Autómatas y Lenguajes Formales
Nota 13. Máquinas de Turing*

Noé Salomón Hernández S.

1. Problemas indecidibles

Hay problemas específicos que no se pueden resolver usando una computadora. Estos problemas son llamados *indecidibles*. Por ejemplo, considere el problema en el que se desea determinar si dado un programa en C, con una entrada dada, imprime en pantalla `hello, world` como los primeros 12 símbolos que imprime. Este problema se conoce como el problema *hola, mundo*. Podemos demostrar que es imposible producir un programa que resuelva el problema *hola, mundo*. Así, dicho problema es indecidible.

Es fácil ver porqué casi todos los problemas deben ser indecidibles para cualquier sistema que involucre programación en una computadora. Recodemos que un *problema* es la pertenencia de una cadena a un lenguaje. El número de lenguajes diferentes sobre un alfabeto Σ de más de un símbolo no es numerable, porque este número equivale a la cardinalidad del conjunto potencia $\mathcal{P}(\Sigma^*)$. Es decir, no hay manera de asignar números enteros a los lenguajes tales que cada lenguaje tenga un entero, y cada entero sea asignado a un lenguaje.

Por otro lado, los programas son numerables ya que son cadenas finitas sobre un alfabeto finito (usualmente un subconjunto del alfabeto ASCII). Es decir, podemos ordenarlos por longitud, y para programas de la misma longitud, los ordenamos lexicográficamente (orden del diccionario). Así podemos hablar de el primer programa, el segundo programa, y en general, de el i-ésimo programa para cualquier entero i.

Sabemos que hay una infinidad menos de programas que de problemas. Si tomamos un lenguaje al azar, casi seguramente corresponderá a un problema que no es un programa, por lo que el problema es indecidible. La única razón por la que la mayoría de los problemas *parecen* ser decibles es que rara vez estamos interesados en problemas al azar. Por lo regular, tendemos a considerar problemas muy sencillos y bien estructurados, y éstos en efecto son decidibles. Sin embargo, incluso entre los problemas que nos interesan y que pueden ser definidos concisa y claramente, encontramos varios que son indecidibles; el problema *hola, mundo* es el caso.

No podemos basar el desarrollo de la teoría de la indecibilidad en términos de programas escritos en C, no sería elegante, o posible en general por las siguientes razones:

A. tendríamos que preocuparnos por el ambiente y errores de ejecución;

B. los elementos del lenguaje son bastante complejos;

*Esta nota se basa en el trabajo del prof. Rajeev Motwani que se encuentra aquí, en los libros *Introduction to Automata Theory, Languages, and Computation*; *Introduction to the Theory of Computation*; *Introduction to Languages and the Theory of Computation* y *Automata and Computability*.

C. tendríamos que preocuparnos por memoria finita;

D. el estado de un programa en C es muy complejo; y

E. se podría argumentar que los resultados se aplicarían únicamente al lenguaje C.

Por lo que basamos nuestra teoría de indecibilidad en un modelo de computación llamado la *máquina de Turing*. Tal modelo tiene las siguientes caracterísitcas:

I. emplea un lenguaje de programación sencillo y universal;

II. su estado es fácil de representar, así como su configuración de ejecución;

III. puede simular cualquier computadora o lenguaje; y

IV. todos los intentos por describir un modelo de computación más poderoso terminan como un caso especial de la máquina de Turing.

La hipótesis de Church-Turing

No existe un modelo de computación más poderoso que las máquinas de Turing. Por lo que este formalismo modela cualquier problema computable.

2. La Máquina de Turing (MT)

Una máquina de Turing (MT) puede representarse por el diagrama en la Figura 1.

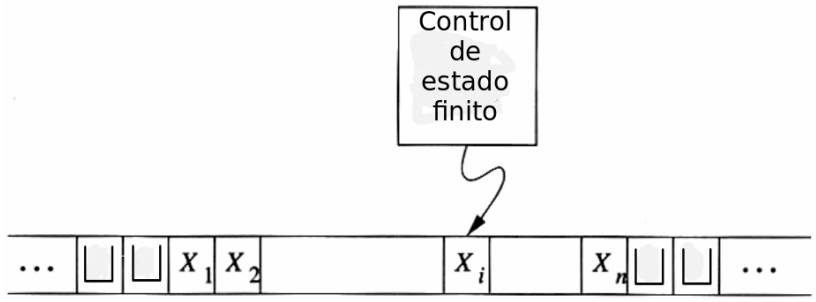


Figura 1: Una Máquina de Turing.

El lenguaje que se usa para programar una MT se codifica en las transiciones. Dichas transiciones dependen del estado actual y del símbolo en la celda que lee la cabeza lectora de la cinta; el efecto que producen es llegar a un estado posiblemente nuevo, sobreescribir la celda en la cinta y mover la cabeza lectora a izquierda o derecha.

Podemos apreciar cierta analogía entre las computadoras reales y las MTs. El CPU corresponde al control de estados finito, y la memoria a la cinta. Aunque en esta abstracción que vamos a estudiar se pierden algunos aspectos:

1) la memoria no es de acceso aleatorio;

2) el conjunto de instrucciones es limitado; y

3) se omiten varias otras características presentes en las computadoras reales.

Sin embargo, la MT captura los aspectos esenciales, y al ser una abstracción sencilla permite un razonamiento mejor y más claro.

Definición 2.1 Formalmente, una máquina de Turing Determinista (MTD) M es una tupla de la forma $M = (Q, \Sigma, \Gamma, \delta, q_0, \sqcup, F)$, donde

- Q, es un conjunto finito de estados;
- Σ, es un conjunto finito de símbolos que forma el alfabeto de entrada;
- Γ, es un conjunto finito de símbolos que forma el alfabeto de la cinta;
- δ, la función de transición, la cual será explicada más adelante;
- $q_0 \in Q$, el estado inicial;
- $\sqcup \in \Gamma$, el símbolo en blanco de la cinta; y
- $F \subseteq Q$, el conjunto de estados finales.

La función de transición tiene la forma:

$$\delta : Q \times \Gamma \longrightarrow Q \times \Gamma \times \{L, R\}$$

donde $\{L, R\}$ representa el posible movimiento a izquierda, L, o a derecha, R, de la cabeza lectora. Luego, $\delta(q, X) = (p, Y, L)$ indica que si se está en el estado q y la cabeza lectora lee el símbolo X, entonces la MT se mueve al estado p, reemplaza X por Y en la celda de la cinta, y la cabeza lectora se mueve una celda a la izquierda. Como estamos definiendo una MT determinista, tenemos por cada $\delta(q, X)$ a lo más un posible movimiento, aunque $\delta(q, X)$ puede no estar definida.

Notemos que la cadena de entrada w forma parte inicial de la cinta, así que $\Sigma \subseteq \Gamma$. De manera que las celdas del resto de la cinta inicialmente contienen a $\sqcup$, por lo que $\sqcup \in \Gamma - \Sigma$. Así, en un principio, una MT se encuentra en el estado q_0, con w en la cinta rodeada de símbolos en blanco, y la cabeza lectora leyendo el símbolo en w más a la izquierda.

¿Cuándo es una cadena w aceptada por una Máquina de Turing Determinista? Acepta cuando la MTD comienza con w en la cinta y en algún momento llega a un estado final. Bien podríamos suponer que los estados finales son *estados de paro*, en el sentido que no hay transiciones definidas que salgan de ellos.

¿Cuándo es una cadena w rechazada por una Máquina de Turing Determinista? Rechaza cuando la MTD se detiene en un estado que no es final, o cuando nunca se detiene al entrar en un ciclo infinito. Aquí hay una diferencia significativa con respecto a los autómatas finitos y a los PDAs, debido a que éstos se detienen cuando se alcanza el fin de la cadena de entrada y entonces se verifica si se llegó a un estado final, si es que antes no se presentó una transición indeterminada. No así en una MT ya que la cadena de entrada no se consume sino que se tiene dada en la cinta, por lo que necesitamos una noción explícita de paro.

Definición 2.2 Una máquina de Turing se detiene o para cuando se encuentra en un estado q y la cabeza lectora lee un símbolo X tal que $\delta(q, X)$ no está definida.

Definición 2.3 Una máquina de Turing M es **total** si se detiene sobre toda cadena de entrada, es decir, para cualquier cadena de entrada, M se detiene aceptando o rechazando.

Como los estados finales no tienen transiciones de salida, entonces la MT siempre se detiene una vez que entra en un estado final.

2.1. Configuraciones

Una configuración es útil para describir de manera concisa la composición de una MT. Este concepto lo estudiamos anteriormente para PDAs.

Definición 2.4 Una configuración se denota como $\alpha_1 p\, \alpha_2$ con $p \in Q$ y $\alpha_1, \alpha_2 \in \Gamma^*$, e indica que la parte de la cinta que no está en blanco contiene a $\alpha_1\alpha_2$, además la cabeza lectora apunta al símbolo más a la izquierda de α_2, y el estado actual es p. Dada la cadena de entrada w, la **configuración inicial** es $q_0 w$. Una **configuración de aceptación** es aquella en la que el estado actual es final.

Para mostrar la ejecución de una MT definimos $\vdash$ de la manera siguiente:

- Si $\delta(q, \alpha_2^1) = (p, \beta, R)$, entonces

$$\alpha_1^1\alpha_1^2 \ldots \alpha_1^n q \alpha_2^1\alpha_2^2 \ldots \alpha_2^m \vdash \alpha_1^1\alpha_1^2 \ldots \alpha_1^n \beta p \alpha_2^2 \ldots \alpha_2^m,$$

- Si $\delta(q, \alpha_2^1) = (p, \beta, L)$, entonces

$$\alpha_1^1\alpha_1^2 \ldots \alpha_1^n q \alpha_2^1\alpha_2^2 \ldots \alpha_2^m \vdash \alpha_1^1\alpha_1^2 \ldots \alpha_1^{n-1} p \alpha_1^n \beta \alpha_2^2 \ldots \alpha_2^m.$$

Definición 2.5 Definimos a $\vdash^*$ como la cerradura reflexiva y transitiva de $\vdash$.

Definición 2.6 Dado una MTD M el lenguaje que acepta es

$$L(M) = \{w \in \Sigma^* \mid q_0 w \vdash^* \alpha_1 p\, \alpha_2, \text{ con } p \in F \text{ y } \alpha_1, \alpha_2 \in \Gamma^*\}$$

Notemos que usamos el reconocimiento de un lenguaje como una notación conveniente de la habilidad de resolver problemas. Sin embargo; una MT puede fácilmente calcular funciones dejando en su cinta el resultado.

Definición 2.7 La clase de lenguajes definidos por las máquinas de Turing se conoce como **lenguajes recursivamente enumerables (r.e.)**, dicho nombre será explicado después. Al complemento de un lenguaje r.e. le llamamos **co-r.e.**

Definición 2.8 La clase de lenguajes definidos por las máquinas de Turing *totales* se conoce como **recursivos**. El término *recursivo* usualmente se refiere a un algoritmo que se llama a sí mismo. El nombre de lenguajes recursivos no tiene nada que ver con este uso. El sentido que aquí le damos es simplemente un nombre para el lenguaje aceptado por una máquina de Turing que siempre se detiene.

3. Trucos de programación

Presentaremos algunas modificaciones en la notación de MTs para facilitar su *programación*, es decir, la definición de la función de transición de las MTs se vuelve una tarea más sencilla con cambios en la notación. Esto también servirá para resaltar el poder y generalidad de las MTs. Básicamente, impondremos una estructura notacional sobre los estados y símbolos de la cinta.

3.1. Resigtros de CPU

Los registros de CPU son componentes adicionales dentro de los estados de una MT. De manera que permitiremos que los nombres de los estados sean de la forma $[q, \mathsf{x}_1, \mathsf{x}_2 \ldots, \mathsf{x}_n]$ donde los x_is actúan como símbolos almacenados.

Ejemplo 3.1 Diseñemos una MT para el lenguaje

$$L = \{ww^{\mathcal{R}} \mid w \in \{0,1\}^*\}.$$

Definimos los elementos de la MT M como

- $Q = \{[q, \_], [q, 0], [q, 1], [r, 0], [r, 1], s, t\}$,
- $\Sigma = \{0, 1\}$,
- $\Gamma = \{0, 1, \sqcup\}$,
- $q_0 = [q, \_]$, y
- $F = \{t\}$.

Dada la cadena de entrada $ww^{\mathcal{R}}$, la función de transición se asegurará de que el símbolo más a la izquierda es igual que el símbolo más a la derecha, y borrará ambos, este proceso se repetirá hasta que se consuman las dos cadenas. Emplearemos registros de CPU para almacenar el símbolo más a la izquierda, mientras recorremos las cadenas hacia la derecha. Las transiciones son las siguientes:

Paso 1. Se almacena el símbolo más a la izquierda en los registros de CPU y se reemplaza por $\sqcup$, o pasamos al estado final t cuando no hay más cadena que leer.

$$\begin{aligned} \delta([q, \_], 0) &= ([q, 0], \sqcup, R), \\ \delta([q, \_], 1) &= ([q, 1], \sqcup, R), \\ \delta([q, \_], \sqcup) &= (t, \sqcup, R). \end{aligned}$$

Paso 2. Se recorre la cadena hasta llegar al símbolo más a la derecha. Esto se logra moviendo la cabeza lectora a la derecha hasta encontrar $\sqcup$, enseguida se mueve una celda a la izquierda.

$$\begin{aligned} \delta([q, i], j) &= ([q, i], j, R), \ \ \forall i, j \in \{0, 1\}, \\ \delta([q, i], \sqcup) &= ([r, i], \sqcup, L). \end{aligned}$$

Paso 3. Nos aseguramos que el símbolo más a la derecha corresponde con el de la izquierda, i, que almacenamos en el registro de CPU.

$$\delta([r, i], i) = (s, \sqcup, L) \ \forall i \in \{0, 1\}.$$

Paso 4. Recorremos la cadena hasta llegar al símbolo más a la izquierda.

$$\begin{aligned} \delta(s, i) &= (s, i, L), \ \ \forall i \in \{0, 1\}, \\ \delta(s, \sqcup) &= ([q, \_], \sqcup, R). \end{aligned}$$

3.2. Múltiples pistas

Consideramos ahora la cinta como si tuviera múltiples pistas y Γ como si tuviera símbolos compuestos. Una representación para esta idea es:

$$\ldots \begin{array}{|c|c|c|c|c|c|c|} \hline & & 0 & \mathrm{Y} & 0 & & \\ \hline & & 1 & 1 & \mathrm{Z} & & \\ \hline & & \mathrm{X} & 1 & 0 & & \\ \hline \end{array} \ldots$$

Los elementos de Γ son de la forma $\begin{bmatrix} 0 \\ 1 \\ \mathrm{X} \end{bmatrix}$, $\begin{bmatrix} \mathrm{Y} \\ 1 \\ 1 \end{bmatrix}$, y $\begin{bmatrix} 0 \\ \mathrm{Z} \\ 0 \end{bmatrix}$.

Ejemplo 3.2 Diseñemos una MT para el lenguaje

$$L = \{ww \,|\, w \in \{0,1\}^+\}$$

Para reconocer estas cadenas necesitamos encontrar el punto medio, y entonces podemos usar un proceso similar al usado en el Ejemplo 3.1 para asegurarnos que la cadena a la izquierda del punto medio es igual a la que está a su derecha. Para encontrar el punto medio vemos a la cinta como si tuviera dos pistas:

$$\ldots \begin{array}{|c|c|c|c|c|c|c|c|} \hline & \sqcup & \sqcup & \sqcup & \star & \sqcup & \sqcup & \\ \hline & 0 & 1 & 1 & 0 & 1 & 1 & \\ \hline \end{array} \ldots$$

donde la pista de arriba sirve para poner marcas sobre los símbolos. Para encontrar el punto medio pondremos marcas sobre los símbolos en los extremos, y moveremos dichas marcar hasta que coincidan. El lugar donde coincidan será el inicio de la segunda cadena.

Los símbolos de la cinta son $\begin{bmatrix} \sqcup \\ \sqcup \end{bmatrix}$, $\begin{bmatrix} \sqcup \\ 0 \end{bmatrix}$, $\begin{bmatrix} \sqcup \\ 1 \end{bmatrix}$, $\begin{bmatrix} \star \\ 0 \end{bmatrix}$, y $\begin{bmatrix} \star \\ 1 \end{bmatrix}$.

Las acciones realizadas por cada estado para encontrar el punto medio se describen a continuación. El estado q_0 marca el símbolo en el extremo izquierdo, q_1 hace un recorrido a la derecha, q_2 marca el símbolo en el extremo derecho, y q_3 hace un recorrido a la izquierda. Las transiciones correspondientes, con $i \in \{0,1\}$, son

$$\begin{aligned}
\delta(q_0, \begin{bmatrix} \sqcup \\ i \end{bmatrix}) &= (q_1, \begin{bmatrix} \star \\ i \end{bmatrix}, R), & \qquad \delta(q_2, \begin{bmatrix} \sqcup \\ i \end{bmatrix}) &= (q_3, \begin{bmatrix} \star \\ i \end{bmatrix}, L), \\
\delta(q_1, \begin{bmatrix} \sqcup \\ i \end{bmatrix}) &= (q_1, \begin{bmatrix} \sqcup \\ i \end{bmatrix}, R), & \qquad \delta(q_3, \begin{bmatrix} \sqcup \\ i \end{bmatrix}) &= (q_3, \begin{bmatrix} \sqcup \\ i \end{bmatrix}, L), \\
\delta(q_1, \begin{bmatrix} \sqcup \\ \sqcup \end{bmatrix}) &= (q_2, \begin{bmatrix} \sqcup \\ \sqcup \end{bmatrix}, L), & \qquad \delta(q_3, \begin{bmatrix} \star \\ i \end{bmatrix}) &= (q_0, \begin{bmatrix} \sqcup \\ i \end{bmatrix}, R). \\
\delta(q_1, \begin{bmatrix} \star \\ i \end{bmatrix}) &= (q_2, \begin{bmatrix} \sqcup \\ i \end{bmatrix}, L), &&
\end{aligned}$$

Al término de las transiciones tendremos a la cabeza lectora apuntando al primer símbolo de la segunda w, que tiene marca $\star$. Además la MT estará en el estado inicial q_0, por lo

que el procesamiento debe continuar definiendo la transición $\delta(q_0, \begin{bmatrix} \star \\ i \end{bmatrix})$. A continuación se procede de manera similar a como se hizo en el Ejercicio 3.1.

Ejercicios 3.1 *¿Qué pasa cuando la MT inicia con la cadena* 1 *en la cinta?*

4. Mejorando MTs

Si la MT estudiada hasta ahora es capaz de capturar todo lo computable, entonces el añadirle características no debería incrementar su poder. Mostraremos que al añadir las siguientes características obtenemos una MT que puede ser "fácilmente" simulada por nuestra MT estándar:

- Múltiples cintas y cabezas lectoras, y
- No determinismo.

4.1. MT con múltiples cintas

Una MT de tres cintas es similar a una MT con una cinta excepto que tiene tres cintas y tres cabezas lectoras independientes. Inicialmente, la cadena de entrada se encuentra en la primer cinta y las otras dos están en blanco. A cada paso, la máquina lee los tres símbolos debajo de sus cabezas, con esta información y con el estado actual, imprime un símbolo en cada cinta, mueve sus cabezas (no necesariamente en la misma dirección), y entra a un estado nuevo. Su función de transición tiene como tipo:

$$\delta : Q \times \Gamma^3 \longrightarrow Q \times \Gamma^3 \times \{L, R\}^3.$$

Denotamos por M a la máquina con tres cintas y tres cabezas lectoras, la cual se ilustra en la Figura 2.

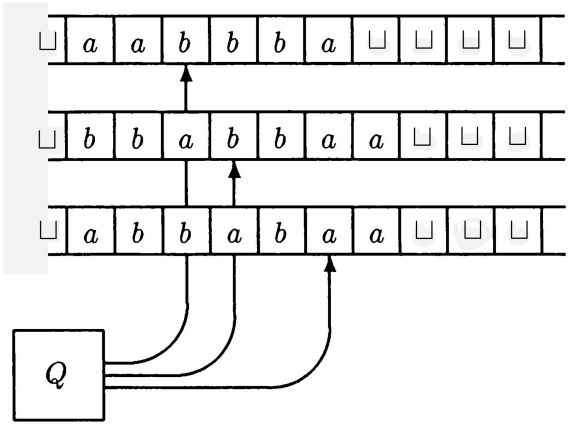


Figura 2: La MT M con tres cintas y tres cabezas lectoras.

Construiremos una máquina N con una sola cinta que simule a M como sigue. La MT N tendrá un alfabeto de la cinta extendido que nos permitirá considerar a su cinta como dividida en tres pistas. Cada pista tendrá el contenido de una de las cintas de M. También marcamos exactamente un símbolo en cada pista con $\widehat{\ }$ para indicar que este símbolo actualmente está siendo leído en la cinta correspondiente en M. La configuración de M en la Figura 2 podría ser simulada por la siguiente configuración de N.

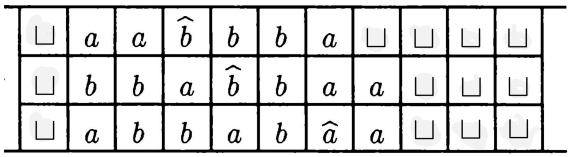
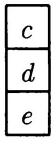


Figura 3: La MT N convencional que simula a M.

Un símbolo de la cinta de N es



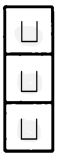
donde c, d, e son símbolos de las cintas de M, cada uno puede estar marcado o no por $\widehat{\ }$. Formalmente, consideramos al alfabeto de la cinta de N como $(\Gamma \cup \Gamma')^3$, donde $\Gamma' = \{\widehat{c} \,|\, c \in \Gamma\}$.

Los tres símbolos en $(\Gamma \cup \Gamma')^3$ representan los símbolos en las posiciones correspondientes de las tres cintas de M, marcados o sin marcar, y

es el símbolo en blanco de N.

Sea $w = a_1 a_2 \dots a_k$ la cadena de entrada. Cada paso en M es simulado por varios pasos en N.

Paso 1. Para simular un paso en M, la máquina N parte de la celda más a la izquierda que no es



.

Paso 2. N entonces hace un recorrido a la derecha hasta que ve las tres marcas en cada pista, recordando cada símbolo marcado, $\widehat{x}$, en su control de estados finitos, es decir, recuerda el símbolo marcado en registros de CPU.

Paso 3. Una vez vistas las tres marcas, determina lo que hay que hacer de acuerdo a la función de transición δ de M, la cual N tiene codificada en su control de estados finito. Con esta información N regresa a la izquierda para encontrar las tres marcas, reescribiendo los símbolos en cada pista y moviendo las marcas apropiadamente, según δ. Luego regresa al Paso 1 para simular el siguiente paso de M.

4.1.1. Tiempo de ejecución de una MT

Si bien el añadir características a la MT no incrementa su poder, sí impacta en su eficiencia.

Definición 4.1 Una MT M tiene **tiempo de ejecución** $T(n)$ si se detiene dentro de $T(n)$ pasos para toda entrada de tamaño n. Observe que $T(n)$ puede ser infinito.

Teorema 4.2 *Sea M la MT con tres cintas y tres cabezas lectoras, y N la MT convencional que simula a M. Si M tiene tiempo de ejecución $T(n)$, entonces N llevará a cabo la simulación de M en un tiempo de ejecución de $\mathcal{O}(T(n)^2)$.*

Demostración. Sea w la cadena de entrada, con $|w| = n$. Sabemos que M ocupa un tiempo de $T(n)$ para procesar w. Notemos que en cada paso, las cabezas de M pueden alejarse a lo más dos celdas. Luego, las cabezas de M no pueden estar más de $2T(n)$ celdas alejadas una de la otra. Por lo que el número de celdas que usa M en cada cinta es a lo más $2T(n)$.

Por cada transición de M sabemos que N realiza dos recorridos, uno a izquierda y otro a derecha. El tiempo que le toma hacer un recorrido es a lo más $2T(n)$, por los dos recorridos, el tiempo empleado es a lo más $4T(n)$. En M el máximo número de transiciones necesarias para procesar w es $T(n)$, por cada una de ellas N ocupa un tiempo de $4T(n)$. Por lo tanto, el tiempo de ejecución que N requiere para simular M es $\mathcal{O}(T(n)^2)$. ⊣

4.2. MTs no deterministas

En cualquier momento de la ejecución de una MT no determinista (MTN) se puede proceder de acuerdo a varias posibilidades. La función de transición para una MT no determinista tiene la forma:

$$\delta : Q \times \Gamma \longrightarrow \mathcal{P}(Q \times \Gamma \times \{L, R\})$$

El cómputo en una MT no determinista es un árbol cuyas ramas corresponden a posibilidades diferentes para la máquina. Si alguna de las ramas conduce a un estado de aceptación, la máquina acepta la cadena de entrada.

Teorema 4.3 *Si N es una MT no determinista, entonces existe MT determinista D equivalente a N.*

Demostración. Veremos el cómputo que realiza N sobre la entrada w como un árbol. Cada una de sus ramas representa una rama del no determinismo. Cada nodo del árbol representa una configuración de N. La raíz del árbol es la configuración inicial.

Considere el árbol de ejecución de N sobre w:

La MT D visitará sistemáticamente todas las configuraciones hasta encontrar una configuración de aceptación. Si no la encuentra, la simulación de D rechazará la cadena de entrada, o bien, no terminará. Debemos realizar esta búsqueda cuidadosamente para garantizar que D no falla en visitar todo el árbol. La manera en la que D recorrerá el árbol es por amplitud. Explorando todos los nodos de la misma profundidad antes de explorar los nodos en la profundidad siguiente. Así nos aseguramos que D visitará cada nodo en el árbol hasta que encuentre una configuración de aceptación.

La MT D tendrá dos cintas,

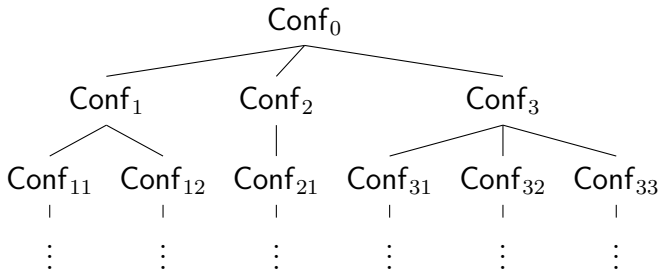


Figura 4: Un árbol de ejecución para una MT no determinista.

- en la primera se guardará una *cola* de las configuraciones de N en orden primero por amplitud, tales configuraciones serán separadas por $\star$, símbolo que no está presente en N. Para indicar la siguiente configuración a explorar usamos $\widehat{\star}$. Todas aquellas configuraciones a la izquierda de $\widehat{\star}$ ya han sido exploradas, y las que están a su derecha faltan por explorar;
- la segunda será una cinta auxiliar.

Al comenzar, la configuración inicial, Conf_0, estará en la cinta 1, la cual al comienzo se ve como

$$\widehat{\star}\ \mathsf{Conf}_0\ \star$$

El algoritmo para D consta de los siguiente pasos:

Paso 1. Examina la configuración actual Conf_A, la que está inmediatamente después de $\widehat{\star}$, y de ella obtiene el estado, q, y el símbolo que lee la cabeza lectora, a. Si $q \in F$, acepta y se detiene.

Paso 2. Si q no es final y hay k posibles transiciones para $\delta(q, a)$, entonces copiamos Conf_A a la cinta 2. Con dicha instancia hacemos k copias nuevas de Conf_A, separadas por $\star$, al final de la cinta 1.

Paso 3. Modificamos las k copias de Conf_A de acuerdo a las k posibles transiciones de N para $\delta(q, a)$.

Paso 4. Movemos $\widehat{\star}$ para que la marca apunte a la siguiente $\star$, tomando así en cuenta la siguiente configuración. Borramos el contenido de la cinta 2 para dejarla en blanco. Regresamos al Paso 1.

La manera en que la cinta 1 se ve al ejecutar el algoritmo anterior sobre el árbol de la Figura 4 es,

$$
\begin{aligned}
& \widehat{\star}\ \mathsf{Conf}_0\ \star \\
\rightsquigarrow\ & \widehat{\star}\ \mathsf{Conf}_0\ \star\ \mathsf{Conf}_0\ \star\ \mathsf{Conf}_0\ \star\ \mathsf{Conf}_0\ \star \\
\rightsquigarrow\ & \widehat{\star}\ \mathsf{Conf}_0\ \star\ \mathsf{Conf}_1\ \star\ \mathsf{Conf}_2\ \star\ \mathsf{Conf}_3\ \star \\
\rightsquigarrow\ & \star\ \mathsf{Conf}_0\ \widehat{\star}\ \mathsf{Conf}_1\ \star\ \mathsf{Conf}_2\ \star\ \mathsf{Conf}_3\ \star \\
\rightsquigarrow\ & \star\ \mathsf{Conf}_0\ \widehat{\star}\ \mathsf{Conf}_1\ \star\ \mathsf{Conf}_2\ \star\ \mathsf{Conf}_3\ \star\ \mathsf{Conf}_1\ \star\ \mathsf{Conf}_1\ \star \\
\rightsquigarrow\ & \star\ \mathsf{Conf}_0\ \widehat{\star}\ \mathsf{Conf}_1\ \star\ \mathsf{Conf}_2\ \star\ \mathsf{Conf}_3\ \star\ \mathsf{Conf}_{11}\ \star\ \mathsf{Conf}_{12}\ \star \\
\rightsquigarrow\ & \star\ \mathsf{Conf}_0\ \star\ \mathsf{Conf}_1\ \widehat{\star}\ \mathsf{Conf}_2\ \star\ \mathsf{Conf}_3\ \star\ \mathsf{Conf}_{11}\ \star\ \mathsf{Conf}_{12}\ \star \\
\rightsquigarrow\ & \star\ \mathsf{Conf}_0\ \star\ \mathsf{Conf}_1\ \widehat{\star}\ \mathsf{Conf}_2\ \star\ \mathsf{Conf}_3\ \star\ \mathsf{Conf}_{11}\ \star\ \mathsf{Conf}_{12}\ \star\ \mathsf{Conf}_2\ \star \\
\rightsquigarrow\ & \star\ \mathsf{Conf}_0\ \star\ \mathsf{Conf}_1\ \widehat{\star}\ \mathsf{Conf}_2\ \star\ \mathsf{Conf}_3\ \star\ \mathsf{Conf}_{11}\ \star\ \mathsf{Conf}_{12}\ \star\ \mathsf{Conf}_{21}\ \star \\
\rightsquigarrow\ & \star\ \mathsf{Conf}_0\ \star\ \mathsf{Conf}_1\ \star\ \mathsf{Conf}_2\ \widehat{\star}\ \mathsf{Conf}_3\ \star\ \mathsf{Conf}_{11}\ \star\ \mathsf{Conf}_{12}\ \star\ \mathsf{Conf}_{21}\ \star \\
& \vdots
\end{aligned}
$$

La simulación anterior es correcta ya que D acepta la cadena w si y sólo si hay una trayectoria de ejecución de N sobre w que la acepte. ⊣

4.2.1. Tiempo de ejecución de la simulación

Supongamos que la MTN N tiene tiempo de ejecución $T(n)$ para las cadenas de entrada w, con $|w| = n$. Encontremos ahora el tiempo de ejecución para la MTD D descrita arriba. Sea m el máximo número de elecciones que N tiene para cualquier $\delta(q, a)$, y sea $t = T(|w|)$. El árbol de ejecución de N sobre w tiene a lo más t niveles. En el comienzo de la ejecución sólo se tiene la configuración inicial de N, en un movimiento se puede llegar como máximo a m configuraciones de N, en dos movimientos se puede llegar como máximo a m^2 configuraciones de N, y así sucesivamente. Después de t movimientos N puede visitar a los más $1+m+m^2+\ldots+m^t = \dfrac{m^{t+1}-1}{m-1} = \mathcal{O}(m^t)$ configuraciones. Así D realiza a lo más $\mathcal{O}(m^t)$ pasos de ejecución sobre w.

Mientras que los DPDAs no pueden simular PDAs, una máquina de Turing determinista sí puede simular a una no determinista. Esto se debe al hecho de que las MTs son más poderosas, por lo que pueden simular el no determinismo, aunque a un costo muy caro pues el tiempo de ejecución es exponencialmente mayor que el de la MTN que simula. No se conoce si este aumento exponencial en el tiempo de ejecución es necesario. Si alguien descubre un mejor método de simular de manera determinista el trabajo de una MTN, las consecuencias impactarían drásticamente a las *clases de complejidad* $\mathcal{P}$ *y* $\mathcal{NP}$, y a la ciencia de la computación.

4.3. Clases de complejidad en el tiempo

La teoría de la complejidad clasifica a todos los problemas con algoritmos de tiempo polinomial como *tratables* o *manejables*. Tales algoritmos son llamados *eficientes*.

Definición 4.4

$$\mathcal{P} = \{L \mid L \text{ es aceptado por una MTD en tiempo polimonial}\}$$

$$\mathcal{NP} = \{L \mid L \text{ es aceptado por una MTN en tiempo polimonial}\}$$

Tanto la MTD como la MTN deben ser máquinas de Turing totales.

Por definición sabemos que

$$\mathcal{P} \subseteq \mathcal{NP}.$$

Estar en $\mathcal{P}$ corresponde a nuestra intuición de tener una solución eficiente, mientras que ser recursivo implica simplemente tener una solución. Informalmente, estar en $\mathcal{NP}$ significa que dada una solución podemos verificar eficientemente que es correcta.

5. Máquinas de enumeración

Definimos los lenguajes recursivamente enumerables (r.e.) como aquellos lenguajes aceptados por máquinas de Turing. El término recursivamente enumerable proviene de un formalismo diferente pero equivalente incorporando la idea de que los elementos de un lenguaje r.e. pueden ser enumerados uno a la vez de forma mecánica.

Definición 5.1 Definimos una *máquina de enumeración* como sigue. Tiene un control finito y dos cintas, una *cinta de trabajo* de escritura y lectura, y una *cinta de salida* de sólo escritura. La cabeza de la cinta de trabajo puede moverse en ambas direcciones, y puede leer y escribir cualquier elemento de Γ. La cabeza de la cinta de salida se mueve a la derecha una celda cuando escribe un símbolo, y sólo puede escribir símbolos de Σ. No hay cadena de entrada, ni estados finales. La máquina comienza en su estado inicial con ambas cintas en blanco. Se mueve de acuerdo a su función de transición como lo haría una MT, ocasionalmente escribe símbolos en la cinta de salida conforme a lo que estipule la función de transición. En un momento puede entrar a un estado especial de *enumeración*, el cual es simplemente un estado que se distingue. Cuando esto ocurre, la cadena actualmente escrita en la cinta de salida es *enumerada*. La cinta de salida se borra automáticamente y su cabeza lectora regresa al inicio de la cinta (la cinta de trabajo permanece intacta), y la máquina continúa su andar desde ese punto. La máquina mantiene su ejecución para siempre. El lenguaje $L(E)$ se define como el conjunto de todas las cadenas en Σ^* que son enumeradas por la máquina de enumeración E. La máquina puede nunca entrar en su estado de enumeración, en cuyo caso $L(E) = \varnothing$, o puede enumerar un número infinito de cadenas. La misma cadena puede ser enumerada más de una vez.

Las máquinas de enumeración y las máquinas de Turing son equivalentes en poder de cómputo.

Teorema 5.2 *La clase de lenguajes enumerados por una las máquinas de enumeración es exactamente la clase de lenguajes r.e. En otras palabras, un lenguaje es $L(E)$ para alguna máquina de enumeración E si y sólo si ese mismo lenguaje es $L(M)$ para alguna máquina de Turing M.*

Demostración. La demostración se encuentra en J. Martin. *Introduction to Languages and the Theory of Computation*, 3/e. McGrawHill 2004. ⊣

6. Lenguajes dependientes del contexto y la jerarquía de Chomsky

Para completar los lenguajes que forman la jerarquía de Chomsky definimos a continuación los lenguajes dependientes del contexto.

6.1. Lenguajes dependientes del contexto

Definición 6.1 Una **gramática dependiente del contexto (GDC)** es una gramática en la que no hay producción que decrezca el tamaño. En otras palabras, las producciones de una GDC son de la forma $\alpha \longrightarrow \beta$, donde $|\beta| \geq |\alpha|$.

Un **lenguaje dependiente del contexto (LDC)** es aquel generado por una gramática dependiente del contexto.

El término *dependiente del contexto* viene de una forma normal para estas gramáticas, en la cual cada producción es de la forma $\alpha_1 A \alpha_2 \longrightarrow \alpha_1 \beta \alpha_2$, con $\beta \neq \varepsilon$. Estas últimas producciones se asemejan a las de las gramáticas libres del contexto, pero permiten la sustitución de la variable A por la cadena β solamente en el "contexto" $\alpha_1 \_ \alpha_2$.

$$S \longrightarrow SABC \,|\, XBC \qquad BA \longrightarrow AB \qquad CA \longrightarrow AC \qquad CB \longrightarrow BC \qquad X \longrightarrow a$$
$$aA \longrightarrow aa \quad aB \longrightarrow ab \quad bB \longrightarrow bb \quad bC \longrightarrow bc \quad cC \longrightarrow cc$$

Figura 5: Una GDC para el lenguaje $\{a^n b^n c^n \,|\, n \geq 1\}$.

Debido a que los lenguajes dependientes del contexto no pueden tener producciones ε, ningún lenguaje conteniendo a ε puede ser un LDC; sin embargo, se puede demostrar que cada lenguaje libre del contexto es un LDC o la unión de $\{\varepsilon\}$ y un LDC. No obstante, hay muchos LDCs que no son libres del contexto, por ejemplo $\{a^n b^n c^n \,|\, n \geq 1\}$, que tiene como GDC la que aparece en la Figura 5. En efecto, casi cualquier lenguaje familiar es dependiente del contexto; en particular lo son los lenguajes de programación. De modo que si queremos encontrar un modelo de computación que corresponda exactamente a los LDCs, tendrá que ser más poderoso que un PDA.

Definición 6.2 Un **autómata linealmente acotado (LBA)** es una máquina de Turing no determinista que satisface las siguientes dos condiciones.

1. Su alfabeto de entrada incluye dos símbolos especiales $\triangleright$ y $\triangleleft$, los señaladores de extremo izquierdo y derecho, respectivamente.

2. El LBA no tiene movimientos a la izquierda de $\triangleright$ ni a la derecha de $\triangleleft$, ni puede imprimir otro símbolo sobre $\triangleright$ y $\triangleleft$.

El autómata linealmente acotado es simplemente una máquina de Turing que, en lugar de tener una cinta potencialemtne infinita sobre la cual hacer cálculos, está restringida a la porción de la cinta que contiene a la entrada w más las dos celdas de la cinta que contienen a los señaladores de los extremos. Restringir a la MT a una cantidad de cinta que, para cada entrada w, está acotada por alguna función lineal del tamaño de w resultaría equivalente a una MT restringida a la porción de la cinta que contiene a w únicamente; de aquí el nombre *autómata linealmente acotado*.

Un LBA será representado por $M = (Q, \Sigma, \Gamma, \delta, q_0, \triangleright, \triangleleft, F)$, donde Q, Σ, Γ, δ, q_0 y F se definen de la misma manera que para una MT no determinista; $\triangleright$ y $\triangleleft$ son símbolos de Σ. El lenguaje aceptado por M es

$$L(M) = \{w \in (\Sigma - \{\triangleright, \triangleleft\})^* \mid q_0 \triangleright w \triangleleft \vdash^* \alpha p \beta, \text{ con } p \in F \text{ y } \alpha, \beta \in \Gamma^*\}.$$

Nótese que los señaladores de los extremos forman parte de la cinta inicialmente pero no se consideran parte de la cadena de entrada que va a ser aceptada o rechazada. Puesto que un LBA no puede moverse fuera de $\triangleright w \triangleleft$, no hay necesidad de suponer que existe un símbolo en blanco en la cinta.

La demostración de los siguientes teoremas se puede consultar en J. E. Hopcroft, J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*, 1a edición.

Teorema 6.3 *Si L es un LDC, entonces L es aceptado por un LBA.*

Teorema 6.4 *Si $L = L(M)$ para el LBA $M = (Q, \Sigma, \Gamma, \delta, q_0, \triangleright, \triangleleft, F)$, entonces $L - \{\varepsilon\}$ es un LDC.*

6.2. Jerarquía de Chomsky

Las cuatro clases de lenguajes de los que trata este curso – regulares, libres de contexto, dependientes del contexto, y recursivamente enumerables – conforman lo que a menudo se conoce como *jerarquía de Chomsky*. Chomsky mismo diseño los cuatro tipos como tipo 3, tipo 2, tipo 1 y tipo 0, en orden del más restrictivo al más general. Cada nivel de la jerarquía, que se resume en la Tabla 1, puede ser caracterizado por una clase de gramática y por un modelo de cómputo específico.

La clase de lenguajes recursivos no figura en la Tabla 1, porque no hay una manera conocida de caracterizar estos lenguajes usando gramáticas. El siguiente teorema nos permite localizar esta clase con respecto a una de las gramáticas en la jerarquía de Chomsky.

Teorema 6.5 *Todo lenguaje dependiente del contexto es recursivo.*

Demostración. La demostración se encuentra en J. Martin. *Introduction to Languages and the Theory of Computation.*

Retomando la jerarquía de Chomsky, tenemos las siguientes contenciones:

$$\mathcal{S}_3 \subseteq \mathcal{S}_2 \subseteq \mathcal{S}_1 \subseteq \mathcal{R} \subseteq \mathcal{S}_0$$

donde $\mathcal{R}$ es la clase de lenguajes recursivos, $\mathcal{S}_i$ es la clase de lenguajes de tipo i para $i = 0$ e $i = 1$, y para $i = 2$ e $i = 3$ tenemos que $\mathcal{S}_i$ es la clase de lenguajes de tipo i que no contienen a ε. Sabemos que las dos primeras contenciones son propias: hay lenguajes libres del contexto que no son regulares, y hay lenguajes dependientes del contexto que no son libres del contexto. Las últimas dos contenciones son también propias, como se verá en la Nota 14.

Tipo	**Lenguajes (Gramáticas)**	**Forma de las producciones en la gramática**	**Modelo que lo acepta**
3	Regular	$A \to aB, A \to \varepsilon$ $(A, B \in V, a \in T)$	Autómata finito
2	Libre del contexto	$A \to \alpha$ $(A \in V, a \in (V \cup T)^*)$	Autómata de pila
1	Dependiente del contexto	$\alpha \to \beta$ $(\alpha, \beta \in (V \cup T)^*, \lvert\beta\rvert \geq \lvert\alpha\rvert,$ α contiene una variable)	Autómata linealmente acotado
0	Recursivamente enumerable (sin restricciones)	$\alpha \to \beta$ $(\alpha, \beta \in (V \cup T)^*,$ α contiene una variable)	Máquina de Turing

Tabla 1: La jerarquía de Chomsky.