in L_1$ se detiene con $f(w)$ en la cinta.

Supongamos primero que L_2 es decidible, y sea M_2 una máquina de turing tal que $L_2 = L(M_2)$. Para mostrar que L_1 es decidible tenemos que construit una máquina M_1 tal que $L_1 = L(M_1)$. Definimos la máquina de la siguiente manera:

Con input w, la máquina M_1 primero ejecuta las instrucciones de M_r. Cuando M_r se detiene, ejecuta luego las instrucciones de M_2, y se detiene cuando se detiene M_2. En otras palabras, para construir M_1, juntamos M_r y M_2 y reemplazamos el estado inicial de M_2 por el final de M_r: si $M_r = (Q_r, \boldsymbol{A}, q_0^r, q_f^r, \delta_r)$ y $M_2 = (Q_2, \boldsymbol{A}, q_0^2, F_2, \delta_2)$, entonces $M_1 = (Q_r \cup Q_2^*, \boldsymbol{A}, q_0^r, F_2, \delta_r \cup \delta_2^*)$, pero en donde Q_2^* y δ_2^* corresponden a reemplazar q_0^2 por q_f^r.

Para mostrar que $L_1 = L(M_1)$ mostramos ambos lados de la inclusión. Primero, para ver que $L_1 \subseteq L(M_1)$, sea $w \in L_1$. Con input w, M_r computa $f(w)$, y por la propiedad de las reducciones sabemos que $f(w) \in L(M_2)$. Por construcción de M_1 deducimos que $w \in L(M_1)$ (por que M_1 es equivalente a correr M_r y sobre eso M_2). Para ver que $L(M_1) \subseteq L_1$, sea $w \in L(M_1)$. Con input w, M_r siempre computa $f(w)$. Luego si $w \in L(M_1)$ entonces $f(w)$ debe pertenecer al lenguaje de M_2. Por la propiedad de las reducciones, y como asumimos que $L_2 = L(M_2)$, obtenemos que w pertenece a L_1.

Supongamos ahora que L_1 es indecidible, y para obtener una contradicción, que L_2 es decidible. Usando el argumento anterior, del hecho que L_2 es decidible podemos deducir que L_1 es decidible, lo que es una contradicción, y por ende L_2 debe ser indecidible (¿puedes reproducir este argumento con el mismo detalle de arriba?)

Problema de la parada es indecidible

Usando el teorema anterior podemos demostrar que H es indecidible. Sea $\mathbf{A} = \{0, 1, \#\}$ y f una función de $\mathbf{A}^*$ en $\mathbf{A}^*$ definida como $f(w) = w\#w$.

Ejercicio. Demuestra que $w \in D$ si y solo si $f(w) \in H$. Usa el teorema 1.2 para concluir que H es indecidible.

2. Relaciones entre lenguajes recursivamente enumerables e indecidibles

Claramente, todo lenguaje decidible también es recursivamente enumerable (RE), pero el teorema de diagonalización nos muestra que hay lenguajes RE que no son decidibles. Podemos explicar como hacer máquinas de turing para mostrar:

Propiedad 1. Si L_1 y L_2 son lenguajes decidibles, entonces $L_1 \cap L_2$, $L_1 \cup L_2$ y el complemento $\bar{L_1}$ son todos decidibles.

Esta propiedad se demuestra construyendo las máquinas adecuadas. Un poco más complicado, pero podemos mostrar también:

Propiedad 2. Si L_1 y L_2 son lenguajes RE, entonces $L_1 \cap L_2$ y $L_1 \cup L_2$ son RE.

Y qué pasa con el complemento de un lenguaje RE? Su complemento no necesariamente es RE. De hecho:

Propiedad 3. Si L es RE, y el complemento $\bar{L}$ es RE, entonces L es decidible.

Demostración. Por definición tenemos una máquina M_L que acepta a una palabra w si y solo si w pertenece a L, y una una máquina $M_{\bar{L}}$ que acepta a una palabra w si y solo si w pertenece a $\bar{L}$. Luego, podemos construir una máquina M que funcione de la siguiente forma: con input w, M echa a correr M_L y $M_{\bar{L}}$ *en paralelo*, una instrucción a la vez. Si M_L acepta, aceptamos, si $M_{\bar{L}}$ acepta, M rechaza. Si $w \in L$ entonces sabemos que la componente M_L de M va a parar, y si $w \notin L$ significa que $w \in \bar{L}$, y por tanto la componente $M_{\bar{L}}$ va a parar. Concluimos que M acepta a L, y se detiene en todas sus entradas, por lo que L es decidible.

En general, la clase de lenguajes cuyo complemento es RE se conoce como coRE, y por la propiedad anterior vemos que RE intersectado con coRE corresponde a la clase de lenguajes decidibles.

3. Más lenguajes indecidibles

Demostremos que esto es indecidible

$$DyD = \{w \in \{0,1\}^* \mid \text{ existen máquinas } M_1, M_2 \text{ tal que } w = C(M_1)0000C(M_2), \text{y}$$
$$M_1 \text{ se detiene con entrada } C(M_1), M_2 \text{ se detiene con entrada } C(M_2)\}$$

Solución. Reduciendo desde D, simplemente haciendo $f(w) = w0000w$. Esta función es computable, y es fácil ver que w pertenece a D si y solo si $f(w)$ pertenece a DyD.

Ejercicio. Muestra la indecidiblidad de

$$DoD = \{w \in \{0,1\}^* \mid \text{ existen máquinas } M_1, M_2 \text{ tal que } w = C(M_1)0000C(M_2) \text{ y}$$
$$\text{o } M_1 \text{ se detiene con entrada } C(M_1) \text{ o } M_2 \text{ se detiene con entrada } C(M_2)\}$$

3.1. Reducciones más complejas

Muestra la indecibilidad de:

$$SIM = \{w \in \{0,1\}^* \mid \text{ existe una MT } M \text{ tal que } w = C(M)0000v \text{ y } M \text{ acepta a } v\}$$

Solución. Vamos a reducir desde H. Sea f la siguiente función. Si w corresponde a la codificación $C(M)0000v$ de una máquina M y un string v, $f(w)$ es la codificación $C(M')0000v$ de una máquina M' que es igual a M solo que todos sus estados son finales y el mismo string v. Si w no corresponde a esa codificación, entonces $f(w) = C(\hat{M})00001$, donde $\hat{M}$ es una maquina con lenguaje vacío.

Demuestra estas dos cosas, con lo que puedes invocar al teorema de reducciones.

1. Comprueba que w pertenece a D si y solo si $f(w)$ pertenece a SIM.
2. Que f es computable por una máquina de turing.

Veamos ahora que este lenguaje también es indecidible:

$$ALL = \{w \in \{0,1\}^* \mid$$
$$\text{existe una MT } M \text{ tal que } w = C(M) \text{ y } M \text{ acepta todas las palabras}\}$$

Solución. Vamos a reducir desde D. Sea f la siguiente función. Si w corresponde a la codificación $C(M)$ de una máquina M, $f(w)$ es la codificación $C(M')$ de una máquina M' que hace lo siguiente: Con cualquier input, M' borra toda la cinta, la reemplaza por w, y luego simula a M con input w, con la excepción de que todos los estados de M son estados finales. Si w no corresponde a esa configuración, entonces $f(w)$ es la codificación de una máquina con lenguaje vacío.

Tenemos que comprobar dos cosas:

1. Comprobar que w pertenece a D si y solo si $f(w)$ pertenece a ALL.
2. Que f es computable por una máquina de turing.

Vemos (1). Supongamos primero que w pertenece a D, luego en particular $w = C(M)$ para alguna máquina M. Entonces, $f(w)$ es la codificación de una máquina que siempre borra el input, y echa a andar a M con input $C(M)$. Si w pertenece a D, entonces M se detiene con input $C(M)$, y por tanto $f(w)$ es la codificación $C(M')$ de una máquina M' que siempre va a aceptar cualquier input, pues siempre va a detenerse en un estado final. Si $f(w)$

pertenece a ALL, eso quiere decir que $f(w) = C(M')$ y M' acepta a todas las palabras. Esto implica que el lenguaje de M' no es vacío, y de acuerdo a la construcción de $f(w)$, tiene que ser que la palabra w corresponda a la codificación de una máquina, es decir, $w = C(M)$, y en ese caso M' es una máquina que llama a M con input $C(M)$. Como M' acepta a todas las palabras, debe ser que M se detiene con input $C(M)$, y por tanto w pertenece a D.

Para (2), lo primero es chequear si w corresponde a la codificación de una máquina, lo que es computable (es un parser). Luego, si w es $C(M)$, $f(w)$ tiene que tomar M y construir M' agregándole a M la funcionalidad de borrar todo el input para correr en vez con input $C(M)$, y luego hacer todos los estados de M finales. Todo esto puede ser construido directamente desde la codificación; en este curso no vamos a pedir mayor detalle que esto. Si w no es la codificación, entonces cualquier máquina con lenguaje vacío basta.