Alfredo Carrillo del Campo

Notas de Clase

Complejidad y Computabilidad

Profesor:
Dr. Rodolfo Martinez Conde

Otoño 2018

Temario

1. Introducción
   1.1. Necesidad de la complejidad complejidad computacional
   1.2. Alfabetos y lenguajes
   1.3. Problemas y su codificación en el alfabeto $\{0, 1\}$
2. Computabilidad
   2.1. Maquinas de Turing
   2.2. Lenguajes computables y computables enumerablemente
   2.3. Imposibilidad del problema de detención
   2.4. Reducciones Computables
3. Clases de Complejidad
   3.1. Notación asintótica y funciones de complejidad apropiadas
   3.2. Clases básicas de complejidad: **TIME(f(n)), SPACE(f(n)), P, NP, L, NL, PSPACE y EXP)**
   3.3. Teoremas del aceleramiento lineal y la jerarquía del tiempo y sus consecuencias.
4. Reducciones y completez
   4.1. Reducciones polinomiales
   4.2. Problemas completos y duros
   4.3. La importancia de los problemas completos
5. Problemas **NP**-completos
   5.1. Teorema de *Cook*
   5.2. Problemas **NP**-completos básicos

Referencias

[1] Michael R Garey and David S Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[2] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[3] Dexter C Kozen. Automata and computability, undergraduate texts in computer science, 1997.

[4] Cristopher Moore and Stephan Mertens. *The nature of computation*. OUP Oxford, 2011.

[5] Christos H Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[6] Michael Sipser. *Introduction to the Theory of Computation*, volume 2. Thomson Course Technology Boston, 2006.

Formas de calificar

1. Ejercicios 40 %
2. Examen Parcial 10 %
3. Exposiciones 30 %
4. Examen final 20 %
5. Apuntes 15 %

1. Introducción

1.1. Necesidad de la complejidad computacional.

Es una materia *teórica* donde se resolverán preguntas como:

- ¿Qué significa que una función sea computable?
- ¿Existen funciones no computables?
- ¿Cómo el poder de computo depende en los constructos de programación?
- ¿Cómo se define el Modelo matemático de computo (Máquina de Turing)?
- Si sí se puede calcular, ¿Cuánto cuesta calcularlo tanto en tiempo como en espacio de memoria? Aquí entran medidas como eficiencia, y un término que no se puede traducir: *untracktable*.[1]

1.2. Alfabetos y Lenguajes

Definición 1.2.1. Un **alfabeto** es un conjunto Σ finito. En símbolos, $|\Sigma| < \infty$.

Ejemplo 1.2.1. Determinamos cuáles conjuntos son alfabetos.

1. Todos los siguientes son alfabetos:
   - El alfabeto binario $\Sigma = \{0, 1\}$.
   - $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.
   - Todas las letras del alfabeto junto con { á, é, í, ó, ú}.
   - $\Sigma = \{a, b, c\}$.
   - $\Sigma = \{a_1, a_2, a_3, \ldots a_n\}$
   - $\Sigma = \emptyset$.
2. El siguiente conjunto no es un alfabeto.
   - $I = [0, 1]$

Definición 1.2.2. A los elementos del conjunto Σ se les llama **símbolos** o **letras**.

Definición 1.2.3. Una **cadena** sobre un alfabeto Σ es cualquier secuencia finita de elementos de Σ. También algunos autores usan el término de **palabra**.

Ejemplo 1.2.2. Sea $\Sigma = \{a, b\}$, entonces $aaba$ es una cadena sobre Σ. Pero $aa1ab$ no es una cadena sobre sigma pues $1 \notin \Sigma$.

Definición 1.2.4. La **longitud** de una cadena x sobre Σ se denota $|x|$ y es el numero de símbolos que contiene x.

[1] Se refiere a que si vale la pena correr el algoritmo. Si se tarda mucho no vale la pena.

Ejemplo 1.2.3. $|aaba| = 4$ o $|aabab| = 5$.

Existe una cadena única sobre Σ llamada cadena vacía y la denotamos por $\lambda(\Sigma)$. Si no hay ambigüedad, simplemente λ. Se caracteriza como aquella tal que $|\lambda| = 0$.

Si Σ es un alfabeto, es conveniente suponer siempre que $\forall a \in \Sigma$, $|a| = 1$. Es decir, ningún símbolo de un alfabeto se puede descomponer en símbolos más simples. Específicamente, no es la concatenación de cadenas no vacías.

Una cadena de la forma $aa \dots a$ donde a se repite n veces se puede denotar por a^n. Por ejemplo, $aa = a^2$ o $aabb = a^2b^2$. Podemos dar una definición inductiva (recursiva) de a^n como sigue:

$$\begin{cases} a^0 & = \lambda \\ a^{n+1} & = aa^n \end{cases}$$

Definición 1.2.5. El conjunto de todas las cadenas sobre un alfabeto lo denotamos por $\Sigma^\star$ y le llamamos **cerradura** de Σ. La cerradura puede considerarse como el conjunto de palabras. En símbolos, $\Sigma^\star = \{x : |x| < \infty, \forall a \in x \Rightarrow a \in \Sigma\}$.

Ejemplo 1.2.4. Se muestran las cerraduras de distintos conjuntos.

- La cerradura de $\{a\}$ es $\{a\}^\star = \{\lambda, a, aa, aaa, aaaa, \dots\} = \{a^n : n \in \mathbb{N} \cup \{0\}\}$.
- La cerradura de $\{a, b\}$ es $\{a, b\}^\star = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, \dots\}$.

Operaciones en cadenas

Definición 1.2.6. Sea $\Gamma = \{a_1, a_2, \dots, a_n\} \cup \{b_1, b_2, \dots, b_m\} \subseteq \Sigma$, éste último un alfabeto. Sean $x = a_1a_2 \dots a_n$ y $y = b_1b_2 \dots b_m \in \Sigma^\star$. La operación $(x \circ y))$ de **concatenación**, con notación más compacta xy, se define como $xy = a_1a_2 \dots a_nb_1b_2 \dots b_m \in \Sigma^\star$.

Ejemplo 1.2.5. Sea $\Sigma = \{0, 1\}$ un alfabeto y $x, y \in \Sigma$ dados por $x = 10111, y = 0001$, entonces $xy = 101110001 \in \Sigma^\star$.

Hasta ahora, el símbolo a^5 es mera notación. Sin embargo, tras introducir la definción previa, puede considerarse como la operación concatenación 5 veces de a con sí mismo. Más aún, podemos extender esta noción a cadenas. Si x es una cadena entonces x^n es la cadena formada tras concatenar a x con si mismo n veces. Por ejemplo, $(aba)^5 = (aba)(aba) \dots (aba) = abaaba \dots aba$.

Definición 1.2.7. Un **monoide** es una pareja $(A, \circ)$ donde A es un conjunto y $\circ$ es una operación binaria $\circ : A \times A \to A$ donde se satisfacen las siguientes tres propiedades: cerradura ($\circ$ está bien definida), asociatividad y

Aquél que conozca de álgebra moderna puede recordarlo como "casi" un grupo pues no pide la existencia de elementos inversos.

Teorema 1.2.1. *Sea Σ un alfabeto y $\circ$ denota la operación de concatenación. Luego $(\Sigma, \circ)$ es un monoide.*

Demostración. Primero, la operación $\circ$ es cerrada porque si $x, y \in \Sigma^\star$, entonces xy es una cadena cuyos símbolos están en Σ, se sigue que $xy \in \Sigma^\star$.

Segundo, la operación es asociativa. Sea Σ un alfabeto y x, y y $z \in \Sigma^\star$. Sea $x = a_1a_2..a_n, y = b_1b_2 \dots b_n$ y $z = c_1c_2 \dots c_n$. Luego

$$\begin{aligned}(xy)z &= (a_1a_2..a_nb_1b_2 \dots b_n)c_1c_2 \dots c_n \\ &= a_1a_2..a_nb_1b_2 \dots b_nc_1c_2 \dots c_n \\ &= a_1a_2..a_n(b_1b_2 \dots b_nc_1c_2 \dots c_n) \\ &= a_1a_2..a_n(yz) \\ &= x(yz)\end{aligned}$$

Tercero, la operación contiene un elemento identidad denominado λ. Esta prueba es trivial pero se hace en el Ejercicio 1. Note que como no existe un elemento inverso, $(\Sigma, \circ)$ no es un **grupo**. ■

Ejercicio 1. Probar que para todo $x \in \Sigma^\star$ se sigue que $x\lambda = \lambda x = x$.

Ejercicio 2. Probar que para toda $x, y \in \Sigma^\star$ se sigue que $|xy| = |x| + |y|$.

Operaciones en conjuntos

Se denotan subconjuntos de $\Sigma^\star$ usualmente con las letras $A, B, C, \dots$. Se define la operación concatenación sobre conjuntos y se definen propiedades y notaciones análogas a las de la sección previa.

Definición 1.2.8. Un **lenguaje** es un subconjunto $A \subseteq \Sigma^\star$ de $\Sigma^\star$.

Como $\Sigma \subseteq \Sigma^\star$ entonces cualquier alfabeto $A \subseteq \Sigma$ es a su vez un lenguaje.

Definición 1.2.9. Sean $A, B \subseteq \Sigma^\star$ dos lenguajes. La operación **concatenación** de A y B se define como $AB = \{xy : x \in A, y \in B\}$.

Como la operación concatenación está definida tanto para cadenas como para conjuntos, es importante distinguir el contexto, ya que la definición depende de éste. En ningún caso la operación es conmutativa.

Ejemplo 1.2.6. Sea $A = \{a, aa\}, B = \{b, bb\}$. Luego $AB = \{ab, abb, aab, aabb\}$.

Ejemplo 1.2.7. Sea $A = \{a, aa\}, B = \{\lambda\}$, se sigue que $AB = A$. Más aún, así como λ es el elemento identidad para la concatenación en cadenas, en el contexto de conjuntos el elemento identidad es $\{\lambda\}$.

Ejercicio 3. Mostrar que si $A \subseteq \Sigma^\star$ y $B = \emptyset$ entonces $AB = \emptyset$.

Notación.

a) Sea $A \subseteq \Sigma$ un alfabeto, luego $A^n = AA \dots A$, n veces.

b) Equivalentemente, $A^n = \{x \in A^\star : |x| = n\}$.

Exhibimos porque ambas notaciones son consistentes. Partiendo de la primera: Sea $A^n = \{a_1 a_2 \dots a_n : a_i \in A\} = \{x : x \in A^\star, |x| = n\}$, que coincide con la segunda notación. Lo importante de esto, es que la primera tiene que ver con n concatenaciones de conjuntos, mientras que la segunda con n la concatenación de símbolos.

Ejemplo 1.2.8. Consideremos al alfabeto $\{a, b\}$. Queremos calcular $|\{a, b\}^n|$.

$$|\{a, b\}^n| = |\{a, b\}\{a, b\} \dots \{a, b\}| = |\{a, b\}||\{a, b\}| \dots |\{a, b\}| = 2 \cdot 2 \dots 2 = 2^n.$$

Un resultado más general es el siguiente. Sea A un alfabeto, luego $|A^n| = |A|^n$.

La definición 1.2.5 de cerradura la vimos estrictamente a partir de alfabetos. Ahora se extiende de forma más general, a también a lenguajes.

Definición 1.2.10. Sea $A \subseteq \Sigma^\star$ un lenguaje. La **cerradura** o **estrella de Kleene** de A es

$$A^\star := \bigcup_{i=0}^{\infty} A^i,$$

donde $A^0 = \{\lambda\}$ y $A^{n+1} = A^n A$. En palabras, podemos pensar a $A^\star$ como todas las cadenas que podemos formar con el lenguaje A. Una notación equivalente en términos de concatenación de cadenas y no de conjuntos es

$$A^\star = \{x_1 x_2 \dots x_n : n \geq 0, x_i \in A\}$$

Expliquemos por qué es conveniente que $A^0 = \{\lambda\}$. Notemos que operaciones del tipo: $A^n A^m = A^{n+m}$. Nos gustaría entonces que $A^0 A^m = A^m$, es decir que A^0 actué como elemento identidad. Éste sabemos que es único y es $\{\lambda\}$. En particular, como $\emptyset$ es un alfabeto entonces $\emptyset^0 = \{\lambda\}$. Por otro lado, para alfabetos no vacíos $A \subseteq \Sigma$, se sigue que $|A| < \infty$ y $|A^\star| = \infty$. El símbolo de * nos dice que el conjunto es infinito.

Ejercicio 4. Probar que si $A \subseteq \Sigma^\star$ entonces $A^\star A^\star = A^\star$.

Propiedades adicionales:

1. $A^{\star\star} = A^\star$ (idempotencia de *).
   Por definición, $A^{\star\star} = \cup_{i=0}^{\infty}(A^\star)^i$. En el ejercicio anterior probamos que $(A^\star)^2 = A^\star$. Usando inducción es fácil probar que $(A^\star)^i = A^\star$, para toda $i \geq 1$ y $(A^\star)^0 = \{\lambda\}$, por definición. Luego $A^{\star\star} = \cup_{i=0}^{\infty}(A^\star)^i = \cup_{i=1}^{\infty} A^\star = A^\star$.

2. $A^\star = \{\lambda\} \cup AA^\star$. Se deduce del siguiente argumento:

   $$AA^\star = A \cup_{i=0}^{\infty} A^i = \cup_{i=0}^{\infty} A^{i+1} = \cup_{i=1}^{\infty} A^i \Rightarrow AA^\star \cup \{\lambda\} = \cup_{i=0}^{\infty} A^i = A^\star.$$

3. $\emptyset^\star = \{\lambda\}$. De la definición tenemos que $\emptyset^\star = \cup_{i=0}^{\infty} \emptyset^i = \emptyset^0 = \{\lambda\}$.

1.3. Problemas y su codificación en el alfabeto {0,1}

En esta sección nos interesa describir un alfabeto dado en términos de otro. Informalmente, dados dos alfabetos A y B, el problema de la *codificación* consiste en definir una función ν donde para cada símbolo $b \in B$, exista un elemento $a \in A$ que le "pegue", es decir $\nu(a) = b$. Intuitivamente estamos buscando una función $\nu : A^\star \to B^\star$ sobre, codificamos el alfabeto B a partir del alfabeto A. Sin embargo, esta definición presentaría el problema que quizá es complicado mapear a todo $A^\star$ además de que no es útil. Por ello, mejor utilizamos sólo las palabras de $A^\star$ que nos sirvan para codificar a B, es decir un subconjunto de $A^\star$.

Definición 1.3.1. Definimos una **codificación válida** del alfabeto B a partir del alfabeto A si existe una función $\nu : \Gamma \to B^\star$, donde $\Gamma \subseteq A^\star$, que sea sobre.

Notación. Quizá, no sea riguroso escribir $\nu : \Gamma \subseteq A^\star \to B^\star$ pero en este contexto es conveniente. Es simplemente para compactar la notación y decir que Γ es el dominio de la función y a su vez es un subconjunto de $A^\star$.

Los siguientes ejemplos codifican respectivamente a: $\mathbb{N}, \mathbb{Z}, \mathbb{Q}$ a partir de alfabetos pequeños, como por ejemplo $\Gamma \subseteq \mathbb{Z}_2^\star = \{0,1\}^\star$. Note que las codificaciones propuestas están bien definidas.

Ejemplo 1.3.1. Codifiquemos a $\mathbb{N}$. Proponemos la función $\nu : \Gamma \subseteq \mathbb{Z}_2^\star \to \mathbb{N}$, donde $\nu(1^n) = n$. Podemos ver que es una codificación válida, en el sentido que es una función sobre. Aquí $\Gamma = \{1^n : n \in \mathbb{N}\}$. Hay codificaciones más eficientes como por ejemplo a partir del sistema binario, donde con n bits podemos representar 2^n números.

Ejemplo 1.3.2. Para codificar a $\mathbb{Z}$, proponemos agregar el símbolo #. Ahora, $\Gamma = \{0, 1, \#\}$, donde "#" se usa para distinguir el signo y hacemos lo mismo que en el Ejemplo 1.3.1. Formalmente, $\nu : \Gamma \subseteq \{0,1\}^\star \to \mathbb{Z}$ donde $\Gamma = \{x1^n, n \in \mathbb{N}, x \in \{\lambda, \#\}\} \cup \{0\}$,

$$\nu(x) = \begin{cases} n & \text{si } x = 1^n \\ -n & \text{si } x = \#1^n \\ 0 & \text{si } x = 0 \end{cases}$$

En los siguientes ejemplo por simplicidad no definiremos más el conjunto Γ. Quedará implícito con la función de codificación.

Ejemplo 1.3.3. Para $\mathbb{Q}$ podemos considerar un símbolo adicional "\". Ahora $\Sigma = \{0, 1, \#, \backslash\}$ y proponemos la función $\nu : \Gamma \subseteq \Sigma^\star \to \mathbb{Q}$, donde

$$\nu(x) = \begin{cases} n\backslash m & \text{si } x = 1^n\backslash 1^m \\ -n\backslash m & \text{si } x = \#1^n\backslash 1^m \\ 0 & \text{si } x = 0 \end{cases}$$

Lema 1.3.1. Si existe una codificación válida $\sigma' : \Gamma' \subseteq A^\star \to B$ válida entonces se puede extender para dar una codificación válida $\sigma : \Gamma \subseteq A^\star \to B^\star$.

Demostración. Se descompone una cadena en $B^\star$ en símbolos, $b = b_1 b_2 \dots b_n$. Por hipótesis existe $a_i \in \Gamma'$ tal que $\sigma'(a_i) = b_i$. Definimos a $\sigma : (\Gamma')^\star \to B^\star$ como sigue $\sigma(a_1 a_2 \dots a_n) = \sigma'(a_1)\sigma'(a_2)\dots\sigma'(a_n) = b_1 b_2 \dots b_n$. Por lo tanto σ es sobre. ■

A continuación mostramos un par de ejemplos con conjuntos menos convencionales.

Ejemplo 1.3.4. Codificar $\{0, 1, \#\}^\star$ a partir de $\Gamma\{0, 1\}^\star$. Por el Lema 1.3.1 basta codificar $\{0, 1, \#\}$. Definimos $\nu : \Gamma \subseteq \{0, 1\}^\star \to \{0, 1, \#\}$ como sigue,

$$\nu(x) = \begin{cases} 0 & \text{si } x = 00 \\ 1 & \text{si } x = 01 \\ \# & \text{si } x = 10 \end{cases}$$

Ejemplo 1.3.5. Consideremos un alfabeto arbitrario $A = \{a_1, a_2, ..., a_n\}$ y el alfabeto $\{0, 1\}$. Propondremos codificaciones en ambas direcciones.

1. Primero definimos $\nu : A \subseteq A^\star \to \{0, 1\}^n$. Sea $\mathbb{N}_n = \{a \in \mathbb{N} : a \leq n\}$. Definimos una codificación auxiliar válida (suprayectiva) $\nu_1 : A \subseteq A^\star \to \mathbb{N}_n$ como sigue, $\nu_1(a_i) = i$. Definimos $\nu_2 : \mathbb{N}_n \to \{0, 1\}^n$ como sigue. Sea k tal que $2^k \geq n$, luego es posible representar a cada número en $\mathbb{N}_n$ en una cinta de longitud fija k, de la forma binaria usual. Se rellena con ceros a la izquierda para que la codificación sea siempre de exactamente k dígitos. Luego $\nu = \nu_2 \circ \nu_1$ es sobre.

2. Ahora definimos una codificación $\omega : \Gamma \subseteq \{0, 1\}^n \to A$. Como ν_1 es biyectiva, definimos $\omega_1 : \mathbb{N}_n \to A$ como sigue $\omega_1(i) = (\nu_1)^{-1}(i) = a_i$. También ν_2 es biyectiva gracias a que usamos una longitud fija para representar a un número. Luego definimos a $\omega_2 : \{0, 1\}^n \to \mathbb{N}_n$ como sigue $\omega_2 = (\nu_2)^{-1}$. Se sigue que $\omega = \omega_1 \circ \omega_2 : \{0, 1\}^n \to A$ es sobre.

   Otra codificación válida directa es definir a ω como sigue: $\omega(0^{n-i}1^i) = a_i$.

Matrices

Si $\mathcal{K}$ es un campo, se definen las matrices $M_{m\times n}(\mathcal{K})$ cuyos elementos $a_{i,j} \in \mathcal{K}$. Los campos con los que estaremos trabajando comúnmente son $\mathbb{Q}, \mathbb{R}, \mathbb{C}, \mathbb{Z}_2$.

$$M = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Para codificar matrices se extiende el diccionario de la forma $\Sigma = \{0, 1, \#, @\}$, donde los últimos dos símbolos son para separar renglones y columnas, respectivamente.

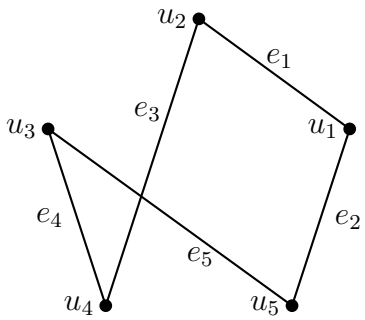
Definición 1.3.2. Una **gráfica** (o **gráfo**) es una pareja ordenada $G = (V, E)$ donde $V = \{v_1, v_2, ..., v_n\}$ es un conjunto no vacío de vértices. Mientras que E es el conjunto de aristas dado como sigue, $E = \{uv : u, v \in V\}$.

En las **gráficas dirigidas** las aristas se llaman **arcos** y varían en que son *dirigidas*, es decir, tienen un sentido. Se denotan así $\vec{uv}$, con origen en u y destino en v.

Definición 1.3.3. La **matriz de adyacencia** de G es la matriz simétrica $A = (a_{ij})$ con $i, j = 1, 2, ..., n$, donde

$$a_{ij} = \begin{cases} 1, & \text{si } i \text{ es adyacente a } j. \\ 0, & \text{en otro caso.} \end{cases}$$

Ejemplo 1.3.6. Ahora se presenta una gráfica y su respectiva matriz de adyacencia.



Descripción gráfica.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

Matriz de adyacencia.

Ejercicio 5. ¿Cuál es el número máximo de vértices de las gráficas que se pueden representar por matriz de adyacencia con $k = 32$ bits y con $k = 64$ bits.

Problemas de Decisión y Lenguajes

Definición 1.3.4. Un **problema de decisión** es una función con un *output* de un sólo bit: "sí" o "no". Para especificar un problema de decisión uno debe definir

- El conjunto A de posibles *inputs* (instancias).
- El subconjunto $Y \subseteq A$ de las denominadas "sí-instancias".

Hay problemas de decisión tanto en las matemáticas, como en la vida real. En este contexto, un *lenguaje* define un *problema* y un *problema* define un *lenguaje*. La idea es convertir una instancia de un problema, por ejemplo una gráfica dada, a una cadena de símbolos, digamos al alfabeto $\{0, 1\}$. Después se computa esta cadena para saber si se "acepta" o se "rechaza". Existe una correspondencia donde la "aceptación" corresponde a una "sí-instancia" mientras que el "rechazo" a una "no-instancia".

Instancia → Traducción a Lenguaje → Lectura de Máquina → Acepta\Rechaza.

Definición 1.3.5. Sea $G = (V, E)$ y $I \subseteq V$. Decimos que I es un **conjunto independiente** si y sólo si para todo $u, v \in I$ se sigue que $uv \notin E$.

Ejemplo 1.3.7. Dada una gráfica, encontrar el conjunto independiente de tamaño máximo (o conjunto máximal) es un problema clásico. Este problema tiene su equivalente problema de decisión, donde aquí además de dar la misma instancia se provee de una $k \in \mathbb{N}$ adicional. La pregunta que se hace es si existe o no un conjunto máximal de tamaño k en la gráfica G. Son *sí-instancias* aquellas que tienen un conjunto independiente de tamaño k. Las vamos a denotar por el conjunto $\langle G, k \rangle$.

Para la última flecha del diagrama anterior podríamos usar una máquina de Turing. En el siguiente capítulo estudiaremos cómo se definen éstas para que su función sea aceptar a elementos que sean la codificación de una sí-instancia.

2. Computabilidad

Autómatas finitos

Intuitivamente, un *estado* de un sistema puede pensarse como una foto (*snapshot*) de éste. Se guarda así su configuración momentánea. Por ejemplo en una partida de ajedrez es cualquier escenario posible válido. De un estado podemos considerar las posibles *transiciones* que pueden proseguir. Note que por más complejo que es el ajedrez existen un número finito de escenarios posibles. A este tipo de sistemas, con un número de estados finitos, se les conoce como *finite-state transition system* y el modelo que le asignamos se le conoce como *autómatas finitos*.

Definición 2.0.1. La definición formal de un **autómata finito determinístico** M es una 5-tupla dada por $M = (Q, \Sigma, \delta, q_0, q_y)$, donde

- Q es un conjunto finito, los elementos son llamados estados.
- Σ es un alfabeto.
- $\delta : Q \times \Sigma \to Q$ es una función de transición. Dado un estado y un input, nos dice cual es el nuevo estado del sistema que se genera.
- $q_0 \in Q$ es el estado inicial.
- $q_y \in Q$ es el estado de aceptación.

Ejemplo 2.0.1. Sea un autómata finito $M = (Q, \Sigma, \delta, q_0, q_y)$ donde $Q = \{0, 1, 2, 3\}$, $\Sigma = \{a, b\}, q_y = \{3\}, q_0 = 0$ y la función delta está dada como sigue:

$$\delta(q, x) = \begin{cases} 1 & x = a, q = 0 \\ 2 & x = a, q = 1 \\ 3 & x = a, q = 2, 3 \\ q & x = b \end{cases}$$

Analizando a la función notamos que si el input es b, la transición nos dice que nos quedamos en el mismo estado. En cambio, cuando es a se abren casos, pero básicamente nos cambiamos de estado hasta quedar en el estado 3.

Informalmente la máquina opera como sigue. El *input* puede ser cualquier cadena x de $\Sigma^\star$. Supongamos que estamos en el estado inicial q_0. Se escanea el *input* x de izquierda a derecha, un símbolo a la vez. Leemos el primer símbolo y hacemos una transición de acuerdo a la función δ. Note que δ recibe un símbolo, no una cadena. Por eso a cada símbolo corresponde una transición. Eventualmente, se termina de leer la cadena x. En ese momento consideramos el estado actual y verificamos si está en q_y o no. En el primer caso decimos que x es *aceptado*, en el segundo que es *rechazado*.

2.1. Máquina de Turing

Se introduce aquí uno de los más poderoso autómatas que estudiaremos: las máquinas de *Turing*. Se llaman así por Alan Turing quien las inventó en 1936. Son capaces de computar cualquier función que nosotros estemos acostumbrados a que una computadora realice. Por ello, tiene sentido decir que una función es computable, si lo es por una Maquina de Turing. Su definición está motivada porque los matemáticos buscaban definir el concepto de *efectividad computacional*. Con ello, plantear un modelo que formalizara qué es una función computable y que no es una función computable. Informalmente, una función no computable sería aquella que no acaba en tiempo finito o una función que no se pueda programar. Nos interesan los límites de estos modelos computacionales, más aún porque se asemejan a las capacidades de una computadora real.

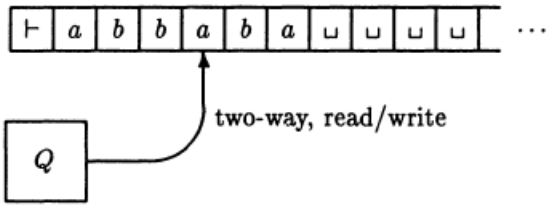
Se plantearon varios modelos, además de las máquinas de Turing. Por ejemplo los *Post systems*, μ-*recursive functions*, λ-*calculus*, *combinatory logic*. Todos son distintos pero es posible emular lo que hace cualquiera de estos modelos por cualquiera de los otros. Realmente, función computable no tiene que definirse relativo a cuál modelo pues son todos equivalentes, aunque aquí se hizo relativo a una máquina de Turing. Se eligió así porque el modelo de máquina de Turing es el que mejor captura la esencia de ser computable. Se define ahora rigurosamente una máquina de Turing.

Definición 2.1.1. Sean Γ un alfabeto y $\Sigma \subseteq \Gamma$. Una **máquina de Turing** (MT) es una 9-tubla, $M = (Q, \Sigma, \Gamma, \vdash, \sqcup, \delta, q_0, q_y, q_n)$, donde

- Q es un conjunto finito (estados).
- Σ es un conjunto finito (el alfabeto del *input*).
- Γ es un alfabeto de cinta ($\Sigma \subseteq \Gamma$).
- $\delta : Q \times \Gamma \to Q \times \Gamma \times \{R, L\}$ función de terminación (programa).
- $\sqcup$ el símbolo de espacio en blanco.
- $\vdash$ el símbolo de inicio hasta en la cinta, indica donde empieza el input.
- $q_0 \in Q$ Estado inicial
- $q_y \in Q$ Estado de aceptación
- $q_n \in Q$ Estado de rechazo

La máquina de Turing es el más similar a una computadora actual entre los otros modelos propuestos. Consiste en una cinta semi-infinita. Decimos *semi* porque la cinta sólo es infinita hacia la derecha. El símbolo $\vdash$ denota el inicio de la cinta. La cabeza de la cinta se puede mover en ambos sentidos y puede tanto leer, como escribir. En la cinta hay una cadena finita de símbolos, que llamamos *input*. El input se escribe justo a partir del símbolo $\vdash$. Cuando acaba, se considera que tenemos

infinitos espacios, denotados por $\sqcup$. La máquina comienza con un estado inicial q_0 y empieza a analizar los símbolos en la cinta de izquierda a derecha. Cada acción consiste en los siguientes pasos en estricto orden: leer el símbolo y estado actual, cambiar o no de estado, mover el lector un sólo espacio, ya sea a la derecha o a la izquierda.



Intuitivamente, $\delta(p, a) = (q, b, R)$ significa: si se está en el estado p viendo en la cinta al símbolo $a \in \Gamma$, entonces se pasa al estado q, se escribe el símbolo "b" sobrescribiendo "a" y se mueve la cabeza lector en la dirección de "R".

Si la maquina entra en el estado q_y decimos que la **cadena es aceptada**. Si entra en el estado q_n decimos que la **cadena es rechazada**. En ambos casos decimos que la máquina M se **detiene**, en el sentido que estos estados son estacionarios. Una vez que caes en ellos, no se cambian jamás bajo ningún número de transiciones. En caso contrario, decimos que la máquina se queda en un **ciclo** (infinito), o que la máquina se **encicla** con esa cadena.

Para que el símbolo $\vdash$ no sea sobreescrito, para toda $p \in Q$ existe $q \in Q$ tal que $\delta(p, \vdash) = (q, \vdash, R)$. Si no la máquina se podría mover hacía la izquierda infinitamente. La estacionariedad de los estados de aceptación q_y y rechazo q_n implican que para toda $b \in \Gamma$, existan $c, c' \in \Gamma$ y $d, d \in \{L, R\}$ tal que

$$\delta(q_y, b) = (q_y, c, d)$$
$$\delta(q_n, b) = (q_n, c', d')$$

Algunos autores representan a la máquina de Turing como una 6-tupla, donde se obvian a los símbolos $\vdash, \sqcup$ y $\Sigma \subseteq \Gamma$.

Definición 2.1.2. Sea Σ un alfabeto y $M = (Q, \Gamma, \delta, q_0, q_y, q_n)$ una MT. Definimos el **lenguaje aceptado** por M como $L(M) = \{w \in \Sigma^\star : M \text{ acepta a } w\} \in \Gamma^\star$.

Definición 2.1.3. Sea M una MT sobre un alfabeto Σ. Definimos una **configuración** (descripción instantánea) de M como un elemento (q, w, n) del conjunto $Q \times \Gamma^\star \times \mathbb{N}$. Donde (q, w, n) significa que la máquina está en el estado q (estado actual) viendo el n-ésimo símbolo en la cadena $w = a_1 a_2 \dots a_n \dots a_N$.

Definición 2.1.4. Sea $(q_1, \vdash w, n)$ una configuración. Tras una acción de la MT se llega a la **siguiente configuración** $(q_2, \vdash w', n \pm 1)$, la cuál tiene un nuevo estado y un nuevo símbolo sobreescrito, y el lector estará situado en el símbolo $n + 1$ o $n - 1$.

Se denota por $(q_1, \vdash w, n) \xrightarrow[M]{1} (q_2, \vdash w', n \pm 1)$, indicando que M, en 1 acción, puede pasar de la primera configuración a la segunda. Note que a lo más, w y w' difieren en un símbolo, el n-ésimo para ser precisos.

Extendemos de forma inductiva esta notación si se pasa de una configuración a otra en más de un paso. Sean α, β configuraciones.

- $\alpha \xrightarrow[M]{0} \alpha$.
- $\alpha \xrightarrow[M]{n+1} \beta$ si $\alpha \xrightarrow[M]{n} \gamma \xrightarrow[M]{1} \beta$, para alguna configuración γ y $n \in \mathbb{N}$.
- $\alpha \xrightarrow[M]{\star} \beta$, si es posible pasar de α a β sin especificar en cuántos pasos.

En las primeros dos puntos se especifica exactamente en cuántas transiciones. En el último punto solo se indica que es posible pasar de una configuración a otra en un número finito de transiciones.

En términos de configuraciones, decimos que $x \in \Sigma^\star$ está en el lenguaje aceptado $L(M)$ si y sólo si $(q_0, \vdash x, 0) \xrightarrow[M]{\star} (q_y, \vdash w, n)$. Análogamente, M rechaza a $x \in \Sigma^\star$ si $(q_0, \vdash x, 0) \xrightarrow[M]{\star} (q_n, \vdash w, n)$.

Un problema común es dado un lenguaje L, definir una máquina de Turing M tal que que $L(M) = L$. A continuación haremos algunos ejemplos partiendo de ese problema.

Ejemplo 2.1.1. Sea un alfabeto $\Sigma = \{a, b, c\}$ y $L = \{a^n b^n c^n : n \geq 0\} \subseteq \{a, b, c\}^\star$. Describimos una MT cuyo lenguaje aceptado sea L. Se propone la siguiente máquina: $M = (Q, \Gamma, \delta, q_o, q_y, q_n)$, $\Sigma = \{a, b, c\}, \Gamma = \Sigma \cup \{\vdash, \sqcup, \dashv\}$, $Q = \{q_0, q_1, \ldots, q_{10}, q_y, q_n\}$, $L = \{a^n b^n c^n : n \geq 0\}$. Finalmente, δ está dada por la siguiente tabla.

	$\vdash$	a	b	c	$\sqcup$	$\dashv$
q_0	$(q_0, \vdash, R)$	(q_0, a, R)	(q_1, b, R)	$(q_n, -, -)$	$(q_3, \dashv, L)$	-
q_1	-	$(q_n, -, -)$	(q_1, b, R)	(q_2, c, R)	$(q_3, \dashv, L)$	-
q_2	-	$(q_n, -, -)$	$(q_n, -, -)$	(q_2, c, R)	$(q_3, \dashv, L)$	-
q_3	$(q_y, -, -)$	$(q_n, -, -)$	$(q_n, -, -)$	$(q_4, \sqcup, L)$	$(q_3, \sqcup, L)$	-
q_4	$(q_n, -, -)$	$(q_n, -, -)$	$(q_5, \sqcup, L)$	(q_4, c, L)	$(q_4, \sqcup, L)$	-
q_5	$(q_n, -, -)$	$(q_6, \sqcup, L)$	(q_5, b, L)	-	$(q_5, \sqcup, L)$	-
q_6	$(q_7, \vdash, R)$	(q_6, a, R)	-	-	$(q_6, \sqcup, L)$	-
q_7	-	$(q_8, \sqcup, R)$	$(q_n, -, -)$	$(q_n, -, -)$	$(q_7, \sqcup, R)$	$(q_y, -, -)$
q_8	-	(q_8, a, R)	$(q_9, \sqcup, R)$	$(q_n, -, -)$	$(q_8, \sqcup, R)$	$(q_n, -, -)$
q_9	-	-	(q_9, b, R)	$(q_{10}, \sqcup, R)$	$(q_9, \sqcup, R)$	$(q_n, -, -)$
q_{10}	-	-	-	(q_{10}, c, R)	$(q_{10}, \sqcup, R)$	$(q_3, \dashv, L)$

Los estados q_0, q_1 y q_2 validan que la cadena sea de la forma $a^{n_1} b^{n_2} c^{n_3}$, si no rechazan. Esta validación se hace de izquierda a derecha. Si se valida se pasa a q_3. Luego q_3,

q_4, q_5 y q_6 consiste en una pasada de derecha a izquierda. Tras finalizar se borra una a, una b y una c, especificamente la primera que leyó respectivamente. Se acepta si no leyó ningun símbolo salvo el $\sqcup$. Si leyó al menos un símbolo a, b o c y no encontró las tres letras entonces se rechaza. Si se valida esta parte se pasa al paso q_7. Luego, q_7, q_8, q_9 y q_{10} hacen lo análogo pero en una pasada de izquierda a derecha. Si la cadena está vacía q_7 lo detecta y acepta, si no, se limpian 3 símbolos y regresamos a q_3 para otra pasada. Consideremos la transición de configuraciones de λ.

$$(q_0, \vdash \sqcup, 0) \underset{M}{\overset{1}{\to}} (q_0, \vdash \sqcup, 1) \underset{M}{\overset{1}{\to}} (q_3, \vdash \dashv, 0) \underset{M}{\overset{1}{\to}} (q_y, \text{-}, \text{-})$$

Ahora a la cadena "abc".

$$(q_0, \vdash abc\sqcup, 0) \underset{M}{\overset{2}{\to}} (q_1, \vdash abc\sqcup, 2) \underset{M}{\overset{2}{\to}} (q_2, \vdash abc\sqcup, 4) \underset{M}{\overset{1}{\to}} (q_3, \vdash abc \dashv, 3) \underset{M}{\overset{3}{\to}}$$
$$(q_6, \vdash \sqcup\sqcup\sqcup \dashv, 0) \underset{M}{\overset{4}{\to}} (q_7, \vdash \sqcup\sqcup\sqcup \dashv, 4) \underset{M}{\overset{1}{\to}} (q_y, \text{-}, \text{-})$$

Ejercicio 6. Sea $\Sigma = \{a, b\}$. Define una MT M tal que $L(M) = \{ww : w \in \Sigma^\star\}$.

Ejercicio 7.

a) Demuestra que el conjunto de MTs es numerable.

b) Demuestra que el conjunto de todos los lenguajes es no numerable

c) Concluir que hay mas problemas que soluciones.

Definición 2.1.5. Una **relación** R de los conjuntos $A_1, A_2, \ldots, A_n$ es un subconjunto del producto cartesiano $A_1 \times A_2 \times \cdots \times A_n$. Decimos que $x_1 R x_2 R \ldots R x_n$ están relacionados si $(x_1, x_2, \ldots, x_n) \in R$.

Recordemos que para que una función esté bien definida, a cada elemento en el dominio corresponde un único elemento en la imagen. Las relaciones sirven para aquellos casos donde queramos que a un elemento x correspondan varias imágenes, por ejemplo y_1 y y_2. Para ello podemos decir que xRy_1, xRy_2 o bien que $(x, y_1), (x, y_2) \in R$.

Las MTs, ahora especificadas como Máquinas de Turing Determínisticas (MTDs), tienen una función δ que define la transición dado un estado y un símbolo. En las *Máquinas de Turing No-Determinísticas* (MTNDs) no existe una función δ sino una relación que indica que dado un símbolo y un estado, existen varias posibilidades de configuraciones siguientes. Sin embargo, este número, aunque posiblemente sea mayor a uno, tiene un número finito de transiciones posibles. Es claro entonces que un caso particular de las MTNDs son las MTDs. En este contexto convendrá enfatizar la distinción con las MDNTs y no escribir simplemente MTs.

Definición 2.1.6. Sea Σ un alfabeto. Una **máquina de Turing no determinística** es una 9-tupla, $N = (Q, \Sigma, \Gamma, R, \vdash, \sqcup, q_0, q_y, q_n)$, definida como en la Definición 2.1.1 de MT, salvo por que aquí la función δ es sustituida por la relación R. Aquí $R \subseteq Q \times \Gamma \times Q \times \Gamma \times \{L, R\}$ es una relación con $|R| < \infty$.

Intuitivamente, los primeros dos argumentos emulan lo que en δ era el dominio y los últimos tres su imagen, sin la restricción de que sea una función. A pesar de esto, es posible denotarla como una función $R : Q \times \Gamma \to 2^{Q\times\Gamma\times\{L,R\}}$. Si $(q, a, s, b, D) \in R$ entonces escribimos $R(q, a) \ni (s, b, D)$. Enfatizamos que $R(q, a)$ no es un elemento de $Q \times \Gamma \times \{L, R\}$, sino un elemento del conjunto potencia de éste. Por ejemplo,

$$R(q, a) = \{(s_1, b_1, D_1), (s_2, b_2, D_2), \ldots, (s_r, b_r, D_r)\} \subseteq Q \times \Gamma \times \{L, R\}$$

El lenguaje de aceptación $L(N)$ para la MTND N y las configuraciones se define de la misma forma que para MTDs. De hecho ambos tienen el mismo poder de computo.

Teorema 2.1.1. *Sea $L \subseteq \Sigma^\star$ un lenguaje. Luego $L(M) = L$, para una MDT M, si y sólo si existe una MTND N, tal que $L(N) = L$.*

Sea Σ un alfabeto y $f : \Sigma^\star \to \Sigma^\star$ una función de cadenas. Una Máquina de Turing que calcula f es una 6-tupla $M = (Q, \Gamma, \delta, q_0, q_h, q_e)$ donde Q, q_0, Γ y δ son como en la definición 2.1.1. $q_h \in Q$ es estado de detención y $q_e \in Q$ es estado de "error".

Ejemplo 2.1.2. Una máquina de Turing que suma dos números naturales. Codificar a $\mathbb{N}$ en $\Sigma = \{0, 1\}$. Decimos, $f : A \subseteq \{1^n01^m : n, m \in \mathbb{N}\} \to \Sigma^\star, f(1^n01^m) = 1^{n+m}$. Formalmente, $M = (Q, \Gamma, \delta, q_0, q_h, q_e), Q = \{q_0, q_1, q_2, q_3, q_4, toq_h, roq_e)\}$ donde δ está dada como sigue.

δ	0	1	$\sqcup$
s	$(r, \cdot, \cdot)$	$(q_1, 1, R)$	$(r, \cdot, \cdot)$
q_1	$(q_2, 1, R)$	$(q_1, 1, R)$	$(r, \cdot, \cdot)$
q_2	$(r, \cdot, \cdot)$	$(q_2, 1, R)$	$(q_3, \sqcup, L)$
q_3	$\cdot$	$(t, \sqcup, R)$	$\cdot$
t	$\cdot$	$\cdot$	$\cdot$
r	$\cdot$	$\cdot$	$\cdot$

Tesis de Church-Turing. Todo problema que puede ser resuelto por un procedimiento de un número *polinomial* de pasos puede ser resuelto por una MT de tiempo polinomial.

Definición 2.1.7. Es posible que una MT U simule paso por paso el comportamiento de cualquier otra MT, $M = (Q, \Gamma, \delta, q_0, q_y, q_n)$.[2] A ésta le llamaremos **Máquina Universal (MUT)**. El alfabeto de la máquina universal U es $\Sigma = \{0, 1, \#\}$. Recibe como argumento una cadena $M\#x$ donde $\#$ es un símbolo separador. La primera parte es una *descripción codificada* de la MT M: el número de estados, símbolos, función de transición, etc. La x es el input de M que simula U. Tanto la x como la M están codificadas en el alfabeto $\{0, 1\}$. Si M acepta o rechaza a x entonces la simulación de U termina y se puede proseguir a otra rutina, posiblemente basada en el resultado. Sin embargo, si M se queda en ciclo entonces U también. La cadena codificada se denota $M\#x$ por obvias razones.

[2] Aquí estamos obviando 3 de los argumentos que definen una MT, pero es por mera simplificación de la notación, no es una notación rigurosa.

El proceso para la descripción codificada de la MT al alfabeto $\{0,1\}$ es el siguiente. Sean Q, Γ, δ de una MT. Sin perdida de generalidad, $Q = \{q_1, q_2, \ldots, q_n\} \mapsto \{1, 2, \ldots, n\}$. Análogamente, $\Gamma = \{1, 2, \ldots, m\}, \{R, L\} \mapsto \{0, 1\}$. El símbolo $\sqcup$ equivale al símbolo 1. El inicio de la codificación es $1^n 01^m 0 \ldots$, indicando a n y m. Finalmente, si $\delta(q_1, a_2) = (q_2, a_4, R)$ entonces $\delta(1, 2) = (2, 4, 1) \mapsto 10110110111101$ (no es binario, es un alfabeto ineficiente donde los 0's son separadores).

El lenguaje $L(U)$ de la máquina universal de Turing U puede ser definida así

$$L(U) = \{M\#x \in \{0, 1, \#\}^\star : M \text{ acepta a } x\}.$$

La máquina universal recibe el *input* $M\#x$, y verifica la primera parte para saber que M tiene sentido. Si no, se rechaza y lo mismo para x. Después procede a hacer la simulación paso por paso. La cinta en U se parte en 3. En la primera parte está la configuración de la máquina M. En la segunda, está x codificado. En la tercer está, sirve para recordar el estado y posición de M, pues no necesariamente son los mismos que los de U.

Ejercicio 8. Construir "grosso modo" una máquina de Turing que simule MTs que calculan funciones de cadena, $f : A \to \Sigma^\star$, $A \subseteq \Sigma^\star$.

2.2. Lenguajes computables enumerablemente y computables

Definición 2.2.1. Sea Σ un alfabeto y M una máquina de Turing sobre Σ.

a) Se dice que M es **total** si y sólo si para todo $x \in \Sigma^\star$, M se detiene en x, es decir, M se acepta o M rechaza a x, pero jamás se queda en ciclo.

b) Se dice que el lenguaje $L \subseteq \Sigma^\star$ es **computable enumerablemente (recursivo enumerablemente**) si existe una maquina de Turing M tal que $L = L(M)$. Cualquier elemento del lenguaje se va a aceptar.

c) **Co-recursivo enumerable** si su alfabeto complemento es recursivo enumerable, i.e. $\Sigma - L$ es recursivo enumerablemente.

d) Se dice que el lenguaje $L \subseteq \Sigma^\star$ es **computable (recursivo)** si existe una máquina de Turing *total* tal que $L = L(M)$.

En el recursivo numerable los elementos en el lenguaje L se aceptan, pero los que que están en $\Sigma^\star - L$ quiza se rechazan o quizá se enciclan. En cambio, en el lenguaje recursivo estos elementos jamas se enciclan, sólo se rechazan.

Definición 2.2.2. Una propiedad lógica P es **decidible** si y sólo si $\{x \in \Sigma^\star : P(x)\}$ es computable. Una propiedad P es **semidecidible** si y sólo si $\{x \in \Sigma^\star : P(x)\}$ es computable enumerablemente. Se dice que P es **no trivial** si no es universalmente cierta o universalmente falsa. Algunas $x \in \Sigma^\star$ la cumplen y otras que no.

Ejemplo 2.2.1.

a) $A = \{w \in \{a,b\}^\star : w = a^n b^b\}$ es computable.

b) $P = \{a^p \in \{a\}^\star : p \text{ es primo}\}$ es computable.[3]

c) Todo conjunto regular (generado por expresiones regulares) es computable.

d) Todo conjunto computable es computable enumerablemente mas no el converso y se probará más adelenta como corolario.

Definición 2.2.3. Sea Σ un alfabeto.

a) Sea $\mathcal{CE} = \{A \subseteq \Sigma^\star : A \text{ es computable enumerablemente}\}$.

b) Sea $\mathcal{C} = \{A \subseteq \Sigma^\star : A \text{ es computable}\}$.

Propiedades de los conjuntos recién definidos

i) $\mathcal{CE}, \mathcal{C} \subseteq 2^{\Sigma^\star}$.

ii) $\mathcal{CE}, \mathcal{C} \neq \emptyset$.

iii) $\mathcal{C} \subseteq \mathcal{CE}$.

Para contestar esta pregunta tomamos dos lenguajes para la máquina universal de Turing U. El conjunto $HP = \{M\#x : M \text{ se detiene en } x\}$ conocido como "El problema de detención", y el lenguaje $MP = \{M\#x : x \in L(M)\}$, conocido como "El problema de la membresia".

Ejercicio 9. Demostrar que $HP, MP \in \mathcal{CE}$.

2.3. Imposibilidad del problema de detención

Un teorema muy importante es que $\mathcal{C} \subsetneq \mathcal{CE}$. Es decir, que existen lenguajes que son decidibles enumerablemente pero no decidibles. En esta sección probamos este teorema. Es conveniente familizarizarse con las pruebas de la siguientes proposiciones para entender mejor el argumento de la prueba del teorema.

Proposición 2.3.1. No existe $f : \mathbb{N} \to 2^{\mathbb{N}}$ biyectiva.

Demostración. Supongamos que si existe f biyectiva entre $\mathbb{N}$ y $2^{\mathbb{N}}$. Formemos una matriz la cual tiene sus columnas indexadas por $\mathbb{N}$ y los renglones por conjuntos $f(0), f(1)...$ Es decir por todos los subconjuntos de $\mathbb{N}$. Llenamos los espacios de la matriz de la siguiente manera. En la posición $(f(k), j)$ colocamos un 1 e si $j \in f(k)$ y 0 si $j \notin f(k)$. Si el natural pertenece o no pertenece al conjunto.

[3]Existe un algoritmo polinomial para saber si un número es primo de 2004.

	1	2	3	4	5	...
f(1)	0	1	1	0	1	...
f(2)	1	1	1	1	1	...
f(3)	0	0	0	1	1	...
f(4)	1	0	0	0	0	...
⋮	⋮	⋮	⋮	⋮	⋮	⋱

Como f es sobre, todo subconjunto de $\mathbb{N}$ está representado por un renglon de la matriz. Sin embargo, consideremos el conjunto $B \subseteq \mathbb{N}$ definido como sigue. Tome la diagonal de la matriz y de esta cadena el complemento. En este caso la diagonales "0100..." y su complemento es "1011...". Esa cadena representa a un subconjunto válido en $2^{\mathbb{N}}$. Sin embargo, no puede estar representado por algún renglón de la matriz porque cada renglon difiere en al menos un elemento de B. Difiere del conjunto $f(k)$ justamente en el elemento k. Por lo tanto el conjunto B no está representado en la matriz. Esto es contradictorio. Por lo tanto no existe f sobre. ■

Ejercicio 10. Probar que no existe $f : A \to 2^A$ biyectiva para cualquier conjunto A.

Teorema 2.3.1. *$HP \notin \mathcal{C}$.*

Demostración. Usaremos como motivación las proposiciones anteriores para esta prueba. Para $x \in \{0,1\}^\star$ consideramos la máquina de Turing M_x con alfabeto $\{0,1\}$, la cual viene descrita (y a su vez identificada por la cadena x). De esta froma obtenemos una lista: $M_\lambda, M_0, M_1, M_{00}, M_{10}, M_{11}, M_{100}, ...$ De todas las posibles máquinas de Turing con alfabeto en $\{0,1\}$, Consideremos una matriz infinita, la cual tiene como renglones a las máquinas de la lista anterior y las columnas son cadenas en $\{0,1\}^\star$. En el renglón i, columna j, obtenemos si la máquina M_x se detiene (H) o se encicla (L) bajo el input j.

	λ	0	1	00	01	10	11	000	...
M_λ	H	L	L	H	H	L	H	L	...
M_0	L	L	H	H	L	H	H	L	...
M_1	L	L	H	H	L	H	H	L	...
M_{00}	L	H	H	L	L	L	H	H	...
M_{01}	H	L	H	L	L	H	L	L	...
M_{10}	H	L	H	H	H	H	H	H	...
M_{11}	L	L	H	L	L	H	L	L	...
M_{000}	H	H	H	L	L	H	H	L	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋱

Por contradicción supongamos que $HP \in \mathcal{C}$, es decir que es computable. Entonces existe una máquina de turing total $\mathcal{K}$ tal que $L(\mathcal{K}) = HP$. Esto es que $\mathcal{K}(M\#x)$ acepta si M se detiene en x y $\mathcal{K}(M\#x)$ rechaza si M se encicla en x.

Construimos una Máquina de Turing $\mathcal{N}$, que en la entrada $x \in \{0,1\}^\star$:

1) Se construye M_x a partir de x.

2) Escribe la cadena $M_x\#x$ en su cinta.

3) Simula a $\mathcal{K}$ en la entrada $M_x\#x$.

4)

$$\mathcal{N}(x) = \begin{cases} \text{Acepta} & \text{si } \mathcal{K}(M_x\#x) \text{ rechaza.} \\ \text{Encicla} & \text{si } \mathcal{K}(M_x\#x) \text{ acepta.} \end{cases}$$

Es como seguir un comportamiento contrario.

Entonces el comportamiento de $\mathcal{N}$ en $x \in \{0,1\}^\star$ en es siguiente. $\mathcal{N}$ se detiene en x si y solo si $\mathcal{K}$ rechaza a $M_x\#x$ si y sólo si M_x se encicla en x. La pregunta para llegar a la contradicción. ¿Qué renglón de la matriz corresponde a $\mathcal{N}$? Usando la autoreferencia. Si $\mathcal{N}$ se detiene en $x_\mathcal{N}$ entonces $\mathcal{N}\#x_\mathcal{N} \in HP = L(\mathcal{K})$, entonces $\mathcal{K}$ acepta a $x_\mathcal{N}$, entonces $\mathcal{N}$ se encicla con $x_\mathcal{N}$. La otra contradicción es análoga en el caso que $\mathcal{N}$ no se detiene en $x_\mathcal{N}$. Por lo tanto $\mathcal{N}$ no existe, pero esto solo puede ocurrir si $\mathcal{K}$ tampoco existe. Por lo tanto HP no es computable. ■

Corolario 2.3.1. $\mathcal{C} \subsetneq \mathcal{CE}$

Ejercicio 11. Probar que el siguiente conjunto está en $\mathcal{C}$

$$HP_T = \{M\#x\#t : M \text{ se detiene en } x \text{ a lo más en } t \text{ pasos}\}$$

Ejercicio 12. Probar que A es computable si y sólo si A es computable enumrablemente y A y $\Sigma^\star - A$ es computable enumerablemente. Concluir que $HP^c \notin \mathcal{CE}$.

2.4. Reducciones Computables

Definición 2.4.1. Sea $\sigma : \{0,1\}^\star \to \{0,1\}^\star$ una función. Decimos que σ es **efectivamente computable** si existe una MT que recibe un input x y se detiene cuando en su cinta esta el resultado $\sigma(x)$.

Definición 2.4.2. Decimos que σ es una **función computable** si es efectivamente computable y la MT M que computa a σ es total.

Definición 2.4.3. Sean Σ_1, Σ_2 alfabetos y $A \subseteq \Sigma_1^\star, B \subseteq \Sigma_2^\star$ subconjuntos. Una **reducción** (muchos a uno) de A en B es una función computable $\sigma : \Sigma_1^\star \to \Sigma_2^\star$ tal que $\forall x \in \Sigma^\star$:

$$x \in A \Leftrightarrow \sigma(x) \in B.$$

Si existe la reducción, decimos que A se **reduce** a B, denotado por $A \leq_m B$, la m por el *many to one*.

Ejercicio 13. Probar que "$\leq_m$" es reflexiva y transitiva.

Teorema 2.4.1. *Sean Σ, Δ alfabetos, $A \subseteq \Sigma^\star, B \subseteq \Delta^\star$ conjuntos y supongamos que $A \leq_m B$. Entonces*

a) Si $B \in \mathcal{CE}$ entonces $A \in \mathcal{CE}$.

b) Si $B \in \mathcal{C}$ entonces $A \in \mathcal{C}$.

Demostración.
a) Por demostrar que $A \in \mathcal{CE}$, es decir que existe una MT M_A tal que $L(M_A) = A$. Por hipótesis $A \leq_m B$ entonces existe $\sigma : \Sigma^\star \to \Delta^\star$ computable tal que $\forall x \in \Sigma^\star$ se sigue que $x \in A \Leftrightarrow \sigma(x) \in B$. Como $B \in \mathcal{CE}$ entonces existe una MT M_B tal que $L(M_B) = B$. Sea M_A una MT tal que en la entrada $x \in \Sigma^\star$ hace lo siguiente:

1) Calcula $\sigma(x)$.

2) Ejecuta a $M_B(\sigma(x))$.

3) M_A Acepta a x si M_B acepta a y.

Como $x \in L(M_A) \Leftrightarrow \sigma(x) \in L(M_B) = B \Leftrightarrow x \in A$, se concluye que $L(M) = A$.

Hasta aquí hemos probado la *correctez* de M. Resta probar que todos los pasos, 1), 2) y 3) se hacen en tiempo polinomial si $x \in A$. El 1) es porque σ es computable. El 2) es porque M_B se hace en tiempo polinomial si $\sigma(x) \in L(M_B)$. El 3) es porque M_A termina poco después de la simulación. Por lo tanto $A \in \mathcal{CE}$.

b) Considere la prueba anterior. Note que en el paso 2) si M_B se encicla con $\sigma(x)$ entonces M_A también. Note que es en este paso el único donde M_A se puede enciclar. Sin embargo, ahora M_B es total, es decir, ya no se puede enciclar. Por lo tanto M_A tampoco en ninguno de sus pasos y se concluye que M_A es total. Con el resto de los argumentos idénticos se concluye que $A \in \mathcal{C}$.

Otra prueba de b) sería como sigue. Por el Ejercicio 12 se sigue que como $B \in \mathcal{C}$ entonces $\Sigma^\star - B$ está en $\mathcal{CE}$. Dada la misma σ del inciso a) se sigue que $x \in \Sigma^\star - A \Leftrightarrow \sigma(x) \in \Sigma^\star - B$. Esto prueba que $\Sigma^\star - A \leq_m \Sigma^\star - B$. Por el inciso a) aplicado dos veces, para A y su complemento, se sigue que $A \in \mathcal{CE}$ y que $\Sigma^\star - A \in \mathcal{CE}$. Nuevamente, por el Ejercicio 12 se sigue que $A \in \mathcal{C}$. ■

Ahora mostraremos cómo usar el Teorema 2.4.1. Por ejemplo, en el Ejercicio 12 concluimos que en particular $HP^c \notin \mathcal{CE}$. Sea $A = HP^c$. Por contrapositiva, si existe un problema B tal que $A \leq_m B$ entonces se sigue que $B \notin \mathcal{CE}$. Podemos usar esa estrategia para probar el siguiente resultado.

Resultado 2.4.1. El conjunto $FIN = \{N : |L(N)| < \infty\} \notin \mathcal{CE}$, con $FIN \subseteq \Sigma^\star$. Es decir, la N denota la codificación de la MT N, no N en sí.

Demostración. Recordemos que $HP^c = \{M\#x : M \text{ no se detiene en } x\} \notin \mathcal{CE}$. Basta dar una reducción $HP^c \leq_m FIN$. Esto es una función $\sigma : \Sigma^\star \to \Sigma^\star$ tal que $M\#x \in$

HP^c si y sólo si $\sigma(M\#x) \in FIN$. En otras palabras, dado $M\#x$ construimos una MT $N = \sigma(M\#x)$ tal que M se cicla en x si y sólo si $|L(N)| < \infty$.

Si la cadena $M\#x$ es invalida σ se pone en estado de error. Supongamos que la cadena es válida. Sea $N = \sigma(M\#x)$ una MT tal que en la entrada $y \in \Sigma^\star$ (1) Borra la entrada y. (2) Escribe a x en la cinta. (3) Ejecuta a M en x. (4) Si M se detiene en x entonces N acepta a y. Se sigue que σ es una reducción computable y total, donde $\sigma(N)$ satisface lo siguiente:

$$L(N) = \begin{cases} \Sigma^\star & \text{si } M\#x \in HP \\ \emptyset & \text{si } M\#x \in HP^c \end{cases}$$

Luego, $M\#x \in HP^c \Leftrightarrow L(N) = \emptyset \Leftrightarrow |L(N)| = 0 < \infty \Leftrightarrow N = \sigma(M\#x) \in FIN$. Por lo tanto $HP^c \leq_m FIN$. Por el Teorema 2.4.1 se concluye que $FIN \notin \mathcal{CE}$. ■

Resultado 2.4.2. El conjunto $FIN^c = \{N : |L(N)| = \infty\} \notin \mathcal{CE}$.

Demostración. Probaremos que $HP^c \leq_m FIN^c$. Por el Teorema 2.4.1 se seguirá que $FIN^c \notin \mathcal{CE}$. Una reducción muy similar a la anterior funcionará para probar que $HP^c \leq_m FIN^c$. Se tiene entonces una reducción $\sigma : \Sigma^\star \to \Sigma^\star$ tal que $x \in HP \Leftrightarrow \sigma(x) \in FIN$. Esto es, dado $M\#x \in HP \Rightarrow \sigma(M\#x) = N$ satisface que $|L(N)| < \infty$.

Se construye la MT N como sigue: 1) Guarda a x en su cinta. 2) Guarda el *input* y en una segunda cinta. 3) Simula a x en M durante $|y|$ pasos. Por cada paso que simula en x borra un elemento en y, es decir lo sustituye por $\sqcup$. Finalmente, N acepta si y sólo si M no se ha detenido tras y pasos.

Resta demostrar que σ funciona. Si $M\#x \in HP^c$, entonces M no se detiene en x jamás. Luego, para toda y la MT N acepta, se sigue que $|L(N)| = \infty$. Se sigue que $\sigma(M\#x) \in FIN^c$. Ahora probamos el converso. Sea $M\#x \in HP$, entonces M se detiene en x, digamos tras exactamente k pasos. Si $|y| < k$, entonces la simulación de M continúa tras y pasos y por lo tanto N acepta a y. Como existe un número finito de cadenas y tal que satisfacen que $|y| < k$, se sigue que $|L(N)| < \infty$. Se tiene entonces que $M\#x \in HP \Rightarrow |L(N)| < \infty \Rightarrow N \in FIN$. Concluimos que $M\#x \in HP \Leftrightarrow N \in FIN$. ■

Resultado 2.4.3. Probar que $MP = \{N\#y : y \in L(N)\} \notin \mathcal{C}$.

Demostración. Mostraremos que $HP \leq_m MP$ y usaremos el hecho que $HP \notin \mathcal{C}$. Sea una reducción $\sigma : \Sigma^\star \to \Sigma^\star$ tal que $M\#x \in HP$ si y sólo si $\sigma(M\#x) = N\#y \in MP$. Esto es, M se detiene en $x \Leftrightarrow y \in L(N)$.

A partir de $M\#x$, se define N idéntico a M, salvo que si M se detiene en x entonces N acepta a x. Hacemos $y = x$ y $\sigma(M\#x) = N\#x$. Como $M\#x \in HP \Leftrightarrow M$ se detiene en $x \Leftrightarrow N$ acepta a $y \Leftrightarrow y \in L(N) \Leftrightarrow N\#y \in MP$. ■

Ejercicio 14. Sea $B = \{M : \lambda \in L(M)\}$, es decir las MT que aceptan a la cadena vacía. Probar que $B \notin \mathcal{C}$.

Teorema 2.4.2. ***Teoema de Rice****. Toda propiedad no trivial de los conjuntos c.e. es no computable.*

Ejercicio 15. Demuestre los incisos.

a) $A = \{M : M \text{ tiene al menos 481 estados}\} \in \mathcal{C}$.

b) $B = \{N : L(N) = \Sigma^\star\} \notin \mathcal{C}$.

3. Clases de Complejidad

3.1. Notación asintótica y funciones de complejidad apropiadas

Esta sección es dentro del contexto $\mathcal{C}$. Sea el alfabeto $\Sigma = \{0, 1\}$. La instancia de un problema se puede representar como una cadena $x \in \Sigma^\star$. Se define el tamaño de la instancia como $n := |x|$. Para medir el tiempo y espacio usado por un algoritmo MT empleamos funciones "funciones de complejidad apropidas" que dependen de n.

Definición 3.1.1. Una función $f : \mathbb{N} \to \mathbb{N}$ es una **función de complejidad apropiada (fca)** si

1. Es positiva, $f > 0$.
2. Es creciente, $f(n) \leq f(n+1), \forall n \in \mathbb{N}$
3. Existe una MTD M_f que calcula el valor $f(n)$ en un número de pasos proporcional al valor $f(n)$.

Definición 3.1.2. El **tiempo** que tarda una MT M en decidir su una cadena $x \in \Sigma^\star$ está o no en $L(M)$ es el transiciones que M ejecuta empezando en su configuración inicial $(q_0 \vdash x)$ para llegar a una configuración de aceptación o rechazo. Diremos que la función $f : \mathbb{N} \to \mathbb{N}$ es **una cota superior (inferior) para el tiempo de ejecución** de M si en el peor de los casos de ejecución, el tiempo que tarda M está acotado superiormente (inferiormente) por f.

Definición 3.1.3. Nuestro modelo usual de MT cambia en este contexto. La cinta *input* se considera *read-only* y se tiene una segunda cinta *write-only* para el output. El **espacio** consumido por una MT M en decidir a $x \in \Sigma^\star$, es el número máximo de celdas usadas por M sobre la cinta de *output* para llegar a alguna de las configuraciones de detención partiendo de su configuración inicial.

Ejemplo 3.1.1. Para la MTD que acepta el lenguaje $\{a^n b^n c^n : n \leq 0\}$:

1. M acepta a λ en dos pasos: $\vdash \sqcup \to \vdash\dashv$, pues recuerde que

$$(q_0, \vdash, 0) \xrightarrow[M]{2} (q_y, \vdash\dashv, \text{-})$$

2. M acepta a abc en 9 pasos, ya que

$$(q_0, \vdash abc, 0) \xrightarrow[M]{5} (q_0, \vdash abc \dashv, 3) \xrightarrow[M]{4} (q_y, \text{-}, \text{-})$$

3. Para $aabbcc$ hacemos 22 pasos:

$$(q_0, \vdash aabbcc, 0) \xrightarrow[M]{8} (q_0, \vdash aabbcc \dashv, 6) \xrightarrow[M]{7} (q., \vdash a \sqcup b \sqcup c \sqcup \dashv, 1) \xrightarrow[M]{7} (q_y, \text{-}, \text{-})$$

4. En general, para $a^n b^n c^n$ hacemos $(n+1)(3n+1) + 1 = 3n^2 + 4n + 2$.

Por lo tanto el tiempo que tarda en decidir si $a^n b^n c^n$ está en L el tiempo total de ejecución es: $f(n) = 3n^2 + 4n + 2$. Para las cadenas que no están en L son rechazadas en un número menor tiempo. Por lo tanto $f(n)$ nos da la medición del peor caso y es una cota superior. Con respecto al espacio, M replica al input de longitud $3n$ y usa a al símbolo $\vdash$ y $\dashv$. Por lo tanto el espacio ocupado por M está acotado por $s(n) = 3n + 2$.

Ejemplo 3.1.2. Calculemos el tiempo y el espacio usados por la MT que suma números naturales. En el caso de elementos del dominio: $S(1^n 01^m) = 1^{n+m}$. Por ejemplo, si recibe la cadena $1^3 01^4$, la MT hace

$$(q_0, \vdash 1^3 01^4 \sqcup, 0) \overset{11}{\underset{M}{\to}} (q_0, \vdash 1^7 0 \dashv, 0).$$

Recorrer todos los símbolos implica $(1 + n + 1 + m + 1)$ movimientos. Se requiere un paso adicional para convertir el último 1 a $\sqcup$. Por lo tanto el tiempo de ejecución de M para cadenas en el dominio de S está acotado por $T(n, m) = n + m + 4$. En términos del espacio, fácilmente vemos que el espacio ocupador por M está acotado por $R(n, m) = n + m + 2$, no escribimos ni el 0, ni $\dashv$.

Introducimos ahora ahora el concepto de notación asintótica, donde se simplificará el análisis de tiempo y la memoria pues bastará clasificar el algoritmo de acuerdo a un orden de complejidad.

Definición 3.1.4. Sea $g : \mathbb{N} \to \mathbb{N}$ una función de complejidad apropiada.

a) Se dice que $g(n)$ es una cota asintótica de $f(n)$ si $\Theta(g(n)) = \{f : \mathbb{N} \to \mathbb{N} : \exists n_0 \in \mathbb{N}, c_1, c_2 \in \mathbb{R}^+, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}$.

b) Se dice que $g(n)$ es una cota superior asintótica de $f(n)$ si $O(g(n)) = \{f : \mathbb{N} \to \mathbb{N}, \exists n_0 \in \mathbb{N}, c \in \mathbb{R}^+ : 0 \leq f(n) \leq cg(n), \forall n \geq n_0\}$.

c) Se dice que $g(n)$ es una cota inferior asintótica de $f(n)$ si $\Omega(g(n)) = \{f : \mathbb{N} \to \mathbb{N}, \exists n_0 \in \mathbb{N}, c \in \mathbb{R}^+ : 0 \leq cg(n) \leq f(n), \forall n \geq n_0\}$.

Ejercicio 16.

a) Para toda función $g : \mathbb{N} \to \mathbb{N}$, los conjuntos $\Theta(g(n)), O(g(n)), \Omega(g(n)) \neq \emptyset$.

b) Para todo g, f funciones de computación apropiadas. $f(n) \in \Theta(g(n)) \Leftrightarrow f(n) \in O(g(n))$ y $f(n) \in \Omega(g(n))$.

Notación: Se denota $f(n) = \Theta(g(n))$ en lugar de $f(n) \in \Theta(g(n))$. Por ejemplo $\log n = O(n^2)$.

Ejemplo 3.1.3.

a) Para el Ejemplo 3.1.1, la función de complejidad y espacio eran $f(n) = 3n^2 + 4n + 2$ y $s(n) = 3n + 2$, respectivamente. Se sigue que $f(n) = \Theta(n^2)$ y $s(n) = \Theta(n)$.

b) Para el Ejemplo 3.1.2, la función de complejidad era $n + m + 4$. Si $p = n + m$, entonces $f(p) = \Theta(p)$. Para el espacio $s(p) = \Theta(p)$, también.

Ejemplo 3.1.4. Demostrar que si $f(n) = \frac{1}{2}n^2 - 3n$ entonces $f(n) = \Theta(n^2)$.

Demostración. Demostrar que existe $n_0 \in \mathbb{N}$ y $c_1, c_2 \in \mathbb{R}^+$ tal que

$$0 < c_1 n^2 \leq \tfrac{1}{2}n^2 - 3n \leq c_2 n^2, \forall n \geq n_0$$

Note que $f > 0$ si $n \geq 7$. Proponemos $n_0 = 7$. Trivialmente, $f(n) \leq \frac{1}{2}n^2$, así que proponemos $c_2 = \frac{1}{2}$. Para c_1 considere la desigualdad $c_1 \leq \frac{1}{2} - \frac{3}{n}$. Si evaluamos el segundo miembro en n_0 obtenemos que $c_1 \leq \frac{4}{14}$. En particular $c_1 = \frac{1}{14}$ funciona. ■

Ejemplo 3.1.5. Probar que $6n^3 \neq \Theta(n^2)$. Por contradicción, supongamos que $6n^3 = \Theta(n^2)$, entonces $\exists n_0 \in \mathbb{N}, c_1, c_2 \in \mathbb{R}^+$, tal que $0 \leq c_1 n^2 \leq n^3 \leq c_2 n^2, \forall n \geq n_0$. Luego $6n^3 \leq c_2 n^2 \Rightarrow 6n \leq c_2, \forall n \geq n_0$. Pero esta desigualdad no puede ser cierto para ninguna $n_0 \in \mathbb{N}$. Por lo tanto $6n^3 \neq \Theta(n^2)$. En particular, $6n^3 \neq O(n^2)$.

Ejemplo 3.1.6. Todo polinomio de la forma $p(n) = \sum_{i=1}^{n} a_i x^i$ cumple con $p(n) = \Theta(x^d)$, si $a_d > 0$.

Ejercicio 17. Probar los incisos:

a) $n^3 \neq \Theta(n^4)$.

b) $\log_2 n = O(n)$.

c) máx$\{f(n), g(n)\} = \Theta(f(n) + g(n)$.

Ejercicio 18. Probar los incisos:

a) Si $f(n) = O(g(n))$ y $g(n) = O(h(n))$ entonces $f(n) = O(h(n))$.

b) $f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$.

Ejercicio 19. Probar los incisos:

a) ¿$2^{n+1} = O(2^n)$?

b) ¿$2^{2n} = O(2^n)$?

c) ¿$3^n = O(2^{n^2})$?

Noción de uso eficiente de los recursos (tiempo y espacio)

$f(n) \setminus n$	10	50	100	1000
$\log(n)$	3	5	6	9
n	10	50	100	10^3
n^2	100	2.5×10^3	10^5	10^6
n^3	10^3	1.25×10^5	10^6	10^9
2^n	10^3	1.1×10^{16}	12.68×10^{29}	10.7×10^{223}

La noción de eficiencia en el uso del tiempo es la polinomial. En el espacio usualmente la noción de eficiencia es la "polilogaritmica", es decir, el algoritmo hace un uso eficiente del espacio si este esta acotado por una función de la forma $c \log^k(n)$.

Problema de Satisfacibilidad booleana

Definición 3.1.5. Sea $U = \{u_1, u_2, \ldots, u_m\}$ un conjunto de variables. Una **asignación de verdad** para U es una función $t : U \to \{V, F\}$. Si $t(u) = V$ decimos que u es "verdadero" bajo t, sino diremos que es "falso" bajo t.

Definición 3.1.6. Si $u \in U$ entonces u y $\bar{u}$ son **literales** sobre U. La variable u y el literal u se denotan igual. El literal u es verdadero bajo t si y sólo si u es verdadero bajo t. El literal $\bar{u}$ es verdadero bajo t si y sólo si u es falsa bajo t.

Definición 3.1.7. La **cláusula** C sobre U es un conjunto de literales sobre U, denotado $(x_1 \vee x_2 \vee \cdots \vee x_n)$. Se dice que C se **satisface** bajo t si y sólo si al menos uno de sus elementos es verdadero bajo t.

Por ejemplo $C = (x_1 \vee x_2)$ se satisface bajo t, si $t(x_1) = V$ y $t(x_2) = F$, pues el literal x_1 en C es verdadero bajo t.

Definición 3.1.8. Una fórmula booleana $\phi = \phi(x_1, x_2, \ldots, x_n)$ está en **Fórmula normal conjuntiva** (FNC). Si ϕ se puede describir como una colección de m disyunciones ($\vee$) conectadas como una gran conjunción $\wedge$.

Ejemplo 3.1.7.

- $(x_1 \vee x_3) \wedge (\bar{x_4} \vee x_2)$ está en FNC.
- $x_1 \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_3 \vee x_1)$ está en FNC
- $x_1 \wedge (\bar{x}_1 \vee (x_3 \wedge x_4))$ no está en FNC.

Una fórmula ϕ en FNC es satisfacilble si existe una asignación de verdadero (T) o falso (F) para las variables de ϕ tal que la fórmula ϕ es verdadera bajo esa asignación.

Teniendo la codificación, definimos

$$\text{SAT} = \{x_1, x_2, \ldots, x_n : n \leq 1 \text{ y } \phi \text{ es satisfacible}\},$$

donde la cadena $x_1, x_2, \ldots, x_n$ es una cadena en $\{0, 1\}^\star$. El orden es de $O(n, m)$ donde n es variables y m es clausulas.

Dada una cadena $x \in \Sigma^\star$, la máquina primera verifica que x representa una fórmula ϕ_x en FNC. Si no es válida, rechaza de inmediato. Si la cadena es una fórmula ϕ_x booleana en FNC. entonces la máquina ejecuta el siguiente ciclo. Para cada posible asignación de verdad de las variables $x_1, x_2, \ldots, x_n$ de ϕ_x, prueba si la asignación satisface ϕ_x. En el caso de que ϕ_x sea satisfacible bajo la asignación, entra en el estado de aceptación. En otro caso continúa el ciclo con la siguiente asignación de verdad para $x_1, x_2, \ldots, x_n$. Aquí termina el ciclo. Si el ciclo termina y ϕ_x no se satisfizo por alguna asignación de verdad, entonces rechaza.

La complejidad de este algoritmo es $O(nm+2^n(n+nm))$. El primer nm corresponde a parsear el input. El 2^n corresponde a cada combinación, dentro de ella, n corresponde

a una asignación y nm corresponde a verificar si esa solución funciona. Se puede probar que $O(nm2^n) = O(2^n)$.

Caso no determinístico
Dado $x \in \Sigma^\star$ tal que x representa a una fórmula en FNC ϕ_x. La máquina "adivina" una posible asignación de verdad para $x_1, x_2, \ldots x_n$ y prueba si ésta satisface a ϕ_x, aceptando si este es el caso y rechazando en otro caso en tiempo polinomial.

Ejercicio 20. ¿Hay una fórmula mecánica eficiente de reconocer fórmulas en FNC satsifacibles?

Ejercicio 21. Dar una codificación de fórmulas FNC en el alfabeto $\{0, 1\}$.

Problema de Coloración

Sea $G = (V, E)$ una gráfica no dirigida y $|V| = n$. Una coloración de los vértices de G con k colores es una función $c : V \to \{1, 2..., k\}$ tal que cumple la siguiente condición: para toda $uv \in E \Rightarrow c(u) \neq c(v)$. Una gráfica G es k-colorable si G tiene una coloración de k colores.

Definición 3.1.9. Se dice que la gráfica $G = (V, E)$ es **completa** si para todo $u, v \in V$ existe $uv \in E$. Se denota la gráfica k_n la gráfica completa de n vértices.

Ejemplo 3.1.8.

- La gráfica k_3 es 3-colorable pero no 2-colorable.
- Los arboles, es decir gráficas conexas acíclicas, son 2-colorables (no se va a probar).

Nos interesa si el conjunto k-colouring $= \{G \in \{0, 1\}^\star : G$ tiene una k-coloración$\}$ es computable. Se analiza la complejidad de ejecutar todas las combinaciones de colores.

Ejercicio 22. Dar un algoritmo determinístico para 2-colouring y 3-colouring. ¿Son de tiempo polinomial?

Ejercicio 23. (Extra +0.5) Dar un algoritmo determinístico polinomial para 2-colouring. Analizarlo para explicar por qué es polinomial.

Problemas de números

Dado un conjunto de números $N = \{0, 1, \ldots, n\} \subseteq \mathbb{N}$, cada entero $i \in N$ tiene un valor v_i y un peso w_i asociado, con $v_i, w_i \in \mathbb{R}$. Se nos pide seleccionar un subconjunto $S \subseteq N$ tal que la suma de los pesos no exceda un límite dado w y además que la suma de valores sea tan grande como sea posible (maximizar la suma de valores). Entonces, el problema es

$$\text{máx} \sum_{i \in S} v_i \qquad \text{s.a.} \sum_{i \in S} w_i \leq w \tag{3.1}$$

Este es un problema de optimización. Adaptamos el problema a una versión de reconocimiento de lenguaje, es decir una versión de decisión y a ésta le llamamos *knapsack*. Aquí, adicionalmente se nos da un entero k y deseamos verificar si existe $S \in\subseteq N$ tal que

$$\sum_{i\in S} v_i \leq k \qquad \text{s.a.} \sum_{i\in S} w_i \leq w \tag{3.2}$$

Rigurosamente definido,

$$\text{KNAPSACK} = \{(N, w, k) : \exists S \subseteq N \text{ tal que satisface (3.2)}\}$$

Ejercicio 24.

a) Dar un algoritmo determinístico para KNAPSACK

b) Dar un algoritmo no determinístico para KNAPSACK

c) Dar cotas asintóticas para los algoritmos a) y b). ¿Son polinomiales?

3.2. Clases Básicas de Complejidad

Una clase de complejidad está dada por varios parámetros:

- El modelo de computo (objeto matemático que representa la computadora).
- La forma de realizar cálculos determinísticos y no-determinísticos-paralelos.

El recurso que medimos es el tiempo, espacio, comunicación, objetos compartidos, etc. Aquí, nuestro modelo será una máquina de Turing con k cintas, esto no afecta en absoluto el análisis, en el sentido de que si en vez se analizara con una MT de una cinta únicamente. A este tipo de máquinas las denotaremos MTDk y MTNDk.

Ejercicio 25. Probar que si $L \subseteq \Sigma^\star$ es aceptado por una MTD de k-cintas en tiempo $O(f(n))$ entonces L es aceptado por una MTD con 1 cinta en tiempo $O(f^2(n))$.

Definición 3.2.1. Sea $f : \mathbb{N} \to \mathbb{N}$ una función de complejidad apropiada (positiva y creciente) y Σ un alfabeto.

1) TIME $(f(n)) = \{L \subseteq \Sigma^\star : \exists M$ MTDk que decide a L en tiempo $f(n)\}$

2) NTIME $(f(n)) = \{L \subseteq \Sigma^\star : \exists M$ MTDNk que decide a L en tiempo $f(n)\}$

3) SPACE$(f(n)) = \{L \subseteq \Sigma^\star : \exists M$ MTDk que decide a L en espacio $f(n)\}$

4) NSPACE$(f(n)) = \{L \subseteq \Sigma^\star : \exists M$ MTDNk que decide a L en espacio $f(n)\}$

5) $\mathbf{P} = \cup_{k>0}\text{TIME}(n^k)$

6) $\mathbf{NP} = \cup_{k>0}\text{NTIME}(n^k)$

7) $\mathbf{L} = \text{SPACE}(\log n)$

8) $\mathbf{NL} = \text{NSPACE}(\log n)$

9) $\text{PSPACE} = \cup_{k>0}\text{SPACE}(n^k)$

10) $\text{NPSPACE} = \cup_{k>0}\text{NSPACE}(n^k)$

11) $\text{EXP} = \cup_{k>0}\text{TIME}(2^{n^k})$

Ejemplo 3.2.1.

- $\mathbf{P} \subseteq \mathbf{NP}$. Sea $L \in \mathbf{P}$. Luego existe M, una MTDk que decide a L en tiempo n^k, para alguna $k \in \mathbb{N}$. Como las MTDks son un caso particular de las MTDNks entonces M decide a L en tiempo n^k y por lo tanto $L \in \mathbf{NP}$.
- $\mathbf{L} \subseteq \mathbf{NL}$. Sea $L \in \mathbf{L}$, luego existe una MTDk M que acepta a L en espacio $\log(n)$. Esta máquina tambien es una MTND. Se sigue que $L \in \mathbf{NL}$.
- PSPACE $\subseteq$ NPSPACE. Es análogo a los anterior. Las MTDks son un caso particular de las MTNDks.
- P $\subsetneq$ EXP. Sea L un lenguaje en $\mathbf{P}$. Luego, existe una MTD tal que decide a L en tiempo menor a n^k, para alguna k. Como $n^k \leq 2^{n^k}$ se sigue que L se decide en éste tiempo. Esto concluye la prueba. La inclusión propia se prueba después.

Definición 3.2.2. Dado $L \subseteq \Sigma^\star$ definimos L-complemento, o simplemente L-co, como el conjunto de cadenas $x \in \Sigma^\star$ que no están en L, pero son entradas legítimas del problema definido por L.

Ejemplo 3.2.2. Los problemas de satisfacibilidad son los problemas válidos que no se satisfacen se definiría como el problema SAT-co.

Dada una clase de complejidad $\mathcal{C}$, definimos co-$\mathcal{C}$ como sigue

$$\textit{co-}\mathcal{C} = \{L \subseteq \Sigma^\star : \text{L-co} \in \mathcal{C}\}$$

Proposición 3.2.1. co-(co-$\mathcal{C}$) $= \mathcal{C}$

Demostración. Usando la definición se tiene que

$$\begin{aligned}\text{co-(co-}\mathcal{C}) &= \{L \subseteq \Sigma^\star : \text{L-co} \in \text{co-}\mathcal{C}\} \\ &= \{L \subseteq \Sigma^\star : (L\text{-co})\text{-co} \in \mathcal{C}\} \\ &= \{L \subseteq \Sigma^\star : L \in \mathcal{C}\} \\ &= \mathcal{C}\end{aligned}$$

■

Proposición 3.2.2. Para toda $f : \mathbb{N} \to \mathbb{N}$, una función de complejidad apropiada.

a) TIME($f(n)$) =coTIME($f(n)$)

b) SPACE($f(n)$) =coSpace($f(n)$) (tiempo y espacio determinístico)

Demostración.

a) Para el primer caso, dado un lenguaje L en alguna de las clases y su correspondiente máquina determinística M que decide el lenguaje L en un tiempo no mayor a $f(n)$, cambiamos todos los estados de aceptación a rechazo y viceversa y a esta máquina le llamamos M'. Se sigue que dado $x \in \Sigma^\star$, M entra en q_n si y solo si M' entra en q_y. Además el tiempo de decisión no excede a $f(n)$ en M'. Se concluye el primer resultado.

b) La prueba es totalmente análoga, salvo por la distinción que la $f(n)$ acota el espacio y no el tiempo.

■

Ejemplo 3.2.3. Si se sabe que SAT $\in$ TIME(2^{n^k}), usando la Proposición 3.2.2, se sigue que SAT $\in$ coTIME(2^{n^k})

Corolario 3.2.1.

*a) co-**P** = **P***

*b) co-**L** = **L***

c) PSPACE = co-PSPACE

d) EXP = co-EXP

Demostración.

a)

$$\begin{aligned}
\text{co-}\mathbf{P} &= \{L \subseteq \Sigma^\star : L\text{-co} \in \mathbf{P}\} \\
&= \{L \subseteq \Sigma^\star : L\text{-co} \in \cup_k \text{TIME}(n^k)\} \\
&= \{L \subseteq \Sigma^\star : L\text{-co} \in \cup_k \text{co-TIME}(n^k)\} \\
&= \{L \subseteq \Sigma^\star : L\text{-co} \in \text{co-}\mathbf{P}\} \\
&= \text{co-(co-}\mathbf{P}) \\
&= \mathbf{P}
\end{aligned}$$

b)

$$\begin{aligned}
\text{co-}\mathbf{L} &= \{L \subseteq \Sigma^\star : L\text{-co} \in \mathbf{L}\} \\
&= \{L \subseteq \Sigma^\star : L\text{-co} \in \text{SPACE}(\log n)\} \\
&= \{L \subseteq \Sigma^\star : L\text{-co} \in \text{co-SPACE}(\log n)\} \\
&= \{L \subseteq \Sigma^\star : L\text{-co} \in \text{co-}\mathbf{L}\} \\
&= \mathbf{L}
\end{aligned}$$

c) Es análogo a a)

d) Es análogo a a)

■

El Teorema de Jerarquía dice que con una cantidad suficientemente más grande de tiempo, las MT pueden hacer tareas más complejas. Sea $f(n) \geq n$ una función de complejidad apropiada. Definimos el lenguaje MP_f como una versión de tiempo del lenguaje del problema de membresia MP.

$$MP_f = \{M\#x : M \text{ acepta a } x \text{ en a lo más } f(|x|) \text{ pasos}\}$$

3.3. Teoremas del aceleramiento lineal y la jeraquía del tiempo y sus consecuencias

Teorema 3.3.1. *(Teorema del Aceleramiento Lineal). Sea $L \in TIME(f(n))$. Entonces, $\forall \epsilon > 0$, $L \in TIME(f_\epsilon(n))$, donde $f_\epsilon(n) = \epsilon f(n) + n + 2$.*

Hemos dicho que las constantes "no importan" para fines prácticos y este Teorema lo fortalece. Los avances en hardware reducen las constantes más no el orden de complejidad de la función. El siguiente lema nos dice que existe una máquina universal que simula MT que tarden $f(n)$ pasos en aceptar, en a lo más $f^3(n)$.

Lema 3.3.1. $MP_f \in \text{TIME}(f^{3+\epsilon}(n))$

Demostración. Sea f una función de complejidad apropiada que satisface que $f(n) \geq n$ y M es una máquina de k-cintas. Construimos una MT U_f con 4 cintas la cual decide a $M\#x$ en tiempo $O(f^3(n))$. Es decir, recibe a $M\#x$, simula, y no tarda más de $f(|x|)$ pasos en decidir. U_f está basada en varias máquinas:

- La máquina universal U de una cinta que simulara a $M\#x$ de k cintas.
- La máquina del teorema del aceleramiento lineal.
- La máquina M_f que calcula a $f(n)$.

Imagine las 4 cintas en paralelo. La máquina U_f hace lo siguiente en este orden.

1. U_f utiliza a M_f para inicializar en su cuarta cinta un "reloj de alarma" de longitud $f(|x|)$, es decir escribe $f(|x|)$ veces el mismo símbolo. Esto lo hace en $O(f(|x|))$, donde la constante depende sólo de f y no de M o x. Si M_f usa más cintas de cuatro cintas, se agregan el número necesario a U_f.
2. U_f copia la descripción de M, la máquina a ser simulada en su tercera cinta y copia (o codifica) a x en su primera cinta, esto toma $O(n)$.

3. U_f inicializa la segunda cinta con el estado inicial q_0 correspondiente a la simulación de M. En este momento U_f puede verificar que su estrada sea válida y rechazar si no es así (lo cual se puede hacer en tiempo lineal en dos cintas). El tiempo usado hasta este momento es de $O(f(|x|)+n) = O(f(n))$. Esta última igualdad porque por hipótesis $f(n) \geq n$ y porque $|x| \leq n$. A continuación recapitulamos que hay en cada una de las cuatro cintas.

   3.1. Codificación de x a binario.

   3.2. Estado de la simulación de M en x y se escribe aquellos símbolos donde estén las cabecillas.

   3.3. Descripción de M.

   3.4. Cinta de Reloj de Alarma con el símbolo $1^{f(|x|)}$.

   A continuación describimos el ciclo de operación de U_f.

4. Se va a simular las k cintas de M en la primera cinta, paso por paso. Primero se hace un escaneo en la primera cinta para saber que símbolos son los que M tiene en las cabezas lectoras y estos son escritos en la segunda cinta. Recuerda que M tiene k cintas pero se está simulando como una única cinta. Ya que transcribe todos los símbolos, considera el estado actual y busca la transición en la 3era cinta y la ejecuta, modificando el contenido de la primera cinta y el estado actual en la segunda cinta. También borra los símbolos después del estado). Por último tacha una unidad de tiempo de la alarma, o en otras palabras incrementa su reloj de alarma en 1.

   Si una máquina termina en $f(|x|)$ pasos entonces una máquina universal de esta forma la puede simular en $O(|M|f(|x|)\log(f(|x|))$ pasos. Es claro que $|M| = |M_k|$. Como la simulación de M para M_k es de orden $O(k^2 f^2(|x|))$ pasos, entonces U_f puede simular a M en x, para $\epsilon > 0$, en orden

$$\begin{aligned} O(|M_k|k^2 f^2(|x|)\log(k^2 f^2(|x|))) &\leq O(n\log^3(f(n))f^2(n)) \\ &\leq O(n^{1+\epsilon}f^2(n) \\ &\leq O(f^{3+\epsilon}(n)) \end{aligned}$$

   Resta justificar que podemos hacer exactamente $f^{3+\epsilon}(n)$ pasos usando el teorema del aceleramiento lineal. Si $0 < c < 1$ no hay necesidad de usar el Teorema. Supongamos que tardamos $cf^3(n)$ pasos con $c > 1$. Se sigue que $L \in TIME(cf^{3+\epsilon}(n))$. Con $\epsilon = 1/2c$ podemos encontrar una máquina de Turing que resuelve el problema en tiempo $\frac{1}{2}f^3(n) + n + 2 \leq f^{3+\epsilon}(n)$. Haciendo pasos de relleno, es posible que esta máquina termine en exactamente $f^{3+\epsilon}(n)$ pasos. Se concluye que $MP_f \in \text{TIME}(f^{3+\epsilon}(n))$.

■

Ejercicio 26. (Extra +0.5) Dada una codificación M de una máquina de Turing razonable, si $n = |M\#x|$, entonces $\log |M| = O(n)$.

Lema 3.3.2. $MP_f \notin \text{TIME}(f(\lfloor \frac{n}{2} \rfloor))$

Demostración. Por contradicción, supongamos que $MP_f \in \text{TIME}\ (f(\lfloor \frac{n}{2} \rfloor))$. Así que existe una MT con k-cintas K_f que decide a MP_f en tiempo $f(\lfloor \frac{n}{2} \rfloor)$. Con la suposición de la máquina K_f, es posible construir una máquina "diagonalizadura" D_f, con el siguiente programa: $D_f(M)$:

K_f acepta a $M\#M$ then
    rechaza
else
    acepta

Note que el input de K_f tiene el doble de longitud más uno, que la entrada de D_f, y sin embargo $D_f(M)$ corre en el mismo tiempo que corre $K_f(M\#M)$. Es decir si $n = |M|$, K_f corre en tiempo $f\left(\lfloor \frac{2n+1}{2} \rfloor\right) = f\left(\lfloor n + \frac{1}{2} \rfloor\right) = f(n)$

Ahora nos preguntamos ¿Qué pasa si D_f corre en sí misma?. Supongamos que D_f acepta, entonces K_f rechaza a la cadena $D_f\#D_f$, es decir $D_f\#D_f \notin MP_f$. Luego D_f no acepta a D_f en tiempo menor a $f(n)$, en otras palabras $D_f(D_f)$ se rechaza, esto es una contradicción con lo que se se supuso al inicio. Por el contrario, si $D_f(D_f)$ se rechaza entonces K_f acepta a la cadena $D_f\#D_f$, esto implica que $D_f(D_f)$ es aceptado. Concluimos que K_f no existe y por lo tanto $MP_f \notin \text{TIME}(f(\lfloor \frac{n}{2} \rfloor))$. ■

Teorema 3.3.2. *(Jerarquía del tiempo) Si $f(n) \geq n$ es una función de complejidad apropiada entonces $TIME(f(n)) \subsetneq TIME(f^3(2n+1))$*

Demostración. Trivialmente, $\text{TIME}(f(n)) \subseteq \text{TIME}(f^3(2n+1))$. En los lemas 3.3.1 y 3.3.2, haciendo $m = 2n+1$, probamos que MP_f está en el segundo y no en el primero. Se concluye que $\text{TIME}(f(\frac{m-1}{2})) \subsetneq \text{TIME}(f^3(m))$ ■

Corolario 3.3.1. $\boldsymbol{P \subsetneq EXP}$

Demostración. Por el Teorema de Jerarquía,

$$\text{P} \subseteq \text{TIME}(2^n) \subsetneq \text{TIME}((2^{2n+1})^3) \subseteq \text{EXP}$$

■

Teorema 3.3.3. *Jerarquía del espacio. Si $s(n)$ es una función de complejidad apropiada. Entonces $SPACE(s(n)) \subsetneq SPACE(s(n)\log(s(n)))$*

Ejercicio 27. (Extra +3) Probar el Teorema 3.3.3.

Problema de alcanzabilidad

El método de "alcanzabilidad". Sea $G = (V, A)$ una digráfica. Es un problema muy común el decidir si dados dos vértices i y j, si existe un camino dirigido de i a j. Este problema lo llamamos REACHABILITY.

Teorema 3.3.4. *Reachability* $\in TIME(n^2)$ *donde* n *es el número de vértices de* G.

Teorema 3.3.5. *Sea* $f : \mathbb{N} \to \mathbb{N}$ *una función de complejidad apropiada entonces:*

a) $SPACE(f(n)) \subseteq NSPACE(f(n))$.

b) $TIME(f(n)) \subseteq NTIME(f(n))$.

c) $NTIME(f(n)) \subseteq SPACE(f(n))$.

d) $NSPACE(f(n)) \subseteq TIME(k^{\log(n)+f(n)})$.

Demostración.

a) Es trivial

b) Es trivial

c) Sea $L \in \text{NTIME}(f(n))$. Luego, existe una MTND que decide en tiempo exactamente "$f(n)$". Se construye una MTD L' que decide L en espacio $f(n)$.

d) Sea $L \in \text{SPACE}(f(n))$, luego existe una MTDN M que decide a L en espacio $f(n)$. Una configuración para M con k cintas es una tupla $w_1, i, u_1, w_2, i, u_2, \ldots w_k, i, u_k$, donde $s \in Q$ y $w_i u_i \in \Gamma^\star$ y el símbolo i indica que la cabeza lectora de la i-ésima cinta está posicionada en el 1er símbolo de u_i para cada cinta. El número de posibles de configuraciones de M se determina como sigue. Hay $|Q|$ posibles estados y $|\Gamma + 1|^{k \cdot f(n)}$, el $+1$ es por el símbolo de espacio, que se considera aparte. La posición de la cabecilla está implica en la notación de la cadena y por ello no se considera como un tercer factor dentro de las combinaciones. El número de configuraciones es $c_1 = |\Gamma + 1|^k \cdot |Q|$, donde este número depende de M y nada más. Definimos la gráfica dirigida de configuraciones de M, denotada por $G(M, x), (x \in \Sigma^\star)$. El conjunto de vértices de $G(M, x)$ es el conjunto de todas las posibles configuraciones y hay un arco de configuración D_1 a D_2.

   El decidir si M acepta a $x \in \Sigma^\star$, es equivalente a decidir si existe un camino dirigido de la configuración $q_0\lambda, i, x\lambda, i, x \ldots \lambda, i, x$ hacia alguna configuración de aceptación de M ($q_y \cdots \in G(M, x)$. En otras palabras, hemos reducido el problema de decidir a L al problema de *Reachability* en una digráfica de $c_1^{f(n)}$ vértices. *Reachability* se puede resolver en $O(n^2)$ tiempo donde n es c_1, y sea c_2 la constante que proviene del orden. Por lo tanto podemos decidir a L en tiempo $c_2 \cdot (c_1^{f(n)})^2$. Si $c = c_1 c_2^2$ entonces se puede decidir a L en tiempo $(c^{f(n)})$.

■

Si Q y Γ son el conjunto de estados y símbolos de N, sean $\sigma \in \Gamma$, $q \in Q$. Sea

$$C_{\sigma,q} = \{(S, \beta, A) : A \in \{L, R\}, (S, \beta, A) \in \delta(q, \sigma)\},$$

un renglón del árbol. Luego $C_{\sigma,q}$ es el conjunto de posibles elecciones no determinísticas para la pareja (q, σ). Como cada $C_{\sigma,q}$ es finito, sea $d = \max_{q,\sigma} |C_{\sigma,q}|$. Construimos una MTD S que decida a L en espacio $f(n)$. S primero calcula el número d, luego en la tercera cinta genera un secuencia de números cada uno entre 0 y $d-1$. Genera $f(n)$ números.

Luego S genera la primer secuencia de números $(O^{f(n)})$ en su tercer cinta y empieza a simular a N, en el camino indicado por la secuencia actual. Si en este camino N acepta, S acepta si S agota todos los posibles caminos de N y esta no acepta, entonces S rechaza su entrada. Es claro que como S reutiliza el espacio de trabajo, el espacio total usado es de $O(f(n))$ (aunque el tiempo puede ser exponencial. Así que $L \in \text{SPACE}(f(n))$.

Teorema 3.3.6. *(Sauitch) El problema Reachability* $\in$ *SPACE*$(\log^2(n))$.

Corolario 3.3.2. *Si* $s : \mathbb{N} \to \mathbb{N}$ *es una función de complejidad apropiada y* $s \geq \log n$*, entonces NSPACE*$(s(n))$ = *SPACE*$(s^2(n))$

Ejercicio 28. Probar que **P** es cerrado bajo unión e intersección de conjuntos. Repetir para **NP**.

Ejercicio 29. (Extra) Sea L un lenguaje y $L^\star$ su cerradura o bien, su estrella de Kleene.

a) **NP** es cerrado bajo estrella de Kleene.

b) **P** es cerrado bajo estrella de Kleene.

4. Reducciones y completez

4.1. Reducciones polinomiales

Cualquier clase de lenguaje, por su mera definición posiblemente contenga una infinidad de lenguajes. Sin embargo, para clases complejidad famosas podemos ir generando una lista de aquellos lenguajes famosos que contengan. Por ejemplo, algunos de los lenguajes que contiene la clase **NP** son

- El problema del agente viajero
- Alcanzabilidad
- SAT, 3SAT
- k-coloring $(k = 2, 3)$
- Knapsack (optimización)

Podríamos tener una lista similar de la clase **P**. Como cualquier lenguaje en **P** está en **NP**, se sigue que todo problema en **P** es a lo más tan difícil como resolver otro en **NP**. De ahí se motiva una idea de poder jerarquizar que problemas son más difíciles que otros. La idea clave es que si podemos transformar un lenguaje L_1 a un lenguaje L_2 entonces resolver L_2 resolverá a L_1 también. Por lo tanto, L_1 es a lo más tan difícil como resolver L_2. En otras palabras, no es posible que L_1 sea más difícil que L_2 porque ya probamos que si resolvemos L_2, entonces se resuelve gratis L_1.

Definición 4.1.1. Sean L_1, $L_2 \subseteq \Sigma^\star$. Definimos que L_1 es reducible en tiempo polinomial a L_2, denotado $L_1 \leq_p L_2 \Leftrightarrow$ existe una función computable en tiempo polinomial $f : \Sigma^\star \to \Sigma^\star$, por una MTDk tal que cumple que $\forall x \in \Sigma^\star$ se siga que $x \in L_1 \Leftrightarrow f(x) \in L_2$).

En capítulos anteriores se dan nociones de reducciones que no tienen un concepto de eficiencia. En este capitulo se exploran este tipo de reducciones eficientes.

Definición 4.1.2. Se define el conjunto de funciones computables en tiempo polinomial FP como sigue

$$FP = \{f : \Sigma^* \to \Sigma^* : \exists M \text{ una MTD de k-cintas que calcula a } f \text{ en tiempo } n^p\}$$

Ejercicio 30. Si $f_1 : \Sigma^* \to \Sigma^*$ es una reducción polinómica de L_1 a L_2 y $f_2 : \Sigma^* \to \Sigma^*$ es una reducción polinomial de L_2 a L_3, entonces $f_2 \circ f_1 : \Sigma^* \to \Sigma^*$ es una reducción polinomial de L_1 a L_3.

Ejercicio 31. Probar que si $A \leq_p B$ y $B \in \mathbf{P}$, entonces $A \in \mathbf{P}$.

Sea $k \geq 1$. Se define el lenguaje:

$$kSAT = \{\phi : \phi \text{ está en FNC, con k literales en cada clausula y } \phi \text{ es satisfacible}\}$$

Lema 4.1.1. SAT $\leq_p$ 3SAT

Demostración. Debemos probar que existe $f \in FP$ tal que $\phi \in$ SAT $\Leftrightarrow f(\phi) \in$ 3SAT. Si alguna clausula de ϕ contiene a más de 2 literales entonces

$$(x_1 \vee \neg x_2) \sim (x_1 \vee \neg x_2 \vee \neg x_3)$$
$$\neg x_3 \sim (\neg x_3 \vee \neg x_3 \vee \neg x_3)$$

Si una clausula C de ϕ contiene más de 3 literales $\Rightarrow$ la convertimos a un número de clausulas equivalentes, agregando variables adecuadas que no están en ϕ, de tal manera que se conserve la satisfacibilidad o no satisfacibilidad de C, por ejemplo:

$$C = (x_1 \vee x_2 \vee \neg x_3 \vee x_4)$$

Se convierte en

$$C = (x_1 \vee x_2 \vee z_2) \wedge (\neg z_1 \vee \neg x_3 \vee x_4)$$

En general, si C es la clausula $C = (a_1 \vee a_2 \vee \cdots \vee a_l), l \geq 4$. Luego C es reemplazado por las $l-2$ clausulas $(a_1 \vee a_2 \vee z_1) \wedge (\neq z_1 \vee a_3 \vee z_4) \cdots \wedge (\neq z_k \vee a_{l-1} \vee a_l)$. El orden de esta transformación es de orden $O(nm) = O(n^2)$ y por lo tanto es polinomial. ■

Teorema 4.1.1. *$\phi \in$ SAT $\Leftrightarrow f(\phi) \in$ 3-SAT*

Demostración. Sea $U = \{u_1, u_2, \ldots, u_n\}$ un conjunto de variables y sea $C = \{c_1, c_2, \ldots, c_m\}$ un conjunto de clausulas generando una instancia arbitraria de SAT (no necesariamente de 3SAT). Construiremos un conjunto de clausulas C' de tamaño 3 tal que sea satisfacible si y solo si C también lo es. La idea será remplazar cualquier clausula que no sea de tamaño 3 por otra u otras equivalentes de tamaño 3. Para ello se usaran variables booleanas auxiliares en U_j, para las clausulas C'_j. Por lo tanto

$$U' = U \cup \left(\cup_{j=1}^{m} U'_j\right) \text{ y } C' = \cup_{j=1}^{m} C'_j$$

Resta mostrar cómo C'_j y U'_j se pueden construir a partir de c_j. Sea c_j dado por $\{z_1, z_2, \ldots, z_k\}$ donde las z_j's son literales derivadas de las variables de U. De esta forma en que C'_j y U'_j se generan del tamaño de la clausula k. Hagamos casos.

1. Si $k = 1$:

$$U'_j = \{y_j^1, y_j^2\} \text{y } C'_j = \{\{z_1, y_j^1, y_j^2\}, \{z_1, y_j^1, \bar{y}_j^2\}, \{\bar{y}_j^1, y_j^2\}, \{z_1, \bar{y}_j^1, \bar{y}_j^2\}\}.$$

2. Si $k = 2$:
$$U'_j = \{y_j\} \text{ y } C'_j = \{\{z_1, z_2, y_j^1\}, \{z_1, z_2, \bar{y}_j^1\}\}$$

3. Si $k = 3$:
$$U'_j = \emptyset \text{ y } C'_j = \{\{c_j\}\}.$$

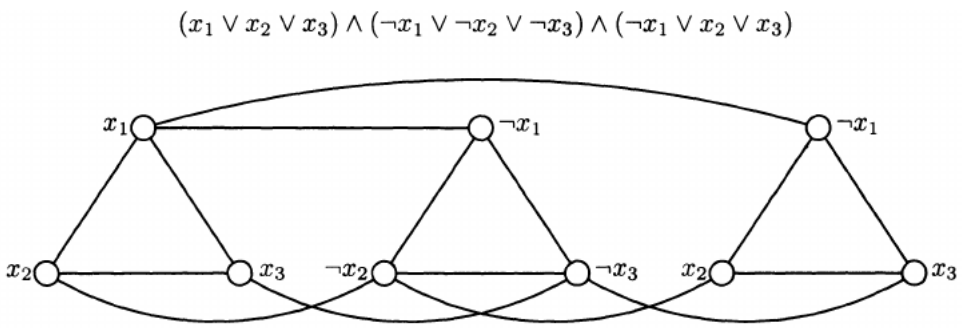
4. Si $k \geq 4$:

$$U'_j = \{y^i_j : 1 \leq i \leq k-3\}$$
$$C'_j = \{\{z_1, z_2, y^1_j\}\} \cup \{\{\overline{y}^i_j, z_{i+2}, y^{i+1}_j\} : 1 \leq i \leq k-4\} \cup \{\{\overline{y}^{k-3}_j, z_{k-1}, z_k\}\}.$$

Se puede probar fácilmente que en efecto C_j se satisface si y sólo si C'_j se satisface. Definiendo una función f que según el caso genere las clausulas respectivas equivalentes de tamaño 3 se obtiene un problema 3SAT. Se concluye que $\phi \in$ SAT $\Leftrightarrow f(\phi) \in$ 3-SAT ■

Teorema 4.1.2. *$\phi \in$ 3-SAT $\Leftrightarrow f(\phi) \in$ INDEPENDENT SET*

Demostración. La prueba usa el artefacto de un *triangulo*. El punto es que si la gráfica contiene un triángulo, entonces cualquier conjunto independiente a lo más puede contener un nodo del triángulo. Hay aún más en este construcción. Para probar que INDEPENDENT-SET es **NPC** lo mejor es restringirla a este tipo de gráficas. Consideramos únicamente aquellas gráficas cuyos nodos pueden ser particionados en m triángulos disjuntos, tal y como se ve en la siguiente figura.



Reduction to INDEPENDENT SET.

Es obvio que un conjunto independiente que a lo más sea de tamaño m existe si y sólo si se puede seleccionar exactamente un nodo en cada triángulo. Intuitivamente pareciera que es más fácil de resolver estos problemas pero computacionalmente no lo son. La reducción polinomial es inmediata. Para cada una de las m clausulas de un predicado booleano ϕ creamos un triángulo en la gráfica G. Cada nodo corresponde a un literal en la clausula. Se añade una arista entre dos nodos en triángulos distintos si y sólo si corresponden a literales opuestos o negados. Intuitivamente, si seleccionamos un literal, ya no podemos seleccionar en ningún otro triángulo el literal negado.

Rigurosamente, dada una instancia ϕ en 3SAT con m clausulas $C_1, C_2, \ldots, C_m$, con cada clausula de la forma $C_1 = \{\alpha_{i1} \vee \alpha_{i2} \vee \alpha_{i3}\}$, con $\alpha'_{ij}s$ literales. Nuestra reducción $R(\phi) = (G, k)$, donde $k = m$ y $G = (V, E)$ es la siguiente gráfica. El conjunto $V = \{v_{ij} : i = 1, 2, \ldots, m; j = 1, 2, 3\}$. El conjunto

$$E = \{v_{ij}v_{ik} : i = 1, 2, \ldots, m, j \neq k\} \cup \{v_{ij}v_{ik} : i \neq l, \alpha_{ij} = \neg\alpha_{ik}\}$$

Nosotros mostramos que existe un conjunto independiente de k nodos en G si y sólo si ϕ es satisfacible. Supongamos que tal instancia I existe. Como $k = m$, I debe contener un nodo en cada uno de los triángulos. Como los nodos están etiquetados con literales, entonces I no contiene ambos literales, para ningún par de triángulos. Por lo tanto, el conjunto de nodos seleccionados es uno y sólo uno de los literales de cada variable booleana. Al seleccionar los literales verdaderos en la gráfica, de acuerdo a la misma etiqueta del predicado, y no seleccionar literales contrarios implícitamente se está seleccionado el valor que hace al predicado verdadero. Conversamente, si ϕ es satisfacible, entonces seleccionar aquellos vértices genere un conjunto independiente de tamaño k. De esta manera podemos generar un conjunto de tamaño $m = k$. ■

Ejercicio 32. (Extra +2) Probar que 2SAT $\in$ **P**.

Definición 4.1.3. Sea G una gráfica. Se define un **cliqué** en G como un subconjunto $C \subseteq V$ tal que C forma una subgráfica completa. Es posible que $C = V$, pues no es necesario que sea subconjunto propio.

Definición 4.1.4. Una **cubierta** de $G = (V, E)$ es un subconjunto $N = V(G)$ tal que para toda arista $v_i v_j$ se cumple que $v_i \in N$ o $v_j \in N$. Este o no es exclusivo, por ello es posible que $N = V$.

Enunciamos dos problemas clásicos de decisión:

1. CLIQUE $= \{(G, k) \in \Sigma^\star : G$ tiene un clique de tamaño $k\}$
2. NODE COVER $= \{(G, k) \in \Sigma^\star : G$ tiene una cubierta de k vértices.$\}$

Ejercicio 33. Probar que

a) CLIQUE $\in$ **NP**.

b) NODE COVER $\in$ **NP**.

Lema 4.1.2.

a) INDEPENDENT SET $\leq_p$ CLIQUE

b) INDEPENDENT SET $\leq_p$ NODE COVER

Ejercicio 34. Probar el Lema 4.1.2.

4.2. Problemas completos y duros

Definición 4.2.1. Sea $L \subseteq \Sigma^*$ y $\mathcal{D}$ una clase de complejidad. Se dice que:

1. $L \in \mathcal{D}-$Hard si $\forall L' \in \mathcal{D}, L \leq_p L$
2. L es $\mathcal{D}$-completo si L es $\mathcal{D}$-Hard y $L \in \mathcal{D}$

Es útil para probar que un lenguaje pertenece a una clase de complejidad dada y otra es para probar igualdades ente clases de complejidad. Esto siempre y cuando las clases de complejidad en cuestión sean cerradas bajo reducciones.

Decimos que una clase de complejidad es cerrado bajo reducciones si dados $L', L \subseteq \Sigma^\star$, $L \in \mathbf{P}$ y $L' \leq_p L$ entonces $L' \in \mathbf{P}$. Todas las clases de complejidad que hemos visto son cerradas bajo reducciones.

Proposición 4.2.1. **P**, **NP**, **L**, **NL**, **coNP**, **PSPACE**, ... y **EXP** son cerradas bajo reducciones.

Ejercicio 35. Probar al menos con dos casos la Proposición 4.2.1.

Teorema 4.2.1. *Sea $L \in$ **NP**. Si L es **NP**-completo y $L \in$**P** entonces **P** = **NP**.*

Demostración. Sea $L' \in \mathbf{NP}$ entonces como L es **NP**-complejo entonces $L' \leq_p L$. Además, $L \in \mathbf{P}$. Se sigue que por la Proposición 4.2.1 $L' \in$**P**. De esto se sigue que $\mathbf{NP} \subseteq \mathbf{P}$. Como ya se tiene la contención en el otro sentido entonces se concluye que $\mathbf{P} = \mathbf{NP}$. ■

5. Problemas NP-completos

5.1. Teorema de *Cook*

En esta sección mostramos uno de los teoremas más importantes de teoría de complejidad. Aquí se demostró que la clase de problemas **NP** completos era no vacía. Esto hace que baste un sólo problema en esta clase para probar que **NP** = **P**. También, por las reducciones polinomiales es más fácil probar que un problema es **NPC**. Por lo tanto, el teorema ayudó a crecer la lista de problemas en esta clase.

Teorema 5.1.1. *SAT es **NP**-completo.*

Demostración. Ya sabemos que SAT $\in$ **NP**. Resta demostrar que SAT es **NP**-hard. Sea $L \in$ **NP**, por demostrar que $L \leq_p$ SAT. Por hipótesis existe una MTND N tal que $L(N) = L$ y N trabaja en tiempo polinomial, digamos n^k. Usaremos el concepto de configuraciones (xqy) y por comodidad asumimos que están escritas entre símbolos #. Configuración $\#xqy\sqcup\#$. x es la cadena, q es el estado y y es la cabecilla. Considere la siguiente tabla (izq)de dimensión $n^k \times n^k$, con celdas $c[i, j]$. Definimos una ventana como cualquier subtabla de dimensión 2×3 que toma 2 y 3 renglones y columna consecutivos respectivamente.

#	q_0	a_1	a_2	...	a_n	$\sqcup$	...	$\sqcup$	#
#									#
...	...	...	...	...	...	...	...	...	...
#									#

Sea $w \in \Sigma^\star$, $w = a_1a_2 \ldots a_n$, $a_i \in \Sigma$ vamos a construir a $f : \Sigma^\star \to \Sigma^\star$, $f \in FP$ y además $w \in L \Leftrightarrow f(w) \in$ SAT. Definimos una tabla para N en la entrada w como un arreglo de dimensión de tamaño $n^k \times n^k$, en la cual los renglones son configuraciones de N en w. la tabla comienza con la configuración inicial y cada renglón subsecuente sigue al renglón anterior, de acuerdo a la función de transición de N. Una tabla de aceptación si tiene un renglón que contiene una configuración de aceptación. Cada tabla de N corresponde a un camino no deterministico de N en w. Entonces el determinar si $w \in$ L es equivalente al problema de determinar si existe una tabla de aceptación par N y w.

A continuación describimos la reducción $f : \Sigma^\star \to \Sigma^\star$, de L a SAT. Describimos primero el conjunto de variables booleanas de ϕ. Sean Q y Γ el conjunto de estados y símbolos de la cinta de N ($\Sigma^\star \subseteq \Gamma$). Sea $C = Q \cup \Gamma \cup \{\#\}$. Entonces el conjunto de varialbes de $f(w) = \phi$ es $\{x_{i,j,s} : 1 \leq i, j \leq n^k, s \in C\}$.

$xi, j, s \leftrightarrow c[i, j] = s$. $x_{i,j,s} = t \Leftrightarrow c[i, j] = s$. Ahora vamos a construir a la fórmula ϕ. Esta tiene la siguiente fórmula general

$$\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{move} \wedge \phi_{accept}.$$

Describimos cómo construir cada una de ellas. Primero, debemos asegurarnos que el hecho que cada celda de la tabla sólo puede contener un sólo símbolo esté reflejado en ϕ (No se vale $x_{i,j,s} = t$ y $x_{i,j,c} = t$ para $s \neq c$. Luego $\phi_{cell} = \wedge_{1 \leq i,j \leq n^k}[\vee_{s \in C} x_{i,j,s} \wedge (\vee_{s,t \in C s \neq t} \neq x_{i,j,s} \vee \neq x_{i,j,t})$.

Con la fórmula ϕ_{start} obligamos a que el primer renglón sea la configuración inicial de N en la cadena $w, \#q_0 w \sqcup \cdots \sqcup \#$. Luego

$$\phi_{start} = x_{1,1,\#} \wedge x_{1,2,q_0} \wedge x_{1,3,a_i} \wedge \cdots \wedge x_{1,n+1,a_n} \wedge x_{1,n+2,\sqcup} \wedge \cdots \wedge x_{1,n^k-1,\sqcup} \wedge x_{1,n^k,\#}$$

Con la fórmula ϕ_{accept} garantizamos que una configuración de aceptación aparezca en la tabla, estipular que alguna de las variables, x_{i,j,q_y} es t.

$$\phi_{accept} = \bigvee_{1 \leq i,j \leq n^k} x_{i,j,q_y}$$

Finalmente, con la fórmula ϕ_{move} garantizamos que cada renglón de la tabla es válido con respecto al renglón anterior, de acuerdo a lo especificado por la función de transición de N. En ϕ_{move} aseguramos que cada ventaina de 2×3 sea legal. Una ventana de 2×3 es legal si en esa ventana no se violan las reglas de transición de N. Es decir, una ventana es legal si puede aparecer cuando la configuración del primer renglón precede correctamente al de abajo.

Ejemplo: Supongamos para propósitos ilustrativos, que N tiene las siguientes transiciones, que N tiene las siguientes transiciones $\delta(q_1, a) = \{q_1, b, R\}$ y $\delta(q_1, b) = \{(q_2, c, L), (q_2, a, R)\}$. Las siguientes ventanas son legales

a	q_1	b
q_2	a	c

a	q_1	b
a	a	q_2

a	a	q
c	a	b

#	b	a
#	b	a

a	b	a
a	b	q_2

Las siguientes son ilegales.

a	b	a
a	a	a

a	q_1	b
q_1	a	a

b	q_1	b
q_2	b	q_2

Afirmamos que si el primer renglón es la confuracion de inicio de N en w y todas las posibles ventanas de 2×3 son legales, entonces cada renglón es válido y sucede correctamente al anterior.

$$\phi_{move} = \bigvee_{1 \leq i,j \leq n^k} (\text{ventana } (i,j) \text{ es legal})$$

Remplazamos el texto "ventana (i, j) es legal" con la siguiente fórmula boolena para cada valor de (i, j) escribiendo el contenido de las celasde la ventana con símbolos $a_1, a_2, \ldots, a_f$.

$$\bigwedge_{\substack{a_1,a_2,\ldots a_f \text{ es} \\ \text{una ventana legal}}} (x_{i-1,j,a_1}) \wedge (x_{i,j,c_{12}}) \wedge (x_{i+1,j,c_{12}}) \wedge (x_{i-1,j+1,a_4}) \wedge (x_{i,j+1,a_5}) \wedge (x_{i+1,j+1,a_6})$$

Analicemos la complejidad de la reducción de f. Sea $l = |C| = |Q \cup \Gamma \cup \{\#\}|$. Notamos que l depende de N y no de w. $\phi = \phi_{cell} \wedge \phi_{start} \wedge \phi_{accept}\phi_{move}$. ϕ_{cell} es una conjunción de fórmulas de n^{2k} fórmulas.

$$\left(\bigwedge_{\substack{s,t \in C \\ s \neq t}} (\neg x_{i,j,s} \vee x_{i,j,t}) \right)$$

Luego, $O(l)+O(l^2) = O(l^2)$. Por lo tanto $O(l^2 \cdot n^{2k}) = O(n^{2k})$. ϕ_{start} se puede generar en tiempo $O(n^k)$. ϕ_{accept} se puede generar en tiempo $O(n^{2k})$. ϕ_{move} se puede generar en tiempo $O(n^{2k})$. Por lo tanto ϕ se genera en tiempo acotado por $O(n^{2k})$. Falto ver que $w \in L \leftrightarrow f(w) = \phi \in$ SAT. Se sigue que $L \in$ SAT y por lo tanto SAT es **NP**-completo. ■

5.2. Problemas NP-completos básicos

Todos los siguientes son problemas **NP** completas.

1. 3 SAT
2. Independent Set
3. Clique
4. Knap Sack
5. Subset Sum