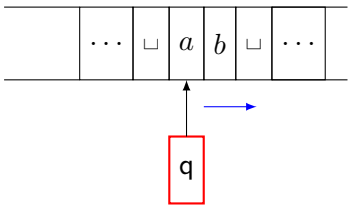
Autómatas y Lenguajes Formales 2019-II
Facultad de Ciencias UNAM
Nota de Clase 9, Máquinas de Turing

Favio E. Miranda Perea A. Liliana Reyes Cabello Lourdes González Huesca

3 de febrero de 2020

1. Máquinas de Turing

Las máquinas de Turing (MT) son máquinas idealizadas capaces de realizar cómputos. Una máquina de Turing consiste de una cinta infinita dividida en sectores (cuadros) y una cabeza de lecto-escritura. Cada sector de la cinta contiene un símbolo de cierto alfabeto de entrada o bien el símbolo blanco. La cabeza lee el sector y puede escribir sobre él así como moverse a la izquierda o a la derecha.



Definición 1 *Una máquina de Turing clásica de una cinta es una tupla de la forma*

$$T = \langle \Sigma, Q, q_0, q_f, \delta \rangle,$$

donde

- $\Sigma \neq \varnothing$ *es un alfabeto finito que contiene un símbolo distinguido* $\sqcup$*, llamado símbolo blanco,*
- $Q \neq \varnothing$ *es el conjunto finito de estados, el cual incluye* q_0 *y* q_f*,*
- q_0 *es el estado inicial,*
- q_f *es el estado final de aceptación,*
- δ *es una función de transición con dominio es un subconjunto de* $Q \times \Sigma$ *y cuyo contradominio es* $Q \times \Sigma \times M$*. De tal forma que si* δ *está definida*[1] *para el par* $(\ell, s) \in Q \times \Sigma$ *entonces:*

$$\delta(\ell, s) = (\ell', s', m)$$

significa que:

[1] El lector experimentado observará entonces que δ es una función parcial con dominio $Q \times \Sigma$, lo cual se suele denotar como $\delta : Q \times \Sigma \rightharpoonup Q \times \Sigma \times M$

- ℓ *es el estado actual,*
- s *el símbolo que está leyendo la cabeza,*
- ℓ' *el estado al cual nos llevará la transición,*
- s' *el símbolo que se escribirá,*
- m *el movimiento que realizará la cabeza.*

En nuestra definición consideramos que $M = \{\leftarrow, \rightarrow, -\}$ *es el conjunto de movimientos realizados por la cabeza lectora ya sea a la izquierda* $\leftarrow$*, a la derecha* $\rightarrow$ *o bien permanecer en la misma posición* $-$*.*

Es importante observar que hay muchas variaciones en lo que respecta a la definición de una máquina de Turing clásica. Por ejemplo, algunas fuentes utilizan un conjunto de estados finales F en vez de un único estado final, mientras que otros distinguen entre el alfabeto de entrada Σ el cúal cumple que $\sqcup \notin \Sigma$ y el alfabeto de la cinta Σ, de tal forma que $\sqcup \in \Sigma$ y $\Sigma \subset \Sigma$. Nosotros preferimos la definición recien dada.

Ejemplo: Considere la siguiente máquina

$$M = \langle \{q_0, q_1\}, \{a, b, \sqcup\}, q_0, q_1, \delta \rangle$$

donde la función de transición está definida por:

$$\delta(q_0, a) = (q_0, b, \rightarrow) \qquad \delta(q_0, b) = (q_0, b, \rightarrow) \qquad \delta(q_0, \sqcup) = (q_1, \sqcup, \leftarrow)$$

Ejemplo: Considere el lenguaje $L = \{a^n b^n \mid n \geq 1\}$, una MT que reconoce cadenas del lenguaje está determinada por la siguiente función de transición donde q_4 es el estado final:

δ	a	b	X	Y	$\sqcup$
q_0	$(q_1, X, \rightarrow)$			$(q_3, Y, \rightarrow)$	
q_1	$(q_1, a, \rightarrow)$	$(q_2, Y, \leftarrow)$		$(q_1, Y, \rightarrow)$	
q_2	$(q_2, a, \leftarrow)$		$(q_0, X, \rightarrow)$	$(q_2, Y, \leftarrow)$	
q_3				$(q_3, Y, \rightarrow)$	$(q_4, \sqcup, \rightarrow)$
q_4					

1.1. Representación gráfica de una Máquina de Turing

Como hemos visto en otras máquinas o autómatas, existe una representación gráfica que modela la transición entre estados:

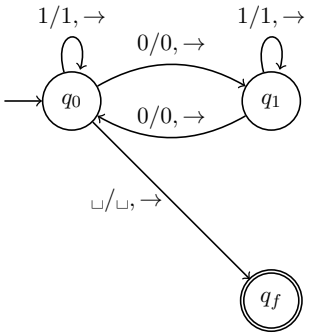
- cada nodo es un estado;
- las aristas dirigidas son las transiciones y están etiquetadas por el símbolo que lee la cabeza, el símbolo que escribirá y el movimiento que realizará;

Ejemplo: Una máquina de Turing que acepta al lenguaje

$$L = \{w \in \{0,1\}^\star \mid w \text{ tiene un número par de ceros }\}$$

está determinada por la función de transición siguiente y se describe gráficamente como:

δ	0	1	$\sqcup$
q_0	$(q_1, 0, \rightarrow)$	$(q_0, 1, \rightarrow)$	$(q_f, \sqcup, \rightarrow)$
q_1	$(q_0, 0, \rightarrow)$	$(q_1, 1, \rightarrow)$	
q_f			



Existen diferentes versiones de máquinas de Turing, cada una para ser utilizada en casos especiales. Estudiemos primero la formalización estándar de ellas y enfatizaremos sus características.

1.2. Máquina Estándar de Turing

La máquina estándar de Turing es una MT con las siguientes características[2]:

- La máquina tiene una cinta infinita en ambas direcciones y se permite un número arbitrario de movimientos en cualquier dirección.
- La máquina es determinista, es decir δ define a lo más un movimiento para cada configuración posible.
- No hay transiciones desde el estado final, es decir, $\delta(q_f, a)$ no está definida en ningún caso.
- No hay un archivo especial de entrada o salida, se asume que la máquina contiene algo al final y al principio del proceso.
- Se considera que la cadena a procesar está almacenada en algún lugar[3] de la cinta, es decir cada símbolo de la cadena está en un sector de la cinta y todos los demás sectores están en blanco, es decir, contienen el símbolo $\sqcup$.
- Al inicio del proceso la cabeza de la cinta se encuentra en el sector inmediato anterior a la cadena.

Definición 2 *Una configuración o descripción instantánea*

$$a_1 a_2 \ldots a_{k-1} \boldsymbol{q} a_k a_{k+1} \ldots a_n$$

está determinada por:

[2]Estás características no son restricciones pero simplifican el razonamiento y las demostraciones
[3]Puesto que el inicio de la cinta no existe

- *El estado actual de la unidad de control (cabeza).*
- *El contenido de la cinta.*
- *La posición de la unidad de control.*
- *La configuración inicial es:* $q_0 w$

Definición 3 *Sea* $T = \langle \Sigma, Q, q_0, q_f, \delta \rangle$ *una máquina de Turing. Una configuración instantanea es un par*

$$\langle q, l\underline{s}r \rangle$$

donde $q \in Q$ *y* $l\underline{s}r \in \Sigma^\star$ *y* s *es el símbolo actual, es decir, el símbolo que está leyendo la cabeza de* T*.*
La relación de transición $\vdash$ *entre configuraciones se define a partir de la función de transición* δ *como sigue: si* $\delta(q, s) = (p, s', \leftarrow)$ *entonces*

$$\langle q, ls'\underline{s}r \rangle \vdash \langle p, l\underline{s'}sr \rangle$$

y análogamente para los movimientos $\rightarrow, - \in M$*, agregando un blanco* $\sqcup$ *al extremo de la cadena en caso de ser necesario.*

Como es costumbre denotamos con $\vdash^\star$ a la cerradura reflexiva transitiva de $\vdash$. Es decir, si c, c' son dos configuraciones instantaneas entonces $c \vdash^\star c'$ si y sólo si existen configuraciones instantaneas $c_0, \ldots, c_n$ tales que $c_0 = c,\ c_n = c'$ y $c_0 \vdash c_1, c_1 \vdash c_2, \ldots, c_{n-1} \vdash c_n$. Obsérvese que en particular se cumple $c \vdash^\star c$ para cualquier configuración instantanea c.

El lenguaje aceptado por una máquina de Turing es intuitivamente el conjunto de todas las cadenas cuya ejecución termina en el estado final de la máquina y se define formalmente mediante la noción de configuración de la siguiente manera

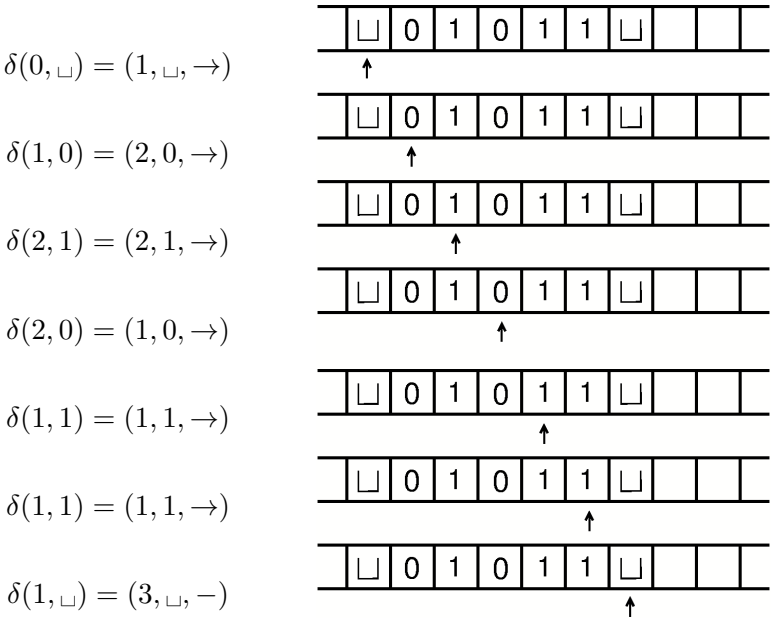
Definición 4 *Sea* $T = \langle \Sigma, Q, q_0, q_f, \delta \rangle$ *una máquina de Turing. El lenguaje aceptado por* T*, denotado* $L(T)$ *se define como*

$$L(T) = \{x \in \Sigma^\star \mid \langle q_0, \underline{\sqcup}x \rangle \vdash^\star \langle q_f, l\underline{s}r \rangle\}$$

Ejemplo 1 *La siguiente máquina de Turing* $T = \langle \{0, 1, \sqcup\}, \{0, 1, 2, 3\}, 0, 3, \delta \rangle$ *verifica que exista un número par de ceros en la cadena de entrada.*

δ	0	1	$\sqcup$
0	$-$	$-$	$(1, \sqcup, \rightarrow)$
1	$(2, 0, \rightarrow)$	$(1, 1, \rightarrow)$	$(3, \sqcup, -)$
2	$(1, 0, \rightarrow)$	$(2, 1, \rightarrow)$	$-$
3	$-$	$-$	$-$

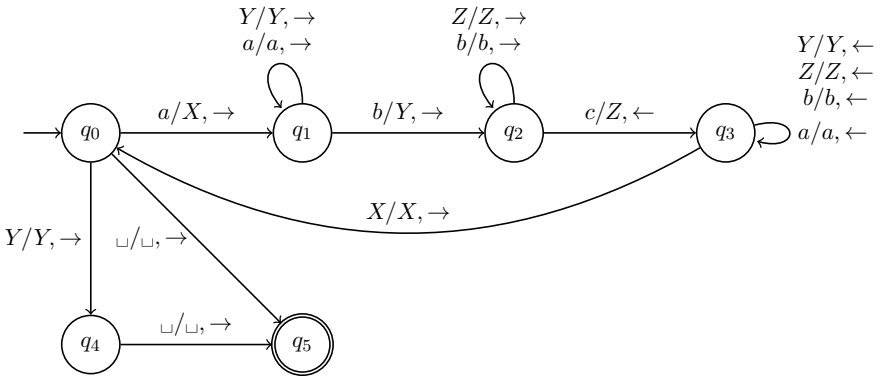
donde 0 *es el estado inicial y* 3 *el estado final. Verifiquemos, por ejemplo que* $\langle 0, \underline{\sqcup}01011 \rangle \vdash^\star \langle 3, \sqcup 01011\underline{\sqcup} \rangle$ *y por lo tanto* $01011 \in L(T)$*. En la siguiente figura en vez de usar las configuraciones directamente apelamos a la idea intuitiva de la cinta donde la flecha indica el sector donde está la cabeza de la máquina.*



Observaciones: **Las máquinas estándar se denominan máquinas aceptadoras ya que determinan si una cadena pertenece a un lenguaje:**

- A diferencia con los autómatas se puede aceptar una cadena en el momento en que el proceso llega a un estado final.
- **NO** es necesario consumir toda la cadena, es decir, se deben contemplar w_1 y w_2 cadenas, inclusive pueden ser una subcadena de w.
- Si $\varepsilon \in L$ para algún lenguaje entonces la cinta no contiene información al inicio y se acepta al avanzar sólo un sector vacío para llegar al estado de aceptación.
- Una máquina que acepta el lenguaje vacío debe aceptar al tener una transición hacia el estado final.
- Recordemos que se asume en el modelo estándar que no hay transición alguna desde un estado final, lo cual evita ambigüedades.
- Si la máquina se detiene en un estado no final o se encuentra bloqueada se puede concluir que la cadena a procesar no pertenece al lenguaje.

Ejemplo: La siguiente máquina acepta el lenguaje $L = \{a^i b^i c^i \mid i \geq 0\}$



Se observa que esta máquina no es estándar (¿ Por qué ?). Modifiquela para que sí lo sea. Además describa la idea de su funcionamiento.

2. Variaciones en MT

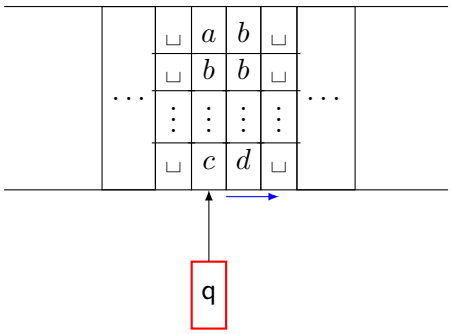
Existen diversas variaciones en la definición de máquinas de Turing. Todas ellas resultan equivalentes, es decir, el poder de computación de cualquier modelo resulta equivalente al de la máquina estándar. Las variaciones son útiles para simplificar la presentación o programación de diversos problemas.

1. MT con múltiples pistas

   - Idea: la cinta se divide en multiples pistas.
   - La función de transición es:

   $$\delta : Q \times \Sigma^n \rightarrow Q \times \Sigma^n \times \{\leftarrow, \rightarrow\}$$

   es decir $\delta(q, \langle a_1, \ldots, a_n \rangle) = (p, \langle b_1, \ldots, b_n \rangle, D)$ con $D \in \{\leftarrow, \rightarrow\}$.



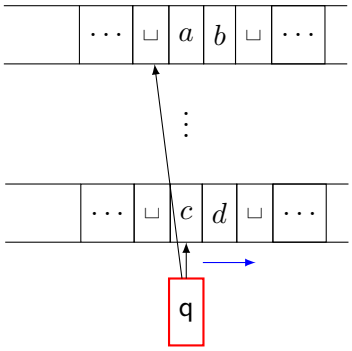
2. MT con múltiples cintas

   - Idea: se agregan más cintas a la máquina.

- La función de transición es:

$$\delta : Q \times \Sigma^n \to Q \times (\Sigma \times \{\leftarrow, \rightarrow\})^n$$

es decir $\delta(q, \langle a_1, \ldots, a_n \rangle) = (p, \langle b_1, D_1 \rangle, \ldots, \langle b_n, D_n \rangle)$ con $D_i \in \{\leftarrow, \rightarrow\}$.



3. MT No-deterministas

- La función de transición es:

$$\delta : Q \times \Sigma^n \to \mathcal{P}(Q \times \Sigma \times \{\leftarrow, \rightarrow\})$$

es decir $\delta(q, \langle a_1, \ldots, a_n \rangle) = \{\langle b_1, D_1 \rangle, \ldots, \langle b_n, D_n \rangle\}$

- Las máquinas no-deterministas juegan un papel central en la teoría de la complejidad.

La MT de Turing tiene suficiente poder para ser usada para otras tareas además de reconocer lenguajes.

3. MT Calculadoras

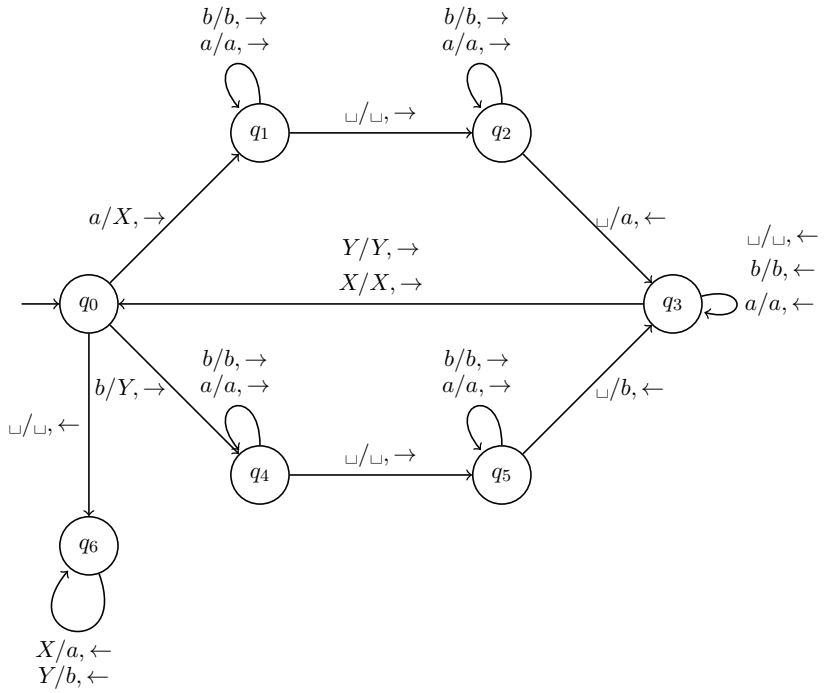
La máquina de Turing

$$M = \langle Q, \Sigma, \delta, q_0, q_f \rangle$$

calcula una función $\mathfrak{F} : \Sigma^\star \to \Sigma^\star$ si

$$\langle q_0, \sqcup w \rangle \vdash^\star \langle q_f, \sqcup v \rangle \quad \text{donde } \mathfrak{F}(w) = v$$

Ejemplo: Un macro que copia una cadena se basa en la idea de recorrer la cadena de entrada almacenada en la cinta símbolo por símbolo, para copiarlo en los sectores posteriores a la cadena pero usando un sector en blanco para separar las cadenas.
La siguiente máquina copia cadenas sobre el alfabeto $\Sigma = \{a, b\}$:



A partir de esta máquina construya una máquina que compute la función $f(x) = 2x$ donde x es un número natural en notación unaria.

4. MT Generadoras

Así como hay variantes en las máquinas aceptadoras, se puede hablar de máquinas generadoras que tienen suficiente poder como para generar lenguajes. Una máquina de Turing M con múltiples cintas genera al lenguaje $L \subseteq \Sigma^\star$ si

- M comienza a operar con las cintas en blanco en q_0.
- Cada vez que M regresa a q_0 hay una cadena de L escrita en alguna de las cintas (la cinta designada). Alternativamente la máquina podría no volver nunca a q_0 y ciclarse en otro estado de forma que al volver a dicho estado la cadena impresa en la cinta designada pertenece al lenguaje
- Eventualmente se generan todas las cadenas de L que se almacenarán en la cinta designada separada por un espacio en blanco o algún otro símbolo para este efecto.

Ejemplo: Analicemos una máquina que genera todas las cadenas del lenguaje: $L = \{a^i b^i c^i \mid i \geq 0\}$. Utilizaremos dos cintas y la idea del diseño es la siguiente:

1. Se inicia con la cinta $C1$ vacía y se dejan dos símbolos # para indicar que la cadena vacía pertenece al lenguaje. $C1$ es la cinta designada
2. Simultáneamente se escribe una a en la cinta $C2$ y la cabeza lectora retrocede

3. A continuación se generarán las cadenas utilizando un ciclo que no termina como sigue:

   a) En la cinta $C1$ se escribe una a por cada a en la cinta $C2$

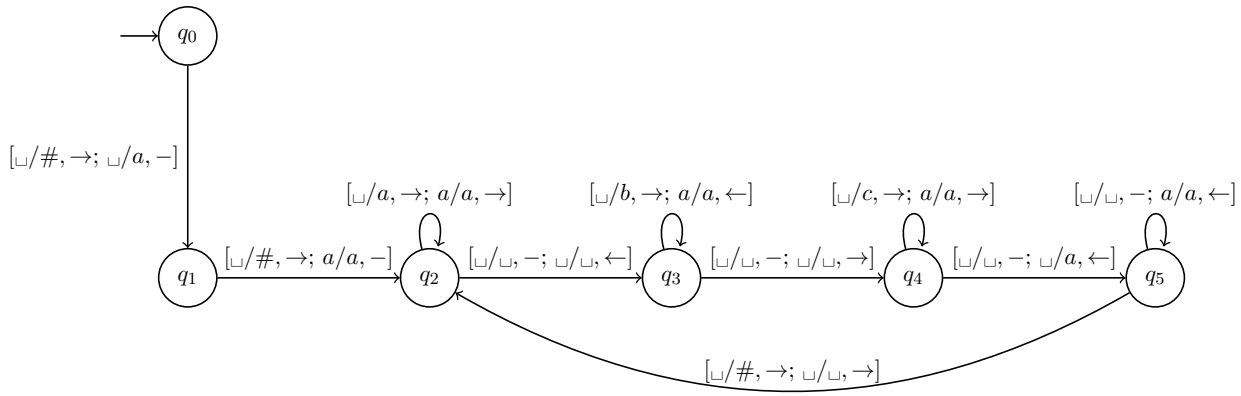
   b) La cabeza en $C2$ regresa a la izquierda para escribir b en $C1$ por cada a que lea

   c) Nuevamente se mueve la cabeza a la derecha en $C2$ para escribir en $C1$ tantas c como a

   d) Se incrementa una a en la cinta $C2$ para generar la siguiente cadena

   e) Se escribe el símbolo # en $C1$ para separar las cadenas

Veamos su diagrama:



5. Lenguajes Recursivos y Recursivamente enumerables

La aceptación en una máquina de Turing definen unas clase de lenguajes muy expresivos:

- Un lenguaje L es **recursivamente enumerable** si es reconocido por una máquina de Turing, es decir, si existe una máquina de Turing M tal que $L = L(M)$.
- Un lenguaje L es **recursivo** si es reconocido por una máquina de Turing que **siempre se detiene**, es decir, si existe una máquina de Turing M que se detiene con todas las cadenas de entrada y $L = L(M)$.

Los lenguajes definidos gozan de algunas propiedades de cerradura:

- Si L es recursivo entonces $\overline{L}$ es recursivo.
- Si L, M son recursivos entonces $L \cup M$ es recursivo.
- Si L, M son rec. enumerables entonces $L \cup M$ es rec. enumerable.
- L es recursivo si y sólo si L y $\overline{L}$ son rec. enumerables.
- Todo lenguaje recursivo es recursivamente enumerable.

- Existen lenguajes recursivamente enumerables que no son recursivos. En particular el lenguaje universal $\mathcal{L}_{\mathcal{U}}$ no es recursivo.
- Existen lenguajes no recursivamente enumerables como el lenguaje diagonal $\mathcal{L}_D$.
- Para definir estos lenguajes es necesario antes presentar a la máquina universal de Turing.

6. La Máquina Universal

6.1. Introducción

- Una computadora es capaz de interpretar algoritmos arbitrarios y obtener la misma respuesta que cada algoritmo particular.
- Entonces una computadora es una máquina útil para propósitos generales
- Las MT en cambio son diseñadas para propositos particulares.
- Conclusión: el poder computacional de las MT no puede ser equiparable al de las computadoras actuales.
- Las computadoras son programables, las MT no.
- ¿Será factible pensar en la existencia de una MT que se comporte de la misma forma que una computadora real?
- Es decir, una MT que sea útil para propósitos multiples.
- Dicha máquina sería capaz de programar y ejecutar máquinas de Turing.
- Tal máquina existe y se conoce como máquina universal de Turing (MUT).
- La MUT recibe como entrada una descripción de una MT, M y una cadena w y simula el comportamiento de M sobre w.
- Los datos de entrada M y w deben ser codificados de manera adecuada.

6.2. Codificación de MT

- Se fija un alfabeto de entrada §.
- Se asume $Q = \{q_0, \ldots, q_n\}$ siendo q_0 el estado inicial y q_1 el **único** estado final.
- El alfabeto es de la forma
$$\Sigma = \{s_1, s_2, \ldots, s_p\}$$
siendo $s_1 = \sqcup$.

$$\Sigma = \{s_1, s_2, \ldots, s_p\}$$

- El símbolo s_i se codifica como 1^i.
- Las cadenas de $\Sigma^\star$ se codifican separando cada símbolo con 0.
- Por ejemplo, si $\Sigma = \{\sqcup, a, b\}$ entonces codificamos $\sqcup := 1,\ a := 11,\ b := 111$.
- La palabra $bab\sqcup aa$ se codifica como:

$$01110110111010110110$$

$$\Sigma = \{s_1, s_2, \ldots, s_p\}$$

- En general, si $w = s_{n_1} s_{n_2} \ldots s_{n_k}$ la codificación de w es:

$$01^{n_1}01^{n_2}0 \ldots 01^{n_k}0$$

$$Q = \{q_0, q_1, \ldots, q_n\}$$

- El estado q_i se codifica, análogamente a los símbolos, mediante cadenas de unos pero con un símbolo más que el índice del estado

$$q_i \text{ se codifica como } 1^{i+1}$$

- Con la convención tomada, el estado inicial q_0 se codifica con 1 y el estado final q_1 con 11.

$$\{\rightarrow, \leftarrow, -\}$$

- $\rightarrow$ se codifica con 1.
- $\leftarrow$ se codifica con 11.
- $-$ se codifica con 111.

$$\delta(q_i, s_k) = (q_j, b_\ell, D)$$

- Los estados, símbolos y dirección de desplazamiento se codifican de la manera indicada anteriormente.
- La transición se codifica escribiendo en orden los códigos respectivos separados por ceros:

$$01^{i+1}01^k01^{j+1}01^\ell01^n0$$

donde $n = 1, 2, 3$.

- Por ejemplo $\delta(q_2, s_3) = (q_0, s_5, \leftarrow)$ se codifica como

$$01^3 01^3 0101^5 01^2 0$$

- Una MT queda completamente determinada mediante su función de transición δ.
- La MT se codifica mediante la sucesión de los códigos de sus transiciones sin separaciones. Es decir, si $C_1, \ldots, C_k$ son los códigos de todas las transiciones de M. Entonces M se codifica mediante

$$C_1 C_2 \ldots C_k$$

- Obsérvese que no hay ambigüedad pues cada transición tiene exactamente seis ceros, además dos ceros consecutivos indican que inicia otra transición.

La máquina M dada por:

$$\delta(q_0, a) = (q_2, b, \rightarrow) \qquad \delta(q_2, b) = (q_3, c, \rightarrow)$$

$$\delta(q_3, a) = (q_1, c, \rightarrow) \qquad \delta(q_1, b) = (q_3, \sqcup, -)$$

se codifica mediante $a := 11,\ b := 111, c := 1111$ como sigue:

$$0101101110111010011101110111101111010$$

$$0111101101101111010011011101111010101110$$

- La codificación de una MT no es única, puesto que el orden de las transiciones no importa y un orden distinto genera una codificación distinta.
- De hecho si M tiene n transiciones, existen $n!$ codificaciones distintas para M.
- El proceso de codificación puede revertirse, no es dificil definir un algoritmo que decida si una secuencia binaria representa un código válido para MT y en tal caso lo decodifique.

6.3. Enumerabilidad de las MT

- En conclusión toda MT puede representarse como una cadena binaria.
- No todas las cadenas binarias representan MT válidas, por ejemplo, las cadenas que empiezan o terminan con 1 o las que tienen más de dos ceros consecutivos.
- Cada cadena binaria, representa por otra parte un número natural y viceversa. Es decir, hay tantas cadenas binarias como números naturales.
- De lo anterior se concluye que hay sólo un número numerable de MT.
- Las cadenas binarias pueden enumerarse en orden lexicográfico con $0 < 1$:

$$0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, 110, 111, \ldots$$

- Cada MT figura varias veces en esta lista.
- Por motivos prácticos si una cadena no codifica directamente a una MT entonces acordamos que codifica a la máquina sin transiciones que acepta el lenguaje vacío.
- De esta manera cada cadena binaria codifica a una MT.
- Dado que las cadenas binarias son tantas como los números naturales y cada cadena codifica a una MT concluimos que sólo hay un número infinito numerable de MT.
- Por otro lado si consideramos las funciones $f : \mathbb{N} \to \mathbb{N}$ es bien sábido que son un número **no** numerable (tantas como números reales)
- De donde se concluye que existen funciones, que **no** pueden calcularse mediante una MT.
- Lo cual bajo la tesis de Church-Turing equivale a que existen funciones que no pueden ser calculadas mediante una computadora.

6.4. Descripción y funcionamiento de MUT

- La máquina universal de Turing $\mathcal{U}$ simula el comportamiento de cualquier MT sobre un alfabeto Σ dado.
- $\mathcal{U}$ recibe como entrada (el código de) una máquina M y una cadena w
- Los datos de entrada se representan como $M0w$, es decir, la entrada es una cadena binaria que consta del código de M y el código de w en orden separados por un cero.
- En tal caso la cadena tiene una única sucesión de tres ceros que separa a M y w.
- $\mathcal{U}$ es una máquina con tres cintas y alfabeto $\Sigma = \{0, 1, \sqcup\}$.
- La primera cinta recibe el código de una máquina cualquiera M.
- La segunda cinta recibe el código de una cadena w.
- La tercera cinta almacena el código del estado actual de M
- $\mathcal{U}$ examina el código de M para saber si es una máquina válida, en caso contrario $\mathcal{U}$ para sin aceptar.
- $\mathcal{U}$ lee las cintas dos y tres para buscar en la cinta uno la transición necesaria.
- Se aplica la transición modificando la cadena en la cinta dos y cambiando el estado en la cinta 3, como M lo haría. La unidad de control vuelve a la primera posición en las cintas uno y tres.
- Si al procesar w $\mathcal{U}$ se detiene en el único estado de aceptación entonces se acepta w, como el estado de aceptación es el mismo que el de M entonces M tambien aceptaba a w.

- Si no hay transición aplicable en la cinta uno o si $\mathcal{U}$ se detiene en un estado distinto al de aceptación w no se acepta, al igual que en M.
- $\mathcal{U}$ acepta a w si y sólo si M acepta a w.

6.5. Lenguaje universal y diagonal

- El lenguaje aceptado por la máquina universal $\mathcal{U}$ se conoce como lenguaje universal, denotado $\mathcal{L}_{\mathcal{U}}$.
$$\mathcal{L}_{\mathcal{U}} = \{M0w \mid M \text{ acepta } w \in \Sigma^\star\}$$
- Por definición $\mathcal{L}_{\mathcal{U}}$ es un lenguaje recursivamente enumerable.
- Consideremos la enumeración de las máquinas de Turing $M_1, M_2, \ldots$ así como la enumeración de todas las cadenas de $\Sigma^\star$, digamos $w_1, w_2, \ldots, w_n, \ldots$.
- Podemos entonces dar como entrada la i-ésima palabra w_i a la i-ésima máquina M_i.
- El lenguaje diagonal se define como:
$$\mathcal{L}_D = \{w_i \mid w_i \text{ no es aceptada por } M_i\}$$
- Es decir $\mathcal{L}_D$ contiene a la i-ésima cadena si y sólo si ésta no es aceptada por la i-ésima máquina.
- Si $\mathcal{L}_D$ fuera R.E. sería aceptado por una MT, digamos la k-ésima máquina M_k.
- En tal caso $\mathcal{L}_D = L(M_k)$.
- Podemos preguntarnos entonces si $w_k \in \mathcal{L}_D$.
  - $w_k \in \mathcal{L}_D \Rightarrow w_k$ no es aceptada por $M_k \Rightarrow w_k \notin L(M_k) = \mathcal{L}_D$.
  - $w_k \notin \mathcal{L}_D \Rightarrow w_k \notin \mathcal{L}_D \Rightarrow w_k$ es aceptada por $M_k \Rightarrow w_k \in L(M_k) = \mathcal{L}_D$.
- Por lo que se tendría
$$w_k \in \mathcal{L}_D \text{ si y sólo si } w_k \notin \mathcal{L}_D$$
- Lo cual es absurdo.

7. ¿Qué es un algoritmo?

- ¿ Podemos reconocer cuando un procedimiento sistemático es un algoritmo?
- ¿Podemos dar una definición precisa del concepto de algoritmo?
- ¿Por qué es importante tener una definición precisa (matemática) de algoritmo?
- Un algoritmo es una colección de instrucciones simples para realizar una tarea o problema particular (procedimientos o recetas)

- Si se tiene un algoritmo para un problema dado P significa tener una manera para calcular efectivamente o resolver P.
- Un algoritmo es un proceso *potencialmente* realizable
  - Las operaciones del proceso se pueden realizar inequívocamente.
  - El número de operaciones o pasos del proceso es finito.

7.1. Existencia y Formalización de Algoritmos

- Décimo problema de Hilbert: Hallar un proceso de acuerdo al cual pueda determinarse en un número finito de pasos (un algoritmo) si un polinomio dado tiene una raiz entera.
- Se creia que todo problema P tenia una solución algorítmica.
- Más aún se pensaba en la existencia de un algoritmo universal U que pudiera resolver todos los problemas matemáticos.
- Los intentos por hallar el algoritmo universal U fallaron. Tal vez U no existía.
- ¿Cómo probar la no existencia de U?
- Era necesario definir el concepto de algoritmo de una manera precisa y hallar un formalismo para poder probar propiedades de los mismos.
- Un formalismo que intente representar el concepto de algoritmo deberia ser preciso y libre de ambigüedades, simple y general.

7.2. Formalismos para representar el concepto de algoritmo

- Máquinas de Turing (Alan Turing, Cambridge 1936)
- Cálculo Lambda (Alonzo Church, Princeton 1936)
- Sistemas de Post (Emile Post)
- Funciones μ-recursivas (Gödel, Herbrand, Kleene)
- Lógica Combinatoria (Curry, Schönfinkel)
- Máquinas de registro (Sheperdson, Sturgis)

Sorprendentemente todos los formalismos han resultado equivalentes. Es decir, un problema tiene solución en un formalismo si y sólo si tiene solución en cualquiera de los otros. Tal afirmación es un teorema, o una serie de teoremas rigurosamente demostrados. Esta supuesta coincidencia nos lleva a conjeturar que existe una única noción de computabilidad, esto se expresa mediante la llamada Tesis de Church-Turing:

7.3. La Tesis de Church-Turing

La tesis de Church-Turing afirma que la noción intuitiva de algoritmo es capturada de manera exacta por la noción matemática de máquina de Turing.

Un problema es soluble algorítmicamente si y sólo si es soluble mediante una máquina de Turing

Es decir, las máquinas de Turing implementan a cualquier algoritmo. Equivalentemente, una función es computable si y sólo si es computable mediante una máquina de Turing.

Las siguientes observaciones son relevantes:

- La afirmación es una tesis indemostrable pues la noción de algoritmo es intuitiva.
- Por otro lado la tesis es refutable y se destruiría mostrando un algoritmo que no pudiera ser implementado en una máquina de Turing.
- Existen fuertes evidencias a favor de la tesis, algunas de éllas son:
  - Intuitivamente cualquier algoritmo detallado para el cálculo manual puede programarse en una MT.
  - La equivalencia con otros formalismos mas modernos.
  - Existen demasiados ejemplos a favor y por supuesto ningún contraejemplo.
  - La comunidad tanto en matemáticas como en ciencias de la computación acepta ampiamente la tesis.

7.4. Computabilidad y Complejidad

Aceptamos la tesis de Church-Turing bajo la cual nos interesa discutir lo siguiente para un problema específico P.

- Computabilidad: ¿Es P efectivamente computable?, es decir, ¿ Podemos resolver P mediante una máquina de Turing ?
- Complejidad: ¿Es P eficientemente computable?, es decir, ¿Puede P calcularse o implementarse de manera eficiente?

7.4.1. Observaciones acerca de la computabilidad

- Según la tesis de Church-Turing, las máquinas de Turing pueden computar todo lo que es computable. Sin embargo existe solamente un número infinito numerable de MT.
- Por otra parte el número de funciones digamos $f : \mathbb{N} \to \mathbb{N}$ **no** es numerable, es decir, hay más funciones que máquinas de Turing.
- Por lo tanto deben existir funciones no computables.
- Ejemplo prominente: el problema de la detención (Halting problem)

7.4.2. Observaciones acerca de la complejidad

- La complejidad es acerca de los problemas no de las soluciones.
- La pregunta de interés es ¿Qué tan complejo es resolver el problema? y **no** ¿Qué tan complejo es implementar la solución?
- La complejidad se divide en clases de problemas que comparten ciertas características computacionales.
- **P**: problemas que pueden resolverse por una MT en tiempo polinomial.
- **NP**: problemas que pueden resolverse por una MT no-determinística en tiempo polinomial.
- El problema más famoso en complejidad es el problema P vs. NP.

8. Computabilidad

8.1. Introducción

- Preguntas:
  - ¿Qué x real cumple $2x^2 - 3x + 5 = 0$?
  - Dadas las ciudades a, b, c, d ¿Cual es la forma óptima de visitarlas sin pasar dos veces por la misma ciudad?
- Una problema es una clase de preguntas:
  - ¿Cuales son las soluciones de ¿$ax^2 + bx + c = 0$?
  - ¿ Dados n vértices en un grafo existira un camino hamiltoniano?
- Cada caso particular de un problema es un ejemplo, ejemplar o instancia de éste.
- La mayoría de los problemas de interén en ciencias de la computación son de dos tipos.
- Problemas de cómputo: obtener el valor de una función en un argumento dado.
  - Por ejemplo obtener la raiz cuadrada de un número dado x con exactitud de milésimas.
- Problemas de decisión: estos problemas tienen como respuesta si o no.
  - Por ejemplo, decidir la existencia de un camino óptimo en costos para recorrer varias ciudades.
- Si bien los problemas de decisión también podrían considerarse problemas de cómputo, es útil hacer la distinción.

8.2. Algoritmos y problemas

- Un algoritmo para un problema consiste de una serie de instrucciones capaces de responder cualquier instancia del problema dado.
- Un problema P se dice soluble si existe un algoritmo para P. En otro caso se dice insoluble.
- Si un problema de decisión P es soluble entonces decimos que P es decidible. En caso contrario el problema se dice indecidible.
- Un proceso o algoritmo para un problema de decisión se conoce como un proceso o algoritmo de decisión.
- Cualquier algoritmo puede verse como una función:

$$\text{entrada } x \rightarrow \text{ algoritmo } \rightarrow \text{ salida } y$$

- La salida está en función de la entrada

$$f(x) = y$$

- Una función es computable si existe un algoritmo que la calcule.
- La teoría de la computabilidad se encarga esencialmente a contestar la pregunta *¿Qué funciones son computables?*
- Lo cual equivale entonces a responder que problemas son solubles.

8.3. Problemas no computables

- ¿Por qué nos interesa averiguar qué problemas no son computables mediante un algoritmo o proceso?
- Tales problemas son aquellos que **no** podemos resolver
- Son resultados fundamentales y debemos conocerlos para tener una visión general de las ciencias de la computación.
- Debemos conocerlos para evitar intentar resolverlos.
- Debemos entender que tales problemas son insolubles independientemente del desarrollo futuro del hardware.
- Considérese una propiedad $\mathcal{P}$ acerca de máquinas de Turing. Es decir una propiedad que es válida o no para una MT y todas sus equivalentes.[4]
- $\mathcal{P}$ genera el problema de decisión siguiente:

[4] Por ejemplo la propiedad de "tener 5 estados" NO es una propiedad de máquinas de Turing, pues puede haber una máquina equivalente a una máquina de 5 estados que tenga más estados (por ejemplo estados inaccesibles)

¿Satisface la máquina M la propiedad $\mathcal{P}$?

- Asumiendo la tesis de Church-Turing, tal problema de decisión será decidible (soluble) si y sólo si el lenguaje

$$L = \{M \mid M \text{ es el código de una MT que satisface } \mathcal{P}\}$$

es recursivo.

8.4. El problema universal

Dada una máquina de Turing cualquiera M y una cadena w
¿Acepta M a w?

- El problema universal equivale a que el lenguaje universal

$$\mathcal{L}_{\mathcal{U}} = \{M0w \mid M \text{ acepta } w \in \Sigma^\star\}$$

sea recursivo.

- Ya mencionamos que $\mathcal{L}_{\mathcal{U}}$ no es recursivo.
- Por lo tanto el problema universal es indecidible.

8.5. El problema de la detención

Dada una máquina M y una cadena w ¿Se detendrá M al procesar w?

- El problema sería soluble si pudieramos hallar una máquina H tal que al recibir como entrada a cualquier cadena $M0w$ se detuviera si y sólo si M se detiene al procesar w.
- Podría pensarse que una máquina universal puede hacer el trabajo.
- El problema de la detención resulta indecidible, es decir, no existe tal máquina H.
- Además es quizas el problema indecidible más relevante en la teoría de la computabilidad.
- Una consecuencia inmediata de su indecidibilidad es que no puede existir un programa que verifique si cualquier programa dado se cicla.
- Para mostrar la indecidibilidad de este problema se usa la técnica de reducibilidad de problemas, resumida a continuación.

8.6. Reducibilidad de problemas

- La reducibilidad de problemas es una técnica de gran utilidad para probar la indecidibilidad de un problema dado a partir de la indedidibilidad de un problema conocido.
- Dados dos problemas $\mathcal{P}_1, \mathcal{P}_2$ decimos que $\mathcal{P}_1$ se reduce a $\mathcal{P}_2$ si un algoritmo para decidir $\mathcal{P}_2$ puede emplearse para decidir $\mathcal{P}_1$.
- Formalmente decimos que $\mathcal{P}_1$ se reduce a $\mathcal{P}_2$, denotado $\mathcal{P}_1 \prec \mathcal{P}_2$ si existe una MT M tal que:
- M recibe como entrada una instancia I_1 de $\mathcal{P}_1$
- M devuelve como salida una instancia I_2 de $\mathcal{P}_2$.
- M decide a I_1 de la misma manera que I_2.
- Es decir, M responde con *sí* a I_1 si y sólo si responde con *sí* a I_2.
- De esta manera se tiene que $\mathcal{P}_1$ es decidible si y sólo si $\mathcal{P}_2$ es decidible.

8.7. Reducibilidad del problema de la detención al problema universal

- El problema universal puede reducirse al problema de la detención
- Supongamos que existe una máquina H que decide el problema de la detención.
- En tal caso H decidiría también al problema universal, lo cual es absurdo.
  - Al recibir una entrada $M0w$, H decide si M se detiene o no con entrada w.
  - Si M no se detiene con w entonces M no acepta a w.
  - Si M se detiene con w entonces M procesa a w y decide si la acepta o no.
- De manera que el problema universal se decide, lo cual contradice la indecidibilidad del problema universal.

8.8. Otros problemas indecidibles

- Detención con cinta en blanco: ¿ se detiene la máquina M al iniciar con la cinta en blanco?
- Impresión de un símbolo: Dada M y $s \in \S$ ¿Escribirá M en algún momento a s sobre la cinta?
- Dada una gramática libre de contexto G ¿Es G ambigua?
- Determinar si dos gramáticas libres de contexto son equivalentes.
- Cualquier propiedad no trivial acerca de MT (Teorema de Rice)

9. Complejidad

9.1. Introducción

- Las MT son el formalismo más útil en el análisis de algoritmos, debido a que modelan de manera precisa los conceptos centrales de cómputo, almacenamiento, espacio y tiempo.
- La noción formal de cómputo permite precisar sin ambigüedades el tiempo de computación de un problema.
- Las celdas de la cinta formalizan de manera clara la noción de espacio de almacenamiento (memoria)
- Así las nociones de tiempo y espacio se modelan de forma muy realista mediante una MT, lo cual permite analizar la complejidad computacional de un problema.
- La teoría de complejidad se interesa por el estudio de la complejidad necesaria para resolver un problema.
- En particular por el consumo de recursos (espacio y tiempo).
- No le conciernen los problemas insolubles.
- Pero tampoco todos los problemas solubles, en la práctica aquellos problemas solubles que no pueden resolverse en un tiempo razonable son tan despreciables como un problema insoluble.
- Los problemas se clasifican en clases de complejidad de acuerdo a que tan dificil es resolverlos.

  La función tiempo de ejecucion de una MT M se define como

$$t_M : \mathbb{N} \to \mathbb{N} \cup \{\infty\}$$

  $t_M(n) :=$ máximo número de pasos de ejecución de M para una entrada de longitud n.

- Una máquina de Turing M corre en tiempo polinomial si:
- Existe un polinomio con coeficientes enteros no negativos

$$p(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_1 x + a_0$$

- tal que $t_M(n) \leq p(n)$ para toda $n \in \mathbb{N}$.
- Es decir, si la función de tiempo de ejecución de M está acotada superiormente por un polinomio.
- Una máquina que corre en tiempo polinomial siempre se detiene.

9.2. Clases de complejidad

- Clase de problemas que pueden ser resueltos eficientemente
- Eficientemente significa que existe un algoritmo que corre en *tiempo polinomial*.
- **P** se conoce también como la clase de problemas *tratables*.
- En contraste, un problema es *intratable* si es soluble pero cualquier solución algorítmica corre en tiempo exponencial en el peor de los casos.
- Un problema intratable es prácticamente insoluble, excepto para entradas muy pequeñas, a no ser que el caso promedio sea mucho mejor que el peor caso.
- Clase de problemas que pueden ser resueltos en tiempo polinomial pero sólo por un algoritmo no determinista.
- Es decir son los problemas solubles por una MT no-determinista en tiempo polinomial.
- Sabemos que las MT no-deterministas equivalen a las MT deterministas.
- Sin embargo no se sabe en general si un problema soluble mediante un algoritmo no-determinista tendrá una solución determinista.
- Dado un problema $\mathcal{P} \in \mathbf{NP}$ en general no sabemos si existirá una solución determinista.
- Si $\mathbf{P} = \mathbf{NP}$ entonces la respuesta es afirmativa.
- Aun no se ha probado ni refutado si **P**=**NP**.
- En caso positivo siempre habría un algoritmo determinista para un problema cuya solución no-determinista se conoce.
- Se sospecha que **P**$\neq$**NP** y esta es la pregunta más importante en la teoria de la complejidad (la respuesta vale 1 millón de dolares)

9.3. Problemas NP-completos

- Los problemas **NP**-completos son los más dificiles de la clase **NP**.
- Cualquier problema de la clase **NP** puede reducirse a cualquier problema **NP**-completo.
- Existen muchos problemas **NP**-completos, el más relevante es probablemente el problema **SAT** (Teorema de Cook).
- Un problema $\mathcal{P}$ es **NP**-completo ($\mathcal{P} \in \mathbf{NPC}$) si:
  - $\mathcal{P} \in \mathbf{NP}$.
  - $\mathcal{Q} \prec \mathcal{P}$ en tiempo polinomial para cualquier problema $\mathcal{Q} \in \mathbf{NP}$.

- Todos los problemas en la clase **NPC** son equivalentes, es decir, si $A, B \in \mathbf{NPC}$ entonces $A \prec B$ y $B \prec A$.

- La segunda condición de la definición se puede intercambiar por: $C \prec \mathcal{P}$ para algún problema **NP**-completo C.

- Dada una fórmula de la lógica proposicional en forma normal conjuntiva:

$$P := C_1 \vee C_2 \vee C_3 \ldots C_n$$

- ¿Existe una asignación de verdad que satisfaga a P?

- Teorema de Cook (1971): **SAT** es **NP**-completo.

- Este es el primer problema **NP**-completo.

- Un circuito hamiltoniano es un cámino que inicia y termina en un mismo vértice de un grafo conexo y que visita a todos los vértices exactamente una vez (el vértice de inicio y fin cuenta solo una vez).

- **PCH**: ¿Dado un grafo conexo G, tiene G un circuito hamiltoniano?

- Se tienen dadas m ciudades, las distancias entre cualesquiera dos de ellas y un entero N.

- **PAV**: ¿Existe un recorrido que visite cada ciudad exactamente una vez y cuya longitud sea menor o igual que N ?