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* (Prueba de existencia de un problema **NP**-completo: SAT)
   5.2. Problemas **NP**-completos básicos: 3SAT, apareamientos, cubierta de vertices, circuito hamiltoniano, clan, etc).
   5.3. Técnicas: Restricción, remplazo local, diseño de componentes.
6. **P** *vs* **NP** y más allá.
   6.1. ¿Son **P** y **NP** iguales?
   6.2. La clase **BPP**
   6.3. Contar soluciones y la clase #**P**

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?
- 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 cualquier conjunto finito.

Ejemplo 1.2.1. Determinamos cuáles conjuntos son alfabetos.

1. Todos los siguientes son alfabetos:
   - $\Sigma = \{0, 1\}$
   - $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
   - 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 pues no es finito.
   - $I = [0, 1]$

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.

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.

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

[1] Se refiere a que si vale la pena correr el algoritmo. Si me voy a tardar mucho no vale la pena.

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

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. En símbolos, $\Sigma^\star = \{x : \forall a \in x \Rightarrow a \in \Sigma\}$. Por convención $\emptyset^\star = \{\lambda\}$.

Ejemplo 1.2.4. Se muestran las cerraduras de distintos conjuntos.

- La cerradura de $\{a\}$ es $\{a\}^\star = \{\lambda, a, aa, aaa, aaaa, \dots\} = \{a^n | n \geq 0\}$.
- 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 $xy = a_1a_2\dots a_nb_1b_2\dots b_m \in \Sigma^\star$.

Notación. Análogo a que denotemos el símbolo $a^5 = aaaaa$, se pueden denotar cadenas. Por ejemplo, $(aba)^5 = (aba)(aba)\dots(aba) = abaaba\dots aba$. Note que puede pensarse como mera notación o como la operación concatenación 5 veces.

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

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, asociatividad y existencia de un elemento neutro en A.

Teorema 1.2.1. *Sea Σ un alfabeto y la operación de concatenación denotada por $\circ$. 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 **??**. Note que como no existe un elemento inverso, $(\Sigma, \circ)$ no es un **grupo**. ■

Operaciones en conjuntos

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

Definición 1.2.8. Sean A y B son subconjuntos de $\Sigma^\star$ ($A, B \subseteq \Sigma^\star$). La operación **concatenación** de A y B se define como $AB = \{xy : x \in A, y \in B\}$.

Es importante distinguir el contexto, si la operación es sobre cadenas o sobre conjuntos, ya que la operación se llama igual, pero se define según el caso. Note como 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 $\{\lambda\}$.

Notación.

a) Sea $A \subseteq \Sigma^\star$, luego $A^n = AA \ldots A$, n veces. Note que si $A = \{a\}$ entonces $A^n = \{a^n\}$.

b) Sea $\Sigma = \{a, b\}$, entonces $\{a, b\}^n = \{x \in \{a, b\}^\star : \ |x| = n\}$.

Note como la notación es consistente. Sea $A = \{a\}$. A continuación partimos de acuerdo al punto b) de la notación anterior y concuerda con a). Se define $A^n = \{x \in \{a\} : |x| = n\} = \{a^n\} = A^n$.

Resultado 1.2.1. $|\{a, b\}^n| = 2^n$.

Definición 1.2.9. La **cerradura** de $A \subseteq \Sigma^\star$ es $A^\star = A^0 \cup A^1 \cup \cdots = \cup_{i=0}^{\infty} A^i$, donde $A^0 = \{\lambda\}$ y $A^{n+1} = A^n A$. En palabras, podemos pensar a $A^\star$ como todas las cadenas que podemos formas con el alfabeto A, quizá una restricción de un alfabeto más grande Σ. Una notación equivalente en símbolos y quizá más intuitiva es

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

Acordamos 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 $\{\lambda\}$. Por eso conviene definirlo de esta manera, $A_0 = \{\lambda\}$. 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, en alfabetos no triviales.
Propiedades adicionales

1. $A^{\star\star} = A^{\star}$ (idempotencia de *).

   Demostración. Por definición, $A^{\star\star} = \cup_{i=0}^{\infty}(A^{\star})^i$ Por el ejercicio anterior, usando inducción es fácil probar que $(A^{\star})^i = A^{\star}$, para toda $i \geq 1$. Finalmente, $(A^{\star})^0 = \{\lambda\}$, por convenio (¿también?). Luego $A^{\star\star} = \cup_{i=0}^{\infty}(A^{\star})^i = \cup_{i=0}^{\infty}(A^{\star}) = A^{\star}$. ■

2. $A^{\star} = \{\lambda\} \cup AA^{\star}$.

   Demostración. Se deduce del siguiente argumento:

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

   Es inmediato que $AA^{\star} \cup \{\lambda\} = \cup_{i=0}^{\infty} A^i = A^{\star}$. ■

3. $\emptyset^{\star} = \{\lambda\}$.

   Demostración. De la definición tenemos que $\emptyset^{\star} = \cup_{i=0}^{\infty}\emptyset^i$. Si $i = 0$, sabemos que para todo subconjunto $A \subseteq \Sigma$ se tiene que $A^0 = \{\lambda\}$, en particular para $\emptyset$. Por otro lado, para toda $i \geq 1$, se tiene que $\emptyset^i = \emptyset$. Se concluye que $\emptyset^{\star} = \{\lambda\}$. ■

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, 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, el problema de la codificación es ligeramente más simple. No requiere mapear a cada elemento de $A^{\star}$ a un elemento de $B^{\star}$. Basta trabajar sobre 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. Sean Σ y Ψ dos alfabetos. La función $\nu : \Gamma \subseteq \Sigma^{\star} \to \Psi^{\star}$ es igual a $\nu : \Gamma \to \Psi^{\star}$, donde $\Gamma \subseteq \Sigma$. La primera es una notación más compacta. Se enfatiza que Γ es el dominio de la función, no $\Sigma^{\star}$. Esta notación será particularmente útil para proponer codificaciones entre alfabetos, pues como mencionamos no es necesario hacerlo usando toda la clausura de Σ, sino basta un subconjunto Γ del mismo.

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

Ejemplo 1.3.1. Codifiquemos a $\mathbb{N}$ a partir del alfabeto $\Sigma = \{0, 1\}$. Proponemos la función $\nu : \Gamma \subseteq \Sigma^{\star} \to \mathbb{N}$, 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. Por ejemplo, ahí con 32 bits podemos generar 2^{32} números.

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

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

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

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

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

Ejemplo 1.3.4. Codificar el $\{0, 1, \#\}$ a partir de $\{0, 1\}$. Proponemos leer en cadenas de dos. Luego $\nu(00) = 0, \nu(01) = 1, \nu(10) = \#$.

Ejemplo 1.3.5. Consideremos un alfabeto arbitrario $A = \{a_1, a_2, ..., a_n\}$ y el alfabeto $\{0, 1\}$. Propongamos primero una codificación de $\{0, 1\}$ a partir de A. Sea $\mathbb{N}_n = \{a \in \mathbb{N} : a \leq n\}$. Definimos una codificación auxiliar válida (suprayectiva), con los naturales $\psi : A^\star \to \mathbb{N}_n$ como sigue: $\psi(a_i) = i$. Basta entonces definir una codificación de $\omega : \mathbb{N}_n \to \{0,1\}^\star$.

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, salvo que llenamos de ceros a la izquierda para que la codificación sea de exactamente k dígitos binarios y la codificación esté bien definida. Definimos $\omega(n) = n_2$, donde n_2 es tal representación binaria de n. Luego $(\omega \circ \psi) : A^\star \to \{0,1\}^\star$ es claramente sobre. Más aún, $(\omega \circ \psi)^-1 : \{0,1\}^\star \to A^\star$ es la codificación inversa, ya que ψ y ω son biyectivas.

Proponemos ahora una segunda codificación de a A a partir de $\{0, 1\}$. La función $\nu : \{0,1\}^\star \to A^\star$ definida como sigue: $\nu(0^{n-i}1^i) = a_i$ es sobre.

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.

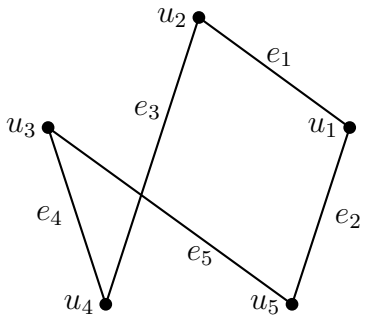
Definición 1.3.2. Una **gráfica** (o **gráfo**) es una pareja ordenada $G = (V, E)$ donde $V = \{v_1, v_2, ..., v_n\}$ es un conjunto no vacío de vértices. Mientras que E es el conjunto de aristas dado como sigue, $E = \{uv : u, v \in V\}$.

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.

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.

Definición 1.3.5. Un **lenguaje** es un subconjunto $A \subseteq \Sigma^\star$ de $\Sigma^\star$.

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 $\rightarrow$ Traducción a Lenguaje $\rightarrow$ Lectura de Máquina $\rightarrow$ Acepta\Rechaza.

Definición 1.3.6. 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 problemas como este podríamos usar una máquina de Turing para resolverlo. En el siguiente capítulo estudiaremos cómo se definen éstas.

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 éste, 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 $M = (Q, \Sigma, \delta, s, F)$, 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.
- $F \subseteq Q$; los elementos de F son llamados estados finales o estados de aceptación.

Ejemplo 2.0.1. Considere una sistema M donde $Q = \{0, 1, 2, 3\}$, $\Sigma = \{a, b\}, F = \{3\}, q_0 = 0$ 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 F 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. Éstas 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.

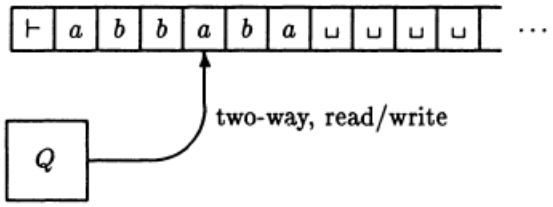
Se plantearon varios modelos, además de las máquinas de Turing. Por ejemplo los *Post systems*, μ-*recursive functions*, λ-*calculus*, *combinatory logic*. Todos son distintos pero es posible emular lo que hace cualquiera de estos modelos por cualquiera de los otros. Realmente, función computable no 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. Podemos pensarla con la siguiente imagen, en su versión clásica. 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 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. También requerimos que si la máquina está en estado de aceptación t o de rechazo b, entonces permanezca en el mismo hasta que se termine de leer la cinta. Esto es que para toda $b \in \Gamma$, existan $c, c' \in \Gamma$ y $d, d \in \{L, R\}$ tal que

$$\delta(t, b) = (t, c, d)$$
$$\delta(r, b) = (r, c', d')$$

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 $L(M) = \{w \in \Sigma^\star : M \text{ acepta a } w\}$.

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)	$(q_{10}, \sqcup, R)$	$(q_3, \dashv, L)$

Consideremos la transición de configuraciones de λ.

$$(q_0, \vdash \sqcup, 0) \xrightarrow[M]{1} (q_0, \vdash \sqcup, 1) \xrightarrow[M]{1} (q_3, \vdash \dashv, 0) \xrightarrow[M]{1} (q_y,$-$,$-$)$$

Ahora a la cadena "*abc*".

$$(q_0, \vdash abc\sqcup, 0) \xrightarrow[M]{2} (q_1, \vdash abc\sqcup, 2) \xrightarrow[M]{2} (q_2, \vdash abc\sqcup, 4) \xrightarrow[M]{1} (q_3, \vdash abc\ \dashv, 3) \xrightarrow[M]{3}$$
$$(q_6, \vdash \sqcup\sqcup\sqcup\ \dashv, 0) \xrightarrow[M]{4} (q_7, \vdash \sqcup\sqcup\sqcup\ \dashv, 4) \xrightarrow[M]{1} (q_y,$-$,$-$)$$

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 xRy están relacionados si $(x, y) \in R$.

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, $N = (Q, \Sigma, \Gamma, R, \vdash, \sqcup, q_0, q_y, q_n)$, donde las definiciones de los argumentos son 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, tipos 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, $f : A \subseteq \{1^n 01^m : n, m \in \mathbb{N}\} \to \Sigma^\star, f(1^n 01^m) = 1^{n+m}$. 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.

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 es $\Sigma = \{0, 1, \#\}$. Recibe como argumento una cadena larga partida por un separador $\#$. La primera parte es una *descripción codificada* de la MT M, el número de estados, símbolos, función de transición, etc. La segunda partes es la cadena x codificada en el alfabeto $\{0, 1\}$ que se desea simular. Sí M acepta a x entonces U también, si M rechaza a x entonces U también, si M se queda en ciclo entonces U también. La cadena codificada se denota $M\#x$ por obvias razones. La máquina universal de Turing MUT es el "equivalente" a una computadora convencional como una PC, lap, tablet, celular int.

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

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

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.

2.2. Lenguajes computables enumerablemente y computables

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

[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.

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.

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 se aceptan, pero los que no están dentro del alfabeto no podemos saber. En cambio, en el lenguaje recursivo nunca caeremos en un ciclo infinito.

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

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

Propiedades de los conjuntos recién definidos

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

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

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".

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

2.3. Imposibilidad del problema de detención

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

Demostración. Usaremos como motivación las proposiciones que se siguen a continuación de esta prueba. Para $x \in \{0,1\}^\star$ consideramos la máquina de Turing $M_\star$ con alfabeto $\{0,1\}$, la cual viene descrita por la cadena δ. 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 la posición $M_{x,y}$ la matriz contiene una H si M_x se detiene en y. Si M_x entra en un ciclo con y entonces la entrada tiene una L.

	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	...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Por contradicción supongamos que $HP \in \mathcal{C}$, es decir que es computable. Entonces existe una máquina de turing total $\mathcal{K}$ tal que $L(\mathcal{K}) = HP$, es decir para $M\#x \in \{0,1,\#\}^\star$. $\mathcal{K}$ se detiene y acepta si M se detiene en x. $\mathcal{K}$ se detiene y rechaza si M entra en un ciclo 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) Ejecuta a $\mathcal{K}$ en la entrada $M_x\#x$ aceptando si K rechaza y entra en un ciclo si $\mathcal{K}$ acepta. 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 $\mathcal{N}$. Podemos ver que $\mathcal{K}$ no existe. Por lo tanto HP no es computable.

Como resumen, 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 $x_\mathcal{K}$, entonces $\mathcal{N}$ se encicla con $x_\mathcal{K}$. La otra contradicción es análoga en el caso que $\mathcal{N}$ no se detiene en $x_\mathcal{N}$. ■

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.

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

Como f es sobre, todo subconjunto de $\mathbb{N}$ es un renglon de la matriz. Pero ahora construimos un conjunto $B \subseteq \mathbb{N}$ que no aparece en la matriz. Este conjunto se forma con el complemento de bits de la diagonal de la matriz. En este caso la diagonales 0100... y su complemento es 1011... Por lo tanto B difiere del conjunto $f(k)$ justamente en el elemento k. Encontré un elemento que no está en la matriz. Por eso f sobre no existe. ■

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

2.4. Reducciones Computables

Definición 2.4.1. Decimos que σ es una **función computable** si es total y es efectivamente computable. Esto quiere decir que σ es computable por una MT total que se detiene cuando en su cinta contiene a $\sigma(x)$.

Definición 2.4.2. Sean Σ, Δ alfabetos y $A \subseteq \Sigma^\star, B \subseteq \Delta^\star$ conjuntos. Una **reducción** (muchos a uno) de A en B es una función computable $\sigma : \Sigma^\star \to \Delta^\star$ 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*.

La reducción construye a las máquinas más no ejecuta la instancia. Si no correría el riesgo de caer en un ciclo infinito.

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 tal que $L(M) = A$. 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 una MT tal que en la entrada $x \in \Sigma^\star$ hace lo siguiente:

1) Calcula $y = \sigma(x)$.

2) Ejecuta a M_B en y.

3) Acepta a x si M_B acepta a y.

Como $x \in L(M) \Leftrightarrow M_B$ acepta a $\sigma(x) \Leftrightarrow \sigma(x) = y \in B \Leftrightarrow x \in A$, 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. El 1) es por hipótesis, está dado implícitamente al existir una reducción de $A \leq_m B$. El 2) es porque M_B se hace en tiempo polinomial. El 3) es porque M termina también cuando la simulación de B termina. Por lo tanto $A \in \mathcal{CE}$

b) Supongamos que B es computable. Por hipótesis, $A \leq_m B$, entonces existe una reducción $\sigma : \Sigma^\star \to \Delta^\star$. Si $B \in \mathcal{C}$ entonces existe una MT M_B total tal que $L(M_B) = B$. Sea M una MT tal que 1) Calcula $y = \sigma(x)$. 2) Ejecuta a M_B en y. 3) Si M_B acepta a y entonces M acepta a x. 4) Si M_B rechaza a y entonces M rechaza a x. La correctez y el tiempo polinomial del algoritmo se argumentan de forma idéntica que en a). Finalmente, M es total porque M_B lo es. Por lo tanto A es computable.

Otra prueba de b) sería como sigue. Por el Ejercicio **??** 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 $x \in \Sigma^\star - A \Leftrightarrow x \notin \Sigma^\star - B$. 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 **??** se sigue que $A \in \mathcal{C}$. ■

Ahora mostraremos cómo usar el Teorema 2.4.1 para determinar. Por ejemplo, en el Ejercicio **??** 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 prima N denota la configuración de N como cadena, no N en sí.*

Demostración. Recordemos que $HP^c = \{M\#x : M \text{ no se detiene en } x\}$. Buscamos una reducimos de HP^c a 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 $|L(N)| < \infty$.

En el caso de cadenas invalidas $M\#x$ podemos suponer que llega a un 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 $FIN^c \notin \mathcal{CE}$. Note que la misma reducción para probar que $HP \leq_m FIN$ funciona para probar que $HP^c \leq_m FIN^c$. Definiremos entonces la primera, esto es una reducción $\sigma : \Sigma^\star \to \Sigma^\star$ tal que $x \in HP \Leftrightarrow \sigma(x) \in FIN$. Esto es, dado $M\#x$, construir una MT $N = \sigma(M\#x)$, tal que $x \in HP \Leftrightarrow |L(N)| < \infty$.

Se define σ similar a como en el Resultado 2.4.1. Se construye la máquina N como sigue: 1) Guarda el *input* y por separado. 2) Guarda a x en su cinta. 3) Simula a x en M por $|y|$ 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 $M\#x \notin HP$, entonces M no se detiene en x jamás. Luego, para toda y la MT N acepta, se sigue que $|L(N)| = \infty$. Sea $M\#x \in HP$, entonces M se detiene en x, digamos tras exactamente k pasos. Si y sólo si $|y| < k$, tras $|y|$ pasos M todavía no se detiene y por lo tanto acepta a y. Existe un número finito de cadenas y tal que satisfacen que $|y| < k$. Se sigue que $|y| < k \Leftrightarrow |L(N)|\infty \Leftrightarrow \sigma(M\#x) \in FIN$. Por lo tanto, si $M\#x \in HP \Leftrightarrow N \in FIN$. ■

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. Entonces si M acepta o rechaza a x, N simplemente acepta. Esto se logra con un paso adicional de N. Definimos $y = x$, así que ahora $\sigma(M\#x) = N\#y$ está definido. Basta notar que, $M\#x \in HP \Leftrightarrow M$ se detiene en $x \Leftrightarrow N$ acepta a $y \Leftrightarrow y \in L(N) \Leftrightarrow N\#y \in MP$. ■

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

3. Clases de Complejidad

3.1. Notación asintótica y funciones de complejidad apropiadas

Toda 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 $x \in \Sigma^\star$. Consideramos, *nautralesn* el tamaño de la instancia, definido como $n := |x|$. Para medir el tiempo y espacio usado por un algoritmo MT empleamos funciones "funciones de complejidad apropidas".

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

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. El **espacio** consumido por una MT M en decidir a $x \in \Sigma^\star$, es el número máximo de celdas usadas por M 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 $L\{a^n b^n c^n : n \leq 0\}$. Tenemos lo siguiente.

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

$$(q_0, \vdash, 0) \underset{M}{\overset{2}{\to}} (q_y, \vdash\dashv, \text{-})$$

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

$$(q_0, \vdash abc, 0) \underset{M}{\overset{5}{\to}} (q_0, \vdash abc \dashv, 3) \underset{M}{\overset{4}{\to}} (q_y, \text{-}, \text{-})$$

3. Para $aabbcc$ hacemos 22 pasos:

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

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.

¿Cuál es el espacio que ocupa M para decidir L? La máquina nunca ocupa fuera del símbolo $\vdash$. 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: $S(1^n 0 1^m) = 1^{n+m}$. Por ejemplo, si recibe la cadena $1^3 0 1^4$, la MT hace

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

luego recorrer todos los símbolos implica $(n+m+1+2)$ movimientos y un paso adicional para convertir el último 1 a 0. 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 $R(n,m) = n+m+3$.

Pasamos ahora al contexto de notación asintótica, donde se simplificará el análisis de tiempo y la memoria pues bastará clasificar el algoritmo según 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 $\Theta(g(n)) = \{f : \mathbb{N} \to \mathbb{N} : \exists n_o \in \mathbb{N}, c_1, c_2 \in \mathbb{R}^+, 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n), \forall n \geq n_0\}$.

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

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

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 era $f(n) = 3n^2+4n+2$. Por lo tanto se sigue que $f(n) = O(n^2)$. El espacio estaba dado por $S(n) = 3n+2$, por lo tanto $S(n) = \Omega(n)$.

b) Para el ejemplo 3.1.2, la función de complejidad era $n+m+2$. Si $p = n+m$, entonces $f(p) = O(p)$. Para el espacio $s(p) = O(p)$, también.

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

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 $a_d > 0$.

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 $n!, \exp, c\log^k n$.

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 continua 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 entra en rechazo.

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.

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.

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,

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

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.

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

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

11) 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 anteriors. 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 a parte.

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

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

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-$\boldsymbol{P} = \boldsymbol{P}$

b) co-$\boldsymbol{L} = \boldsymbol{L}$

c) PSPACE = co-PSPACE

d) EXP = co-EXP

Demostración. Sólo se prueba a), el resto son análogos.

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

■

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.

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

Asumimos que el lenguaje de cada MT M está compuesto por símbolos usados para codificar cosas "útiles" (0,1,(,),",,"), entonces $x \in \Sigma^\star$ es dado tal cual.

Teorema 3.2.1. *(Jerarquía del tiempo) Si $f(n) \geq n$ es una función de complejidad apropiada entonces TIME$(f(n)) \subsetneq$ TIME$(f^3(2n+1))$*

Hemos dicho que las constantes "no importan" para fines prácticos y este Teorema fortalece lo que decimos. Los avances en hardware reducen las constantes. 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 3.2.1. *$MP_f \in$ TIME$(f^3(n))$*

Demostración. Construimos una MT U_f con 4 cintas la cual decide a $M_f\#x$ en tiempo $O(f^3(n))$. Es decir, recibe a $M_f\#x$, simula, y no tarda más de $f(|x|)$ pasos en decidir. U_f está basada en varias máquinas:

- La máquina universal U.
- El simulador de máquinas con k-cintas en una máquina con 1 cinta.
- La máquina del teorema del aceleramiento lineal.
- La máquina que calcula a $f(n)$.

Si alguna máquina usa más cintas, agregamos más cintas a U_f.

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 $f(|x|)$. Esto lo hace en $O(f(|x|))$, donde la constante depende sólo de f y no de M_f o x.
2. U_f copia la descripción de M_f, la máquina a ser simulada en su tercera cinta.
3. U_f inicializa la segunda cinta con el estado inicial q_0 correspondiente a la simulación de M_f. 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 $O(f(|x|)+|x|) = O(f(|x|)$. Esta última igualdad porque por hipótesis $f(n) \geq n$. A continuación describimos el ciclo de operación de U_f:

4. Primero se hace un escaneo en la primera cinta para saber que símbolos son los que M_f tiene en las cabezas lectoras y estos son escritos en la cinta numérica 2. Recuerda que se simula una máquina de k cintas. Ya que los transcribe en la segunda cinta, busca la combinación de estados y símbolos de la segunda cinta en las transiciones de la codificación de M_f en la tercera cinta y ejecuta la transición apropiada, modificando el contenido de la primera cinta y el estado actual en la segunda cinta. También borra los símbolos después del estado). Por último tacho una unidad de tiempo de la alarma, o en otras palabras incrementa su reloj de alarma en 1.

   ¿Cuánto tiempo tarda U_f en simular un paso de M_f? Definimos k el número de cintas de la máquina M_f y l es la longitud de la descripción de estados y símbolos de M_f. Luego U_f tarda $lk^2f(n)$ para ejecutar un paso de M_f. Se puede probar que k y l están acotados por $\log|M|$. Pero $k^2l \in O(\log|M|) \subseteq O(n) \subseteq O(f(n))$ Demostrar la primera inclusión para puntos extra. Por lo tanto, simular un paso es de orden $O(f^2(n))$.

   El tiempo total de la simulación será de $O(f^3(n))$ (pues U_f simulará a lo más $f(|x|)$ pasos de M, por el reloj de alarma, donde $f(|x|) = O(f(n))$, porque x es una parte proporcional de n. Se puede hacer el tiempo de simulación algo muy cercano a $f^3(n)$ modificando a U_f con el Teorema del aceleramiento lineal. Por lo tanto $M_f \in \text{TIME}(f^3(n))$.

■

Lema 3.2.2. *$MP_f \notin TIME(f(\lfloor \frac{n}{2} \rfloor))$*

Demostración. Por contradicción, supongamos que $MP_f \in$TIME$(f(\lfloor \frac{n}{2} \rfloor))$. Así que existe una MT con k-cintas K_f que decide a MP_f en tiempo $f(\lfloor \frac{n}{2} \rfloor)$. La máquina K_f nos lleva a construir una máquina "diagonalizadura" D_f, con el siguiente programa:
$D_f(M)$:
    K_f acepta a $M\#M$ then
        rechaza
    else
        acepta

En la entrada M, D_f corre en el mismo tiempo que corre K_f, es decir si $n = |M|$, K_f corre en tiempo

$$f\left(\lfloor\frac{2n+1}{2}\rfloor\right) = f\left(\lfloor n + \frac{1}{2}\rfloor\right)$$

Ahora nos preguntamos ¿Qué pasa si D_f corrre en sí misma?. Supogamos 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 $f(n)$, pero supongamos que

$$L(K_f) = MP_f = \{M\#x : M \text{ acepta a } x \text{ en tiempo } f(|x|)\}$$

.
D_f acepta a D_f y esto debe pasar en tiempo acotado por $f(n)$. Esto es una contradicción. Similarmente, llegamos a una contradicción. Si suponemos que D_f rechaza a D_f. Concluimos que K_f no existe y por lo tanto $MP_f \notin \text{TIME}(f(\lfloor \frac{n}{2} \rfloor))$. ■

Teorema 3.2.2. *(Teorema del Aceleramiento Lineal). Sea $L \in TIME(f(n))$. Entonces, $\forall \epsilon > 0$, $L \in TIME(f_\epsilon(n))$, donde $f_\epsilon(n) = \epsilon f(n) + n + 2$.*

Demostración. Por los lemas anteriores se tiene que TIME $(f(\lfloor \frac{m}{2} \rfloor)) \subseteq \text{TIME}(f^3(n))$, haciendo $m = 2n + 1$, tenemos que

$$f\left(\lfloor \frac{m}{2} \rfloor\right) = f\left(\lfloor \frac{2n+1}{2} \rfloor\right) = f\left(\lfloor n + \tfrac{1}{2} \rfloor\right) = f(n)$$

Se concluye que $\text{TIME}(f(n)) \subsetneq \text{TIME}(f^3(2n+1))$ ■

Corolario 3.2.2. *$P \subsetneq EXP$*

Demostración. Por el teorema de jerarquía, P $\subseteq$ TIME(2^n) $\subsetneq$ TIME$((2^{2n+1})^3)$ Se concluye que P $\subsetneq$ TIME$((2^{2n+1})^3) \subseteq$ EXP ■

Teorema 3.2.3. *Jerarquía del espacio. Si $f(n)$ es una función de complejidad apropiada. Entonces $SPACE(f(n)) \subsetneq SPACE(f(n)\log(f(n)))$*

Demostración. Prueba vale 3 puntos Hartmanis. ■

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. En la gráfica hay un camino dirigido de 1 al 5, pero si cambiamos el arco (4,3) de dirección a (3,4) entonces no hay camino dirigido de 1 al 5.

Teorema 3.2.4. *Reachability $\in TIME(n^2)$ donde n es el número de vértices de G.*

Teorema 3.2.5. *Sea $f : \mathbb{N} \to \mathbb{N}$ una funcion 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) $NSPACE(f(n)) \subseteq TIME(k^{f(n)})$.

Demostración.

a) Es trivial

b) Es trivial

c) Sea $L \in \text{NTIME}(f(n))$. Así que existe N una máquina de Turing no determinística que decide a 2 en tiempo "$f(n)$". Construimos una MTD S que decide L en espacio $f(n)$.

■

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 $d = \text{máx}_{q,\sigma} |C_{\sigma,q}|$. 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 exponeicial. Así que $L \in$SPACE$(f(n))$.