Universidad del Valle de Guatemala
Teoría de la Computación, CC2019
Docentes: Alan Reyes; Tomás Gálvez P.
Semestre 2, 2022

UVG
UNIVERSIDAD DEL VALLE DE GUATEMALA

Computabilidad: máquinas de Turing

Las **funciones efectivamente calculables** son las funciones matemáticas para cada una de las cuales existe un procedimiento efectivo o mecánico que calcula sus imágenes. Cuando existe un algoritmo para realizar una tarea argumentamos que se puede construir una máquina hipotética y determinística que realice dicha tarea, llamada **máquina de Turing**.

Las máquinas de Turing tienen una caracterización (descripción) matemática precisa, y esto impone una limitante igualmente precisa sobre las funciones que pueden computar. Serán éstas las **funciones computables**, e identificamos al conjunto de las funciones computables con el de las efectivamente calculables por convención.

Ejercicio de práctica A: ¿por qué es convencional la identificación entre funciones computables y funciones efectivamente calculables? ¿Si no hay un algoritmo que resuelva un problema o compute una función, podemos afirmar que dicho problema o función no es computable?

Un computador es algo que computa. En efecto, puede ser cualquier cosa desde una máquina hasta una persona. Computar es seguir un procedimiento efectivo o mecánico. Usualmente se requiere algo que computar (*input*) y se produce un resultado *(output*).

Ejercicio de práctica B: proponga los materiales que necesitaría, como mínimo, para ejecutar un algoritmo "a mano". Responda: ¿por qué puede una computadora ejecutar algoritmos mucho más rápido que usted, si su cerebro humano es mucho más poderoso que dicha computadora?

Producir un algoritmo para hacer algo equivale a construir una máquina de Turing que logre ese algo, demostrando en el camino que ese algo es computable. Existen diferentes formas de definir una máquina de Turing y también diferentes tipos de máquinas. Las características básicas que poseen son:

- Al menos una cinta infinita dividida en secciones iguales (que normalmente llamamos **casillas**).
- Al menos un **lector** de símbolos en cada cinta.
- Un conjunto finito de símbolos (**alfabeto**) que se escriben y leen sobre las casillas de la cinta.
- Un conjunto finito de **estados**, en uno de los cuales se encuentra la máquina en todo momento.
- Operaciones de:
    - Lectura de símbolos en la cinta.
    - Escritura de símbolos en la cinta (reemplaza el símbolo que esté bajo el lector).
    - Desplazamiento del lector una casilla a la derecha.
    - Desplazamiento del lector una casilla a la izquierda.
    - Cambio de estado.

Representemos el alfabeto con S y los símbolos que a él pertenecen con un subíndice numérico (*i.e.*, $S_i \in S \mid i \in X \subset \mathbf{N}$). Se usa un **símbolo nulo**, externo al alfabeto, para describir casillas "en blanco". En nuestro caso, usamos el símbolo *B* como nulo. También definimos un conjunto de estados Q y los elementos que a él pertenecen como $q_j \mid j \in Y \subset \mathbf{N}$; y las operaciones de movimiento hacia la derecha e izquierda con los símbolos R y L respectivamente. Una **expresión** será cualquier combinación de símbolos, estados y operaciones R, L.

Un tipo especial de expresiones son las **cuádruplas,** ya que permiten representar una máquina de Turing formalmente. Cada cuádrupla está compuesta por cuatro elementos y tiene una de las siguientes cuatro formas:

1. $q_i S_j S_k q_l$
2. $q_i S_j R q_l$
3. $q_i S_j L q_l$
4. $q_i S_j q_k q_l$

Los símbolos de las primeras tres formas presentan, en orden, un estado esperado, un símbolo esperado, una operación realizada ante esa combinación de símbolo y estado; y un estado destino. La última es una cuádrupla especial que da paso al concepto de computabilidad relativa y representa una condicional que permite a una máquina de Turing "comunicarse con el mundo exterior".

Ejercicio de práctica C: identifique a qué operaciones de una máquina de Turing corresponde cada una de las primeras tres cuádruplas.

Nótese que en ninguna cuádrupla se impide que el estado destino y el estado esperado sean iguales, efectivamente expresando que dicha cuádrupla mantiene el estado de la máquina. Lo mismo aplica a los símbolos que la máquina lee y escribe. Una máquina de Turing será, entonces, un conjunto finito y no vacío de cuádruplas. Nuestra máquina de Turing será **simple** si no posee cuádruplas de la forma cuatro. Ningún par de cuádruplas en la misma máquina debe compartir los primeros dos elementos. Permitir esto abriría la posibilidad de dos o más comportamientos distintos a partir de un mismo input, *i.e.*, a partir de una misma combinación de estado y símbolo leído.

Los estados son una característica de la máquina, como una etiqueta que nos sirve de parámetro al decidir operaciones subsecuentes. En un estado no necesariamente encontramos información sobre la máquina de Turing en sí. Para ello empleamos **descripciones instantáneas**, expresiones que son conformadas por el estado actual de la máquina y los símbolos del alfabeto que describen el contenido de la cinta, en orden. Identificamos al símbolo a la derecha del estado como el símbolo bajo el lector. Ejemplo de una descripción instantánea que representa una máquina en el estado q_0 con los símbolos 1101 sobre su cinta, el lector en el primer 1:

$$q_0 1101$$

Como el símbolo inmediatamente a la derecha del estado en la descripción instantánea será, por convención, el símbolo bajo el lector, y siempre debe haber un símbolo bajo el lector, una descripción instantánea con el estado escrito al final es inválida. Cuando el lector de la máquina se halla sobre el último símbolo en la cinta, el estado se coloca en la descripción instantánea justo antes del último símbolo. Cuando el lector se encuentra más allá del contenido sobre la cinta, lo indicamos con el símbolo B a la derecha del estado, ya que las casillas donde no haya símbolos estarán "vacías".

Una descripción instantánea se suele abreviar para incluir sólo una porción de la cinta alrededor del lector. La cinta es infinita, por lo que las descripciones instantáneas agregan símbolos en blanco conforme sea necesario según el movimiento del lector sobre las casillas más allá del contenido en la cinta. Como observación, escribir B en una sección es interpretado como borrar su contenido.

EEEENTONCES... aplicamos las operaciones permitidas por la máquina de Turing (expresadas en sus cuádruplas) para cambiar entre diferentes descripciones instantáneas. La aplicación es automática: la descripción instantánea muestra la configuración actual de la máquina y sólo una cuádrupla de la máquina de Turing en cuestión debe ser aplicable para esa descripción. El "trabajo" que conlleva la aplicación está en modificar la descripción instantánea para expresar la configuración resultante de la máquina.

En la tabla a continuación presentamos descripciones instantáneas en las columnas de los extremos. La columna del medio muestra la cuádrupla que debe poseer la máquina de Turing para "pasar" de la descripción instantánea α a la descripción instantánea β. Interprétense P y Q como todos los símbolos que se encuentran respectivamente antes y después de los demás símbolos en la descripción instantánea.

Ejercicio de práctica D: llene las celdas pendientes en la tabla que describe los cambios propiciados por cada tipo de cuádrupla sobre una descripción instantánea dada.

Descripción instantánea $\boldsymbol{\alpha}$	Regla (cuádrupla) aplicada	Descripción instantánea β
Pq_iS_jQ	$q_iS_jS_kq_l$	
$Pq_iS_jS_kQ$		$PS_jq_lS_kQ$
Pq_iS_j		PS_jq_lB
	$q_iS_jLq_l$	$Pq_lS_kS_jQ$
	$q_iS_jLq_l$	

Una **computación** será una secuencia de descripciones instantáneas, cada una de las cuales es obtenida a partir de su predecesora por medio de aplicación de una cuádrupla. La última descripción instantánea de la computación es la **descripción instantánea terminal**, de la que no se puede obtener otra descripción instantánea. La descripción instantánea terminal contiene el **resultado** de aplicar la máquina de Turing a la descripción instantánea inicial α_1. Por definición, α_1 contiene los símbolos de entrada y el estado de inicio a la izquierda de ellos, representando el lector al inicio de la cinta.

Ejercicio de práctica E: ¿cómo sabemos cuándo una máquina de Turing ha terminado? ¿En qué ocasiones afirmaríamos que nuestra máquina de Turing ha sido incapaz de realizar una tarea? Explique el porqué de sus respuestas.

Para computar funciones numéricas optamos, a lo largo del tema, por no usar a los números como nuestro alfabeto porque ejemplificar el diseño y funcionamiento de las máquinas nos será más fácil con la representación **unaria** (*e.g.*, $1 = 1, 2 = 11, 3 = 111$, etc.). Para abreviar, usamos la notación $1^n = \overbrace{111\ldots1}^{n}$, pero $1^0 = B$, lo que nos deja sin una representación para el cero. Entonces usamos una codificación unaria modificada, agregando un '1' de más. Denotamos esta codificación como $\bar{n}$, y esto sería igual a una secuencia de $n + 1$ dígitos '1' (*i.e.*, $\bar{n} = 1^{n+1}$). De esta forma nos es fácil armar una expresión para la cinta de nuestra máquina de Turing que represente al número n. Si necesitamos representar una secuencia de números basta con separarlos usando secciones en blanco, *i.e.*, $\overline{(n_1, n_2, n_3, n_4 \ldots)} = \overline{n_1}B\overline{n_2}B\overline{n_3}B\overline{n_4}$ ... Con esto podemos fabricar máquinas de Turing que acepten varios argumentos.

Cuando la máquina de Turing inicia operaciones, los n argumentos de entrada, representados como una tupla $(m_1 \dots m_n)$, se convierten en una descripción instantánea $\alpha_1 = q_0\overline{(m_1, \dots, m_n)}$ donde q_0 es el estado inicial. Luego procedemos a aplicar las cuádruplas sobre la descripción instantánea hasta alcanzar una descripción instantánea terminal. Lo que nuestra máquina de Turing provea como salida será la cantidad de unos en la descripción instantánea terminal (la cantidad total de unos, ignorando la codificación convenida).

Será posible que nuestra misma α_1 sea la terminal en la computación, pero también será posible que nuestra computación sea infinita al no alcanzar nunca una descripción instantánea terminal. En ese caso, el resultado de la máquina bajo esa entrada queda **indefinido**. Así, las funciones serán **parcialmente computables** cuando las máquinas de Turing que las calculan tengan resultados indefinidos ante ciertas entradas, pero si la función en cuestión es total (es decir, no tiene resultados indefinidos) dicha función será **totalmente computable** o, simplemente, **computable**. Veamos dos ejemplos: la función de suma $f(x, y) = x + y$ & la función de resta positiva $g(x, y) = x - y$.

Función de suma

Observemos que la descripción instantánea inicial será $\alpha_1 = q_1\overline{(x, y)} = q_1 11^x B 11^y$. El número de 1's en α_1 es $x + y + 2$, por lo que sólo necesitamos quitarnos ese 2 procedente del '1' extra en las codificaciones de x y de y. Nuestro proceder será eliminar el primer 1 de x, luego leer la secuencia hasta encontrar los 1's que corresponden a y (identificables porque inician luego de un espacio en blanco), y eliminar el primer 1 de y. A continuación, las cuádruplas que nos permiten realizar estos pasos:

Cuádrupla	Explicación
$q_1 1 B q_1$	Se aplica sobre α_1 y borra el primer 1 de x.
$q_1 B R q_2$	Salta el espacio en blanco recién creado. Cambia de estado para no reaplicar la cuádrupla anterior.
$q_2 1 R q_2$	Salta los 1's de x.
$q_2 B R q_3$	Encuentra y salta el espacio que separa x de y. Cambia de estado para borrar el 1 que viene.
$q_3 1 B q_3$	Borra el primer 1 de y. Como no hay una cuádrupla que empiece con $q_3 B$, esto nos dejará una descripción instantánea terminal.

Para demostrar que esto funciona escribimos la siguiente computación:

$$\alpha_1 = \overline{(x, y)} = q_1 11^x B 11^y$$
$$\rightarrow q_1 B 1^x B 11^y$$
$$\rightarrow B q_2 1^x B 11^y$$
$$\rightarrow B 1 q_2 1^{x-1} B 11^y$$
$$\dots$$
$$\rightarrow B 1^x q_2 B 11^y$$
$$\rightarrow B 1^x B q_3 11^y$$
$$\rightarrow B 1^x B q_3 B 1^y$$

¿Cuántos 1's quedan en la expresión final? El resultado es $x + y$. Con la limitación de uso de enteros no negativos impuesta por nuestra codificación, esta función es computable.

Función de resta

Para computar esta función necesitaremos eliminar 1's de la cinta hasta que quede la diferencia $x - y$. Elegimos ir eliminando 1's de los extremos de la cinta alternativamente:

Cuádrupla	Explicación
$q_1 1 B q_1$ $q_1 B R q_2$	Borrar un 1 del extremo izquierdo y saltar a la derecha.
$q_2 1 R q_2$ $q_2 B R q_3$	Encontrar la separación entre x e y.
$q_3 1 R q_3$ $q_3 B L q_4$	Encontrar el extremo derecho de la expresión en la cinta.
$q_4 1 B q_4$ $q_4 B L q_5$	Borrar un 1 de extremo derecho, y saltar a la izquierda.
$q_5 1 L q_6$	Si hay un uno, saltar que todavía no hemos terminado. Como no hay cuádrupla que empiece con $q_5 B$, cuando se agoten los 1's de y se habrá alcanzado una terminal.
$q_6 1 L q_6$ $q_6 B L q_7$	Encontrar y saltar la separación entre x e y (de regreso).
$q_7 1 L q_8$ $q_7 B R q_9$	Si se encuentra un espacio en blanco se agotó x antes que y, y estamos condenados; de lo contrario seguir.
$q_8 1 L q_8$ $q_8 B R q_1$	Encontrar el extremo izquierdo de la expresión en la cinta, y repetir el procedimiento.
$q_9 1 L q_9$ $q_9 B R q_9$	Somos una desgracia para nuestro clan. *Harakiri* computacional.

La eliminación se hace alternando extremos porque así se reduce nuestra expresión en la cinta sin perder el formato conocido que separa los argumentos con un espacio en blanco.

Ahora, la demostración de funcionamiento. En el primer caso, $x \geq y$, por lo que $y + k = x$ para algún $k \in$ **N**:

$$\alpha_1 = q_1\overline{(x,y)} = q_1 11^y 1^k B 1^y 1$$

$$\rightarrow q_1 B 1^y 1^k B 1^y 1$$

$$\rightarrow B q_2 1^y 1^k B 1^y 1$$

...

$$\rightarrow B 1^y 1^k q_2 B 1^y 1$$

$$\rightarrow B 1^y 1^k B q_3 1^y 1$$

...

$$\rightarrow B 1^y 1^k B 1^y 1 q_3 B$$

$$\rightarrow B 1^y 1^k B 1^y q_4 1 B$$

$$\rightarrow B 1^y 1^k B 1^y q_4 B B$$

...

$$\rightarrow B 1^y 1^k B q_6 1^y B B$$

$$\rightarrow B 1^y 1^{k-1} q_7 1 B 1^y B B$$

...

$$\rightarrow q_8 B 1^y 1^k B 1^y B B$$

$$\rightarrow B q_1 1 1^{y-1} 1^k B 1^y B B$$

...

$$\rightarrow B^{y+1} 1^k B q_4 1 B^{y+1}$$

$$\rightarrow B^{y+1} 1^k B q_4 B B^{y+1}$$

$$\rightarrow B^{y+1} 1^k q_5 B B B^{y+1}$$

Aquí, el número de 1's es igual a k, con $k = x - y$.

Nuestra computación debe detenerse cuando hayamos borrado los 1's de y. ¿Qué pasa si agotamos los unos de x primero? Como nuestra codificación no describe números negativos necesitamos expresar que, si $x < y$, el resultado queda indefinido, para lo cual introdujimos cuádruplas que provocan una computación infinita.

Entonces, cuando $x < y$ sabemos que $x + k = y$ para algún $k \in \mathbf{N}$:

$$\alpha_1 = q_1\overline{(x,y)} = q_1 11^x B 1^k 1^x 1$$

$$\rightarrow q_1 B 1^x B 1^k 1^x 1$$

$$\rightarrow B q_2 1^x B 1^k 1^x 1$$

...

$$\rightarrow B q_1 1^x B 1^k 1^x BB$$

...

$$\rightarrow B^x q_1 1 B 1^k 1 B^{x+1}$$

$$\rightarrow B^x q_1 BB 1^k 1 B^{x+1}$$

$$\rightarrow B^x B q_2 B 1^k 1 B^{x+1}$$

$$\rightarrow B^x BB q_3 1^k 1 B^{x+1}$$

...

$$\rightarrow B^x BB 1^k q_4 1 B^{x+1}$$

$$\rightarrow B^x BB 1^k q_4 BB^{x+1}$$

...

$$\rightarrow B^x B q_6 B 1^k BB^{x+1}$$

$$\rightarrow B^x q_7 BB 1^k BB^{x+1}$$

$$\rightarrow B^x B q_9 B 1^k BB^{x+1}$$

$$\rightarrow B^x BB q_9 1^k BB^{x+1}$$

$$\rightarrow B^x B q_9 B 1^k BB^{x+1}$$

...

Como esta computación sigue para siempre, tenemos un resultado indefinido y, por lo tanto, la función es parcialmente computable.

Universidad del Valle de Guatemala
Teoría de la Computación, CC2019
Docentes: Alan Reyes; Tomás Gálvez P.
Semestre 2, 2022

Vamos a regresar a, y a elaborar sobre, el concepto de computabilidad relativa. La **computabilidad relativa** se refiere a computabilidad dependiente de algo, y para este fin se creó el concepto de **oráculos**: entes a los que se puede preguntar algo y obtener la respuesta correcta. Una **máquina de Turing oráculo** (más fácil de decir en inglés, ***oracle machine***) es una máquina capaz de consultar a un oráculo durante sus computaciones. Los oráculos resuelven problemas de decisión, que se resumen en los problemas que se responden con "sí" o "no". Este tipo de problemas suele plantearse como preguntas de pertenencia de una cadena de símbolos a un lenguaje formal.

Retomando las cuádruplas que definen máquinas de Turing, recordemos que no hemos usado las cuádruplas "condicionales" cuya forma es:

$$q_i S_j q_k q_l$$

Dependiendo del resultado de una pregunta al oráculo, la máquina de Turing podría pasar al estado k o al estado l, suponiendo un estado actual q_i y un símbolo S_j bajo el lector. La pregunta que será respondida por el oráculo en los casos de este documento será la pertenencia de un número a un conjunto A predeterminado; dicho número será la cantidad de unos en nuestra descripción instantánea actual.

Si la respuesta del oráculo es positiva, nuestra descripción instantánea pasa de $Pq_i S_j Q$ a $Pq_k S_j Q$; de lo contrario, pasa a $Pq_l S_j Q$. El calificativo "terminal" aplica a la última descripción instantánea en una computación sobre máquinas de Turing simples, pero cuando una descripción instantánea es terminal y, además, no se le pueden aplicar cuádruplas condicionales, se le llama descripción instantánea **final.**

Ejercicio de práctica F: argumentamos que a nuestras *oracle machines* se les puede determinar un conjunto A de modo que su comportamiento pueda ser replicado por una máquina simple. ¿Cuál sería ese A y qué cambiaría en las cuádruplas de la máquina simple que reproduce el comportamiento de la *oracle machine* con ese A?

Las máquinas de Turing simples son un subconjunto de las máquinas de Turing en general debido a que las máquinas oráculo pueden hacer lo que las máquinas simples hacen, y más. Con esto podemos argumentar convincentemente que la computabilidad simple es un caso especial de la computabilidad relativa. **Nota**: también se podrían convertir las cuádruplas condicionales a $q_i S_j S_j q_k$ si tomamos $A = \mathbf{N}$. Esto aplica a las convenciones y definiciones de este documento, pero no en general.

Ejercicio de práctica G: defina una máquina de Turing que compute la función "sucesor" $S(x) = x + 1$. Demuestre que funciona tal como se demostró el funcionamiento de la máquina de Turing para sumar.

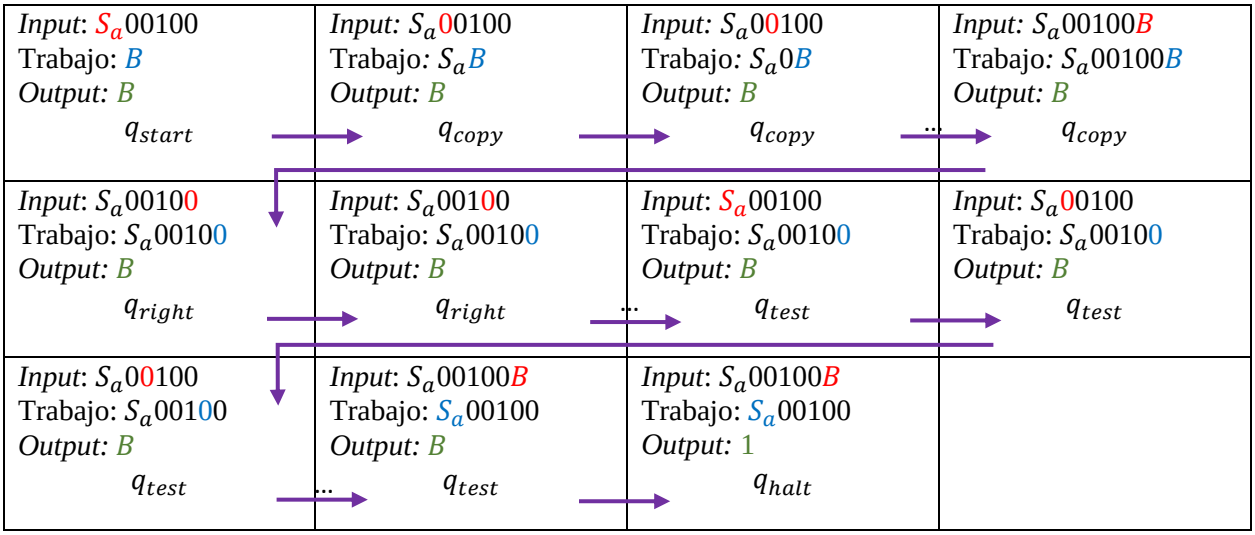
Arora, *et. al.*, proveen una definición un poco más amplia e intuitiva de una máquina de Turing. Ellos proponen que una máquina de Turing posee un número k de cintas en lugar de sólo una, y que habitualmente una cinta sirve únicamente para leer la entrada. Las otras son "escribibles" y sirven para trabajar, una de ellas siendo la cinta de donde se leerá el resultado final.

En la propuesta formal, una máquina de Turing es una tupla (S, Q, δ) donde S y Q son el alfabeto y el conjunto de estados que hemos usado hasta ahora, pero δ es una **función de transición** con la forma $\delta: Q \times S^k \to Q \times S^{k-1} \times \{L, S, R\}^k$. Lo que la función hace es relacionar un estado y los símbolos leídos en las k cintas con: un nuevo estado, los $k-1$ símbolos que reemplazan (por medio de escritura) los símbolos en las cintas de trabajo; y un movimiento (***L**eft*, ***S**tay*, ***R**ight*) en cada una de las k cintas.

Un ejemplo de uso de esta forma es la máquina que identifica palíndromos. Esta máquina posee tres cintas y su configuración inicial es un símbolo S_a seguido por la entrada en la cinta de *input* (todo lo demás con símbolos nulos). El conjunto de estados es $\{q_{start}, q_{copy}, q_{right}, q_{test}, q_{halt}\}$, con q_{start} como estado inicial. Las reglas de la máquina son:

1. Si el estado actual es q_{start}, moverse a la derecha en la cinta de *input*, escribir S_a en la cinta de trabajo; y moverse a la derecha en la cinta de trabajo. Cambiar el estado a q_{copy}.
2. Si el estado actual es q_{copy}:
    a. Si el símbolo leído en la cinta de *input* no es nulo, escribir en la cinta de trabajo el símbolo leído y moverse a la derecha en ambas cintas.
    b. Si el símbolo leído en la cinta de *input* sí es nulo, moverse a la izquierda en ambas cintas. Cambiar el estado a q_{right}.
3. Si el estado actual es q_{right}:
    a. Si el símbolo leído en la cinta de *input* no es S_a, moverse a la izquierda en la cinta de *input*.
    b. Si el símbolo leído en la cinta de *input* sí es S_a, moverse a la derecha en la cinta de *input*. Cambiar el estado a q_{test}.
4. Si el estado actual es q_{test}:
    a. Si el símbolo leído en la cinta de *input* es nulo y el símbolo leído en la cinta de trabajo es S_a, escribir 1 en la cinta de resultado. Cambiar el estado a q_{halt}.
    b. Si el símbolo leído en la cinta de *input* no es igual al símbolo leído en la cinta de trabajo, escribir 0 en la cinta de resultado. Cambiar el estado a q_{halt}.
    c. Si el símbolo leído en la cinta de *input* sí es igual al símbolo leído en la cinta de trabajo, moverse a la derecha en la cinta de *input* y a la izquierda en la cinta de trabajo

Como conocemos el conjunto de estados, la definición formal de esta máquina de Turing requeriría que especifiquemos el alfabeto y la función de transición. La función de transición sería simplemente un *mappeo* o asociación explícita entre el dominio y el contradominio de δ. Probemos esta definición con el *input* 00100. Los símbolos con color representan la posición del lector en la cinta correspondiente:



El embrollo de la computabilidad está en saber si existen formas puramente mecánicas de resolver ciertos problemas (en otras palaras, si son efectivamente calculables o resolubles por medio de procedimientos efectivos). Goldreich separa estos problemas en dos tipos: **problemas de búsqueda** y **problemas de decisión**. Los problemas de búsqueda son los más fáciles de reconocer, pues son aquellos donde buscamos obtener la respuesta que corresponde a una entrada específica. Los problemas de decisión, como mencionado, tienen por objetivo responder si la entrada pertenece o no a algún conjunto predeterminado.

Goldreich también provee una definición más abstracta de computación: un proceso que modifica un ambiente aplicando repetidamente alguna(s) regla(s) simple(s), que es decir que depende y afecta una pequeña porción del ambiente. De acuerdo con esta idea, un **algoritmo** es un conjunto de reglas de computación que permiten obtener una funcionalidad deseada dentro de un ambiente artificial.

Entonces, la máquina de Turing es un **modelo de computación** que formaliza el ambiente y el tipo de reglas que definen una computación. Las diferentes posibles máquinas de Turing corresponden a algoritmos específicos. Con las máquinas de Turing que hemos definido podemos observar que cualquier función computable puede ser vista como efectivamente calculable ya que si una función es computable entonces existe una máquina de Turing que la compute, cuyo comportamiento es mecánico. Por otra parte, necesitamos un argumento para apoyar la identificación de funciones efectivamente calculables con funciones computables. Más adelante veremos que modelos de computación aparentemente más poderosos que las máquinas de Turing (e incluso modificaciones a las propias máquinas de Turing) pueden ser simulados por máquinas de Turing como las que hemos definido. Esto nos llevará a la siguiente conclusión: si alguna otra formalización de procedimiento efectivo permite calcular los resultados de una función específica, dicha función es también computable por una máquina de Turing.

La identificación de funciones efectivamente calculables con funciones computables es la esencia de la famosa **tesis de Church-Turing**, y su naturaleza (hacer equivalencia entre una noción intuitiva y una formal) le impide una demostración formal. Nada quita la posibilidad de que exista un dispositivo teórico o metodología que resuelva mecánicamente funciones consideradas no-computables, pero se ha encontrado mucha evidencia que apoya su inexistencia, como la equivalencia entre diferentes formalizaciones de procedimiento efectivo o función efectivamente calculable.

La máquina de Turing es una entre otras formalizaciones de procedimiento efectivo. Existen, por ejemplo:

- Recursividad general (Gödel-Kleene-Herbrand): lo veremos en el curso a profundidad. Define ciertas funciones básicas que permiten construir otras funciones, cada una equivalente a una máquina de Turing.
- Cálculo Lambda (Church): un modelo de computación basado en funciones. Un truco importante en ello es ***currying***, parecido a lo visto en la Regla de Horner: $(x, y) \rightarrow x^2 + y^2 \equiv x \rightarrow y \rightarrow x^2 + y^2$. Aunque no trataremos el tema directamente, un concepto fundamental es que las funciones se tratan como información, y la aplicación de funciones expresa cantidades. Aquí unos ejemplos:
    - El número 0 se expresa como $0(f, x) \equiv \lambda f.\lambda x.x$. Esto es una función (llamada "cero"). La función requiere un parámetro: f, que es otra función. Lo que hace la función cero con f es devolver la definición de otra función que toma un parámetro x y lo devuelve. O, sea, no usa f. Por tanto, "cero" es una función que expresa la aplicación de otra función f cero veces sobre un parámetro x.

- El número 1 se expresa como $1(f, x) \equiv \lambda f.\, \lambda x.\, (fx)$: la función "uno" es aquella que toma una función f y devuelve otra función que toma un parámetro x y que aplica f sobre x una única vez. "Aplicar" f dos veces será aplicar f sobre el resultado de aplicar f a una entrada. De esta forma se pueden definir funciones para cada número natural.
- La función sucesor es $S(n, f, x) \equiv \lambda n.\, \lambda f.\, \lambda x.\, \Big(f\big(n\, f(x)\big)\Big)$: n, un número, es una función que toma una función f y devuelve otra función que toma x y aplica f sobre x n veces. El número sucesor es aplicar f al resultado de esto; es decir, aplicar f un total de $n + 1$ veces sobre x.

Ejercicio de práctica H: defina las funciones lambda de suma de dos números y multiplicación de dos números.

Pese a sus diferencias, un argumento que apoya la equivalencia entre estas formalizaciones es que comparten las siguientes características:

- El conjunto de operaciones realizables (o instrucciones) es finito.
- Los pasos son discretos: no hay puntos intermedios entre un paso y otro.
- La computación es determinística: el resultado producido es siempre el mismo si se aplican las mismas condiciones.
- Memoria y tiempo se consideran no acotados, aunque no se debe depender de cantidades infinitas de ellos.
- Los datos involucrados en cada paso son finitos.

Fuentes:

- Arora, S., & Barak, B. (2007). Computational Complexity: A Modern Approach (Web Draft). Princeton University.
- Davis, M. (1958). Computability and Unsolvability. McGraw-Hill.
- Goldreich, O. (2008). Computational Complexity: A Conceptual Perspective. New York: Cambridge University Press.
- Har-Peled, S., & Parthasarthy, M. (2009, march 31). CS 373: Lecture Schedule. Retrieved from Illinois College of Engineering Course Websites: https://courses.engr.illinois.edu/cs373/sp2009/lectures/lect_18.pdf
- Jones, N. D. (1997). Computability and Complexity from a Programming Perspective. MIT Press.
- Martin, J. C. (2003). Introduction to Languages and the Theory of Computation. McGraw-Hill.
- Sipser, M. (2013). Introduction to the Theory of Computation. Boston: CENGAGE Learning.