Capítulo 1

Modelos de Computación

1.1 Introducción

En esta sección trataremos el concepto de "computación" o "algoritmo". Aunque este concepto es fundamental para nuestro objetivo, no lo definiremos formalmente. Lo que haremos es considerarlo como una noción intuitiva que es posible formalizar de muchas maneras. Lo que realmente necesitamos para nuestro propósito de estudiar complejidad de problemas es fijar un modelo de computación estándar. Nosotros elegimos, las máquinas de Turing y el modelo RAM que explicamos en esta sección.

Un *algoritmo* significa un procedimiento que sirve para una computación o una construcción (la computación de alguna función), la cual puede ser llevada a cabo "mecánicamente", sin pensar. Otro punto de vista es decir que es "una prescripción exacta del orden determinado en que ha de ejecutarse un sistema de operaciones para resolver todos los problemas de un cierto tipo." Esta no es realmente una definición rigurosa, pero uno de los propósitos de este curso es demostrar que posible hacer una discusión general en estos términos. (Esta discusión es a menudo formulada como la *tesis de Church.*) Un programa en un lenguaje de programación (C, JAVA o cualquier otro) es un buen ejemplo de una especificación de un algoritmo. Debido a que la naturaleza "mecánica" de un algoritmo es su característica más importante, para precisar la noción de algoritmo introduciremos varios conceptos de máquina abstracta que cumplen estas características.

Las *máquinas abstractas* o matemáticas computan alguna salida a partir de alguna entrada. La entrada y la salida puede ser una *palabra* de un alfabeto finito, es decir, una secuencia finita de símbolos del alfabeto, muchas veces también llamada *cadena.* Las máquinas abstractas son muy parecidas a los computadores que el lector conoce pero algo idealizadas: omitimos algunas características innecesarias para nuestra discusión (por ejemplo fallas del hardware) y agregamos una memoria que es infinitamente expansible.

El siguiente es un típico problema que a menudo resolvemos en los computadores: dada una lista de nombres, ordenarlos alfabéticamente. La entrada es una cadena que consiste de nombres como Roberto, Carlos, Alicia. La salida es también una cadena. El problema es computar una función que asigne a cada cadena de nombres su copia alfabéticamente ordenada.

En general, un problema algorítmico típico es realmente un conjunto de problemas del mismo

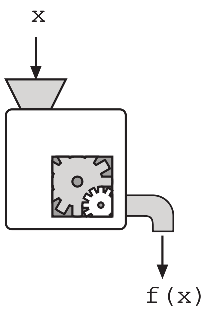

Figura 1.1: Una máquina abstracta

tipo. Por ejemplo, "encontrar la raíz cuadrada de un número" es un problema que realmente consiste de infinitos problemas particulares: uno para cada número. Estos problemas particulares se llaman *instancias* del problema general, y su número puede ser infinito, y por lo tanto pueden tener un tamaño arbitrariamente grande. Por eso debemos considerar una familia infinita de computadores finitos de tamaño creciente, o algún computador infinito idealizado. La última aproximación tiene la ventaja de que evita las preguntas sobre qué familias infinitas son las permitidas.

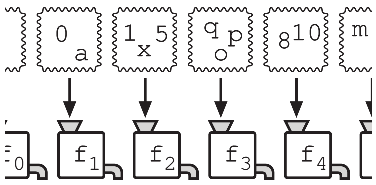

Figura 1.2: Un computador infinito

Históricamente, el primer modelo infinito puro de computación fue la *máquina de Turing*, introducida por el matemático inglés Alan Turing en 1936, antes de la creación de los computadores programables. Esencialmente el modelo consta de una unidad central que es finita (con una estructura independiente de la entrada) y una capacidad de almacenamiento (memoria) infinita. Más exactamente, la memoria es un arreglo uni-dimensional infinito de celdas. La unidad de control es un autómata finito capaz de hacer cambios locales arbitrarios a la celda de memoria explorada y de cambiar gradualmente la posición explorada. En las máquinas de Turing, es posible efectuar todos los cálculos que pueden ejecutarse en *cualquier otro* modelo de máquina abstracta. Esta noción de máquina es usada principalmente en investigaciones teóricas. Este modelo es menos apropiado para la definición de algoritmos concretos ya que su descripción es artificiosa, y sobretodo difiere de los computadores actuales en varios aspectos importantes.

La debilidad más importante de la máquina de Turing en comparación con los computadores reales es que su memoria no es accesible inmediatamente: para leer una celda de memoria distante, hay que recorrer todas las celdas intermedias. Esto es remediado por la *máquina de acceso aleatorio* (en inglés Random Access Machine, RAM).[1] La RAM puede alcanzar cualquier celda de memoria en un solo paso. Puede ser considerada un modelo simplificado de un computador del mundo real con la abstracción adicional de que tiene memoria ilimitada y la capacidad de almacenar números enteros de gran tamaño en cada una de sus celdas de memoria. La RAM puede ser programada en un lenguaje de programación arbitrario. Para la descripción de algoritmos, es práctico usar la RAM ya que es lo más

[1] Shepherdson y Sturgis en 1963 introdujeron este modelo que es el más similar al moderno computador digital, o más bien una abstracción de él. A diferencia de otros modelos, éste fué obtenido como una abstracción de los computadores digitales existentes ya en esa época.

cercano a la estructura de un programa real. Veremos que la máquina de Turing y la RAM son equivalentes desde varios puntos de vista, siendo el más importante entre ellos que las funciones que es posible computar en las máquinas de Turing y en la RAM son las mismas.

A pesar de sus aparentes limitaciones teóricas, un tercer modelo de computación muy usado son los *circuitos lógicos*. Un circuito lógico dado permite sólo entradas de un tamaño fijo, y por lo tanto, puede resolver sólo un número finito de problemas. Sin embargo, será evidente que para entradas de un tamaño fijado, cualquier función es computable por algún circuito lógico. Más aún, si restringimos el tiempo de cálculo, entonces la diferencia entre los problemas resolubles por circuitos lógicos o por máquinas de Turing o por RAM no es esencial. Dado que la estructura y el trabajo de los circuitos lógicos son muy transparentes y manejables, ellos juegan un rol muy importante en las investigaciones teóricas (especialmente en las demostraciones de cotas inferiores en complejidad).[2] En estos apuntes no los trataremos en profundidad.

Un concepto clave usado en la discusión de modelos de máquinas es el de *simulación*. Esta noción no será definida en su completa generalidad, ya que también se refiere a máquinas o lenguajes que todavía no han sido inventados. Pero su significado es posible aclararlo informalmente ahora: Decimos que la máquina M simula a la máquina N si a la máquina M se le pueden asociar los estados internos y las transiciones de N de manera tal que para una misma entrada, M compute la misma salida que N.

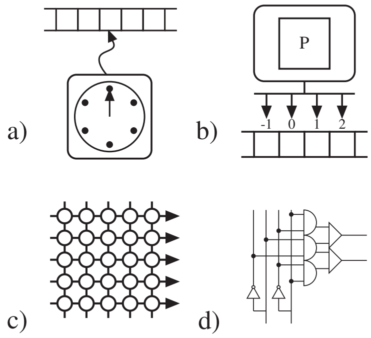

Figura 1.3: Modelos de computación: a) Máquina de Turing; b) Máquina de acceso aleatario; c) autómata celular; d) circuito lógico.

1.2 La máquina de Turing

Informalmente, una *máquina de Turing* es un autómata finito equipado con memoria ilimitada. Esta memoria está dada en la forma de una o más cintas, las cuales son infinitas en ambas direcciones. Cada cinta tiene una celda de partida especial que también podemos llamar la celda 0-ésima. Un número finito de celdas contiene símbolos de un alfabeto fijo Σ, y el resto el símbolo especial $*$, que que denota la "celda vacía".

Para acceder a la información en las cintas, asociado a cada cinta existe un cabezal de lectura-escritura que en cada paso se sitúa en una celda de la cinta. En esa posición, es posible leer el contenido y sobreescribir un nuevo símbolo en dicha celda. Los cabezales de lectura-escritura están conectados a una unidad de con-

[2]Si se agrega un reloj y un registro de memoria a un circuito lógico llegaremos a un *autómata finito* interconectado que conforma los componentes de hardware típicos de los computadores de hoy. Dejemos claro que un autómata finito dado, cuando es usado en entradas de tamaño arbitrario, puede computar sólo funciones muy primitivas, y no es un modelo de computación adecuado. Uno de los modelos más simples para una máquina infinita es conectar un número infinito de autómatas similares dentro de un arreglo. De esta manera obtenemos un *autómata celular*.

trol que cual es un autómata finito. Sus posibles estados forman un conjunto finito Γ. Hay un estado de partida "START" y un estado de parada "STOP". Inicialmente, la unidad de control está en el estado "START", y los cabezales se ubican en las celdas de partida de las cintas.

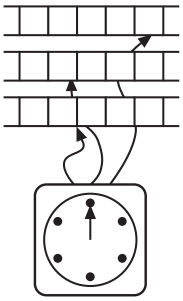

Figura 1.4: Máquina de Turing con tres cintas

En cada paso, cada cabezal lee el símbolo que contiene la celda de la cinta correspondiente, y lo envía a la unidad de control. Dependiendo de este símbolo y de su propio estado, la unidad de control efectúa tres operaciones (ver Figura **??**):

1. Envía un símbolo a cada cabezal para sobreescribir el símbolo en la cinta;

2. Envía a cada cabezal uno de los comandos "MOVER A DERECHA", "MOVER A IZQUIERDA", "PERMANECER";

3. Hace una transición a un nuevo estado (que puede ser el mismo que el anterior).

Por supuesto, los cabezales se encargan de estos comandos, los cuales completan un paso de la computación. La máquina se detiene cuando la unidad de control alcanza el estado "STOP". Aunque la descripción informal de arriba usa alguna jerga de ingeniería, no es difícil traducirla a términos puramente formales (matemáticos). Formalmente, una máquina de Turing T está completamente especificada por los siguientes datos: $T = (k, \Sigma, \Gamma, \alpha, \beta, \gamma)$ donde $k \geq 1$ es un número natural, Σ y Γ son conjuntos finitos, $* \in \Sigma$, START, STOP $\in \Gamma$, y α, β y γ son funciones arbitrarias,

$$\alpha : \Gamma \times \Sigma^k \longrightarrow \Gamma,$$
$$\beta : \Gamma \times \Sigma^k \longrightarrow \Sigma^k,$$
$$\gamma : \Gamma \times \Sigma^k \longrightarrow \{-1, 0, 1\}^k,$$

Aquí α especifica el nuevo estado, β entrega los símbolos a ser escritos en la cinta y γ especifica cómo se mueven los cabezales (Figura 1.5).

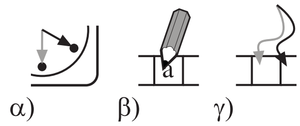

Figura 1.5: Tres funciones de una máquina de Turing: α) cambiar de estado, β) sobreescribir una cinta, γ) mover el cabezal una posición.

En lo que sigue fijaremos el alfabeto Σ y asumiremos que contiene, además del símbolo blanco $*$, por lo menos otros dos símbolos, digamos 0 y 1 (en la mayoría de los casos estos dos símbolos serán suficientes).

Por *entrada* de la máquina de Turing, entendemos las k palabras inicialmente escritas en las cintas. Siempre asumimos que éstas son escritas en las k cintas partiendo desde la posición 0. Así, formalmente la entrada de la máquina de Turing de k cintas es una k-tupla ordenada, cada elemento de los cuales es una palabra en Σ^*. A menudo la entrada consiste de una sola palabra x, en cuyo caso por "la entrada x" entenderemos la entrada consistente en la k-tupla $(x, 0, \ldots, 0)$. La *salida* de la máquina es una k-tupla ordenada que consiste de las palabras que quedan en las cintas cuando la máquina se detiene. Frecuentemente, estamos realmente interesados sólo en una palabra, el resto es “basura”. Cuando decimos que la salida es una palabra única y no especificamos cuál, entonces entenderemos que es la palabra escrita en la última cinta.

Es útil asumir que las palabras de entrada no contienen el símbolo $*$, pues de otro modo, no sería posible saber dónde está el fin de la palabra de entrada. Un simple problema tal como “encontrar el largo de la entrada” no sería soluble: como no hay manera de saber cuán lejos se ha movido el cabezal, no sabríamos si la entrada ya ha terminado. También asumiremos que durante su trabajo, la máquina de Turing lee su entrada completa: con esto, sólo estamos dejando de lado casos triviales. Denotaremos el alfabeto $\Sigma \setminus \{*\}$ por Σ_0.

Es posible definir las máquinas de Turing de muchas maneras, pero afortunadamente desde todos los puntos de vista importantes estas definiciones son equivalentes. Generalmente, las cintas son infinitas sólo en una dirección; su número puede ser virtualmente siempre restringido a dos y en muchos casos incluso a uno; podemos asumir que aparte del símbolo $*$ (el cual en este caso lo identificamos con el 0) el alfabeto contiene sólo el símbolo 1; sobre algunas cintas, podemos estipular que la máquina sólo puede leer desde ellas o sólo puede escribir en ellas. La equivalencia de estas variantes (desde el punto de vista de las computaciones realizables en ellas) puede ser verificada con mayor o menor trabajo, pero sin grandes dificultades. En este sentido, demostraremos en estas notas sólo lo que vayamos necesitando, pero creemos que esto da una familiaridad suficiente con las herramientas de tales simulaciones para que el lector haga algunos ejercicios por su cuenta.

1.3 Máquina de Turing Universal

Basado en lo precedente, podemos notar una diferencia significativa entre las máquinas de Turing y los computadores reales: para el cómputo de cada función, hay que construir una máquina de Turing separada, mientras que en los computadores reales controlados por programas, es suficiente escribir un programa apropiado. Mostraremos ahora que las máquinas de Turing también pueden ser operadas de esta manera: es posible construir una máquina de Turing en la cual, usando programas adecuados, es posible computar todo lo que es computable en cualquier máquina de Turing. Tales máquinas de Turing son interesantes no sólo porque semejan más a los computadores programables, sino porque también juegan un rol muy importante en muchas demostraciones.

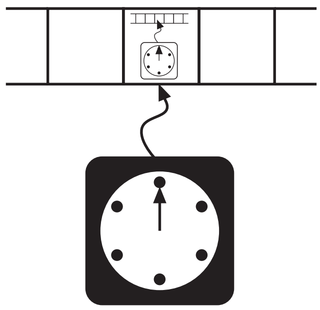

Figura 1.6: Una máquina de Turing Universal

Sean $T = (k+1, \Sigma, \Gamma_T, \alpha_T, \beta_T, \gamma_T)$ y $S = (k, \Sigma, \Gamma_S, \alpha_S, \beta_S, \gamma_S)$ dos máquinas de Turing ($k \geq 1$). Sea $p \in \Sigma_0^*$. Decimos que T *simula a S con el programa p* si para palabras arbitrarias $x_1, \ldots, x_k \in \Sigma_0^*$ la máquina T se detiene en un número finito de pasos con la entrada $(x_1, \ldots, x_k, p)$ si y sólo si S se detiene con la entrada $(x_1, \ldots, x_k)$ y si al momento de parar, las primeras k cintas de T tienen el mismo contenido que las cintas de S. Decimos que una máquina de Turing T de $(k+1)$ cintas es *universal* (con respecto a máquinas de Turing de k cintas) si para cada máquina de Turing S de k cintas sobre Σ hay una palabra (programa) p con el cual T simula a S.

Teorema 1 *Para cada número $k \geq 1$ y cada alfabeto Σ existe una máquina de Turing universal de $(k+1)$ cintas.*

DEMOSTRACIÓN. La idea básica de la construcción de una máquina de Turing universal consiste en escribir en la cinta $k+1$ una tabla que describa el trabajo de la máquina de Turing S a ser simulada (esencialmente las funciones α, β, γ). Aparte de esto, la máquina de Turing universal T debe llevar la cuenta del estado en que está en cada momento la máquina simulada S. (Esto no es sencillo, pues aunque cada máquina tiene un número finito de estados, como la máquina fija T debe simular *todas* las máquinas, no podría retener los estados de cada una). En cada paso, basándose en el estado de la máquina simulada S y en los símbolos leídos en las otras cintas, busca en la tabla: (1) el estado al cual S hace la transición, (2) lo que tiene que escribir en las cintas y (3) hacia dónde debe mover los cabezales. Daremos la construcción usando $k+2$ cintas.[3]

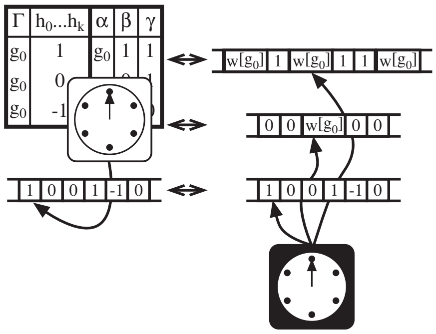

Figura 1.7: Codificación de una máquina de Turing

Para simplificar, asumiremos que S contiene los símbolos 0, 1 y -1. Sea $S = (k, \Sigma, \Gamma_S, \alpha_S, \beta_S, \gamma_S)$ una máquina de Turing de k cintas arbitraria. Codificaremos cada elemento g del conjunto de estados $\Gamma_S \setminus \{\text{STOP}\}$ con una palabra $w[g]$ sobre el alfabeto Σ_0^*. El "código" de una configuración[4] dada de la máquina S será la siguiente palabra:

$$w[g]h_1 \cdots h_k w[\alpha_S(g, h_1, \ldots, h_k)] \\ \beta_S(g, h_1, \ldots, h_k)\gamma_S(g, h_1, \ldots, h_k), \quad (1.1)$$

donde $g \in \Gamma_S$ es el estado dado de la unidad de control y $h_1, \ldots, h_k \in \Sigma$ son los símbolos leídos por cada cabezal.

Concatenando todas esas palabras (una por cada configuración posible de la máquina S) obtenemos una palabra a_S. Esto es lo que escribi-

[3]Podemos deshacernos fácilmente de la cinta $(k+2)$: su contenido (el cual siempre son sólo r celdas) será puesto en las celdas $-1, -2, \ldots, -r$ de la cinta $k+1$. Sin embargo, parece que todavía necesitamos dos cabezales en esta cinta: una se mueve en la mitad positiva mientras que la otra en la negativa (no necesitan cruzarse). Resolveremos esto duplicando cada celda: el símbolo original se queda en la mitad izquierda, y en la mitad derecha hay un 1 si el cabezal correspondiente estuviera ahí (la otra mitad derecha de celdas quedan vacías). No es difícil describir como un cabezal debe moverse en esta cinta para que pueda simular los movimientos de ambos cabezales originales.

[4]Una máquina de Turing queda completamente definida por las funciones α, β y γ. Luego una tabla que contenga como columnas el estado, los símbolos leídos por cada uno de los cabezales, y los resultados de las funciones α, β, γ aplicadas sobre esos datos da una descripción completa de esa máquina. La *configuración* de una máquina es una "foto" en un instante dado: su estado y los símbolos que leen sus cabezales en ese momento. Ver Figura 1.8.

Γ	Σ	α	β	γ
a	d	b	o	0
a	o	b	d	-1
b	d	c	d	0
b	o	c	d	-1
c	d	a	o	1
c	o	a	d	1

Figura 1.8: Tabla de Transiciones; Configuración.

mos en la cinta $k+1$; mientras en la cinta $k+2$ guardamos el estado (codificado) de la máquina S, inicialmente el estado START.

A continuación, construimos la máquina de Turing T que simula un paso de S como sigue. En en la cinta $k+1$ busca la palabra $w[g]$ correspondiente al estado g guardado en la cinta $k+2$ y a los símbolos leídos por los primeros k cabezales. Entonces lee desde ahí lo que tiene que hacer, y escribe el nuevo estado en la cinta $k+2$, deja que los primeros k cabezales escriban el símbolo apropiado y se mueve en la dirección apropiada. ⊙

1.4 Varias Cintas Versus Una Cinta

El siguiente resultado muestra que, en cierto sentido, no es esencial cuántas cintas tenga una máquina de Turing.

Teorema 2 *Para cada máquina de Turing S de k cintas existe una máquina de Turing T de una cinta que reemplaza a S de la siguiente manera: para cada palabra $x \in \Sigma_0^*$, la máquina S se detiene en un número finito de pasos en la entrada x si y sólo si T se detiene en la entrada x, y al momento de parar, la ultima cinta de S tiene escrito lo mismo que la cinta de T. Además si S demora N pasos entonces T demora $\mathcal{O}(N^2)$ pasos.*

Demostración. Debemos almacenar el contenido de las cintas de S en la única cinta de T. Para esto, primero debemos estirar la entrada escrita en la cinta de T: copiamos el símbolo encontrado en la i-ésima celda a la celda $(2ki)$-ésima. Esto puede ser hecho de la siguiente manera: primero empezando desde el último símbolo y saltando hacia la derecha copiamos cada símbolo $2k$ posiciones a la derecha. Mientras tanto, escribimos $*$ en las posiciones $1, 2, \dots, 2k-1$. Después, empezando desde el último símbolo, movemos cada símbolo del último bloque de no-blancos $2k$ posiciones a la derecha, etc.

Ahora, la posición $2ki + 2j - 2$ $(1 \leq j \leq k)$ corresponderá a la i-ésima celda de la cinta j, y la posición $2k + 2j - 1$ tendrá un 1 o una $*$ dependiendo de que si el cabezal correspondiente de S, en el paso correspondiente a la computación de S, esté explorando esa celda o no. También, marcamos con un 0 la primera celda par de los finales vacíos de las cintas. De esta forma asignamos una configuración de T a cada configuración de las computaciones de S.

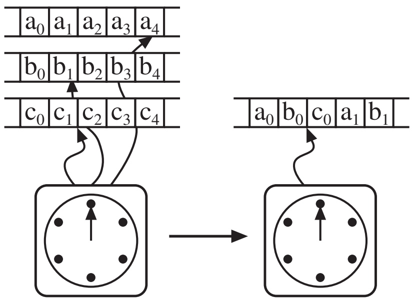

Figura 1.9: Simulación de varias cintas con una

Ahora mostraremos cómo T puede simular los pasos de S. Primero que nada T retiene el estado en que se encuentra S. También sabe cuál es el resto del número de la celda módulo $2k$ que fue explorado por su propio cabezal. Empezando desde la derecha, dejamos que el cabezal ahora haga una pasada a través de la cinta completa. En el momento en que alcance el fin sabrá cuáles son los símbolos leídos por los cabezales de S en este paso. Desde aquí, computa cuál será el nuevo estado de S, lo que deben escribir los cabezales y en qué dirección se deben mover. Empezando desde atrás, por cada 1 encontrado en una celda impar, puede sobreescribir correspondientemente la celda que esta antes de esta y también puede mover el 1 en $2k$ posiciones hacia la izquierda o hacia la derecha si esto fuese necesario. (Si en el intertanto, pudiera pasar más allá del 0 del principio o del final, entonces también podría moverla en 2k posiciones en la dirección apropiada.)

Cuando la simulación de la computación de S haya terminado, el resultado todavía debe ser comprimido: el contenido de la celda $2ki$ debe ser copiado a la celda i. Esto puede ser hecho en forma similar al "estiramiento" inicial. Obviamente, la máquina T descrita arriba computará las mismas cosas que la máquina S. El número de pasos está compuesto de tres partes: los tiempos tomados por el "estiramientos", la simulación y la "compresión". Sea M el número de celdas en la máquina T que siempre serán exploradas por la máquina; obviamente $M = \mathcal{O}(N)$. El "estiramiento" y la "compresión" necesitan tiempo $\mathcal{O}(M^2)$. La simulación de un paso de S necesita $\mathcal{O}(M)$ pasos, por lo tanto, la simulación necesita $\mathcal{O}(MN)$ pasos. Todo esto junto suma $\mathcal{O}(N^2)$ pasos. $\odot$

Como hemos visto, la simulación de una máquina de Turing de k cintas por una máquina de Turing de 1 cinta no es completamente satisfactoria: el número de pasos aumenta cuadráticamente. Esto no es sólo una debilidad de la construcción específica que hemos descrito; de hecho existen tareas computacionales que pueden ser resueltas en una máquina de Turing de 2 cintas en sólo N pasos, pero para las cuales cualquier máquina de Turing de 1 cinta necesita N^2 pasos. Describiremos un ejemplo sencillo de tales problemas. Un *palíndrome* es una palabra (digamos, sobre el alfabeto $\{0,1\}$) que no cambia cuando se lee en orden reverso. Por ejemplo, $x_1 \cdots x_n$ es un palíndrome si y sólo si $x_i = x_{n-i+1}$ para todo i. Analicemos el problema de reconocer un palíndrome.

Teorema 3 *1. Existe una máquina de Turing de 2 cintas que decide si una palabra de entrada $x \in \{0,1\}^n$ es un palíndrome en $\mathcal{O}(n)$ pasos.*

2. Cada máquina de Turing de 1 cinta que decida si una palabra de entrada $x \in \{0,1\}^n$ es un palíndrome tiene que hacer $\Omega(n^2)$ pasos en el peor de los casos.

DEMOSTRACIÓN. La parte (1) es fácil: por ejemplo, podemos copiar la entrada en la segunda cinta en n pasos, entonces movemos el primer cabezal hasta el principio de la entrada en n pasos más allá (dejamos el segundo cabezal en el final de la palabra), y comparamos x_1 con x_n, x_2 con x_{n-1} etc., en otros n pasos. Todo junto, esto tomaría sólo $3n$ pasos.

La parte (2) es más difícil de demostrar. Considere cualquier máquina de Turing de 1 cinta que reconozca palíndromes.

Para ser más específicos, diremos que termina escribiendo un "1" en el campo de partida en la cinta si la palabra de entrada es un palíndrome, y un "0" si no lo es.

Vamos a demostrar que para cualquier entrada de largo n, la máquina tendrá que hacer $\Omega(n^2)$ movimientos.

Será conveniente asumir que n es divisible por 3 (el argumento es muy similar en el caso general). Sea $k = n/3$. Restringimos las entradas a palabras en las cuales el tercio del medio son sólo ceros, es decir, a palabras de la forma: $x_1 \cdots x_k 0 \cdots 0 x_{2k+1} \cdots x_n$. (Si podemos mostrar que entre medio ya existen tales palabras, existe una para la cual la máquina debe trabajar $\Omega(n^2)$, estamos listos.) Fijemos cualquier j tal que $k \leq j \leq 2k$. Llamemos a la línea divisora entre los campos j y $j+1$ de la cinta el *corte* después de j.

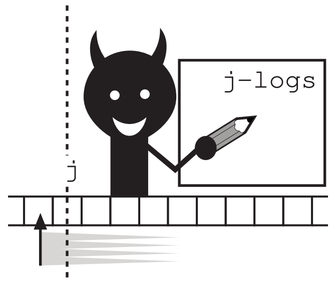

Figura 1.10: Punto de partida de la demostración

Imaginemos que tenemos un pequeño demonio sentado en esto y que está grabando el estado de la unidad central todo el tiempo que el cabezal pase entre estos campos.

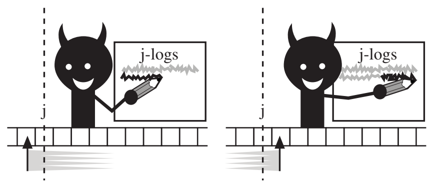

Figura 1.11: Demonio marcando el estado para formar el j-log

Al final de la computación tendremos la secuencia $g_1g_2\cdots g_t$ de elementos de Γ (el largo t de la secuencia puede ser diferente para distintas entradas), el j-log de la entrada dada.

La clave para la demostración es el Lema 1 que demostramos más adelante. Usando este lema, es fácil probar que para una entrada de largo m dado, el número de diferentes j-logs de largo menor que m es a lo más

$$1+|\Gamma|+|\Gamma|^2+\cdots+|\Gamma|^{m-1}=\frac{|\Gamma|^m-1}{|\Gamma|-1}<2|\Gamma|^{m-1}.$$

Esto es cierto para cualquier elección de j; por lo tanto, el número de palíndromes en que el j-log, para algún j, tiene largo menor que m es a lo más $2(k+1)|\Gamma|^{m-1}$. Existen 2^k palíndromes del tipo considerado, y también el número de palíndromes para los cuales j-log tiene largo al menos m para todo j es a lo menos

$$2^k-2(k+1)|\Gamma|^{m-1}. \tag{1.2}$$

Así pues si eligiéramos m número positivo, entonces habría un palíndrome para el cual el j-log tendría largo de a lo menos m para todo j. Esto implica que los demonios registrarían por lo menos $(k+1)m$ movidas, para que la computación tome a lo menos $(k+1)(m+1)$ pasos. Es fácil verificar que la elección de $m=n/(6\log|\Gamma|)$ hace la expresión (1.2) positiva, y entonces hemos encontrado una entrada para la cual, la computación toma al menos $(k+1)m>n^2/(6\log|\Gamma|)$ pasos.

Lema 1 *Sea* $x = x_1\cdots x_k0\cdots0x_k\cdots x_1$ *y* $y = y_1\cdots y_k0\cdots0y_k\cdots y_1$ *dos palíndromes distintos y* $k\le j\le 2k$. *Entonces sus* j*-logs son distintos.*

Demostración. Supongamos que los j-logs de x e y son los mismos, digamos $g_1\cdots g_t$. Considere la entrada $z=x_1\cdots x_k0\cdots0y_k\cdots y_1$. Notar que en estas entradas, todos los x_i están ubicados a la izquierda del corte y todos los y_i están ubicados hacia la derecha. Mostraremos que la máquina concluirá que z es un palíndrome, lo cual es una contradicción.

¿Qué pasa cuando iniciamos la máquina con la entrada z? Por un momento, el cabezal se moverá en los campos a la izquierda del corte, y por lo tanto, la computación procederá exactamente como si estuviera con la entrada x. Cuando el cabezal alcance primero el campo $j+1$, entonces estará en el estado g_1 por el j-log de x. Después, el cabezal utilizará su tiempo mirando hacia la derecha del corte. Esta parte de la computación será idéntica a la parte correspondiente de la computación con la entrada y: comienza en el mismo estado que la parte correspondiente de la computación de y lo hizo, y lee los mismos caracteres de la cinta, hasta que el cabezal regresa al campo j de nuevo. Podemos seguir la computación en la entrada z similarmente, y ver que la porción de la computación durante sus m pasos que queda a la izquierda del corte es idéntica a la porción correspondiente de la computación con la entrada x, y que la porción de la computación durante sus m pasos que queda a la derecha del corte es idéntica a la porción correspondiente de la computación con la entrada y. Dado que la computación con la entrada x termina escribiendo un "1" en el campo inicial, la computación con la entrada z termina de la misma manera. Esto es una contradicción. ⊙ ⊙

1.5 La máquina de Acceso Aleatorio (RAM)

Cuando se diseñan máquinas de Turing para diversas tareas, uno nota que una máquina de Turing gasta una gran cantidad de tiempo en mover sus cabezales de lectura y escritura de un extremo de la cinta al otro. Es posible diseñar artificios para evitar algunos de estos movimientos, pero siguiendo esta línea nos alejaríamos cada vez más de los computadores reales, los cuales tienen una memoria de "acceso aleatorio", es decir, que pueden acceder a *cualquier* campo de su memoria en un solo paso. Por lo tanto, uno puede querer modificar la manera en que ha diseñado las máquinas de Turing con memoria de manera que puedan alcanzar una celda de memoria arbitraria en un solo paso.

Por supuesto, la máquina debe saber cuáles celdas acceder y por lo tanto, tenemos que asignar direcciones a las celdas. Deseamos seguir

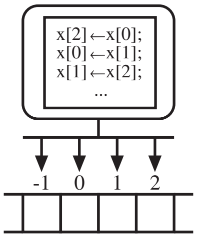

Figura 1.12: Máquina de acceso aleatorio, RAM

considerando la memoria como una memoria no acotada; por lo tanto, debemos permitir cualquier número entero como dirección válida. La dirección de la celda a acceder debe ser ella misma almacenada en alguna parte, por lo tanto, necesitamos que cualquier número entero pueda ser almacenado en una celda (en vez de un solo símbolo de un alfabeto finito, como es el caso de una máquina de Turing). Finalmente, haremos el modelo más parecido a las máquinas de hoy en día al hacerlo programable (nótese que esto le da el carácter análogo a la máquina de Turing universal). De esta manera obtenemos la noción de una *Máquina de Acceso Aleatorio* o máquina RAM (de las siglas en inglés *Random Access Machine*). Ahora, seamos más precisos. La *memoria* de una Máquina de Acceso Aleatorio es una secuencia doblemente infinita $\ldots, x[-1], x[0], x[1], \ldots$ de registros de memoria. Cada registro puede almacenar un entero arbitrario. En cualquier momento dado, sólo una cantidad finita de los números almacenados en la memoria son distintos de 0. La memoria de programa es una secuencia infinita (en un solo sentido) de registros llamados *líneas*. Aquí escribimos un programa de algún largo finito en cierto lenguaje de programación similar al lenguaje *assembler* de las máquinas reales. Es suficiente, por ejemplo, permitir las siguientes instrucciones:

$$x[i] \leftarrow 0;$$
$$x[i] \leftarrow x[i] \pm 1;$$
$$x[i] \leftarrow x[i] \pm x[j];$$
$$x[i] \leftarrow x[x[j]];$$
$$x[x[i]] \leftarrow x[j];$$
$$\texttt{IF } x[i] \leq 0 \texttt{ THEN GOTO } p.$$

Aquí, i y j son direcciones de los registros de memoria (enteros arbitrarios), p es la dirección de alguna línea de programa (un número natural arbitrario). La penúltima instrucción garantiza la posibilidad de acceso directo. Con esto, la memoria se comporta como un arreglo en un lenguaje de programación convencional como C. El conjunto exacto de instrucciones básicas es importante sólo en la medida en que sea suficientemente simple para implementar, suficientemente expresivo para hacer posibles las computaciones deseadas, y de cardinalidad finita. Por ejemplo, sería suficiente restringir los valores de i, j a $-1, -2, -3$. También podríamos omitir las operaciones de suma y resta entre los elementos unitarios ya que es posible escribir un programa para ellas. Por otro lado, también podríamos incluir la multiplicación, etc. La *entrada* de una Máquina de Acceso Aleatorio es una secuencia finita de números naturales escritos en registros de memoria $x[0], x[1], \ldots$ La Máquina de Acceso Aleatorio lleva un programa finito arbitrario. Ella se detiene cuando llega a una línea del programa que no contine ninguna instrucción. La *salida* está definida como el contenido que queda en los registros $x[i]$ después que el programa se detiene.

Es sencillo escribir subrutinas RAM para tareas simples que ocurren repetidamente en programas que resuelven cosas más complejas. Varias de éstas las dejamos para los ejercicios.

Aquí discutiremos tres tareas que necesitaremos posteriormente en este capítulo.

Ejemplo (asignación de valores). Sean i y j dos enteros. Entonces, la asignación

$$x[i] \leftarrow j$$

puede ser realizada por el siguiente programa RAM si j es positivo:

$$x[i] \leftarrow 0;$$

$$\left.\begin{array}{c} x[i] \leftarrow x[i]+1; \\ \vdots \\ x[i] \leftarrow x[i]+1; \end{array}\right\} j \ \textit{veces}$$

Si j es negativo, sencillamente reemplazamos la instrucción $x[i] \leftarrow x[i]+1$ por $x[i] \leftarrow x[i]-1$ y la ejecutamos $|j|$ veces.

Ejemplo (Suma de una constante). Sean i y j dos enteros. Entonces, la declaración

$$x[i] \leftarrow x[i] + j$$

puede ser realizada de la misma manera que en el ejemplo anterior, sólo que omitiendo la primera línea.

Ejemplo (Ramificaciones múltiples). Sean $p_0, p_1, \ldots, p_r$ índices de líneas de programa y supóngase que sabemos que los contenidos de los registros i satisfacen la condición $0 \leq x[i] \leq r$. Entonces, la declaración

$$\texttt{GOTO } p_{x[i]}$$

puede ser realizada por el programa RAM

$$\begin{array}{l} \texttt{IF } x[i] \leq 0 \texttt{ THEN GOTO } p_0; \\ x[i] \leftarrow x[i]-1; \\ \texttt{IF } x[i] \leq 0 \texttt{ THEN GOTO } p_1; \\ x[i] \leftarrow x[i]-1; \\ \vdots \\ \texttt{IF } x[i] \leq 0 \texttt{ THEN GOTO } p_r; \end{array}$$

Se debe tener cuidado cuando se incluye este último segmento de programa en otro programa, ya que modifica el contenido de $x[i]$. Si deseamos preservar el contenido de $x[i]$, pero tener un registro "borrador", digamos $x[-1]$, entonces podemos hacer

$$\begin{array}{l} x[-1] \leftarrow x[i]-1; \\ \texttt{IF } x[-1] \leq 0 \texttt{ THEN GOTO } p_0; \\ x[1] \leftarrow x[-1]-1; \\ \texttt{IF } x[-1] \leq 0 \texttt{ THEN GOTO } p_1; \\ x[-1] \leftarrow x[-1]-1; \\ \vdots \\ \texttt{IF } x[-1] \leq 0 \texttt{ THEN GOTO } p_r; \end{array}$$

Si no tenemos un registro "borrador" tenemos que hacer espacio para uno (debido a que no necesitaremos esos detalles, los dejaremos para los ejercicios).

Ahora mostraremos que las RAM y las máquinas de Turing pueden computar esencialmente las mismas funciones y sus tiempos de ejecución no difieren notablemente. Consideremos (por simplicidad) una máquina de Turing de una cinta, con alfabeto $\{*, 0, 1\}$.

Cada entrada $x_1, \ldots, x_n$ de la máquina de Turing (que es una secuencia de símbolos 0 y 1) puede ser interpretado como una entrada de la

RAM de dos maneras distintas: podemos escribir los números n, $x_1, \ldots, x_n$ en los registros $x[0], \ldots, x[n]$ o podemos asignar a la secuencia $x_1, \ldots, x_n$ un solo número natural, aquel cuya expresión en binario es $1x_1 \cdots x_n$. La salida de la máquina de Turing puede ser interpretada similarmente a la salida de la RAM.

Estudiaremos la correspondencia precisa para la primera interpretación.

Teorema 4 *Para cada máquina de Turing T (con múltiples cintas) sobre el alfabeto $\{*, 0, 1\}$, se puede construir un programa en la máquina RAM que computa para cada entrada las misma salida que la máquina de Turing T. Más aún, si T realiza N pasos entonces la RAM realiza $\mathcal{O}(N)$ pasos con números binarios de $\mathcal{O}(\log N)$ dígitos.*

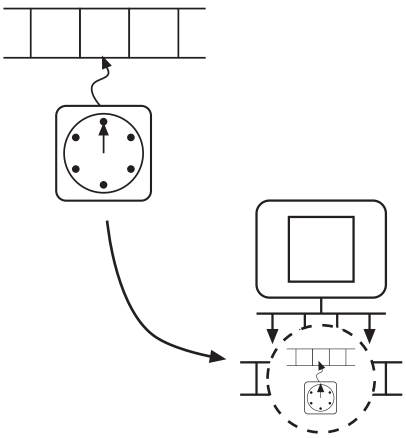

Figura 1.13: Una máquina de Turing puede ser simulada por una máquina RAM

Demostración. Sea $T = \langle 1, \{*, 0, 1\}, \Gamma, \alpha, \beta, \gamma \rangle$ y $\Gamma = \{1, \ldots, r\}$, donde $1 =$ START y $r =$ STOP. Durante la simulación de la computación de la máquina de Turing T, en el registro $x[i]$ de la RAM encontraremos el mismo número (0 ó 1) que en la i-ésima celda de la máquina de Turing, ó 2 si la celda i-ésima contiene $*$. El registro $x[-1]$ almacenará la posición del cabezal en la cinta, y el estado de la unidad de control será determinado según la línea que estemos en el programa. Nuestro programa estará compuesto de trozos $Q_{i,j}$ los cuales simulan la acción de la máquina de Turing cuando se encuentra en el estado i $(1 \leq i < r)$ y está leyendo el símbolo $j \in \{0, 1, 2\}$, y las líneas P_i que saltan a $Q_{i,j}$ si la máquina de Turing se encuentra en el estado i y leyendo el símbolo j. Ambas son sencillas de construir. P_i es simplemente

$$\texttt{GOTO } Q_{i,x[x[-1]]}$$

para $1 \leq i < r$ y P_r consiste de una sola línea de programa vacía. Las partes $Q_{i,j}$ del programa no son mucho más complicadas:

$$x[x[-1]] \leftarrow \beta(i,j);$$
$$x[-1] \leftarrow x[-1] + \gamma(i,j);$$
$$\texttt{GOTO } P_{\alpha(i,j)};$$

El programa mismo se vería como sigue:

$$x[-1] \leftarrow 0;$$
$$P_1$$
$$\vdots$$
$$P_r$$
$$Q_{1,0}$$
$$\vdots$$
$$Q_{r-1,2}$$

Con esto, hemos descrito la simulación de la máquina de Turing por la RAM. Para analizar el número de pasos y el tamaño de los números utilizados, es suficiente notar que en N pasos, la máquina de Turing puede escribir en a lo más $\mathcal{O}(N)$ registros. Luego en cada paso de la máquina de Turing se trabaja con números de largo $\mathcal{O}(\log N)$. $\odot$

Otra interpretación de la entrada de la máquina de Turing es, como se mencionó más arriba, mirar la entrada como un solo número natural a e ingresarlo a la RAM como tal. Entonces, este número a estará en el registro $x[0]$. En este caso, lo que podemos hacer es computar los dígitos de a con la ayuda de un programa sencillo y luego escribirlos (borrando el 1 de la primera posición) en los registros $x[1], \ldots, x[n]$ y aplicar la construcción descrita en el Teorema 4.

Nota 1 *En la demostración del Teorema 4, no usamos la instrucción $x[i] \leftarrow x[i]+x[j]$; esta instrucción es necesaria sólo si nos preocupa una buena contabilidad del número de pasos realizados. De hecho, sin esta instrucción, si permitimos números arbitrarios como entradas para la RAM, entonces el tiempo de ejecución y el número de pasos obtenidos pueden ser exponenciales incluso para problemas muy sencillos. Un ejemplo de ello es el problema de sumar el contenido a del registro $x[1]$ con el contenido b del registro $x[0]$. Esto es fácil de llevar a cabo en la RAM en un número constante de pasos. Pero si excluimos la instrucción $x[i] \leftarrow x[i]+x[j]$ entonces el tiempo requerido es al menos $\min\{|a|, |b|\}$.*

A continuación mostraremos el resultado inverso del Teorema 4. Observe que para cada programa dado para la RAM, podemos interpretar su entrada y su salida como palabras en el alfabeto $\{0, 1, -, \#\}^*$ (denotando todas las ocurrencias de enteros en binario, si se necesita, con signo y separados entre ellos por #).

Teorema 5 *Para cada programa RAM existe una máquina de Turing que computa para cada entrada la misma salida. Si la máquina RAM realiza N pasos, entonces la máquina de Turing corre en $\mathcal{O}(N^2)$ pasos.*

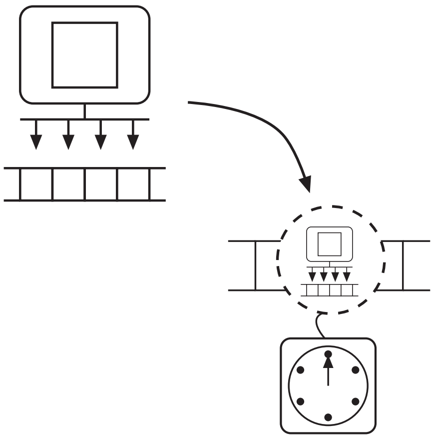

Figura 1.14: Una máquina RAM puede ser simulada por una máquina de Turing

Demostración. Simularemos la computación de la RAM por una máquina de Turing de cuatro cintas. Escribimos en la primera cinta el contenido de los registros $x[i]$ (en binario, y con signo si es negativo). Podemos representar el contenido de todos los registros (representando, digamos, el contenido 0 por el símbolo $*$). Sin embargo, esto puede causar un problema debido a la característica de acceso inmediato ("aleatorio") de la RAM. Por ejemplo, la RAM puede escribir incluso en el registro $x[2^N]$ usando sólo un paso con un entero de N bits. Por supuesto, el contenido de la mayoría de los registros con índices menores seguirían siendo 0 durante toda la computación. No es útil mantener el contenido de éstos en la cinta dado que la cinta sería muy larga y tomaría tiempo exponencial al cabezal para dirigirse al lugar donde debe escribir. Por lo tanto, almacenaremos en la cinta de la máquina de Turing sólo el contenido de aquellos registros en los cuales la máquina escribe. Por supuesto, también debemos registrar el número del registro en cuestión.

Lo que haremos entonces es que cuando la RAM escriba un número y en el registro $x[z]$, la máquina de Turing simule esto escribiendo la cadena $\#\#y\#z$ al final de su primera cinta (esto nunca reescribe esta cinta). Si la RAM lee el contenido de algún registro $x[z]$, entonces, en la primera cinta de la máquina de Turing, partiendo por atrás, el cabezal busca la primera cadena de la forma $\#\#u\#z$; este valor u muestra lo que fué escrito en el z-ésimo registro la última vez. Si no encuentra tal cadena entonces trata a $x[z]$ como 0.

Cada instrucción del "lenguaje de programación" de la RAM es fácil de simular por una máquina de Turing apropiada usando sólo las otras tres cintas. Nuestra máquina de Turing será una "supermáquina" en la cual un conjunto de estados corresponden a cada línea de programa. Estos estados generan una máquina de Turing que lleva la instrucción en cuestión y entonces trae los cabezales al fin de la primera cinta (a su última celda no vacía) y a la celda 0 de las otras cintas. El estado STOP de cada una de tales máquinas de Turing es identificado con el estado START de la máquina de Turing correspondiente a la próxima línea. (En caso de salto condicional, si $x[i] \leq 0$, la supermáquina se va al estado inicial de la máquina de Turing correspondiente a la línea p). El START de la máquina de Turing correspondiente a la línea 0 también es el START de la supermáquina. En vez de esto tenemos aún otro estado, STOP, que corresponde a la línea de programa vacía.

Es fácil de ver que la máquina de Turing así construida simula el trabajo de la RAM paso por paso. Esto conduce a que la mayoría de las líneas de programa se ejecuta en un número de pasos proporcional al número de dígitos de los números que ocurren en éste, es decir, al tiempo de ejecución que la RAM gasta en éste. La excepción es la lectura del siguiente, para la cual, posiblemente debe ser buscado en la cinta completa. Dado que el largo de la cinta es N, el número total de pasos es $\mathcal{O}(N^2)$. $\odot$

1.6 Otros modelos

Los modelos de computación están estrechamente ligados a la "definición" de algoritmo. Los lógicos formalizaron este concepto con diversas definiciones que capturan o aproximan el concepto de "procedimiento mecánico" en términos matemáticos. En este marco aparecieron casi simultáneamente, a partir de los años treinta del siglo XX, varias definiciones que intentaban capturar el concepto de algoritmo. A continuación mencionaremos algunas de las más importantes.

Funciones recursivas. Este es el intento más antiguo y el que contiene las ideas seminales sobre este tema.

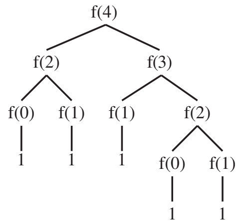

Figura 1.15: Una función recursiva

Es obra de los lógicos K. Gödel y J. Herbrand, que investigaron que tipo de funciones de números son "calculables" por medios "finitos".[5] Entonces, un procedimiento de decisión es una función "calculable" $f : \mathbb{N} \longrightarrow \{0, 1\}\}$, que a cada número (problema) le asigna 1 (Sí) o 0 (No).

A partir de esto, la intuición es la siguiente: cualquier función que sea "calculable" debe serlo debido a un proceso parecido a las definiciones por inducción matemática. Por ejemplo las ecuaciones $m(x, 1) = x$ y $m(x, y+1) = m(x, y) + x$ definen inductivamente la multiplicación a partir de la suma. No es difícil ver que todas las funciones definibles por ecuaciones de este tipo son calculables mecánicamente (por ejemplo usando las ecuaciones del ejemplo: calculamos $m(2, 3) = m(2, 2) + 2$; luego $m(2, 2) = m(2, 1) + 2$, y finalmente $m(2, 1) = 2$ y luego sumamos todos los resultados correspondientes).

Las *funciones recursivas* se obtienen generalizando estas ideas al permitir usar parámetros y ecuaciones funcionales implícitas.

$$((\lambda n.(n^3)\ \lambda n.(2n))\ 5) = (\lambda n.(n^3)\ (\lambda n.(2n)\ 5))$$

Figura 1.16: Cálculo λ

Cálculo λ. Este es el enfoque desde el punto de vista de los lenguajes funcionales y de la teoría de tipos, y fué inventado por A. Church siguiendo ideas de Shönfinkel y Curry.

[5] Esta restricción a números no es importante pues es posible codificar palabras por números. Las oraciones son sucesiones de palabras, las palabras son sucesiones de símbolos (letras, espacio en blanco, puntuación). Si le asignamos un número a cada símbolo, no es difícil asignarle números únicos a sucesiones de símbolos, etc.

Toda función calculable se puede expresar como una composición sucesiva de funciones más simples. Para formalizar esto, necesitamos desarrollar un sistema simbólico para expresar funciones y su composición. Esto es precisamente el *cálculo lambda* (cálculo λ). Las ideas básicas son: (1) Ua función consta de una entrada y una salida, y necesitamos una notación que indique esto explícitamente. Church eligió $\lambda x.f(x)$, donde la primera x indica el parámetro de entrada, lo que viene después del punto es la expresión a ser evaluada (la salida). (2) Estudiar la mecánica de la composición y evaluacion de funciones; Por ejemplo, ¿cómo se evalúa $(f \circ g)(5)$, donde $f(n) = n^3$ y $g(n) = 2n$? Si queremos analizar procesos mecánicos de cálculo, hay muchas soluciones. Por ejemplo, ¿evaluamos $2n = 10$ y luego multiplicamos $10 \times 10 \times 10$?, o ¿evaluamos $g(n)^3 = 2n \times 2n \times 2n = 8n^3$ y luego 5^3, etc.?

Es posible acercarse a este enfoque desde un punto de vista algo más abstracto (aunque clave en lenguajes de computación): nótese la odiosa diferencia que hacemos en general entre una función y los elementos a los que se aplica. Si les damos a las funciones y a sus argumentos los mismos "derechos de ciudadanía" tenemos una teoría general de funciones. Así podemos hablar de funciones aplicándose sobre funciones, etc. Si se llevan estas ideas a su límite, se obtiene lo que se conoce como *cálculo* λ.

$$\begin{array}{rcl} \mathrm{I} & \longrightarrow & \mathrm{ED} \\ \mathrm{L} & \longrightarrow & \mathrm{PI} \\ \mathrm{OT} & \longrightarrow & \mathrm{R} \\ & \longrightarrow & \mathrm{LOTO} \end{array}$$

Figura 1.17: Algoritmo de Markov

Algoritmos de Markov. Es posible ver un cómputo como la aplicación sucesiva de reglas. Este es el enfoque desde el punto de vista de los lenguajes y las gramáticas formales. Fué introducido por Markov en los años cuarenta.

Un proceso de cómputo mecánico puede verse como un proceso de transformaciones simbólicas $A_1 \cdots A_n \longrightarrow B_1 \cdots B_m$, donde hay muchas reglas como ésta. Un número finito de ellas determina un algoritmo; si uno agrega ciertas condiciones de normalización se obtienen los *algoritmos de Markov*. Este ha sido un enfoque muy productivo, especialmente en el estudio de gramáticas formales y reescritura.

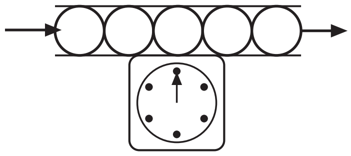

Figura 1.18: Sistema de Post

Sistemas de Post. Tempranamente ciertas personas se dieron cuenta que la computación no tenía que ver sólo con números. (Hagamos notar que incluso Turing presentó su primera máquina como calculando números.) El lógico E. Post introdujo ciertos sistemas en la década del cuarenta, que contienen un fuerte componente simbólico, es decir, calculan con símbolos. En ciertos aspectos, es similar en espíritu a los algoritmos de Markov. Es muy útil para codificar problemas insolubles.

Los sistemas anteriores representan diversos puntos de vista, tanto desde el punto de vista de los datos que manejan, como de los procesos en que están basado. ¿Cómo compararlos? Es posible hacerlo haciendo una correspondencia de números con palabras y viceversa, y notando que en el fondo todos pueden considerarse como funciones con ciertas entradas y salidas. Usando estas técnicas es posible demostrar:

Teorema 6 (Gödel, Turing, Church, Post, Markov, etc.) *Si codificamos adecuadamente las entradas y salidas, cada uno de los sistemas anteriores define exactamente la misma clase de funciones.*

Es decir, las definiciones son equivalentes. Esta es una gran sorpresa, considerando la diversidad de puntos de vista que representan. Las funciones de números que calculan estos sistemas se llaman *recursivas* y todas son calculables por algoritmos. ¿Hay algoritmos fuera de los representados por estas funciones? Obsérvese que de nuevo tenemos el mismo problema: ¿que se entiende por algoritmo? Respondiendo a esta pregunta, A. Church y A. Turing presentaron en forma independiente y casi paralelamente la siguiente conjetura:

Tesis de Church. *Las funciones de números susceptibles de ser calculadas por algoritmos son* precisamente *aquellas definidas por el teorema anterior.*

Nótese el carácter de la *tesis de Church*: Una afirmación que no es posible demostrar ni refutar (pues no tenemos una definición formal de algoritmo). Aún a pesar

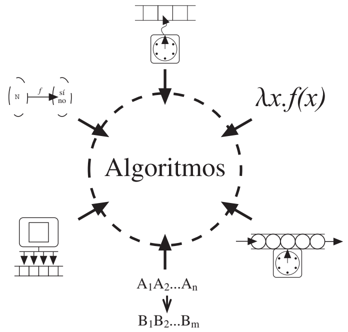

Figura 1.19: La tesis de Church

de esto, la tesis rápidamente obtuvo aprobación general debido a la abrumadora cantidad de evidencia en su favor (en particular el teorema antes mencionado).[6]

1.7 Ejercicios

1. Construya una máquina de Turing que calcule las siguientes funciones:
   (a) $x_1 \cdots x_m \mapsto x_m \cdots x_1$.
   (b) $x_1 \cdots x_m \mapsto x_1 \cdots x_m x_1 \cdots x_m$.
   (c) $x_1 \cdots x_m \mapsto x_1 x_1 \cdots x_m x_m$.
   (d) Para cada entrada de largo m que consiste sólo en 1's, la forma binaria de m; para todas las otras entradas, nunca se detiene.

2. Asuma que tenemos dos máquinas de Turing, calculando las funciones $f : \Sigma_0^* \longrightarrow \Sigma_0^*$ y $g : \Sigma_0^* \longrightarrow \Sigma_0^*$. Construya una máquina de Turing que calcule la función $f \circ g$.

3. Construya una máquina de Turing que realice $2^{|x|}$ pasos para cada entrada x.

4. Construya una máquina de Turing que con la entrada x, se detenga en un número finitos pasos si y sólo si el símbolo 0 aparece en x.

5. Escriba una simulación de una máquina de Turing con cinta infinita para ambos lados por una maquina Turing con una cinta que es infinita en una sola dirección.

6. Muestre que si simulamos una máquina con k cintas en la máquina de Turing universal de $k+1$ cintas, entonces para una entrada arbitraria, el número de pasos aumenta sólo por un factor multiplicativo proporcional al largo del programa simulado.

7. Sean T y S dos máquinas de Turing con una cinta. Decimos que T simula el trabajo de S por un programa p (aquí $p \in \Sigma_0^*$) si para todas las palabras $x \in \Sigma_0^*$, la máquina T se detiene en las entradas $p * x$ en un número finito de pasos si y sólo si S se detiene en la entrada x, y al momento de parar, la cinta de T y la cinta de S tienen el mismo contenido.

   Demuestre que existe una máquina de Turing T de una cinta que puede simular el trabajo de cualquier otra máquina de Turing de una cinta en este sentido.

8. En la simulación de máquinas de k cintas por máquinas de una cinta presentada en el

[6] En la época que se formuló la tesis, el lógico húngaro Kalmar fué unos de los pocos que se opuso a aceptarla debido a consideraciones lógicas.

Es bueno hacer notar que detrás de la tesis de Church hay una serie de consideraciones de carácter *físico*: determinismo, finitud, comunicación de información, etc. No es sorprendente por ello que cuando la validez de la universalidad de estos conceptos se pone en duda, aparezcan nuevos modelos de computación. Nótese que la computación digital moderna es la prueba más fehaciente de que la tesis de Church domina el ámbito de la computación.

texto, el control finito de la máquina simuladora T es un poco más grande que la de la máquina simulada S: más aún, el número de estados de la máquina simuladora depende de k. Demuestre que esto no es necesario: existe una máquina de una cinta que puede simular cualquier máquina de k cintas.

9. (*) Muestre que toda máquina de Turing de k cintas puede ser simulada por una de dos cintas de manera que si en alguna entrada, la máquina de k cintas hace N pasos, entonces la de dos cintas hace a lo más $\mathcal{O}(N \log N)$ pasos. (Ayuda: en vez de mover los cabezales simulados, mueva las cintas simuladas (Hennie-Stearns)).

10. Cinta bidimensional.

    (a) Defina la noción de máquina de Turing con cinta bidimensional.

    (b) Muestre que una máquina de Turing de dos cintas puede simular una máquina de Turing con cinta bidimensional.

    (c) Estime la eficiencia de la simulación anterior.

11. Sea $f : \Sigma_0^* \longrightarrow \Sigma_0^*$ una función. Una máquina de Turing *online* contiene, aparte de las cintas usuales, dos cintas extras. La cinta de entrada es legible sólo en una dirección, la cinta de salida puede ser escrita sólo en una dirección. Una máquina de Turing online T computa la función f si en una sola corrida, para cada n, después de haber recibido n símbolos $x_1, \ldots, x_n$, escribe $f(x_1, \ldots, x_n)$ en la cinta de salida, terminada con un blanco.

    Encuentre un problema que puede ser resuelto más eficientemente en una máquina de Turing online con cinta de trabajo bidimensional que con una cinta de trabajo unidimensional. (Ayuda: en una cinta bidimensional, cualquiera de los n bits puede ser accesados en $\sqrt{n}$ pasos. Para aprovechar esto, deje a la entrada representar una secuencia de operaciones en una "base de datos": inserciones y consultas, y defina f como la interpretación de estas operaciones.)

12. Cinta de árbol.

    (a) Defina la noción de una máquina de Turing con una cinta de árbol.

    (b) Muestre que una máquina de Turing con dos cintas puede simular una máquina de Turing con cintas de árbol. (Ayuda: Almacene en la cinta 1 cada símbolo de la cinta bidimensional, un número arbitrario que identifique su posición original y los números que identifican estos padres e hijos.)

    (c) Estime la eficiencia de la simulación anterior.

    (d) Encuentre un problema que pueda ser resulto más eficientemente con una cinta de árbol que con cualquier cinta de dimensión finita.

13. Escriba un programa para la RAM que para un número positivo dado a

    (a) determine el número m más grande con $2^m \leq a$;

    (b) compute su representación en base 2.

14. Sea $p(x) = a_0 + a_1 x + \cdots + a_n x^n$ un polinomio con coeficientes enteros $a_0, \ldots, a_n$. Escriba un programa RAM que compute los coeficientes del polinomio $(p(x))^2$ a partir de

aquellos de $p(x)$. Estime el tiempo de ejecución de su programa en términos de n y $K = \max\{|a_0|, \ldots, |a_n|\}$.

15. Demuestre que si a una RAM no se le permite utilizar la instrucción $x[i] \leftarrow x[i] + x[j]$, entonces sumar el contenido a de $x[1]$ al contenido de b de $x[2]$ toma al menos $\min\{|a|, |b|\}$ pasos.

16. Dado que la RAM es una sóla máquina, el problema de la universalidad no puede ser tomado exactamente de la misma manera que para las máquinas de Turing: en cierto sentido esta máquina RAM es universal. Por lo tanto, la siguiente propiedad de "auto-simulación" de la RAM se acerca más. Para un programa RAM p y una entrada x, sea $R(p, x)$ la salida de la RAM. Sea $\langle p, x\rangle$ la entrada de la RAM que obtenemos escribiendo los símbolos de p uno a uno en los registros $1, 2, \ldots$ seguido por un símbolo $\#$ y entonces por registros conteniendo la secuencia original x. Demuestre que existe un programa RAM u tal que para todos los programas RAM p y entradas x se cumple que $R(u, \langle p, x\rangle) = R(p, x)$.

17. (Máquina puntero). Después de haber visto cintas dimensionalmente finitas y una cinta de árbol podríamos buscar considerar una máquina con un grafo dirigido más general como su medio de almacenamiento. Cada celda c tiene un número fijo de aristas numeradas $1, \ldots, r$. Cuando el cabezal explora una cierta celda se puede mover a cualquiera de las celdas $\lambda(c, i)(i = 1, \ldots, r)$ alcanzables desde ella a través de las aristas salientes. Dado que parece imposible concordar en el mejor grafo, introducimos un nuevo tipo de operación elemental: cambiar la estructura de almacenamiento del grafo localmente, alrededor de el cabezal explorada. Transformaciones arbitrarias pueden ser logradas aplicando repetidamente las siguientes tres operaciones (e ignorando los nodos que están inaccesibles): $\lambda(c, i) \leftarrow$ New, donde New es un nuevo nodo; $\lambda(c, i) \leftarrow \lambda(\lambda(c, j))$ y $\lambda(\lambda(c, i)) \leftarrow \lambda(c, j)$. Una máquina con esta estructura de almacenamiento y estas tres operaciones además de las usuales operaciones de las máquinas de Turing será llamada una *Máquina Puntero.* Llamemos RAMá la RAM en que las operaciones de suma y resta son omitidas; sólo la operación $x[i] \leftarrow x[i] + 1$ es permitida. Demuestre que la Máquina Puntero es equivalente a la RAM', en el siguiente sentido. Para cada Máquina Puntero existe un programa RAM'que computa para cada entrada la misma salida. Si la Máquina Puntero corre en tiempo N entonces la RAM'corre en $\mathcal{O}(N)$ pasos. Para cada programa RAMéxiste una Máquina Puntero que computa para cada entrada la misma salida. Si el programa RAM'corre en tiempo N entonces la Máquina Puntero corre en $\mathcal{O}(N)$ pasos.