8

Introducción a las máquinas de Turing

En este capítulo vamos a cambiar significativamente de dirección. Hasta ahora, hemos estado interesados fundamentalmente en clases simples de lenguajes y en las formas en que pueden utilizarse para problemas relativamente restringidos, tales como los protocolos de análisis, las búsquedas de texto o el análisis sintáctico de programas. Ahora, vamos a centrarnos en qué lenguajes pueden definirse mediante cualquier dispositivo computacional, sea cual sea. Este tema es equivalente a preguntarse qué pueden hacer las computadoras, ya que el reconocimiento de cadenas de un lenguaje es una manera formal de expresar cualquier problema y la resolución de un problema es un sustituto razonable de lo que hacen las computadoras.

Comenzaremos con una exposición informal, utilizando los conocimientos de la programación en C, para demostrar que existen problemas específicos que no se pueden resolver con una computadora. Estos problemas se conocen como "indecidibles". Después presentaremos un venerable formalismo para computadoras: la máquina de Turing. Aunque una máquina de Turing no se parece en nada a un PC, y sería enormemente ineficiente en el caso de que alguna empresa emprendedora las decidiera fabricar y vender, hace tiempo que la máquina de Turing se ha reconocido como un modelo preciso para representar lo que es capaz de hacer cualquier dispositivo físico de computación.

En el Capítulo 9, utilizamos la máquina de Turing para desarrollar una teoría acerca de los problemas "indecidibles", es decir, problemas que ninguna computadora puede resolver. Demostramos que una serie de problemas que son fáciles de expresar son de hecho problemas indecidibles. Un ejemplo de ello es el problema de determinar si una gramática es ambigua, y además veremos muchos otros.

8.1 Problemas que las computadoras no pueden resolver

El propósito de esta sección es proporcionar una introducción informal basada en la programación en C a la demostración de un problema específico que las computadoras no pueden resolver. El problema particular que vamos a abordar es si lo primero que imprime un programa C es `hola, mundo`. Aunque podríamos imaginar que la simulación del programa nos permitirá decir qué hace, en realidad tendremos que enfrentarnos a programas que tardan mucho tiempo en producir una respuesta. Este problema (no saber cuándo va a ocurrir algo, si es que alguna vez ocurre) es la causa última de nuestra incapacidad de decir lo que hace un programa. Sin embargo,

demostrar formalmente que no existe un programa que haga una tarea establecida es bastante complicado, por lo que necesitamos desarrollar algunos mecanismos formales. En esta sección, proporcionamos el razonamiento intuitivo que hay detrás de las demostraciones formales.

8.1.1 Programas que escriben "Hola, mundo"

En la Figura 8.1 se muestra el primer programa en C con el que se encuentran los estudiantes que leen el clásico texto de Kernighan y Ritchie.[1] Es fácil descubrir que este programa imprime `hola, mundo` y termina. Este programa es tan transparente que se ha convertido en una práctica habitual presentar los lenguajes mostrando cómo escribir un programa que imprima `hola, mundo` en dichos lenguajes.

```
main()
{
    printf("hola, mundo\n");
}
```

Figura 8.1. Programa hola-mundo de Kernighan y Ritchie.

Sin embargo, existen otros programas que también imprimen `hola, mundo`; a pesar de que lo que hacen no es ni mucho menos evidente. La Figura 8.2 muestra otro programa que imprime `hola, mundo`. Toma una entrada n, y busca las soluciones enteras positivas de la ecuación $x^n + y^n = z^n$. Si encuentra una, imprime `hola, mundo`. Si nunca encuentra los enteros x, y y z que satisfacen la ecuación, continúa buscando de forma indefinida y nunca imprime `hola, mundo`.

Para entender qué hace este programa, primero observe que `exp` es una función auxiliar para el cálculo de potencias. El programa principal necesita buscar en los tripletes (x, y, z) en un orden tal que estemos seguros de obtener todos los tripletes de enteros positivos. Para organizar la búsqueda adecuadamente, utilizamos una cuarta variable, `total`, que se inicia con el valor 3 y, en el bucle *while*, se incrementa en una unidad en cada pasada, tomando siempre valores enteros finitos. Dentro del bucle *while*, dividimos `total` en tres enteros positivos x, y y z, haciendo primero que x varíe entre 1 y `total-2`, y dentro del bucle for, hacemos que y varíe entre 1 y una unidad menos de lo que queda de `total` después de haberle restado el valor actual de x. A z se le asigna la cantidad restante, que tiene que estar comprendida entre 1 y `total-2`.

En el bucle más interno, se comprueba si (x, y, z) cumple la ecuación $x^n + y^n = z^n$. En caso afirmativo, el programa imprime `hola, mundo` y no imprime nada, en caso contrario.

Si el valor de n que lee el programa es 2, entonces encontrará combinaciones de enteros tales como `total` = 12, $x = 3$, $y = 4$ y $z = 5$, para las que $x^n + y^n = z^n$. Por tanto, para la entrada 2, el programa *imprime* `hola, mundo`.

Sin embargo, para cualquier entero $n > 2$, el prograna nunca encontrará tres enteros positivos que satisfagan la ecuación $x^n + y^n = z^n$, y por tanto no imprimirá `hola, mundo`. Lo interesante de esto es que hasta hace unos pocos años no se sabía si este programa imprimiría o no el texto `hola, mundo` para enteros n grandes. La afirmación de que no lo haría, es decir, que no existen soluciones enteras para la ecuación $x^n + y^n = z^n$ si $n > 2$, fue hecha por Fermat hace 300 años, pero hasta hace muy poco no se ha demostrado. Esta proposición normalmente se conoce como el "último teorema de Fermat".

Definamos el *problema de hola-mundo*: determinar si un programa C dado, con una entrada dada, imprime en primer lugar los primeros 11 caracteres de `hola, mundo`. En lo que sigue, a menudo diremos, para abreviar, que un programa imprime `hola, mundo` para indicar que lo primero que imprime son los 11 caracteres de `hola, mundo`.

[1] B. W. Kernighan y D. M. Ritchie, *The C Programming Language*, 1978, Prentice-Hall, Englewood Cliffs, NJ.

```
int exp(int i, n)
/* calcula i a la potencia n */
{
    int ans, j;
    ans = 1;
    for (j=1; j<=n; j++) ans *= i;
    return(ans);
}

main ()
{
    int n, total, x, y, z;
    scanf("%d", &n);
    total = 3;
    while (1) {
        for (x=1; x<=total-2; x++)
            for (y=1; y<=total-x-1; y++) {
                z = total - x - y;
                if (exp(x,n) + exp(y,n) == exp(z,n))
                    printf("hola, mundo\n");
            }
        total++;
    }
}
```

Figura 8.2. El último teorema de Fermat expresado como un programa hola-mundo.

Parece probable que, si los matemáticos tardaron 300 años en resolver una pregunta acerca de un único programa de 22 líneas, entonces el problema general de establecer si un determinado programa, para una entrada dada, imprime `hola, mundo` tiene que ser realmente complicado. De hecho, cualquiera de los problemas que los matemáticos todavía no han podido solucionar puede transformarse en una pregunta de la forma "¿imprime este programa, con esta entrada, el texto `hola, mundo`?". Por tanto, sería totalmente extraordinario que consiguiéramos escribir un programa que examinara cualquier programa P y la entrada I para P, y estableciera si P, ejecutado para la entrada I, imprime o no `hola, mundo`. Demostraremos que tal programa no existe.

8.1.2 Comprobador hipotético de "hola, mundo"

La imposibilidad de crear un comprobador de hola-mundo se demuestra por reducción al absurdo. Es decir, suponemos que existe un programa, llamado H, que toma como entrada un programa P y una entrada I, y establece si P para la entrada I imprime `hola, mundo`. La Figura 8.3 es una representación de lo que hace H. En concreto, la única salida que proporciona H es o imprimir los caracteres `sí` o imprimir los caracteres `no`. Siempre hace una cosa o la otra.

Si un problema tiene un algoritmo como H, que siempre establece correctamente si un caso del problema tiene como respuesta "sí" o "no", entonces se dice que el problema es "decidible". En caso contrario, el problema es "indecidible". Nuestro objetivo es demostrar que H no existe; es decir, que el problema de hola-mundo es indecidible.

Para demostrar dicha afirmación por reducción al absurdo, vamos a hacer algunos cambios en H, construyendo un problema relacionado denominado H_2 que demostraremos que no existe. Puesto que los cambios en H

¿Por qué tienen que existir problemas indecidibles?

Aunque es complicado demostrar que un problema específico, tal como el "problema de hola-mundo" que acabamos de ver, es indecidible, es bastante sencillo ver, mediante cualquier sistema que implique programación, por qué casi todos los problemas tienen que ser indecidibles. Recuerde que hemos establecido que un "problema" puede definirse como una cuestión acerca de si una cadena pertenece a un lenguaje. El número de lenguajes diferentes sobre cualquier alfabeto de más de un símbolo es no numerable. Es decir, no hay ninguna manera de asignar enteros a los lenguajes tal que todo lenguaje tenga asignado un entero y todo entero esté asignado a un lenguaje.

Por otro lado, los programas que tienen cadenas finitas construidas con un alfabeto finito (normalmente un subconjutno del alfabeto ASCII) *son* contables. Es decir, podemos ordenarlos de acuerdo con su longitud, y los programas con la misma longitud, podemos ordenarlos lexicográficamente. Así, podemos hablar del primer programa, el segundo programa y, en general, del programa i-ésimo para cualquier entero i.

Como resultado, sabemos que existen infinitos menos programas que problemas. Si elegimos un lenguaje al azar, casi seguro que será un problema indecidible. La única razón por la que la mayoría de los problemas *parecen* ser decidibles es porque rara vez estaremos interesados en problemas elegidos al azar. En lugar de ello, tendemos a buscar problemas sencillos y bien estructurados, que a menudo son decidibles. Sin embargo, incluso entre los problemas de nuestro interés que pueden definirse de manera clara y sucinta, nos encontraremos con muchos que son indecidibles; el problema de hola-mundo es uno de ellos.

son sencillas transformaciones que pueden realizarse en cualquier programa C, la única cuestión es la existencia de H, por lo que esta suposición es la que nos llevará a una contradicción.

Para simplificar la explicación, vamos a hacer varias suposiciones acerca de los programas en C. Estas suposiciones facilitan el trabajo de H en lugar de complicarlo, por lo que si podemos demostrar que no existe un "comprobador de hola-mundo" para estos programas restringidos, entonces podremos asegurar que no existe un comprobador que funcione para una clase más amplia de programas. Las suposiciones son las siguientes:

1. Toda salida se obtiene en modo carácter, es decir, no utilizamos un paquete gráfico o cualquier otra facilidad que proporcione una salida que no sean caracteres.
2. Toda salida en modo carácter se obtiene utilizando la instrucción `printf`, en lugar de `putchar()` o cualquier otra función de salida basada en caracteres.

Ahora suponemos que el programa H existe. La primera modificación que vamos a realizar consiste en cambiar la salida `no`, que es la respuesta que H proprociona cuando el programa de entrada P no imprime

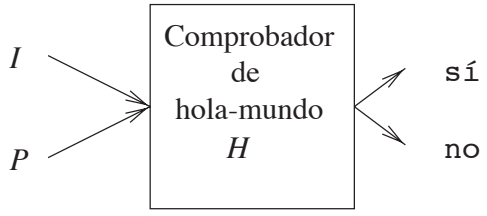

Figura 8.3. Hipotético programa H detector de hola-mundo.

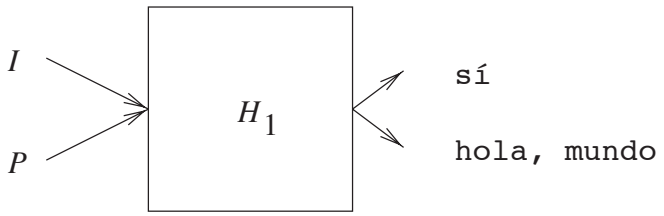

Figura 8.4. H_1 se comporta como H, pero dice `hola, mundo` en lugar de `no`.

`hola, mundo` como primera salida en respuesta a la entrada I. Tan pronto como H imprime "n", sabemos que seguirá con el carácter "o".[2] Por tanto, podemos modificar cualquier instrucción `printf` de H que imprima "n" para que en su lugar imprima `hola, mundo`. Cualquier otra instrucción `printf` que imprima una "o" pero no la "n" se omite. Como resultado, el nuevo programa, al que denominamos H_1, se comporta como H, excepto en que imprime `hola, mundo` exactamente cuando H imprimiría `no`. La Figura 8.4 muestra el comportamiento de H_1.

La siguiente transformación del programa es un poco más complicada; se trata básicamente de la misma acción que permitió a Alan Turing demostrar su resultado sobre la indecidibilidad acerca de las máquinas de Turing. Puesto que realmente estamos interesados en programas que tomen otros programas como entrada y nos informen de algo acerca de ellos, retringiremos H_1 de modo que:

a) Sólo tome la entrada P, no P e I.

b) Se le pregunte a P qué haría si su entrada fuera su propio código; es decir, ¿qué haría H_1 si tanto el programa como la entrada I fuesen la misma cadena P?

Las modificaciones que deben realizarse sobre H_1 para generar el programa H_2 mostrado en la Figura 8.5 son las siguientes:

1. Primero, H_2 lee la entrada completa P y la almacena en una matriz A, utilizando la función "malloc".[3]

2. H_2 entonces simula H_1, pero cuando H_1 lea la entrada P o I, H_2 leerá de la copia almacenada en A. Para hacer un seguimiento de cuánto H_1 ha leído de P y de I, H_2 puede mantener dos cursores que marquen las posiciones correspondientes en A.

Ahora ya estamos preparados para demostrar que H_2 no puede existir. Luego, H_1 tampoco puede existir, y de la misma manera, H tampoco. El núcleo del argumento está en prever lo que hace H_2 si se le proporciona como entrada su propio código. En la Figura 8.6 se refleja esta situación. Recuerde que H_2, dado cualquier programa P como entrada, proporciona como salida `sí` si P imprime `hola, mundo` cuando su entrada es él mismo. Además, H_2 imprime `hola, mundo` si la primera salida producida por P, con él mismo como entrada, no es `hola, mundo`.

Supongamos que el programa H_2 representado por la caja de la Figura 8.6 proporciona la salida `sí`. Entonces lo que el programa H_2 está diciendo acerca de su entrada H_2 es que H_2, cuando se proporciona su propio código como entrada, imprime `hola, mundo` como su primera salida. Pero hemos supuesto que la primera salida de H_2 en esta situación es `sí` en lugar de `hola, mundo`.

[2] Muy probablemente, el programa introducirá `no` en un comando `printf`, pero podría imprimir la "n" con un `printf` y la "o" con otro.

[3] La función UNIX de sistema `malloc` asigna un bloque de memoria de un tamaño especificado en la llamada a `malloc`. Esta función se utiliza cuando no puede determinarse el espacio de almacenamiento que se necesita hasta que se ejecuta el programa, como sería el caso en que se fuera a leer una entrada de longitud arbitraria. Normalmente, se invoca a `malloc` varias veces, a medida que se va leyendo la entrada y se necesita más espacio.

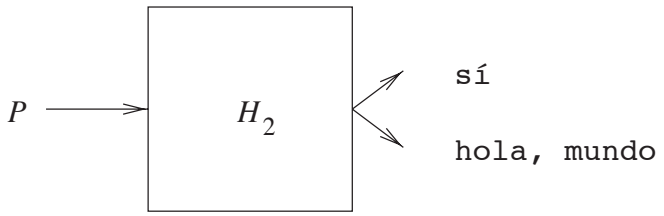

Figura 8.5. H_2 se comporta como H_1, pero utiliza su entrada P como P e I.

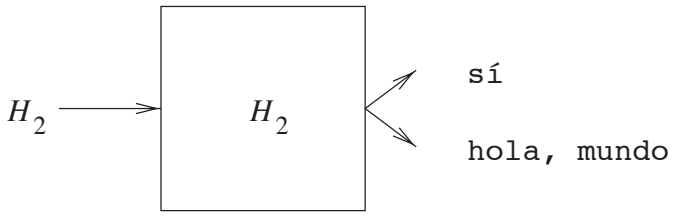

Figura 8.6. ¿Qué hace H_2 si se le proporciona su propio código como entrada?

Por tanto, parece que en la Figura 8.6 la salida de la caja es `hola, mundo`, ya que tiene que ser una u otra. Pero si H_2, teniendo como entrada su propio código, imprime en primer lugar `hola, mundo`, entonces la salida de la caja de la Figura 8.6 tiene que ser `sí`. Sea cual sea la salida que supongamos que proporciona H_2, podemos argumentar que proporciona la otra.

Esta situación es paradójica, por lo que concluimos que H_2 no puede existir. Como resultado, hemos llegado a una contradicción de la suposición de que H existe. Es decir, hemos demostrado que ningún programa H puede informar de si un determinado programa P con entrada I imprime o no como primera salida `hola, mundo`.

8.1.3 Reducción de un problema a otro

Ahora, tenemos un problema (¿lo primero que escribe un determinado programa para una entrada dada es `hola, mundo`?) que sabemos que ningún programa informático puede resolver. Un problema que no puede ser resuelto por una computadora se dice que es *indecidible*. En la Sección 9.3 proporcionamos la definición formal de problema "indecidible", pero por el momento, vamos a emplear el término de manera informal. Suponga que queremos determinar si algún otro problema puede ser solucionado o no por una computadora. Podemos probar a escribir un programa para resolverlo, pero si no se nos ocurre cómo hacerlo, entonces podemos intentar demostrar que no existe tal programa.

Podríamos demostrar que este nuevo problema es indecidible mediante una técnica similar a la que hemos empleado en el problema de hola-mundo: supongamos que existe un programa que resuelve el problema y desarrollamos un programa paradójico que tiene que hacer dos cosas contradictorias, al igual que el programa H_2. Sin embargo, una vez que tenemos un problema que sabemos que es indecidible, ya no tenemos que demostrar la existencia de una situación paradójica. Basta con demostrar que si pudiéramos resolver el nuevo problema, entonces utilizaríamos dicha solución para resolver un problema que ya sabemos que es indecidible. La estrategia se muestra en la Figura 8.7; esta técnica se conoce como *reducción* de P_1 a P_2.

Suponga que sabemos que el problema P_1 es indecidible y sea P_2 un nuevo problema que queremos demostrar que también es indecidible. Supongamos que existe el programa representado en la Figura 8.7 por el rombo etiquetado con "decide"; este programa imprime `sí` o `no`, dependiendo de si la entrada del problema P_2 pertenece o no al lenguaje de dicho problema.[4]

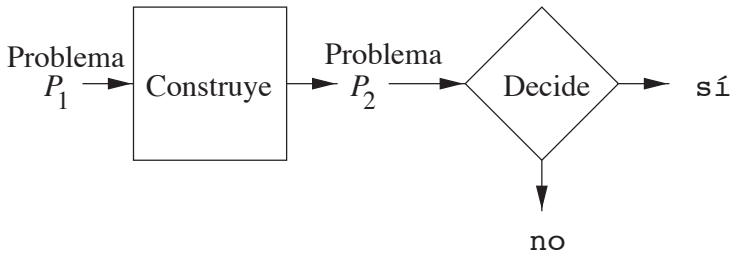

Figura 8.7. Si pudiéramos resolver el problema P_2, entonces prodríamos emplear su solución para resolver el problema P_1.

¿Puede realmente una computadora hacer todo?

Si examinamos un programa como el de la Figura 8.2, podríamos preguntarnos si realmente busca contraejemplos para el último teorema de Fermat. Después de todo, los enteros son sólo 32 bits en una computadora típica, y si el contraejemplo más pequeño implica un orden de magnitud de miles de millones, se producirá un error de desbordamiento antes de que fuera posible encontrar la solución. De hecho, se podría argumentar que una computadora con 128 megabytes de memoria principal y un disco de 30 gigabytes, tiene "sólo" $256^{30128000000}$ estados y es, por tanto, un autómata finito.

Sin embargo, no es productivo tratar las computadoras como autómatas finitos (o tratar el cerebro como un autómata, que es de donde procede la idea de autómata finito). El número de estados implicado es tan grande y los límites son tan poco claros, que no es posible extraer ninguna conclusión útil. De hecho, existen razones para creer que, si lo deseamos, podríamos expandir el conjunto de estados de una computadora de forma arbitraria.

Por ejemplo, podemos representar los números enteros como listas de dígitos enlazadas de longitud arbitraria. Si nos quedamos sin memoria, el programa puede imprimir una solicitud dirigida al usuario para que desmonte el disco, almacene su contenido y lo reemplace por un disco vacío. A medida que pase el tiempo, la computadora podría solicitar el intercambio entre varios discos de acuerdo con las necesidades de la computadora. Este programa sería mucho más complejo que el de la Figura 8.2, pero podríamos escribirlo. Otros trucos similares permitirían a cualquier otro programa evitar las limitaciones finitas relacionadas con el tamaño de memoria o con el tamaño de los enteros y otros elementos de datos.

Para demostrar que el problema P_2 es indecidible, tenemos que inventar una construcción, la cual se ha representado en la Figura 8.7 mediante un cuadrado, que convierta casos de P_1 en casos de P_2 que proporcionan la misma respuesta. Es decir, cualquier cadena del lenguaje P_1 se convierte en una cadena del lenguaje P_2, y cualquier cadena sobre el alfabeto de P_1 que *no* pertenezca al lenguaje P_1 se convierte en una cadena que no pertenece al lenguaje P_2. Una vez que tengamos esta construcción, podemos resolver P_1 como sigue:

1. Dado un programa P_1, es decir, dada una cadena w que puede o no pertenecer al lenguaje P_1, se aplica el algoritmo de construcción para generar una cadena x.
2. Se comprueba si x pertenece a P_2, y se aplica la misma respuesta de w y P_1.

[4]Recuerde que, en realidad, un problema es un lenguaje. Cuando hablamos del problema de decidir si un determinado programa con una entrada dada imprime como su primera salida `hola, mundo`, realmente estamos hablando de cadenas formadas por un programa fuente en C seguidas de cualquier archivo de entrada que el programa lee. Este conjunto de cadenas es un lenguaje sobre el alfabeto de los caracteres ASCII.

El sentido de una reducción es importante

Es un error común intentar demostrar que un problema P_2 es indecidible reduciendo P_2 a un problema indecidible conocido P_1; es decir, demostrando la proposición "si P_1 es decidible, entonces P_2 es decidible". Esta proposición, aunque seguramente es verdadera, no es útil, ya que la hipótesis "P_1 es decidible" es falsa.

La única forma de demostrar que un nuevo problema P_2 es indecidible es reduciéndolo a un problema indecidible conocido P_1. Dicho de otra forma, demostramos la proposición "si P_2 es decidible, entonces P_1 es decidible". La conversión contradictoria de esta proposición es "si P_1 es indecidible, entonces P_2 es indecidible". Dado que sabemos que P_1 es indecidible, podemos deducir que P_2 es indecidible.

Si w pertenece a P_1, entonces x pertenece a P_2, por lo que este algoritmo proporciona `sí` como respuesta. Si w no pertenece a P_1, entonces x no pertenece a P_2, y el algoritmo proporciona `no` como respuesta. De cualquier forma, dice la verdad acerca de w. Puesto que hemos supuesto que no existe ningún algoritmo que nos permita decidir si existe una cadena que pertenezca a P_1 , tenemos una demostración por reducción al absurdo de que el hipotético algoritmo de decisión para P_2 no existe; es decir, P_2 es un indecidible.

EJEMPLO 8.1

Utilicemos esta metodología para demostrar que la pregunta de si "un programa Q, dada una entrada y, hace una llamada a la función `foo`" es indecidible. Observe que Q puede no contener una función `foo`, en cuyo caso el problema es fácil, pero los casos complicados aparecen cuando Q contiene una función `foo` pero puede o no invocarla cuando la entrada es y. Dado que sólo conocemos un problema indecidible, el papel que desempeñará P_1 en la Figura 8.7 es el del problema hola-mundo. P_2 será el *problema llamar-a-foo* que acabamos de mencionar. Supongamos que existe un programa que resuelve el problema llamar-a-foo. Nuestro trabajo consiste en diseñar un algoritmo que convierta el problema hola-mundo en el problema llamar-a-foo.

Es decir, dado el programa Q y su entrada y, tenemos que construir un programa R y una entrada z tal que R, con la entrada z, llame a la función `foo` si y sólo si Q con la entrada y imprime `hola, mundo`. La construcción no es difícil:

1. Si Q tiene una función denominada `foo`, renombramos dicha función, así como todas las llamadas a la misma. Evidentemente, el nuevo programa Q_1 hace exactamente lo mismo que Q.
2. Añadimos a Q_1 una función `foo`. Esta función no hará nada y además no se la invoca. El programa resultante será Q_2.
3. Modificamos Q_2 para recordar los 11 primeros caracteres que imprime y los almacenamos en una matriz A. El programa resultante será Q_3.
4. Modificamos Q_3 de modo que siempre que ejecute una instrucción de salida, compruebe el contenido de la matriz A para ver si contiene los 11 caracteres o más y, en caso afirmativo, si `hola-mundo` son los 11 primeros caracteres escritos. En dicho caso, se llama a la nueva función `foo` que se ha añadido en el punto (2). El programa resultante será R, y la entrada z es igual que y.

Supongamos que Q con la entrada y imprime `hola, mundo` como primera salida. Entonces, por construcción, R llamará a `foo`. Sin embargo, si Q con la entrada y no imprime `hola, mundo` como primera salida, entonces R nunca llamará a `foo`. Si podemos decidir si R con la entrada z llama a `foo`, entonces también sabemos si Q con la entrada y (recuerde que $y = z$) imprime `hola, mundo`. Dado que sabemos que no existe

ningún algoritmo que permita decidir el problema hola-mundo y que los cuatro pasos de la construcción de R a partir de Q podrían realizarse mediante un programa que editara el código de los programas, nuestra hipótesis de que existe un comprobador para llamar-a-foo es errónea. Por tanto, como no existe tal programa, el problema llamar-a-foo es indecidible. □

8.1.4 Ejercicios de la Sección 8.1

Ejercicio 8.1.1. Reduzca el problema hola-mundo a cada uno de los siguientes problemas. Utilice el estilo informal de esta sección para describir las posibles transformaciones del programa y no se preocupe por los límites prácticos, como el tamaño máximo de archivo o el tamaño de memoria que imponen las computadoras reales.

***!** a) Dados un programa y una determinada entrada, ¿el programa termina deteniéndose; es decir, el programa no entra en un bucle infinito al recibir dicha entrada?

b) Dados un programa y una determinada entrada, ¿llega el programa a generar *alguna* salida?

! c) Dados un programa y una determinada entrada, ¿producen ambos programas la misma salida para la entrada dada?

8.2 La máquina de Turing

El propósito de la teoría de los problemas indecidibles no es sólo establecer la existencia de tales problemas (una idea excitante por sí misma desde el punto de vista intelectual) sino proporcionar también una guía a los programadores sobre lo que se puede o no conseguir a través de la programación. La teoría también tiene un gran impacto práctico, como veremos en el Capítulo 10, al tratar problemas que aunque sean decidibles, requieren mucho tiempo para ser resueltos. Estos problemas, conocidos como "problemas intratables", suelen plantear una mayor dificultad al programador y al diseñador de sistemas que los problemas indecidibles. La razón de ello es que mientras que los problemas indecidibles normalmente suelen resultar obvios y habitualmente no se intentan resolver, los problemas intratables se presentan continuamente. Además, a menudo dan lugar a pequeñas modificaciones de los requisitos o a soluciones heurísticas. Por tanto, el diseñador se enfrenta con frecuencia a tener que decidir si un problema es o no intratable, y qué hacer si lo es.

Necesitamos herramientas que nos permitan determinar cuestiones acerca de la indecidibilidad o intratabilidad todos los días. La tecnología presentada en la Sección 8.1 resulta útil para cuestiones que tratan con programas, pero no se puede trasladar fácilmente a problemas en otros dominios no relacionados. Por ejemplo, tendríamos grandes dificultades para reducir el problema de hola-mundo a la cuestión de si una gramática es ambigua.

Como resultado, necesitamos reconstruir nuestra teoría sobre la indecidibidad, no basándonos en programas en C o en otro lenguaje, sino en un modelo de computadora muy simple: la máquina de Turing. Básicamente, este dispositivo es un autómata finito que dispone de una única cinta de longitud infinita en la que se pueden leer y escribir datos. Una ventaja de la máquina de Turing sobre los programas como representación de lo que se puede calcular es que la máquina de Turing es lo suficientemente simple como para que podamos representar su configuración de manera precisa, utilizando una notación sencilla muy similar a las descripciones instantáneas de un autómata a pila. En cambio, aunque los programas en C tienen un estado, que implica a todas las variables en cualquier secuencia de llamadas a función que se realice, la notación para describir estos estados es demasiado compleja como para poder realizar demostraciones formales comprensibles.

Con la notación de la máquina de Turing, demostraremos que ciertos problemas, que aparantemente no están relacionados con la programación, son indecidibles. Por ejemplo, demostraremos en la Sección 9.4 que el "problema de la correspondencia de Post", una cuestión simple que implica a dos listas de cadenas, es

indecidible, y que este problema facilita la demostración de que algunas cuestiones acerca de las gramáticas, como por ejemplo la ambiguedad, sean indecidibles. Del mismo modo, al presentar los problemas intratables comprobaremos que ciertas cuestiones, que parecen tener poco que ver con la computación (como por ejemplo, si se satisfacen las formulas booleanas), son intratables.

8.2.1 El intento de decidir todas las cuestiones matemáticas

A finales del siglo XX, el matemático D. Hilbert se preguntó si era posible encontrar un algoritmo para determinar la verdad o falsedad de cualquier proposición matemática. En particular, se preguntó si existía una forma de determinar si cualquier fórmula del cálculo de predicados de primer orden, aplicada a enteros, era verdadera. Dado que el cálculo de predicados de primer orden sobre los enteros es suficientemente potente para expresar proposicones como "esta gramática es ambigua" o "este programa imprime `hola, mundo`", si Hilbert hubiera tenido éxito, existirían algoritmos para estos problemas que ahora sabemos que no existen.

Sin embargo, en 1931, K. Gödel publicó su famoso teorema de la incompletitud. Construyó una fórmula para el cálculo de predicados aplicada a los enteros, que afirmaba que la propia fórmula no podía ser ni demostrada ni refutada dentro del cálculo de predicados. La técnica de Gödel es similar a la construcción del programa auto-contradictorio H_2 de la Sección 8.1.2, pero trabaja con funciones sobre los enteros, en lugar de con programas en C.

El cálculo de predicados no era la única idea que los matemáticos tenían para "cualquier computación posible". De hecho, el cálculo de predicados, al ser declarativo más que computacional, entraba en competencia con una variedad de notaciones, incluyendo las "funciones recursivas parciales", una notación similar a un lenguaje de programación, y otras notaciones similares. En 1936, A. M. Turing propuso la máquina de Turing como modelo de "cualquier posible computacion". Este modelo es como una computadora, en lugar de como un programa, incluso aunque las verdaderas computadoras electrónicas o incluso electromecánicas aparecieron varios años después (y el propio Turing participó en la construcción de estas máquinas durante la Segunda Guerra Mundial).

Lo interesante es que todas las propuestas serias de modelos de computación tienen el mismo potencial; es decir, calculan las mismas funciones o reconocen los mismos lenguajes. La suposición no demostrada de que cualquier forma general de computación no permite calcular sólo las funciones recursivas parciales (o, lo que es lo mismo, que las máquinas de Turing o las computadoras actuales pueden calcular) se conoce como *hipótesis de Church* (por el experto en lógica A. Church) o *tésis de Church*.

8.2.2 Notación para la máquina de Turing

Podemos visualizar una máquina de Turing como se muestra en la Figura 8.8. La máquina consta de una *unidad de control*, que puede encontrarse en cualquiera de un conjunto finito de estados. Hay una *cinta* dividida en cuadrados o *casillas* y cada casilla puede contener un símbolo de entre un número finito de símbolos.

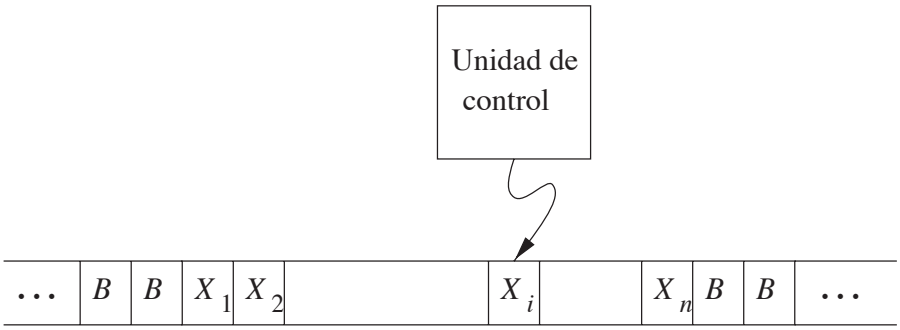

Figura 8.8. Una máquina de Turing.

Inicialmente, la *entrada*, que es una cadena de símbolos de longitud finita elegidos del *alfabeto de entrada*, se coloca en la cinta. Las restantes casillas de la cinta, que se extiende infinitamente hacia la izquierda y la derecha, inicialmente almacenan un símbolo especial denominado *espacio en blanco*. El espacio en blanco es un *símbolo de cinta*, pero no un símbolo de entrada, y pueden existir también otros símbolos de cinta además de los símbolos de entrada y del espacio en blanco.

Existe una *cabeza de la cinta* que siempre está situada en una de las casillas de la cinta. Se dice que la máquina de Turing *señala* dicha casilla. Inicialmente, la cabeza de la cinta está en la casilla más a la izquierda que contiene la entrada.

Un *movimiento* de la máquina de Turing es una función del estado de la unidad de control y el símbolo de cinta al que señala la cabeza. En un movimiento, la máquina de Turing:

1. Cambiará de estado. El siguiente estado puede ser opcionalmente el mismo que el estado actual.
2. Escribirá un símbolo de cinta en la casilla que señala la cabeza. Este símbolo de cinta reemplaza a cualquier símbolo que estuviera anteriormente en dicha casilla. Opcionalmente, el símbolo escrito puede ser el mismo que el que ya se encontraba allí.
3. Moverá la cabeza de la cinta hacia la izquierda o hacia la derecha. En nuestro formalismo, exigiremos que haya un movimiento y no permitiremos que la cabeza quede estacionaria. Esta restricción no limita lo que una máquina de Turing puede calcular, ya que cualquier secuencia de movimientos con una cabeza estacionaria podría condensarse, junto con el siguiente movimiento de la cabeza de la cinta, en un único cambio de estado, un nuevo símbolo de cinta y un movimiento hacia la izquierda o hacia la derecha.

La notación formal que vamos a emplear para una *máquina de Turing* (MT) es similar a la que hemos empleado para los autómatas finitos o los autómatas a pila. Describimos un MT mediante la siguiente séptupla:

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$$

cuyos componentes tienen el siguiente significado:

Q El conjunto finito de *estados* de la unidad de control.

Σ El conjunto finito de *símbolos de entrada*.

Γ El conjunto completo de *símbolos de cinta*; Σ siempre es un subconjunto de Γ.

δ La *función de transición*. Los argumentos de $\delta(q, X)$ son un estado q y un símbolo de cinta X. El valor de $\delta(q, X)$, si está definido, es (p, Y, D), donde:

1. p es el siguiente estado de Q.
2. Y es el símbolo de Γ, que se escribe en la casilla que señala la cabeza y que sustituye a cualquier símbolo que se encontrara en ella.
3. D es una *dirección* y puede ser L o R, lo que que nos indica la dirección en que la cabeza se mueve, "izquierda" (L) o "derecha" (R), respectivamente.

q_0 El *estado inicial*, un elemento de Q, en el que inicialmente se encuentra la unidad de control.

B El símbolo *espacio en blanco*. Este símbolo pertenece a Γ pero no a Σ; es decir, no es un símbolo de entrada. El espacio en blanco aparece inicialmente en todas las casillas excepto en aquéllas que se almacenan los símbolos de la entrada.

F El conjunto de los estados *finales* o *de acepatción*, un subconjunto de Q.

8.2.3 Descripciones instantáneas de las máquinas de Turing

Para describir formalmente lo que hace una máquina de Turing, necesitamos desarrollar una notación para las configuraciones o *descripciones instantáneas*, al igual que la notación que desarrollamos para los autómatas a pila. Dada una MT que, en principio, tiene una cinta de longitud infinita, podemos pensar que es imposible de forma sucinta describir las configuraciones de dicha MT. Sin embargo, después de cualquier número finito de movimientos, la MT puede haber visitado únicamente un número finito de casillas, incluso aunque el número de casillas visitadas puede crecer finalmente por encima de cualquier límite finito. Por tanto, en cada configuración, existe un prefijo y un sufijo infinitos de casillas que nunca han sido visitadas. Todas estas casillas tienen que contener espacios en blanco o uno de los símbolos de entrada. En consecuencia, en una configuración, sólo mostramos las casillas comprendidas entre el símbolo más a la izquierda y el símbolo más a la derecha que no seas espacios en blanco. Cuando se dé la condición especial de que la cabeza está señalando a uno de los espacios en blanco que hay antes o después de la cadena de entrada, también tendremos que incluir en la configuración un número finito de espacios en blanco.

Además de representar la cinta, tenemos que representar la unidad de control y la posición de la cabeza de la cinta. Para ello, incluimos el estado en la cinta y lo situamos inmediatamente a la izquierda de la casilla señalada. Para que la cadena que representa el estado de la cinta no sea ambigua, tenemos que asegurarnos de que no utilizamos como estado cualquier símbolo que sea también un símbolo de cinta. Sin embargo, es fácil cambiar los nombres de los estados, de modo que no tengan nada en común con los símbolos de cinta, ya que el funcionamiento de la MT no depende de cómo se llamen los estados. Por tanto, utilizaremos la cadena $X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n$ para representar una configuración en la que:

1. q sea el estado de la máquina de Turing.
2. La cabeza de la cinta esté señalando al símbolo i-ésimo empezando por la izquierda.
3. $X_1X_2\cdots X_n$ sea la parte de la cinta comprendida entre los símbolos distintos del espacio en blanco más a la izquierda y más a la derecha. Como excepción, si la cabeza está a la izquierda del símbolo más a la izquierda que no es un espacio en blanco o a la derecha del símbolo más a la derecha que no es un espacio en blanco, entonces un prefijo o un sufijo de $X_1X_2\cdots X_n$ serán espacios en blanco e i será igual a 1 o a n, respectivamente.

Describimos los movimientos de un máquina de Turing $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ utilizando la notación $\underset{M}{\vdash}$ que hemos empleado para los autómatas a pila. Cuando se sobreentienda que hacemos referencia a la MT, sólo usaremos $\vdash$ para indicar los movimientos. Como es habitual, utilizaremos $\overset{*}{\underset{M}{\vdash}}$, o sólo $\overset{*}{\vdash}$, para indicar cero, uno o más movimientos de la MT M.

Supongamos que $\delta(q, X_i) = (p, Y, L)$; es decir, el siguiente movimiento se realiza hacia la izquierda. Entonces:

$$X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n \underset{M}{\vdash} X_1X_2\cdots X_{i-2}pX_{i-1}YX_{i+1}\cdots X_n$$

Observe cómo este movimiento refleja el cambio al estado p y el hecho de que la cabeza de la cinta ahora señala a la casilla $i-1$. Existen dos excepciones importantes:

1. Si $i = 1$, entonces M se mueve al espacio en blanco que se encuentra a la izquierda de X_1. En dicho caso,
$$qX_1X_2\cdots X_n \underset{M}{\vdash} pBYX_2\cdots X_n$$
2. Si $i = n$ e $Y = B$, entonces el símbolo B escrito sobre X_n se añade a la secuencia infinita de los espacios en blanco que hay después de la cadena de entrada y no aparecerá en la siguiente configuración. Por tanto,
$$X_1X_2\cdots X_{n-1}qX_n \underset{M}{\vdash} X_1X_2\cdots X_{n-2}pX_n$$

Supongamos ahora que $\delta(q,X_i) = (p,Y,R)$; es decir, el siguiente movimiento es hacia la derecha. Entonces,

$$X_1X_2\cdots X_{i-1}qX_iX_{i+1}\cdots X_n \underset{M}{\vdash} X_1X_2\cdots X_{i-1}YpX_{i+1}\cdots X_n$$

En este caso, el movimiento refleja el hecho de que la cabeza se ha movido a la casilla $i+1$. De nuevo, tenemos dos excepciones importantes:

1. Si $i = n$, entonces la casilla $i+1$ almacena un espacio en blanco, por lo que dicha casilla no formaba parte de la configuración anterior. Por tanto, tenemos que:

$$X_1X_2\cdots X_{n-1}qX_n \underset{M}{\vdash} X_1X_2\cdots X_{n-1}YpB$$

2. Si $i = 1$ e $Y = B$, entonces el símbolo B escrito sobre X_1 se añade a la secuencia infinita de los espacios en blanco anteriores a la cadena de entrada y no aparecerá en la siguiente configuración. Por tanto,

$$qX_1X_2\cdots X_n \underset{M}{\vdash} pX_2\cdots X_{n-1}$$

EJEMPLO 8.2

Vamos a diseñar una máquina de Turing y a ver cómo se comporta con una entrada típica. La MT que vamos a construir aceptará el lenguaje $\{0^n1^n \mid n \geq 1\}$. Inicialmente, se proporciona a la cinta una secuencia finita de ceros y unos, precedida y seguida por secuencias infinitas de espacios en blanco. Alternativamente, la MT cambiará primero un 0 por X y luego un 1 por una Y, hasta que se hayan cambiado todos los ceros y los unos.

Más detalladamente, comenzando por el extremo izquierdo de la entrada, se cambia sucesivamente un 0 por una X y se mueve hacia la derecha pasando por encima de todos los ceros y letras Y que ve, hasta encontrar un 1. Cambia el 1 por una Y y se mueve hacia la izquierda pasando sobre todas las letras Y y ceros hasta encontrar una X. En esta situación, busca un 0 colocado inmediatamente a la derecha y, si lo encuentra, lo cambia por una X y repite el proceso, cambiando el 1 correspodiente por una Y.

Si la entrada no es de la forma $\mathbf{0}^*\mathbf{1}^*$, entonces la MT terminará no haciendo el siguiente movimiento y se detendrá sin aceptar. Sin embargo, si termina cambiando todos los ceros por X en la misma iteración en la que cambia el último 1 por una Y, entonces determina que la entrada era de la forma 0^n1^n y acepta. La especificación formal de la máquina de Turing M es:

$$M = (\{q_0,q_1,q_2,q_3,q_4\},\{0,1\},\{0,1,X,Y,B\},\delta,q_0,B,\{q_4\})$$

donde δ se especifica en la tabla de la Figura 8.9.

	Símbolo				
Estado	0	1	X	Y	B
q_0	(q_1,X,R)	–	–	(q_3,Y,R)	–
q_1	$(q_1,0,R)$	(q_2,Y,L)	–	(q_1,Y,R)	–
q_2	$(q_2,0,L)$	–	(q_0,X,R)	(q_2,Y,L)	–
q_3	–	–	–	(q_3,Y,R)	(q_4,B,R)
q_4	–	–	–	–	–

Figura 8.9. Una máquina de Turing que acepta $\{0^n1^n \mid n \geq 1\}$.

Mientras M realiza las operaciones anteriores, la parte de la cinta que ya ha sido recorrida por la cabeza de la misma corresponderá siempre a una secuencia de símbolos descrita por la expresión regular $\mathbf{X}^*\mathbf{0}^*\mathbf{Y}^*\mathbf{1}^*$. Es decir, habrá ceros que han sido sustituidos por X, seguidos de ceros que todavía no han sido sustituidos por X. Luego se encontrarán algunos unos que han sido sustituidos por Y, y unos que todavía no lo han sido. A continuación, puede que haya o no algunos ceros y unos.

El estado q_0 es el estado inicial y M entra en el estado q_0 cada vez que vuelve al cero más a la izquierda que queda. Si M está en el estado q_0 y se señala un 0, la regla de la esquina superior izquierda de la Figura 8.9 indica que M tiene que pasar al estado q_1, cambiar el 0 por una X y moverse hacia la derecha. Una vez que está en el estado q_1, M se mueve hacia la derecha saltándose todos los ceros y las Y que encuentra en la cinta, permaneciendo en el estado q_1. Si M ve una X o una B, se detiene. Sin embargo, si M ve un 1 estando en el estado q_1, cambia dicho 1 por una Y, pasa al estado q_2 y comienza a moverse hacia la izquierda.

En el estado q_2, M se mueve hacia la izquierda pasando por encima de los ceros y las Y, permaneciendo en el estado q_2. Cuando M alcanza la X que está más a la derecha, la cual marca el extremo derecho del bloque de ceros que ya han sido cambiados por X, M vuelve al estado q_0 y se mueve hacia la derecha. Hay dos casos:

1. Si ahora M ve un 0, entonces repite el ciclo de sustituciones que acabamos de describir.

2. Si M ve una Y, entonces es que ha cambiado todos los ceros por X. Si todos los unos han sido cambiados por Y, entonces la entrada era de la forma 0^n1^n, y M acepta. Por tanto, M pasa al estado q_3, y comienza a moverse hacia la derecha, pasando por encima de las Y. Si el primer símbolo distinto de una Y que M encuentra es un espacio en blanco, entonces existirá el mismo número de ceros que de unos, por lo que M entra en el estado q_4 y acepta. Por otro lado, si M encuentra otro 1, entonces es que hay demasiados unos, por lo que M deja de operar sin aceptar. Si encuentra un 0, entonces la entrada era de la forma errónea y M también se detiene.

He aquí un ejemplo de un cálculo de aceptación de M. Su entrada es 0011. Inicialmente, M se encuentra en el estado q_0, señalando al primer 0, es decir, la configuración inicial de M es q_00011. La secuencia completa de movimientos de M es:

$$q_00011 \vdash Xq_1011 \vdash X0q_111 \vdash Xq_20Y1 \vdash q_2X0Y1 \vdash$$
$$Xq_00Y1 \vdash XXq_1Y1 \vdash XXYq_11 \vdash XXq_2YY \vdash Xq_2XYY \vdash$$
$$XXq_0YY \vdash XXYq_3Y \vdash XXYYq_3B \vdash XXYYBq_4B$$

Veamos otro ejemplo. Consideremos lo que hace M para la entrada 0010, que no pertenece al lenguaje aceptado.

$$q_00010 \vdash Xq_1010 \vdash X0q_110 \vdash Xq_20Y0 \vdash q_2X0Y0 \vdash$$
$$Xq_00Y0 \vdash XXq_1Y0 \vdash XXYq_10 \vdash XXY0q_1B$$

El comportamiento de M para 0010 se parece al comportamiento para 0011, hasta que llega a la configuración $XXYq_10$ M y señala al 0 final por primera vez. M tiene que moverse hacia la derecha permaneciendo en el estado q_1, lo que corresponde a la configuración $XXY0q_1B$. Sin embargo, en el estado q_1, M no realiza ningún movimiento si el símbolo de la entrada al que señala es B; por tanto, M deja de funcionar y no acepta su entrada. □

8.2.4 Diagramas de transición para las máquinas de Turing

Podemos representar gráficamente las transiciones de una máquina de Turing, al igual que hicimos con los autómatas a pila. Un *diagrama de transiciones* consta de un conjunto de nodos que se corresponden con los estados de la MT. Un arco que va desde un estado q al estado p se etiqueta con uno o más elementos de la forma

X/YD, donde X e Y son símbolos de cinta y D especifica una dirección, bien L (izquierda) o R (derecha). Es decir, siempre que $\delta(q,X) = (p,Y,D)$, nos encontraremos con la etiqueta X/YD sobre el arco que va desde q hasta p. Sin embargo, en nuestros diagramas, la dirección D está representada gráficamente por $\leftarrow$ para indicar hacia la "izquierda" y por $\rightarrow$ para la "derecha".

Al igual que en los otros diagramas de transiciones, representamos el estado inicial mediante la palabra "Inicio" y una flecha que entra en el estado. Los estados de aceptación se indican mediante círculos dobles. Por tanto, la única información acerca de la máquina de Turing que no podemos leer directamente del diagrama es el símbolo utilizado para el espacio en blanco. Supondremos que dicho símbolo es B a menos que se indique otra cosa.

EJEMPLO 8.3

La Figura 8.10 muestra el diagrama de transiciones para la máquina de Turing del Ejemplo 8.2, cuya función de transición se ha proporcionado en la Figura 8.9. □

EJEMPLO 8.4

Aunque actualmente consideramos más adecuado pensar en las máquinas de Turing como en reconocedores de lenguajes, o lo que es lo mismo, solucionadores de problemas, el punto de vista original de Turing sobre esta máquina era el de una computadora capaz de procesar funciones de enteros con valor. En su esquema, los enteros se representaban en código unario, como bloques de un sólo carácter, y la máquina calculaba cambiando las longitudes de los bloques o construyendo nuevos bloques en cualquier lugar de la cinta. En este sencillo ejemplo, vamos a ver cómo una máquina de Turing puede calcular la función $\dot{-}$, que recibe el nombre de *sustracción propia* y se define mediante $m \dot{-} n = \text{máx}(m-n, 0)$. Es decir, $m \dot{-} n$ es $m-n$ si $m \geq n$ y 0 si $m < n$.

La especificación de una MT que realiza esta operación es como sigue:

$$M = (\{q_0, q_1, \ldots, q_6\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B)$$

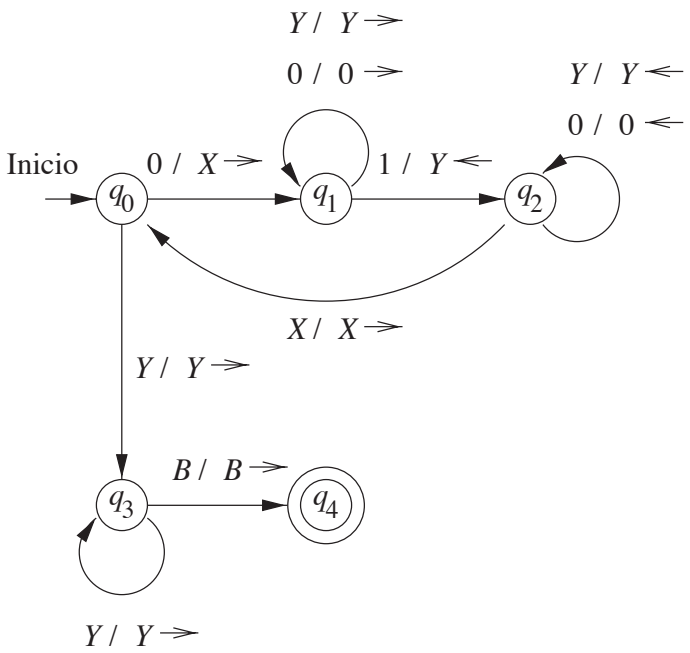

Figura 8.10. Diagrama de transiciones de una MT que acepta cadenas de la forma 0^n1^n.

Observe que, dado que esta MT no se emplea para aceptar entradas, hemos omitido la séptima componente, que es el conjunto de los estados de aceptación. M empezará a trabajar con una cinta que conste de 0^m10^n rodeada de espacios en blanco. M se parará cuando el contenido de la cinta sea $0^{m \dot{-} n}$ rodeado de espacios en blanco.

M encuentra repetidamente el 0 más a la izquierda que queda y lo reemplaza por un espacio en blanco. A continuación, se mueve hacia la derecha, en busca de un 1. Después de encontrar un 1, continúa moviéndose hacia la derecha hasta que llega a un 0, que sustituye por un 1. M vuelve entonces a moverse hacia la izquierda, buscando el 0 más a la izquierda, el cual identifica cuando encuentra un espacio en blanco a su izquierda, y luego se mueve una casilla hacia la derecha. Este proceso se termina si:

1. Buscando hacia la derecha un 0, M encuentra un espacio en blanco. Esto significa que los n ceros de 0^m10^n han sido todos ellos sutituidos por unos, y $n+1$ de los m ceros han sido sustituidos por B. M reemplaza los $n+1$ unos por un 0 y n símbolos B, dejando $m-n$ ceros en la cinta. Dado que $m \geq n$ en este caso, $m-n = m \dot{-} n$.
2. Al comenzar un ciclo, M no puede encontrar un 0 para sustituirlo por un espacio en blanco, porque los m primeros ceros ya han sido sustituidos por símbolos B. Entonces $n \geq m$, por lo que $m \dot{-} n = 0$. M reemplaza todos los unos y ceros restantes por símbolos B y termina con una cinta completamentamente en blanco.

La Figura 8.11 proporciona las reglas de la función de transición δ. También, en la Figura 8.12, hemos representado δ como un diagrama de transiciones. A continuación se resume el papel desempeñado por cada uno de los siete estados:

q_0 Este estado inicia el ciclo y también lo interrumpe cuando es necesario. Si M está señalando a un 0, el ciclo debe repetirse. El 0 se reemplaza por B, la cabeza se mueve hacia la derecha y se pasa al estado q_1. Por el contrario, si M está señalando un espacio en blanco B, entonces es que se han hecho todas las posibles correspondencias entre los dos grupos de ceros de la cinta y M pasa al estado q_5 para dejar la cinta en blanco.

q_1 En este estado, M busca hacia la derecha atravesando el bloque inicial de ceros y buscando el 1 más a la izquierda. Cuando lo encuentra, M pasa al estado q_2.

q_2 M se mueve hacia la derecha, saltando por encima de los unos hasta que encuentra un 0. Cambia dicho 0 por un 1, vuelve a moverse hacia la izquierda y entra en el estado q_3. Sin embargo, también es posible que no exista ningún cero más a la izquierda después del bloque de unos. En este caso, M encuentra un espacio en blanco cuando está en el estado q_2. Estamos entonces en el caso (1) descrito anteriormente, donde los n ceros del segundo bloque de ceros se han empleado para cancelar n de los m ceros del primer bloque, y la sustracción ya ha finalizado. M entra en el estado q_4, cuyo propósito es el de convertir los unos de la cinta en espacios en blanco.

q_3 M se mueve a la izquierda, saltando por encima de los ceros y los unos, hasta encontrar un espacio en blanco. Cuando encuentra un espacio en blanco B, se mueve hacia la derecha y vuelve al estado q_0, comenzando de nuevo el ciclo.

q_4 Aquí, el proceso de sustracción está terminado, pero uno de los ceros del primer bloque se ha sustituido incorrectamente por un espacio en blanco B. Por tanto, M se mueve hacia la izquierda, cambiando los unos por espacios en blanco B, hasta encontrar un espacio B sobre la cinta. Cambia dicho espacio en blanco por 0 y entra en el estado q_6, en el que M se detiene.

q_5 Desde el estado q_0 se pasa al estado q_5 cuando se comprueba que todos los ceros del primer bloque han sido cambiados por espacios en blanco B. En este caso, descrito en el punto (2) anterior, el resultado de la sustracción propia es 0. M cambia todos los ceros y unos por espacios en blanco B y pasa al estado q_6.

Estado	Símbolo 0	1	B
q_0	(q_1,B,R)	(q_5,B,R)	–
q_1	$(q_1,0,R)$	$(q_2,1,R)$	–
q_2	$(q_3,1,L)$	$(q_2,1,R)$	(q_4,B,L)
q_3	$(q_3,0,L)$	$(q_3,1,L)$	(q_0,B,R)
q_4	$(q_4,0,L)$	(q_4,B,L)	$(q_6,0,R)$
q_5	(q_5,B,R)	(q_5,B,R)	(q_6,B,R)
q_6	–	–	–

Figura 8.11. Una máquina de Turing que calcula la función de sustracción propia.

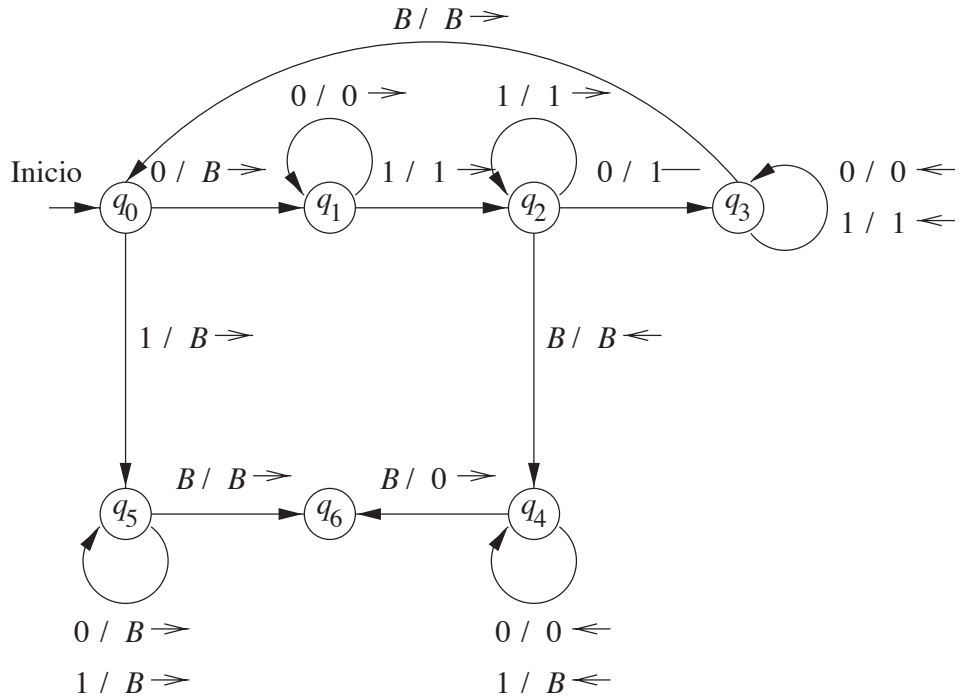

Figura 8.12. Diagrama de transiciones para la MT del Ejemplo 8.4.

q_6 El único propósito de este estado es permitir que M se detenga cuando haya terminado su tarea. Si la sustracción ha sido una subrutina de alguna función más compleja, entonces q_6 iniciará el siguiente paso de dicho cálculo más largo. □

8.2.5 El lenguaje de una máquina de Turing

Intuitivamente, hemos sugerido la forma en que una máquina de Turing acepta un lenguaje. La cadena de entrada se coloca en la cinta y la cabeza de la cinta señala el símbolo de entrada más a la izquierda. Si la MT entra en un estado de aceptación, entonces la entrada se acepta, y no se acepta en cualquier otro caso.

Más formalmente, sea $M = (Q,\Sigma,\Gamma,\delta,q_0,B,F)$ una máquina de Turing. Entonces $L(M)$ es el conjunto de cadenas w de Σ^* tales que $q_0w \overset{*}{\vdash} \alpha p\beta$ para algún estado p de F y cualesquiera cadenas α y β. Esta definición

Convenios de notación para las máquinas de Turing

Los símbolos que normalmente se emplean para las máquinas de Turing son parecidos a los que se usan en otros tipos de autómatas que ya hemos visto.

1. Las letras minúsculas del principio del alfabeto se emplean para los símbolos de entrada.
2. Las letras mayúsculas, normalmente las próximas al final del alfabeto, se emplean para los símbolos de cinta que pueden o no ser símbolos de entrada. Sin embargo, B suele utilizarse para designar el espacio en blanco.
3. Las letras minúsculas del final del alfabeto se emplean para designar cadenas de símbolos de entrada.
4. Las letras griegas se utilizan para las cadenas de símbolos de cinta.
5. Letras como q, p, y próximas a éstas se utilizan para los estados.

se ha dado por supuesto al abordar la máquina de Turing del Ejemplo 8.2, la cual acepta cadenas de la forma 0^n1^n.

El conjunto de lenguajes que podemos aceptar utilizando una máquina de Turing a menudo se denominan *lenguajes recursivamente enumerables* o lenguajes RE. El término "recursivamente enumerable" viene de los formalismos de computación que precedieron a la máquina de Turing pero que definen la misma clase de lenguajes o funciones aritméticas. En un recuadro de la Sección 9.2.1 se exponen los orígenes de este término.

8.2.6 Máquinas de Turing y parada

Existe otro concepto de "aceptación" que normalmente se emplea con las máquinas de Turing: aceptación por parada. Decimos que una MT *se para* si entrada en un estado q, señalando a un símbolo de cinta X, y no existe ningún movimiento en esa situación; es decir, $\delta(q,X)$ no está definida.

EJEMPLO 8.5

La máquina de Turing M del Ejemplo 8.4 no estaba diseñada para aceptar lenguajes; como hemos visto, en lugar de ello calculaba una función aritmética. Observe, no obstante, que M se detiene para todas las cadenas de ceros y unos, ya que independientemente de qué cadena encuentre en su cinta, finalmente cancelará el segundo grupo de ceros, si puede encontrar dicho grupo, con el primero, y luego llega al estado q_6 y se para. □

Siempre podemos suponer que una MT se para si acepta. Es decir, sin cambiar el lenguaje aceptado, podemos hacer $\delta(q,X)$ indefinida cuando q es un estado de aceptación. En general, a menos que se diga lo contrario:

- Suponemos que una máquina de Turing siempre se detiene cuando está en un estado de aceptación.

Lamentablemente, no siempre es posible exigir que una MT se pare si no acepta. Los lenguajes reconocidos por máquinas de Turing que no se detienen, independientemente de si aceptan o no, se denominan *recursivos*, y veremos sus importantes propiedades a partir de la Sección 9.2.1. Las máquinas de Turing que siempre se paran, independientemente si aceptan o no, son un buen modelo de "algoritmo". Si existe un algoritmo que permite resolver un determinado problema, entonces decimos que el problema es "decidible", por lo que las máquinas de Turing que siempre se paran desempeñan un papel importante en la teoría de la decidibilidad que se aborda en el Capítulo 9.

8.2.7 Ejercicios de la Sección 8.2

Ejercicio 8.2.1. Determine las configuraciones de la máquina de Turing de la Figura 8.9 si la cinta de entrada contiene:

* a) 00.

b) 000111.

c) 00111.

! **Ejercicio 8.2.2.** Diseñe máquinas de Turing para los siguientes lenguajes:

* a) El conjunto de cadenas con el mismo número de ceros que de unos.

b) $\{a^n b^n c^n \mid n \geq 1\}$.

c) $\{ww^R \mid w$ es cualquier cadena de ceros y unos$\}$.

Ejercicio 8.2.3. Diseñe una máquina de Turing que tome como entrada un número N y le añada 1 en binario. Para ser más precisos, inicialmente la cinta contiene el símbolo de \$ seguido por N en binario. La cabeza de la cinta inicialmente está señalando al símbolo \$ estando en el estado q_0. La MT debe pararse cuando en la cinta haya $N+1$, en binario y esté señalando al símbolo más a la izquierda de $N+1$, estando en el estado q_f. Si fuera necesario, en el proceso de creación de $N+1$ se puede destruir \$. Por ejemplo, $q_0\$10011 \vdash^* \$q_f10100$ y $q_0\$11111 \vdash^* q_f100000$.

a) Determine las transiciones de la máquina de Turing y explique el propósito de cada estado.

b) Muestre la secuencia de configuraciones de la máquina de Turing para la entrada \$111.

*! **Ejercicio 8.2.4.** En este ejercicio exploraremos la equivalencia entre el cálculo de funciones y el reconocimiento de lenguajes para las máquinas de Turing. Con el fin de simplificar, sólo consideraremos funciones de enteros no negativos a enteros no negativos, aunque las ideas de este problema se aplican a cualquier función computable. He aquí las dos definiciones fundamentales:

- Se define el *grafo* de una función f como el conjunto de todas las cadenas de la forma $[x, f(x)]$, donde x es un entero no negativo en binario y $f(x)$ es el valor de la función f con x como argumento, también expresado en binario.
- Una máquina de Turing se dice que *calcula* una función f si, partiendo de una cinta que tiene un entero no negativo x, en binario, se para (en cualquier estado) cuando la cinta contiene $f(x)$ en binario.

Responda a las siguientes cuestiones con construcciones informales pero claras.

a) Dada una MT que calcula f, muestre cómo se puede construir una MT que acepte el grafo de f como lenguaje.

b) Dada una MT que acepta el grafo f, muestre cómo se puede construir una MT que calcule f.

c) Se dice que una función es *parcial* si puede no estar definida para algunos argumentos. Si extendemos las ideas de este ejercicio a las funciones parciales, entonces no se exige que la MT que calcula f se pare si su entrada x es uno de los enteros para los que $f(x)$ no está definida. ¿Funcionan las construcciones de los apartados (a) y (b) si la función f es parcial? Si la respuesta es no, explique cómo se podrían modificar para que funcionaran.

Ejercicio 8.2.5. Considere la máquina de Turing

$$M = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B, \{q_f\})$$

De manera informal y clara describa el lenguaje $L(M)$ si δ consta de los siguientes conjuntos de reglas:

* a) $\delta(q_0,0) = (q_1,1,R)$; $\delta(q_1,1) = (q_0,0,R)$; $\delta(q_1,B) = (q_f,B,R)$.

b) $\delta(q_0,0) = (q_0,B,R)$; $\delta(q_0,1) = (q_1,B,R)$;
$\delta(q_1,1) = (q_1,B,R)$; $\delta(q_1,B) = (q_f,B,R)$.

! c) $\delta(q_0,0) = (q_1,1,R)$; $\delta(q_1,1) = (q_2,0,L)$;
$\delta(q_2,1) = (q_0,1,R)$; $\delta(q_1,B) = (q_f,B,R)$.

8.3 Técnicas de programación para las máquinas de Turing

Nuestro objetivo es proporcionarle la idea de cómo se puede utilizar una máquina de Turing para hacer cálculos de una manera muy diferente a como lo hace una computadora convencional. En particular, veremos que la máquina de Turing puede realizar, sobre otras máquinas de Turing, el tipo de cálculos que puede realizar mediante un programa al examinar otros programas, como vimos en la Sección 8.1.2. Esta habilidad "introspectiva" de las máquinas de Turing y los programas de computadora es lo que nos permite demostrar la indecibilidad de algunos problemas.

Para clarificar cuál es la capacidad de una MT, presentaremos una serie de ejemplos de cómo tenemos que pensar en términos de la cinta y la unidad de control de la máquina de Turing. Ninguno de estos trucos extiende el modelo básico de la MT; son únicamente convenios de notación, que emplearemos más adelante para simular los modelos extendidos de la máquina de Turing que tienen características adicionales (por ejemplo, con más de una cinta que el modelo básico).

8.3.1 Almacenamiento en el estado

Podemos emplear la unidad de control no sólo para representar una posición en el "programa" de la máquina de Turing, sino también para almacenar una cantidad finita de datos. La Figura 8.13 muestra esta técnica (además de otra idea: pistas múltiples). En ella vemos que la unidad de control consta de no sólo un estado de "control" q, sino además de tres elementos de datos A, B y C. Esta técnica no requiere extensiones del modelo de la MT; basta con pensar en el estado como en una tupla. En el caso de la Figura 8.13, el estado podría ser $[q,A,B,C]$. Considerar los estados de esta manera nos permite describir las transiciones de una forma más sistemática, haciendo que la estrategia subyacente al programa de la MT resulte, a menudo, más transparente.

EJEMPLO 8.6

Diseñaremos una máquina de Turing

$$M = (Q, \{0,1\}, \{0,1,B\}, \delta, [q_0,B], \{[q_1,B]\})$$

que recuerde en su unidad de control el primer símbolo (0 o 1) que ve y compruebe que no aparece en ningún otro lugar de su entrada. Por tanto, M acepta el lenguaje $\mathbf{01}^* + \mathbf{10}^*$. La aceptación de lenguajes regulares como éste no sirve para subrayar la capacidad de las máquinas de Turing, pero servirá como demostración sencilla.

El conjunto de estados Q es $\{q_0,q_1\} \times \{0,1,B\}$. Es decir, puede pensarse en los estados como en pares de dos componentes:

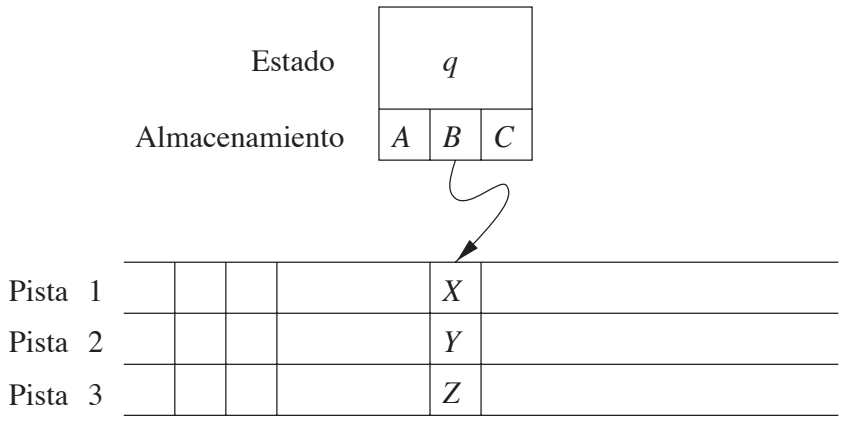

Figura 8.13. Una máquina de Turing con almacenamiento en la unidad de control y pistas múltiples.

a) Una parte de control, q_0 o q_1, que recuerda lo que está haciendo la MT. El estado de control q_0 indica que M todavía no ha leído su primer símbolo, mientras que q_1 indica que *ya* lo ha leído, y está comprobando que no aparece en ningún otro lugar, moviéndose hacia la derecha con la esperanza de alcanzar una casilla con un espacio en blanco.

b) Una parte de datos, que recuerda el primer símbolo que se ha visto, que tiene que ser 0 o 1. El símbolo B en esta componente indica que no se ha leído ningún símbolo.

La función de transición δ de M es la siguiente:

1. $\delta([q_0,B],a) = ([q_1,a],a,R)$ para $a = 0$ o $a = 1$. Inicialmente, q_0 es el estado de control y la parte de datos del estado es B. El símbolo que está siendo señalado se copia en la segunda componente del estado y M se mueve hacia la derecha entrando en el estado de control q_1.
2. $\delta([q_1,a],\overline{a}) = ([q_1,a],\overline{a},R)$ donde $\overline{a}$ es el "complementario" de a, es decir, 0 si $a = 1$ y 1 si $a = 0$. En el estado q_1, M salta por encima de cada símbolo 0 o 1 que es diferente del que tiene almacenado en su estado y continúa moviéndose hacia la derecha.
3. $\delta([q_1,a],B) = ([q_1,B],B,R)$ para $a = 0$ o $a = 1$. Si M llega al primer espacio en blanco, entra en el estado de aceptación $[q_1,B]$.

Observe que M no está definida para $\delta([q_1,a],a)$ para $a = 0$ o $a = 1$. Luego si M encuentra una segunda aparición del símbolo almacenado inicialmente en su unidad de control, se para sin entrar en el estado de aceptación. □

8.3.2 Pistas múltiples

Otro "truco" práctico es el de pensar que la cinta de una máquina de Turing está compuesta por varias pistas. Cada pista puede almacenar un símbolo, y el alfabeto de cinta de la MT consta de tuplas, con una componente para cada "pista". Por ejemplo, la casilla señalada por la cabeza de la cinta en la Figura 8.13 contiene el símbolo $[X,Y,Z]$. Al igual que la técnica de almacenamiento en la unidad de control, el uso de múltiples pistas no es una extensión de lo que puede hacer la máquina de Turing. Se trata simplemente de una forma de ver los símbolos de la cinta y de imaginar que tienen una estructura útil.

EJEMPLO 8.7

Un uso común de las pistas múltiples consiste en considerar una pista que almacena los datos y una segunda

pista que almacena una marca. Podemos marcar cada uno de los símbolos como "utilizado", o podemos seguir la pista de un número pequeño de posiciones dentro de los datos marcando sólo dichas posiciones. En los Ejemplos 8.2 y 8.4 se ha visto esta técnica, aunque no se ha mencionado explícitamente que la cinta estuviera formada por varias pistas. En este ejemplo, utilizaremos explícitamente una segunda pista para reconocer el lenguaje no independiente del contexto,

$$L_{wcw} = \{wcw \mid w \text{ pertenece a } (\mathbf{0}+\mathbf{1})^+\}$$

La máquina de Turing que vamos a diseñar es:

$$M = (Q, \Sigma, \Gamma, \delta, [q_1, B], [B, B], \{[q_9, B]\})$$

donde:

Q El conjunto de estados es $\{q_1, q_2, \ldots, q_9\} \times \{0, 1\}$, es decir, pares que constan de un estado de control q_i y una componente de datos, 0 ó 1. De nuevo, utilizamos la técnica de almacenamiento en la unidad de control, permitiendo que el estado recuerde un símbolo de entrada 0 ó 1.

Γ El conjunto de símbolos de cinta es $\{B, *\} \times \{0, 1, c, B\}$. La primera componente, o pista, puede ser o un espacio en blanco o un símbolo "marcado" (que se ha revisado), y se representan respectivamente mediante los símbolos B y $*$. Utilizamos $*$ para marcar como revisados los símbolos del primer y segundo grupos de ceros y unos, confirmando así que la cadena a la izquierda del marcador central c es la misma que la cadena situada a su derecha. La segunda componente del símbolo de cinta es el propio símbolo de cinta. Es decir, interpretamos el símbolo $[B, X]$ como el símbolo de cinta X para $X = 0, 1, c, B$.

Σ Los símbolos de entrada son $[B, 0]$ y $[B, 1]$, los cuales, como ya se ha dicho, se identifican con 0 y 1, respectivamente.

δ La función de transición δ se define de acuerdo con las siguientes reglas, en las que a y b pueden tomar los valores 0 ó 1.

1. $\delta([q_1, B], [B, a]) = ([q_2, a], [*, a], R)$. En el estado inicial, M lee el símbolo a (que puede ser 0 o 1), lo almacena en su unidad de control, pasa al estado de control q_2, "marca como revisado" el símbolo que acaba de leer y se mueve hacia la derecha. Observe que cambiando la primera componente del símbolo de cinta de B a $*$, se lleva a cabo la revisión del símbolo.
2. $\delta([q_2, a], [B, b]) = ([q_2, a], [B, b], R)$. M se mueve hacia la derecha en busca del símbolo c. Recuerde que a y b pueden ser 0 o 1, de forma independiente, pero no pueden ser c.
3. $\delta([q_2, a], [B, c]) = ([q_3, a], [B, c], R)$. Cuando M encuentra el símbolo c, continúa moviéndose hacia la derecha, pero cambia al estado de control q_3.
4. $\delta([q_3, a], [*, b]) = ([q_3, a], [*, b], R)$. En el estado q_3, M pasa por encima de todos los símbolos que ya se han revisado.
5. $\delta([q_3, a], [B, a]) = ([q_4, B], [*, a], L)$. Si el primer símbolo no revisado que encuentra M es el mismo que el símbolo que se encuentra en su unidad de control, marca este símbolo como revisado, ya que está emparejado con el símbolo correspondiente del primer bloque de ceros y unos. M pasa al estado de control q_4, eliminando el símbolo de su unidad de control y comienza a moverse hacia la izquierda.
6. $\delta([q_4, B], [*, a]) = ([q_4, B], [*, a], L)$. M se mueve hacia la izquierda pasando sobre todos los símbolos revisados.
7. $\delta([q_4, B], [B, c]) = ([q_5, B], [B, c], L)$. Cuando M encuentra el símbolo c, pasa al estado q_5 y continúa moviéndose hacia la izquierda. En el estado q_5, M tiene que tomar una decisión, que depende de si el símbolo inmediatamente a la izquierda del símbolo está marcado o no como revisado. Si está

revisado, entonces quiere decir que ya hemos considerado el primer bloque completo de ceros y unos (aquellos que están a la izquierda de c). Tenemos que asegurarnos de que todos los ceros y unos situados a la derecha del símbolo c también están revisados, y aceptar si no queda ningún símbolo no revisado a la derecha de c. Si el símbolo inmediatamente a la izquierda de la c no está revisado, buscaremos el símbolo no revisado más a la izquierda, lo leeremos e iniciaremos el ciclo que comienza en el estado q_1.

8. $\delta([q_5,B],[B,a]) = ([q_6,B],[B,a],L)$. Esta rama cubre el caso en que el símbolo a la izquierda de c no está revisado. M pasa al estado q_6 y continúa moviéndose hacia la izquierda, en busca de un símbolo revisado.
9. $\delta([q_6,B],[B,a]) = ([q_6,B],[B,a],L)$. Siempre y cuando existan símbolos no revisados, M permanece en el estado q_6 y continúa moviéndose hacia la izquierda.
10. $\delta([q_6,B],[*,a]) = ([q_1,B],[*,a],R)$. Cuando se encuentra el símbolo revisado, M entra en el estado q_1 y se mueve hacia la derecha hasta llegar al primer símbolo no revisado.
11. $\delta([q_5,B],[*,a]) = ([q_7,B],[*,a],R)$. Ahora, elegimos la rama que sale del estado q_5, que corresponde al caso en que M se mueve hacia la izquierda de c y se encuentra un símbolo revisado. Comenzamos de nuevo a movernos hacia la derecha pasando al estado q_7.
12. $\delta([q_7,B],[B,c]) = ([q_8,B],[B,c],R)$. En el estado q_7 seguramente veremos el símbolo c. M entrará en el estado q_8 y continúa moviéndose hacia la derecha.
13. $\delta([q_8,B],[*,a]) = ([q_8,B],[*,a],R)$. En el estado q_8, M se mueve hacia la derecha, saltando sobre todos los ceros y unos revisados que ya ha encontrado.
14. $\delta([q_8,B],[B,B]) = ([q_9,B],[B,B],R)$. Si M llega a una casilla en blanco estando en el estado q_8 sin encontrar ningún 0 ni 1 no revisado, entonces acepta. Si M encuentra un 0 o un 1 no revisado, entonces quiere decir que los bloques anterior y posterior al símbolo c no concuerdan y M se parará sin aceptar. □

8.3.3 Subrutinas

Como ocurre en general con los programas, resulta de ayuda pensar que las máquinas de Turing se construyen a partir de una colección de componentes interactivos, o "subrutinas". Una subrutina de una máquina de Turing es un cojunto de estados que realiza un determinado proceso útil. Este conjunto de estados incluye un estado inicial y otro estado en el que temporalmente no existen movimientos, y que sirve como estado de "retorno" para pasar el control a cualquier otro conjunto de estados que llame la subrutina. La "llamada" de una subrutina se produce cuando existe una transición a su estado inicial. Dado que la MT no tiene ningún mecanismo que la permita recordar una "dirección de retorno"; es decir, un estado al que volver una vez que haya terminado, si la llamada a una MT que actúa como subrutina tiene que hacerse desde varios estados, se pueden hacer copias de la subrutina utilizando un nuevo conjunto de estados para cada copia. Las "llamadas" se realizan a los estados iniciales de las distintas copias de la subrutina y cada copia "vuelve" a un estado distinto.

EJEMPLO 8.8

Vamos a diseñar una MT para implementar la función "multiplicación". Es decir, la MT tendrá inicialmente en la cinta 0^m10^n y al final tendrá 0^{mn}. Un esquema de la estrategia es la siguiente:

1. En general, la cinta contendrá una cadena sin ningún espacio en blanco de la forma $0^i10^n10^{kn}$ para un cierto k.
2. En el paso básico, cambiamos un 0 del primer grupo por espacios en blanco B y añadimos n ceros al último grupo, obteniendo así una cadena de la forma $0^{i-1}10^n10^{(k+1)n}$.

3. Como resultado, se habrá copiado m veces el grupo de n ceros al final, una por cada vez que hayamos cambiado un 0 del primer grupo por un símbolo B. Cuando todo el primer grupo de ceros se haya cambiado por espacios en blanco, habrá mn ceros en el último grupo.

4. El último paso consiste en cambiar la cadena 10^n1 del principio por espacios en blanco.

El núcleo de este algoritmo es una subrutina, que llamaremos `Copia`. Esta subrutina implementa el paso (2) anterior, copiando el bloque de n ceros al final. Dicho de forma más precisa, `Copia` convierte una configuración de la forma $0^{m-k}1q_10^n10^{(k-1)n}$ en una configuración de la forma $0^{m-k}1q_50^n10^{kn}$. La Figura 8.14 muestra las transiciones de la subrutina `Copia`. Esta subrutina marca el primer 0 con una X, se mueve hacia la derecha permaneciendo en el estado q_2 hasta que encuentra un espacio en blanco, allí copia el 0 y se mueve hacia la izquierda en el estado q_3 hasta encontrar el marcador X. Repite este ciclo hasta que encuentra un 1 estando en el estado q_1 en lugar de un 0. En este punto, utiliza el estado q_4 para cambiar de nuevo las X por ceros y termina en el estado q_5.

La máquina de Turing completa para la multiplicación parte del estado q_0. Lo primero que hace es ir, en varios pasos, desde la configuración $q_00^m10^n$ a la configuración $0^{m-1}1q_10^n1$. Las transiciones necesarias se muestran en la Figura 8.15, en la parte que se encuentra a la izquierda de la llamada a la subrutina; estas transiciones sólo implican a los estados q_0 y q_6.

En la parte de la derecha de la llamada a la subrutina en la Figura 8.15, podemos ver los estados q_7 hasta q_{12}. El propósito de los estados q_7, q_8 y q_9 es el de tomar el control justo después de que `Copia` haya copiado un bloque de n ceros y se encuentre en la configuración $0^{m-k}1q_50^n10^{kn}$. Finalmente, estos estados nos llevan al estado $q_00^{m-k}10^n10^{kn}$. En este punto, el ciclo comienza de nuevo y se llama a `Copia` para copiar de nuevo el bloque de n ceros.

Como excepción, en el estado q_8, la MT puede encontrarse con que los m ceros se han cambiado por espacios en blanco (es decir, $k = m$). En este caso, se produce una transición al estado q_{10}. Este estado, con ayuda del estado q_{11}, cambia la cadena 10^n1 del principio por espacios en blanco y lleva al estado de parada q_{12}. En este punto, la MT se encuentra en la configuración $q_{12}0^{mn}$, y su trabajo ha terminado. □

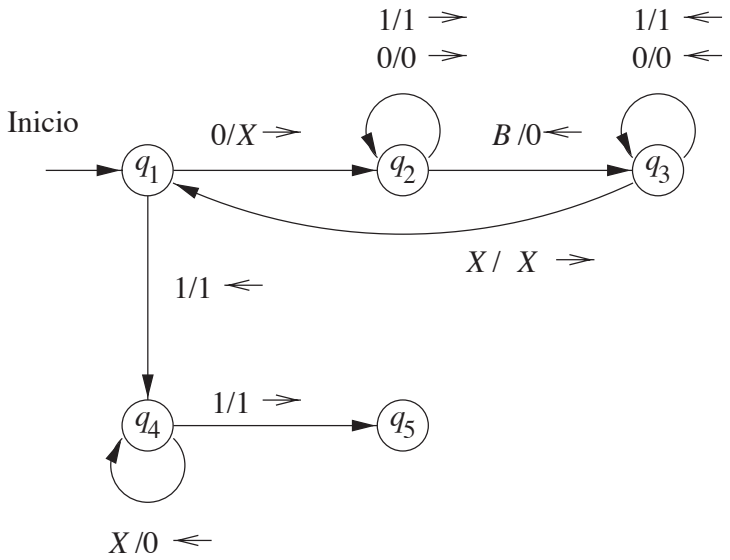

Figura 8.14. La subrutina `Copia`.

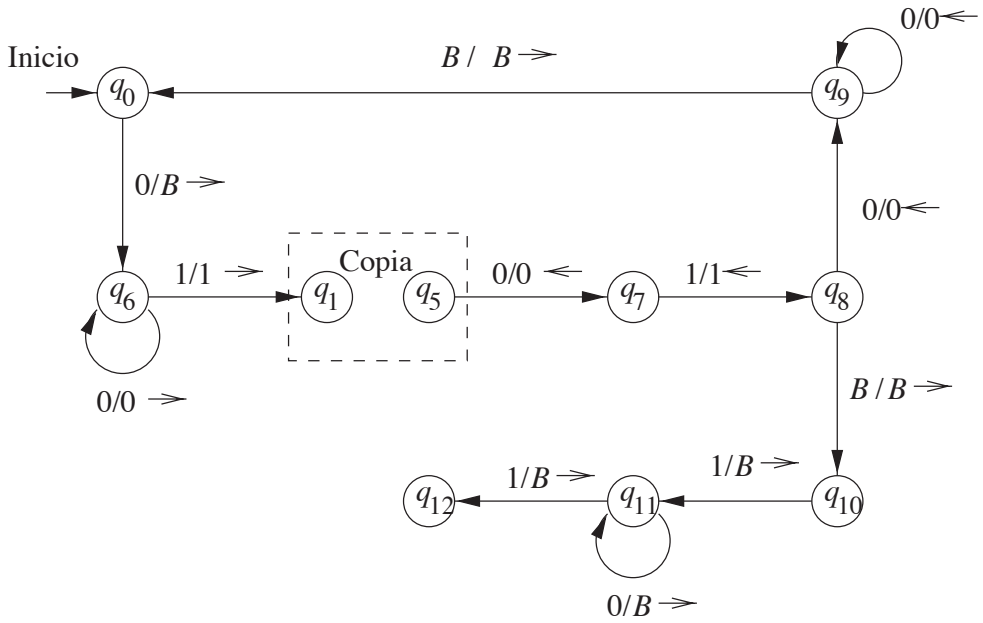

Figura 8.15. El programa de multiplicación completo utiliza la subrutina `Copia`.

8.3.4 Ejercicios de la Sección 8.3

! Ejercicio 8.3.1. Rediseñe las máquinas de Turing del Ejercicio 8.2.2 para aprovechar las técnicas de programación vistas en la Sección 8.3.

! Ejercicio 8.3.2. Una operación habitual en los programas de las máquinas de Turing es el "desplazamiento". Idealmente, estaría bien crear una casilla adicional en la posición de la cabeza actual de la cinta, en la que podríamos almacenar algún carácter. Sin embargo, no es posible modificar la cinta de esta forma. En lugar de ello, tenemos que mover, una casilla a la derecha, el contenido de cada una de las casillas hacia la derecha respecto de la posición actual de la cabeza y luego encontrar una forma de volver a la posición actual de la cabeza. Indique cómo realizar esta operación. *Consejo*: deje un símbolo especial para marcar la posición a la que debe volver la cabeza.

*** Ejercicio 8.3.3.** Diseñe una subrutina para mover la cabeza de una MT desde su posición actual hacia la derecha, saltando por encima de todos los ceros, hasta llegar a un 1 o un espacio en blanco. Si la posición actual no almacena un 0, entonces la MT tiene que detenerse. Puede suponer que no existe ningún símbolo de cinta distinto de 0, 1 y B (espacio en blanco). Después, emplee esta subrutina para diseñar una MT que acepte todas las cadenas de ceros y unos que no contengan dos unos en una fila.

8.4 Extensiones de la máquina de Turing básica

En esta sección veremos algunos modelos de computadora que están relacionados con las máquinas de Turing y que tienen la misma funcionalidad de reconocimiento de lenguajes que el modelo básico de la MT con la que hemos estado trabajando. Uno de estos modelos es la máquina de Turing de varias cintas, y es importante porque es mucho más fácil ver cómo una MT de varias cintas puede simular computadoras reales (u otras clases de máquinas de Turing), en comparación con el modelo de una sola cinta que hemos estudiado. No obstante, las cintas adicionales no añaden potencia al modelo, en lo que se refiere a la capacidad de aceptar lenguajes.

Consideramos entonces la máquina de Turing no determinista, una extensión del modelo básico que permite elegir un movimiento entre un conjunto finito de posibles movimientos en una situación dada. Esta extensión

también facilita la "programación" de las máquinas de Turing en algunas circunstancias, aunque no añade al modelo básico potencia en lo que se refiere a la definición de lenguajes.

8.4.1 Máquina de Turing de varias cintas

En la Figura 8.16 se muestra una máquina de Turing de varias cintas. El dispositivo tiene una unidad de control (estado) y un número finito de cintas. Cada cinta está dividida en casillas, y cada casilla puede contener cualquier símbolo del alfabeto de cinta finito. Al igual que en la MT de una sola cinta, el conjunto de símbolos de la cinta incluye el espacio en blanco y también dispone de un subconjunto de símbolos de entrada, al que no pertenece el espacio en blanco. El conjunto de estados incluye un estado inicial y varios estados de aceptación. Inicialmente:

1. La entrada, una secuencia finita de símbolos de entrada, se coloca en la primera cinta.
2. Todas las casillas de las demás cintas contienen espacios en blanco.
3. La unidad de control se encuentra en el estado inicial.
4. La cabeza de la primera cinta apunta al extremo izquierdo de la entrada.
5. Las cabezas de las restantes cintas apuntan a una casilla arbitraria. Puesto que las cintas distintas de la primera están completamente en blanco, no es importante dónde se sitúe inicialmente la cabeza; todas las casillas de estas cintas "parecen" idénticas.

Un movimiento de la MT de varias cintas depende del estado y del símbolo señalado por las cabezas de cada una de las cintas. En un movimiento, esta MT hace lo siguiente:

1. La unidad de control entra en un nuevo estado, que podría ser el mismo que en el que se encontraba anteriormente.
2. En cada cinta, se escribe un nuevo símbolo de cinta en la casilla señalada por la cabeza. Estos símbolos pueden ser los mismos que estaban escritos anteriormente.

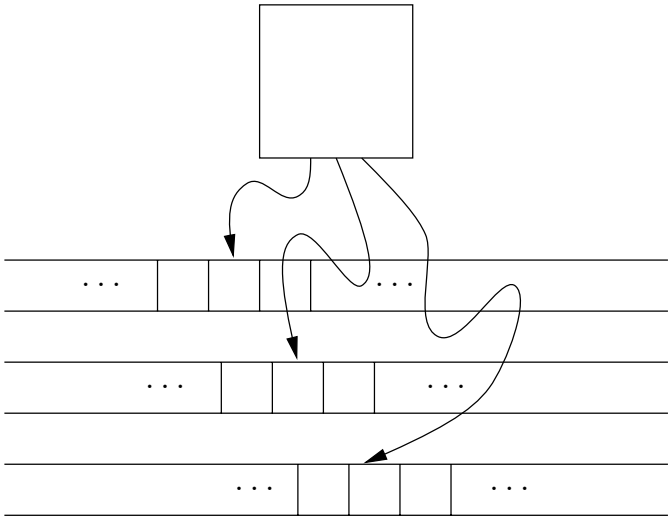

Figura 8.16. Máquina de Turing de varias cintas.

3. Cada una de las cabezas de las cintas realizan un movimiento, que puede ser hacia la izquierda, hacia la derecha o estacionario. Las cabezas se mueven de manera independiente, por lo que pueden moverse en direcciones diferentes y alguna puede no moverse en absoluto.

No vamos a proporcionar la notación formal de las reglas de transición, cuya forma es una generalización directa de la notación para la MT de una cinta, excepto en lo que se refiere a las direcciones que ahora los movimientos pueden tomar: L (izquierda), R (derecha) o S (estacionaria). En la máquina de una sola cinta, la cabeza no podía permanecer estacionaria, por lo que la opción S no fue mencionada. Debe ser capaz de imaginar una notación apropiada para las descripciones instantáneas de una MT de varias cintas; esta notación no la vamos especificar formalmente. Las máquinas de Turing de varias cintas, al igual que las de una sola cinta, aceptan al alcanzar un estado de aceptación.

8.4.2 Equivalencia entre las MT de una sola cinta y de varias cintas

Recuerde que los lenguajes recursivamente enumerables se definen como aquellos que son aceptados por una MT de una sola etapa. Es prácticamente seguro que las MT de varias cintas aceptan todos los lenguajes recursivamente enumerables, ya que una MT de una sola cinta *es* un caso particular de una MT de varias cintas. Sin embargo, ¿existen lenguajes que no sean recursivamente enumerables que sean aceptados por las máquinas de Turing de varias cintas? La respuesta es "no" y vamos a demostrar este hecho mostrando cómo simular una MT de varias cintas mediante una MT de una sola cinta.

TEOREMA 8.9

Todo lenguaje aceptado por una MT de varias cintas es recursivamente enumerable.

DEMOSTRACIÓN. La demostración se muestra en la Figura 8.17. Supongamos que el lenguaje L es aceptado por una MT de k cintas M. Simulamos M mediante una MT de una única cinta N, cuya cinta consta de $2k$ pistas. La mitad de estas pistas almacenan las cintas de M, y la otra mitad de las pistas almacena, cada una de ellas, sólo un único marcador que indica donde se encuentra actualmente la cabeza de la cinta correspondiente de M. La Figura 8.17 supone $k = 2$. La segunda y la cuarta pistas almacenan los contenidos de la primera y la segunda cintas de M, la pista 1 almacena la posición de la cabeza de la primera cinta y la pista 3 almacena la posición de la cabeza de la segunda cinta.

Para simular un movimiento de M, la cabeza de N tiene que acceder a los k marcadores de cada cabeza. Con el fin de que N no se pierda, tiene que recordar en cada momento cuántos marcadores quedan a su izquierda; esta cuenta se almacena en una componente de la unidad de control de N. Después de acceder a cada marcador de cabeza y almacenar el símbolo al que señalan en una componente de su unidad de control, N sabe cuáles son los símbolos señalados por cada una de las cabezas de M. N también conoce el estado de M, que se almacena en la propia unidad de control de N. Por tanto, N sabe qué movimiento realizará M.

Ahora N vuelve a acceder a cada uno de los marcadores de cabeza de su cinta, cambia el símbolo de la pista que representa a las cintas correspondientes de M y mueve los marcadores de cabeza hacia la izquierda o hacia la derecha, si fuera necesario. Por último, N cambia el estado de M de acuerdo con lo que se haya registrado en su propia unidad de control. En este punto, N ha simulado un movimiento de M.

Seleccionamos como estados de aceptación de N todos aquellos estados que registran el estado de M como uno de sus estados de aceptación. Por tanto, cuando la máquina de Turing M simulada acepta, N también acepta y no aceptará en cualquier otro caso. □

8.4.3 Tiempo de ejecución en la construcción que pasa de muchas cintas a una

Ahora vamos a introducir un concepto que resultará de gran importancia más adelante: la "complejidad temporal" o "tiempo de ejecución" de una máquina de Turing. Decimos que el *tiempo de ejecución* de la MT M para la

entrada w es el número de pasos que M realiza antes de pararse. Si M no se para con la entrada w, entonces el tiempo de ejecución de M para w es infinito. La *complejidad temporal* de la MT M es la función $T(n)$ que es el máximo, para todas las entradas w de longitud n, de los tiempos de ejecución de M para w. En las máquinas de Turing que no se detienen para todas las entradas, $T(n)$ puede ser infinito para algún o incluso para todo n. Sin embargo, pondremos una atención especial en las MT que se paran para todas las entradas, y en concreto, en aquellas que tengan una complejidad temporal polinómina $T(n)$. En la Sección 10.1 se inicia este estudio.

La construcción del Teorema 8.9 puede parecer un poco torpe. De hecho, la MT de una cinta construida puede tener un tiempo de ejecución mayor que la MT de varias cintas. Sin embargo, los tiempos invertidos por las dos máquinas de Turing guardan cierta proporción: la MT de una cinta necesita un tiempo que no es mayor que el cuadrado del tiempo que necesita la máquina de varias cintas. Aunque el hecho de que la relación sea "cuadrática" no ofrece muchas garantías, mantiene el tiempo de ejecución polinómico. Veremos en el Capítulo 10 que:

a) La diferencia entre un tiempo polinómico y otras tasas de crecimiento más rápidas en el tiempo de ejecución es lo que realmente establece la separación entre lo que podemos resolver con una computadora y lo que no es resoluble en la práctica.

b) A pesar del gran número de investigaciones realizadas, no se ha conseguido que el tiempo de ejecución necesario para resolver muchos problemas no rebase un tiempo polinómico. Así, la cuestión de si estamos empleando una MT de una cinta o de varias cintas para resolver el problema no es crucial cuando examinamos el tiempo de ejecución necesario para dar solución a un problema concreto.

La demostración de que el tiempo de ejecución de la máquinas de Turing de una cinta es del orden del cuadrado del tiempo de ejecución de una MT de varias cintas es como sigue:

TEOREMA 8.10

El tiempo invertido por la MT de una cinta N del Teorema 8.9 para simular n movimientos de la MT de k cintas M es $O(n^2)$.

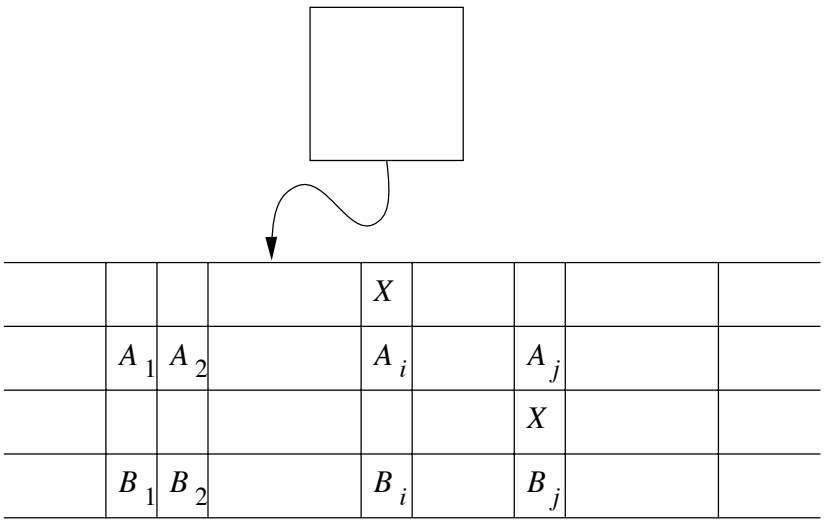

Figura 8.17. Simulación de una máquina de Turing de dos cintas mediante una máquina de Turing de una sola cinta.

Recordatorio sobre lo que es finito

Un error habitual es el de confundir un valor que es finito en cualquier instante de tiempo con un conjunto finito de valores. La construcción de una MT de varias cintas a partir de otra de una sola cinta puede ayudarnos a ver la diferencia. En esta construcción, hemos utilizado pistas de la cinta para registrar las posiciones de las cabezas de la cinta. ¿Por qué no podríamos almacenar estas posiciones como enteros en la unidad de control? En principio, podríamos argumentar que después de n movimientos, las posiciones de la cabeza de la cinta de la MT tienen que encontrarse como mucho a n posiciones de distancia de las posiciones originales de las cabezas, y de este modo la cabeza sólo tendría que almacenar enteros hasta n.

El problema es que, mientras que las posiciones son finitas en cualquier instante, el conjunto completo de posibles posiciones en cualquier instante es infinito. Si el estado sirve para representar cualquier posición de la cabeza, entonces tiene que existir una componente de datos del estado cuyo valor sea cualquier número entero. Esta componente fuerza a que el conjunto de estados sea infinito, incluso si sólo se puede emplear un número finito de ellos en cualquier momento. La definición de una máquina de Turing requiere que el *conjunto* de estados sea finito. Por tanto, no es posible almacenar la posición de la cabeza de la cinta en la unidad de control.

DEMOSTRACIÓN. Después de n movimientos de M, los marcadores de la cabeza de las cintas no pueden estar separados una distancia mayor que $2n$ casillas. Luego si M comienza en el marcador más a la izquierda, no puede moverse más de $2n$ casillas hacia a la derecha, para encontrar todos los marcadores de cabeza. Puede entonces realizar una excursión hacia la izquierda cambiando el contenido de las cintas simuladas de M, y moviendo los marcadores de cabeza hacia la izquierda o hacia la derecha según sea necesario. Este proceso no requiere más de $2n$ movimientos hacia la izquierda, más un máximo de $2k$ movimientos para invertir el sentido de movimiento y escribir un marcador X en la casilla situada a la derecha (en el caso de que la cabeza de una cinta de M se mueva hacia la derecha).

Por tanto, el número de movimientos que N necesita para simular uno de los n primeros movimientos no es mayor que $4n + 2k$. Dado que k es una constante, independiente del número de movimientos simulado, esta cantidad de movimientos es $O(n)$. Para simular n movimientos no necesita más de n veces esta cantidad, es decir, $O(n^2)$. □

8.4.4 Máquinas de Turing no deterministas

Una máquina de Turing *no determinista* (*MTN*) se diferencia de la máquina determinista que hemos estudiado hasta el momento en que tiene una función de transición δ tal que para el estado q y símbolo de cinta X, $\delta(q, X)$ es un conjunto de tuplas:

$$\{(q_1, Y_1, D_1),\ (q_2, Y_2, D_2), \ldots, (q_k, Y_k, D_k)\}$$

donde k es cualquier número entero finito. La MTN puede elegir, en cada paso, cuál de las tuplas será el siguiente movimiento. Sin embargo, no puede elegir un estado de una, un símbolo de cinta de otra y una dirección de una tercera.

El lenguaje aceptado por una MTN M se define, como era de esperar, de forma análoga a otros dispositivos no deterministas, tales como los AFN y los autómatas a pila que ya hemos estudiado. Es decir, M acepta una entrada w si existe cualquier secuencia de movimientos que lleva desde la configuración inicial con w como entrada hasta una configuración con un estado de aceptación. La existencia de otras opciones de movimientos que *no* lleven a un estado de aceptación es irrelevante, al igual que en el caso de los AFN y los autómatas a pila.

Las MTN no aceptan ningún lenguaje que no sea aceptado por una MT determinista (o *MTD* si necesitamos resaltar que se trata de una máquina determinista). La demostración implica demostrar que para toda MTN M_N, podemos construir una MTD M_D que explore las configuraciones a las que M_N puede llegar mediante cualquier secuencia posible de movimientos. Si M_D encuentra una secuencia que tiene un estado de aceptación, entonces M_D entra en uno de sus propios estados de aceptación. M_D tiene que ser sistemática, colocando las nuevas configuraciones en una cola, en lugar de en una pila, de modo que después de cierto tiempo finito, M_D habrá simulado todas las secuencias que consten como mucho de k movimientos de M_N, para $k = 1, 2, \ldots$.

TEOREMA 8.11

Si M_N es una máquina de Turing no determinista, entonces existe una máquina de Turing determinista M_D tal que $L(M_N) = L(M_D)$.

DEMOSTRACIÓN. Diseñaremos M_D como una máquina de Turing de varias cintas, como la mostrada en la Figura 8.18. La primera cinta de M_D almacena una secuencia de configuraciones de M_N, incluyendo el estado de M_N. Una configuración de M_N está marcada como configuración "actual", estando las subsiguientes configuraciones en el proceso de ser descubiertas. En la Figura 8.18, la tercera configuración está marcada mediante una x junto con el separador inter-configuraciones, que es el símbolo $*$. Todas las configuraciones situadas a la izquierda de la actual han sido exploradas y, por tanto, pueden ser ignoradas.

Para procesar la configuración actual, M_D hace lo siguiente:

1. M_D examina el estado y el símbolo al que señala la cabeza de la cinta de la configuración actual. La unidad de control de M_D conoce los movimientos de M_N para cada estado y símbolo. Si el estado de la configuración actual es de aceptación, entonces M_D acepta y termina la simulación de M_N.

2. Sin embargo, si el estado es de no aceptación y la combinación estado-símbolo da lugar a k movimientos, entonces M_D utiliza su segunda cinta para copiar la configuración y hacer a continuación k copias de dicha configuración al final de la secuencia de configuraciones de la cinta 1.

3. M_D modifica cada una de estas k configuraciones de acuerdo con una de las k diferentes secuencias de movimientos que M_N puede realizar partiendo de su configuración actual.

4. M_D devuelve la configuración actual marcada, borra la marca y mueve la marca a la siguiente configuración situada a la derecha. El ciclo se repite entonces desde el paso (1).

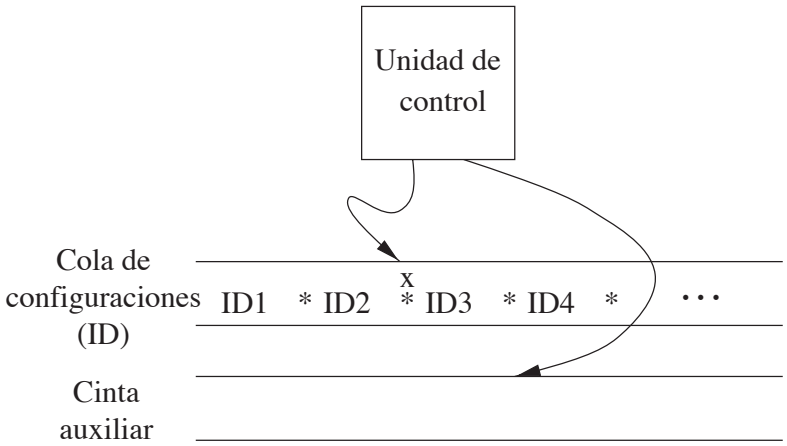

Figura 8.18. Simulación de una MTN mediante una MTD.

Debería estar claro que la simulación es precisa en el sentido de que M_D sólo aceptará si comprueba que M_N puede entrar en una configuración de aceptación. Sin embargo, tenemos que confirmar que si M_N entra en una configuración de aceptación después de una secuencia de n de sus propios movimientos, entonces dicha configuración llegará finalmente a ser la configuración actual marcada por M_D que también aceptará.

Supongamos que m es el número máximo de movimientos que M_N tiene en cualquiera de sus configuraciones. Entonces existe una configuración inicial de M_N, un máximo de m configuraciones que M_N podría alcanzar después de un movimiento, un máximo de m^2 configuraciones que M_N podría alcanzar después de dos movimientos, y así sucesivamente. Por tanto, después de n movimientos, M_N puede alcanzar como máximo $1+m+m^2+\cdots+m^n$ configuraciones. Este número corresponde, como máximo, a nm^n configuraciones.

El orden en que M_D explora las configuraciones de M_N es "por anchura"; es decir, explora todas las configuraciones alcanzables mediante cero movimientos (es decir, la configuración inicial), luego todas las configuraciones alcanzables mediante un movimiento, después aquellas que son alcanzables mediante dos movimientos, etc. En particular, M_D procesará como configuración actual toda configuración que se pueda alcanzar después de hasta n movimientos antes de considerar cualquier otra configuración que sólo sea alcanzable con más de n movimientos.

En consecuencia, M_D considerará la configuración de aceptación de M_N de entre las nm^n primeras configuraciones. Sólo debemos preocuparnos de que M_D considere esta configuración en un tiempo finito y este límite sea suficiente para asegurarnos de que finalmente se accederá a la configuración de aceptación. Por tanto, si M_N acepta, entonces M_D también lo hará. Dado que ya hemos comprobado que M_D aceptará sólo cuando M_N acepte, podemos concluir que $L(M_N) = L(M_D)$. □

Fíjese en que la MT determinista construida puede emplear un tiempo exponencialmente mayor que la MT no determinista. No se sabe si esta ralentización exponencial es o no necesaria. De hecho, el Capítulo 10 está dedicado a este tema y a las consecuencias que tendría que alguien descubriese una forma mejor de simular de forma determinista una MTN.

8.4.5 Ejercicios de la Sección 8.4

Ejercicio 8.4.1. De manera informal pero clara describa las máquinas de Turing de varias cintas que aceptan cada uno de los lenguajes del Ejercicio 8.2.2. Intente que cada una de las máquinas de Turing descritas opere en un tiempo proporcional a la longitud de la entrada.

Ejercicio 8.4.2. He aquí la función de transición de una MT no determinista $M = (\{q_0,q_1,q_2\},\{0,1\},\{0,1,B\},\delta,q_0,B,\{q_2\})$:

δ	0	1	B
q_0	$\{(q_0,1,R)\}$	$\{(q_1,0,R)\}$	$\emptyset$
q_1	$\{(q_1,0,R),\ (q_0,0,L)\}$	$\{(q_1,1,R),\ (q_0,1,L)\}$	$\{(q_2,B,R)\}$
q_2	$\emptyset$	$\emptyset$	$\emptyset$

Determine la configuración alcanzable a partir de la configuración inicial si la entrada es:

* a) 01.

b) 011.

! **Ejercicio 8.4.3.** De manera informal pero clara describa las máquinas de Turing no deterministas (de varias cintas si lo prefiere) que aceptan los siguientes lenguajes. Intente aprovechar el no determinismo para evitar la iteración y ahorrar tiempo en el sentido no determinista. Es decir, es preferible que la MTN tenga muchas ramas, siempre que cada una de ellas sea corta.

* a) El lenguaje de todas las cadenas de ceros y unos que tengan alguna cadena de longitud 100 que se repita, no necesariamente de forma consecutiva. Formalmente, este lenguaje es el conjunto de cadenas de ceros y unos de la forma $wxyxz$, donde $|x| = 100$ y w, y y z tienen una longitud arbitraria.

b) El lenguaje de todas las cadenas de la forma $w_1\#w_2\#\cdots\#w_n$, para cualquier n, tal que cada w_i sea una cadena de ceros y unos, y para cierto j, w_j es la representación binaria del entero j.

c) El lenguaje de todas las cadenas de la misma forma que en el apartado (b), pero para al menos dos valores de j, tenemos que w_j es igual a la representación binaria de j.

! **Ejercicio 8.4.4.** Considere la máquina de Turing no determinista

$$M = (\{q_0, q_1, q_2, q_f\}, \{0,1\}, \{0,1,B\}, \delta, q_0, B, \{q_f\})$$

De manera informal pero clara describa el lenguaje $L(M)$ si δ consta de los siguientes conjuntos de reglas: $\delta(q_0,0) = \{(q_0,1,R),\ (q_1,1,R)\}$; $\delta(q_1,1) = \{(q_2,0,L)\}$; $\delta(q_2,1) = \{(q_0,1,R)\}$; $\delta(q_1,B) = \{(q_f,B,R)\}$.

* **Ejercicio 8.4.5.** Considere una MT no determinista cuya cinta sea infinita en ambos sentidos. En un determinado instante, la cinta está completamente en blanco excepto por una casilla en la que se almacena el símbolo \$. La cabeza de la cinta se encuentra actualmente señalando a una casilla en blanco y el estado es q.

a) Escriba las transiciones que permitirán a la MTN entrar en el estado p, cuando la cabeza de la cinta apunte a \$.

! b) Suponga que la MT fuera determinista. ¿Cómo la configuraría para que encontrara el símbolo \$ y pasara al estado p?

Ejercicio 8.4.6. Diseñe una MT con dos cintas que acepte el lenguaje de todas las cadenas que tienen el mismo número de ceros que de unos. La primera cinta contiene la entrada y se explora de izquierda a derecha. La segunda cinta se emplea para almacenar el exceso de ceros respecto de unos, o viceversa, que existe en la parte de la entrada examinada hasta el momento. Especifique los estados, las transiciones y el propósito de cada estado.

Ejercicio 8.4.7. En este ejercicio implementaremos una pila utilizando una MT especial de tres cintas.

1. La primera cinta se utilizará sólo para almacenar y leer la entrada. El alfabeto de entrada consta del símbolo $\uparrow$, que interpretaremos como "extraer de la pila" y los símbolos a y b, que se interpretan como "introducir una a (una b) en la pila".

2. La segunda cinta se emplea para almacenar la pila.

3. La tercera cinta es la cinta de salida. Cada vez que se extrae un símbolo de la pila, éste tiene que escribirse en la cinta de salida después de todos los símbolos que anteriormente se hayan escrito.

Se requiere que la máquina de Turing se inicie con una pila vacía e implemente la secuencia de operaciones de inserción y extracción de la pila, tal y como especifique la entrada, leyendo de izquierda a derecha. Si la entrada hace que la MT intente hacer una extracción de la pila y ésta está vacía, entonces tiene que pararse en un estado especial que indique error q_e. Si la entrada completa hace que al terminar la pila esté vacía, entonces la entrada se acepta y se pasa al estado final q_f. Describa de manera informal y clara la función de transición de la MT. Proporcione también un resumen del propósito de cada uno de los estados que utilice.

Ejercicio 8.4.8. En la Figura 8.17 se ha mostrado un ejemplo de la simulación general de una MT de k cintas mediante un MT de una sola cinta.

* a) Suponga que esta técnica se emplea para simular una MT de cinco cintas con un alfabeto de cinta de siete símbolos. ¿Cuántos símbolos de cinta debería tener la MT de una sola cinta?

* b) Una forma alternativa de simular las k cintas mediante una sola sería utilizar la $(k+1)$-ésima pista para almacenar las posiciones de la cabeza de las k cintas, mientras que las k primeras pistas simulan las k cintas de la manera habitual. Observe que en la pista $(k+1)$-ésima, hay que diferenciar entre las cabezas de cinta y permitir la posibilidad de que dos o más cabezas estén señalando a la misma casilla. ¿Reduce este método el número de símbolos de cinta necesario de la MT de una sola cinta?

c) Otra forma de simular k cintas mediante una sola sería evitando almacenar las posiciones de las cabezas juntas. Para ello, la pista $(k+1)$-ésima se emplea sólo para marcar una casilla de la cinta. En todo momento, cada cinta simulada se coloca sobre su pista de modo que la cabeza señale la casilla marcada. Si la MT de k cintas mueve la cabeza de la cinta i, entonces la simulación de la MT de una sola cinta desplaza todo el contenido que no son espacios en blanco de la pista i-ésima una casilla en el otro sentido, de manera que la casilla marcada continúe marcando la casilla señalada por la cabeza de la cinta i-ésima de la MT de k cintas. ¿Ayuda este método a reducir el número de símbolos de cinta de la MT de una sola cinta? ¿Presenta algún inconveniente si se compara con los restantes métodos estudiados?

! **Ejercicio 8.4.9.** Una máquina de Turing de *k-cabezas* tiene k cabezas para leer las casillas de una sola cinta. Un movimiento de esta MT depende del estado y del símbolo señalado por cada una de las cabezas. En un movimiento, la MT puede cambiar el estado, escribir un nuevo símbolo en la casilla señalada por cada una de las cabezas y mover cada una de las cabezas hacia la izquierda, la derecha o dejarla estacionaria. Puesto que varias cabezas pueden señalar a la misma casilla, suponemos que éstas están numeradas desde 1 hasta k y que el símbolo que finalmente aparecerá en dicha casilla será el escrito por la cabeza con el número más alto. Demuestre que los lenguajes aceptados por las máquinas de Turing de k cabezas son los mismos que los que aceptan las MT ordinarias.

!! **Ejercicio 8.4.10.** Una máquina de Turing *bidimensional* tiene la unidad de control usual, pero una cinta que es una rejilla de casillas bidimensional, infinita en todas las direcciones. La entrada se coloca en una de las filas de la rejilla con la cabeza señalando a su extremo izquierdo y la unidad de control en el estado inicial, como es habitual. También, como es usual, la aceptación se consigue cuando se entra en un estado final. Demuestre que los lenguajes aceptados por las máquinas de Turing bidimensionales son los mismos que los aceptados por las MT ordinarias.

8.5 Máquinas de Turing restringidas

Hasta aquí hemos visto generalizaciones de la máquina de Turing que no le añaden potencia en lo que respecta al reconocimiento de lenguajes. Ahora, vamos a considerar algunos ejemplos de aparentes restricciones sobre la MT que proporcionan exactamente la misma potencia en lo que se refiere al reconocimiento de lenguajes. La primera restricción que vamos a ver es poco importante pero resulta útil en una serie de construcciones que veremos más adelante: reemplazamos la cinta de la MT que es infinita en ambos sentidos por una cinta que es infinita sólo hacia la derecha. También prohibimos a esta MT restringida que sustituya símbolos de la cinta por espacios en blanco. Estas restricciones permiten suponer que las configuraciones constan de sólo símbolos distintos del espacio en blanco y que siempre comienzan por el extremo izquierdo de la entrada.

Exploramos a continuación determinadas clases de máquinas de Turing de varias cintas que se comportan como autómatas a pila generalizados. En primer lugar, restringimos las cintas de la MT para que se comporten como pilas. A continuación, restringimos para que las cintas se comporten como "contadores"; es decir, sólo pueden representar un entero y la MT sólo puede distinguir si un contador tiene o no el valor cero. La importancia de esto es que existen varios tipos muy sencillos de autómatas que poseen toda la potencia de cualquier computadora. Además, los problemas de indecibilidad de las máquinas de Turing, que hemos visto en el Capítulo 9, también se aplican a estas máquinas sencillas.

8.5.1 Máquinas de Turing con cintas semi-infinitas

Aunque la cabeza de una máquina de Turing puede moverse hacia la izquierda o la derecha respecto de su posición inicial, en este caso basta con que la cabeza de la MT pueda moverse a las posiciones situadas a la derecha de la posición inicial de la cabeza. De hecho, podemos suponer que la cinta es *semi-infinita*, es decir, no existe ninguna casilla a la izquierda de la posición inicial de la cabeza. En el siguiente teorema, proporcionamos una construcción que demuestra que una MT con una cinta semi-infinita puede simular una MT cuya cinta es infinita en ambas direcciones, al igual que en el modelo de MT original.

Esta construcción se basa en el uso de dos pistas en la cinta semi-infinita. La pista superior representa las casillas de la MT original situadas en o a la derecha de la posición inicial de la cabeza. La pista inferior representa las posiciones a la izquierda de la posición inicial, pero en orden inverso. En la Figura 8.19 se muestra la disposición exacta. La pista superior representa las casillas $X_0, X_1, \ldots$, donde X_0 es la posición inicial de la cabeza; X_1, X_2, etc., son las casillas situadas a su derecha. Las casillas X_{-1}, X_{-2}, etc., representan las casillas situadas a la izquierda de la posición inicial. Fíjese en el símbolo $*$ almacenado en la casilla más a la izquierda de la pista inferior. Este símbolo sirve como marcador de final e impide que la cabeza de la MT semi-infinita se salga accidentalmente del extremo izquierdo de la cinta.

Vamos a aplicar una restricción más a nuestra máquina de Turing: no puede escribir nunca un espacio en blanco. Esta sencilla restricción, junto con la restricción de que la cinta sea semi-infinita, implica que la cinta en todo momento contendrá un prefijo de símbolos distintos del espacio en blanco seguido de una cantidad infinita de espacios en blanco. Además, la secuencia de símbolos distintos del espacio en blanco siempre comienza en la posición inicial de la cinta. En el Teorema 9.19, y también en el Teorema 10.9, veremos lo útil que resulta suponer que las configuraciones o descripciones instantáneas tienen esta forma.

TEOREMA 8.12

Todo lenguaje aceptado por una MT M_2 también es aceptado por una MT M_1 con las siguientes restricciones:

1. La cabeza de M_1 nunca se mueve hacia la izquierda de su posición inicial.
2. M_1 nunca escribe un espacio en blanco.

DEMOSTRACIÓN. La condición (2) es bastante sencilla. Consiste en crear un nuevo símbolo de cinta B' que se comporte como un espacio en blanco, pero no es el espacio en blanco B. Es decir:

a) Si M_2 tiene una regla $\delta_2(q,X) = (p,B,D)$, esta regla se cambia por $\delta_2(q,X) = (p,B',D)$.

b) Luego, $\delta_2(q,B')$ se hace igual a $\delta_2(q,B)$, para todo estado q.

La condición (1) requiere algo más de trabajo. Sea

$$M_2 = (Q_2, \Sigma, \Gamma_2, \delta_2, q_2, B, F_2)$$

la MT M_2 que incluye las modificaciones anteriores, de modo que nunca escribe espacios en blanco B. Construimos:

$$M_1 = (Q_1, \Sigma \times \{B\}, \Gamma_1, \delta_1, q_0, [B,B], F_1)$$

donde:

X_0	X_1	X_2	$\ldots$
$*$	X_{-1}	X_{-2}	$\ldots$

Figura 8.19. Una cinta semi-infinita puede simular una cinta infinita en ambos sentidos.

Q_1 Los estados de M_1 son $\{q_0, q_1\} \cup \big(Q_2 \times \{U, L\}\big)$. Es decir, los estados de M_1 son el estado inicial q_0, otro estado q_1 y todos los estados de M_2 con una segunda componente de datos cuyo valor puede ser U (*upper*, superior) o L (*lower*, inferior). Esta segunda componente nos dice si M_2 está explorando la pista superior o inferior, como se muestra en la Figura 8.19. Dicho de otra manera, U indica que la cabeza de M_2 se encuentra en la posición inicial o en una posición a la derecha de la misma, y L indica si se encuentra a la izquierda de dicha posición.

Γ_1 Los símbolos de cinta de M_1 son todos los pares de símbolos de Γ_2, es decir, $\Gamma_2 \times \Gamma_2$. Los símbolos de entrada de M_1 son aquellos pares con un símbolo de entrada de M_2 como primera componente y un espacio en blanco en la segunda componente, es decir, pares de la forma $[a, B]$, donde a pertenece a Σ. El espacio en blanco de M_1 contiene espacios en blanco en ambos componentes. Adicionalmente, para cada símbolo X de Γ_2, existe un par $[X, *]$ de Γ_1. Aquí, $*$ es un nuevo símbolo que no pertenece a Γ_2, y sirve para marcar el extremo izquierdo de la cinta de M_1.

δ_1 Las transiciones de M_1 son las siguientes:

1. $\delta_1(q_0, [a, B]) = (q_1, [a, *], R)$, para cualquier a de Σ. El primer movimiento de M_1 coloca el marcador * en la pista inferior de la casilla más a la izquierda. El estado pasa a ser q_1 y la cabeza se mueve hacia la derecha, porque no puede moverse hacia la izquierda o quedar estacionaria.
2. $\delta_1(q_1, [X, B]) = ([q_2, U], [X, B], L)$, para cualquier X de Γ_2. En el estado q_1, M_1 establece las condiciones iniciales de M_2, devolviendo la cabeza a su posición inicial y cambiando el estado a $[q_2, U]$, es decir, al estado inicial de M_2 y señalando a la pista superior de M_1.
3. Si $\delta_2(q, X) = (p, Y, D)$, entonces para todo Z de Γ_2:
   a) $\delta_1([q, U], [X, Z]) = ([p, U], [Y, Z], D)$ y
   b) $\delta_1([q, L], [Z, X]) = ([p, L], [Z, Y], \overline{D})$,

   donde $\overline{D}$ es el sentido opuesto a D, es decir, L si $D = R$ y R si $D = L$. Si M_1 no está en la casilla más a la izquierda, entonces simula M_2 en la pista apropiada (la pista superior si la segunda componente del estado es U y la pista inferior si la segunda componente es L). Observe sin embargo que cuando trabaja sobre la pista inferior, M_1 se mueve en el sentido opuesto al que se mueve M_2. Esto resulta lógico ya que el contenido de la mitad izquierda de la cinta de M_2 se ha colocado en orden inverso a lo largo de la pista inferior de la cinta de M_1.
4. Si $\delta_2(q, X) = (p, Y, R)$, entonces:

   $$\delta_1([q, L], [X, *]) = \delta_1([q, U], [X, *]) = ([p, U], [Y, *], R)$$

   Esta regla cubre uno de los casos de cómo se emplea el marcador final izquierdo *. Si M_2 se mueve hacia la derecha a partir de su posición inicial, entonces independientemente de si anteriormente había estado a la izquierda o a la derecha de dicha posición (como refleja el hecho de que la segunda componente del estado de M_1 puede ser L o U), M_1 tiene que moverse hacia la derecha y apuntar a la pista superior. Es decir, M_1 continuará estando en la posición representada por X_1 en la Figura 8.19.
5. Si $\delta_2(q, X) = (p, Y, L)$, entonces:

   $$\delta_1([q, L], [X, *]) = \delta_1([q, U], [X, *]) = ([p, L], [Y, *], R)$$

   Esta regla es similar a la anterior, pero cubre el caso en que M_2 se mueve hacia la izquierda de su posición inicial. M_1 tiene que moverse hacia la derecha de su marcador final, pero ahora apuntando a la pista inferior; es decir, la casilla indicada mediante X_{-1} en la Figura 8.19.

F_1 Los estados de aceptación de F_1 son aquellos estados de $F_2 \times \{U, L\}$, es decir, todos los estados de M_1 cuya primera componente es un estado de aceptación de M_2. En el momento de la aceptación, M_1 puede estar apuntando a la pista superior o a la inferior.

Ahora, la demostración del teorema está completa. Podemos observar por inducción sobre el número de movimientos realizados por M_2 que M_1 reproducirá las configuraciones de M_2 sobre su propia cinta. Para ello, basta con tomar la pista inferior, invertirla y concatenarla con la pista superior. Observemos también que M_1 pasa a uno de sus estados de aceptación exactamente cuando lo hace M_2. Por tanto, $L(M_1) = L(M_2)$. □

8.5.2 Máquinas con varias pilas

Ahora vamos a ocuparnos de varios modelos de computación que están basados en generalizaciones del autómata a pila. En primer lugar, consideremos lo que ocurre cuando se le proporcionan al autómata a pila varias pilas. Del Ejemplo 8.7, ya sabemos que una máquina de Turing puede aceptar lenguajes que no son aceptados por algunos autómatas de una sola pila. Resulta que si proporcionamos al autómata dos pilas, entonces puede aceptar cualquier lenguaje que pueda aceptar una MT.

A continuación vamos a considerar una clase de máquinas conocidas como "máquinas contadoras". Estas máquinas sólo tienen la capacidad de almacenar un número finito de enteros ("contadores"), y realizan diferentes movimientos dependiendo de si alguno de los contandores actualmente tiene el valor 0. La máquina contadora sólo puede sumar o restar uno al contador y no puede distinguir entre dos valores distintos de cero. En efecto, un contador es como una pila en la que podemos colocar sólo dos símbolos: un marcador del fondo de pila que sólo aparece en la parte inferior y otro símbolo que puede introducirse o extraerse de la pila.

No vamos a llevar a cabo un tratamiento formal de la máquina de varias pilas, aunque la idea se sugiere en la Figura 8.20. Una máquina de k pilas es un autómata a pila determinista con k pilas. Obtiene su entrada, al igual que lo hace un autómata a pila, de una fuente de entrada, en lugar de tenerla colocada sobre una cinta o en una pila, como es el caso de una MT. La máquina multipila dispone de una unidad de control, que se encuentra en uno de los estados de su conjunto finito de estados. Tiene un alfabeto de pila finito, que utiliza para todas sus pilas. Un movimiento de la máquina de varias pilas está basado en:

1. El estado de la unidad de control.
2. El símbolo de entrada leído, el cual se elige del alfabeto de entrada finito. Alternativamente, la máquina de varias pilas puede realizar un movimiento utilizando la entrada ε, pero para ser una máquina determinista, no se puede permitir que en alguna situación pueda realizar un movimiento con la entrada ε y a la vez con una entrada distinta de ε.

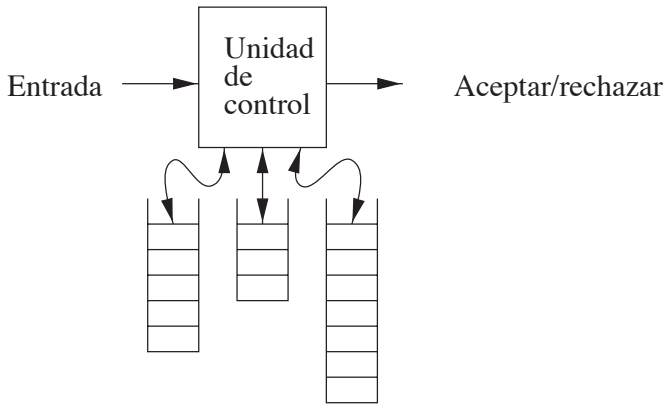

Figura 8.20. Una máquina con tres pilas.

3. El símbolo superior de la pila en cada una de sus pilas.

En un movimiento, una máquina de varias pilas puede:

a) Cambiar a un nuevo estado.

b) Reemplazar el símbolo superior de cada pila por una cadena de ceros o más símbolos de pila. Puede existir (y normalmente existe) una cadena de sustitución diferente para cada pila.

Por tanto, una regla de transición típica para una máquina de k pilas será similar a:

$$\delta(q,a,X_1,X_2,\ldots,X_k) = (p,\gamma_1,\gamma_2,\ldots,\gamma_k)$$

La interpretación de esta regla es que en el estado q, estando X_i en la cima de la pila i, para $i = 1,2,\ldots,k$, la máquina puede consumir a (que es un símbolo de entrada o ε) de su entrada, pasar al estado p y reemplazar el símbolo X_i de la cima de la pila i por la cadena γ_i, para $i = 1,2,\ldots,k$. La máquina de varias pilas acepta al alcanzar un estado final.

Añadamos ahora una capacidad a esta máquina determinista que simplifique el procesamiento de entrada: supongamos que existe un símbolo especial \$, denominado *marcador de final*, que sólo aparece al final de la entrada y que no forma parte de la misma. La presencia de este marcador de final nos permite saber cuándo hemos consumido toda la entrada disponible. Veremos en el siguiente teorema cómo el marcador de final facilita a la máquina de varias pilas simular una máquina de Turing. Observe que la MT convencional no necesita ningún marcador de final especial, ya que el primer espacio en blanco sirve para indicar el final de la entrada.

TEOREMA 8.13

Si un lenguaje L es aceptado por una máquina de Turing, entonces L es aceptado por una máquina de dos pilas.

DEMOSTRACIÓN. La idea principal es que dos pilas pueden simular la cinta de una máquina de Turing, almacenando en una pila lo que está a la izquierda de la cabeza y en la otra pila lo que se encuentra a la derecha de la cabeza, excepto para el caso de cadenas de espacios en blanco infinitas situadas más a la izquierda y más a la derecha de los caracteres no blancos. Más detalladamente, sea L el lenguaje $L(M)$ para una MT (de una cinta) M. La máquina de dos pilas S hará lo siguiente:

1. S comienza con un *marcador de fondo de pila* en cada pila. Este marcador puede ser el símbolo inicial de las pilas y no tiene que aparecer en ninguna otra posición de las mismas. De aquí en adelante, diremos que una "pila está vacía" cuando sólo contenga el marcador de fondo de pila.

2. Supongamos que w\$ es la entrada de S. S copia w en su primera pila, y deja de copiar cuando lee el marcador de final en la entrada.

3. S extrae cada símbolo por turno de la primera pila y lo introduce en la segunda. Así, la primera pila estará vacía y la segunda almacenará w, estando el extremo izquierdo de w en la cima.

4. S pasa al estado inicial (simulado) de M. Tiene una primera pila vacía, lo que representa el hecho de que M no contiene nada más que espacios en blanco a la izquierda de la casilla señalada por la cabeza de la cinta. S tiene una segunda pila que almacena w, lo que representa el hecho de que w aparece en la casilla señalada por la cabeza de la cinta de M y las casillas situadas a su derecha.

5. S simula un movimiento de M como sigue.

   a) S conoce el estado de M, supongamos que es q, porque S simula el estado de M en su propia unidad de control.

b) S sabe cuál es el símbolo X señalado por la cabeza de la cinta de M; es el símbolo colocado en la cima de la segunda de pila de S. Excepcionalmente, si la segunda pila sólo contiene el marcador de fondo de pila, quiere decir que M acaba de llegar a un espacio en blanco; S interpreta el símbolo señalado por M como el espacio en blanco.

c) Por tanto, S conoce el siguiente movimiento de M.

d) El siguiente estado de M se registra en un componente de la unidad de control de S, en el lugar del estado anterior.

e) Si M reemplaza X por Y y se mueve hacia la derecha, entonces S introduce Y en su primera pila, representando así el hecho de que ahora Y está a la izquierda de la cabeza de M. X se extrae de la segunda pila de S. Sin embargo, existen dos excepciones:

1) Si la segunda pila sólo tiene un marcador de fondo de pila (y, por tanto, X es el espacio en blanco), entonces la segunda pila no se modifica, ya que quiere decir que M se ha movido un espacio en blanco más hacia la derecha.
2) Si Y es un espacio en blanco y la primera pila está vacía, entonces dicha pila permanece vacía. La razón de ello es que sólo quedan blancos a la izquierda de la cabeza de M.

f) Si M reemplaza a X por Y y se mueve hacia la izquierda, S extrae el símbolo situado en la cima de la primera pila, por ejemplo, Z, y a continuación reemplaza X por ZY en la segunda pila. Este cambio refleja el hecho de que lo que se encontraba en una posición inmediatamente a la izquierda de la cabeza de la cinta ahora es lo que señala la cabeza. Excepcionalmente, si Z es el marcador de fondo de pila, entonces M debe introducir BY en la segunda pila y no extraer ningún símbolo de la primera.

6. S acepta si el nuevo estado de M es un estado de aceptación. En cualquier otro caso, S simula otro movimiento de M de la misma forma. □

8.5.3 Máquinas contadoras

Una *máquina contadora* puede verse de dos maneras:

1. La máquina contadora tiene la misma estructura que la máquina de varias pilas (Figura 8.20), pero cada pila ahora es un contador. Los contadores almacenan cualquier entero no negativo, pero sólo podemos distinguir entre contadores que están a cero y contadores que no están a cero. Es decir, el movimiento de la máquina contadora depende de su estado, el símbolo de entrada y de los contadores (si existen) que estén a cero. En un movimiento, la máquina contadora puede:

   a) Cambiar de estado.

   b) Sumar o restar 1 de cualquiera de sus contadores de manera independiente. Sin embargo, no está permitido que un contador tome un valor negativo, por lo que no puede restar 1 de un contador que esté a 0.

2. Una máquina contadora también puede verse como una máquina de varias pilas restringida. Las restricciones son las siguientes:

   a) Sólo existen dos símbolos de pila, a los que haremos referencia como Z_0 (el *marcador de fondo de pila*) y X.

   b) Inicialmente, Z_0 está en cada pila.

   c) Sólo podemos reemplazar Z_0 por una cadena de la forma X^iZ_0, para $i \geq 0$.

d) Sólo podemos reemplazar X por X^i para $i \geq 0$. Es decir, Z_0 sólo aparece en el fondo de cada una de las pilas y los restantes símbolos de pila, si existen, son X.

Utilizaremos la definición (1) para las máquinas contadoras, pero ambas definiciones dan lugar claramente a máquinas de potencia equivalente. La razón de ello es que la pila X^iZ_0 puede identificarse con el contador i. En la definición (2), podemos diferenciar un contador que esté a 0 de otros contadores, ya que el contador que está a cero, tendrá Z_0 en la cima de la pila, y en cualquier otro caso, estaría el símbolo X. Sin embargo, no podemos distinguir dos contadores positivos, ya que ambos tendrán X en la cima de la pila.

8.5.4 La potencia de las máquinas contadoras

Hay varias observaciones acerca de los lenguajes aceptados por las máquinas contadoras que aunque resultan evidentes merece la pena mencionar:

- Todo lenguaje aceptado por una máquina contadora es recursivamente enumerable. La razón de ello es que una máquina contadora es un caso especial de una máquina de pila y una máquina de pila es un caso especial de una máquina de Turing de varias cintas, que acepta sólo lenguajes recursivamente enumerables de acuerdo con el Teorema 8.9.

- Todo lenguaje aceptado por una máquina con un contador es un LIC. Fíjese en que un contador, de acuerdo con la definición (2), es una pila, por lo que una máquina de un contador es un caso especial de una máquina de una pila; es decir, un autómata a pila. De hecho, los lenguajes de las máquinas de un contador son aceptados por los autómatas a pila deterministas, aunque la demostración de esto es sorprendentemente compleja. La dificultad de la demostración resulta del hecho de que las máquinas contadoras y de varias cintas tienen un marcador de final \$ al final de su entrada. Una autómata a pila no determinista puede suponer que ha visto el último símbolo de entrada al ver el marcador \$; por tanto, es evidente que un autómata a pila no determinista sin el marcador de final puede simular un APD con marcador de final. Sin embargo, la demostración complicada que no vamos a abordar, es la que consiste en demostrar que un APD sin marcador de final puede simular un APD que *sí* lo tenga.

El resultado sorprendente acerca de las máquinas contadoras es que dos contadores son suficientes para simular una máquina de Turing y, por tanto, para aceptar todos los lenguajes recursivamente enumerables. Ahora nos vamos a centrar en este resultado y vamos a demostrar, primero, que tres contadores son suficientes para conseguir lo anterior y luego simularemos los tres contadores mediante dos contadores.

TEOREMA 8.14

Todo lenguaje recursivamente enumerable es aceptado por una máquina de tres contadores.

DEMOSTRACIÓN. Partimos del Teorema 8.13, que establece que todo lenguaje recursivamente enumerable es aceptado por una máquina de dos pilas. Necesitamos entonces ver cómo simular una pila mediante contadores. Supongamos que la máquina de pila utiliza $r-1$ símbolos de cinta. Podemos identificar los símbolos con los dígitos de 1 hasta $r-1$, e interpretar el contenido de la pila $X_1X_2\cdots X_n$ como enteros en base r. Es decir, esta pila (cuya cima se encuentra en el extremo izquierdo, como es habitual) se representa mediante el entero $X_nr^{n-1}+X_{n-1}r^{n-2}+\cdots+X_2r+X_1$.

Utilizamos dos contadores para almacenar los enteros que representan a cada una de las dos pilas. El tercer contador se utiliza para ajustar los otros dos contadores. En particular, necesitamos el tercer contador cuando bien dividimos o multiplicamos por r.

Las operaciones sobre una pila pueden clasificarse en tres categorías: extracción del símbolo de la cima de la pila, cambio del símbolo de la cima e introducción de un símbolo en la pila. Un movimiento de la máquina

de dos pilas puede implicar varias de estas operaciones; en concreto, reemplazar el símbolo de la cima de la pila X por una cadena de símbolos precisa reemplazar X y luego introducir símbolos adicionales en la pila. Estas operaciones sobre una pila representada por un contador i se realizan de la forma siguiente. Observe que es posible utilizar la unidad de control de la máquina de varias pilas para realizar cada una de las operaciones necesarias para contar hasta r o un valor menor.

1. Para extraer un símbolo de la pila tenemos que reemplazar i por i/r, despreciando el resto, que es X_1. Teniendo inicialmente el tercer contador el valor 0, decrementamos repetidamente el contador i en r e incrementamos el tercer contador en 1. Cuando el contador que originalmente almacena i alcanza el valor 0, nos detenemos. A continuación, aumentamos repetidamente el contador original en 1 y disminuimos el tercer contador en 1, hasta que este último alcanza de nuevo el valor 0. En esta situación, el contador que hemos empleado para almacenar i almacena i/r.

2. Para cambiar X por Y en la cima de una pila que está representada por el contador i, incrementamos o decrementamos i en una cantidad pequeña, que seguro no será mayor que r. Si $Y > X$, siendo dígitos, i se incrementa en $Y - X$; si $Y < X$, entonces i se decrementa en $X - Y$.

3. Para introducir X en una pila que inicialmente almacena i, tenemos que reemplazar i por $ir + X$. En primer lugar multiplicamos por r. Para ello, decrementamos repetidamente el contador i en una unidad e incrementamos el tercer contador (que, como siempre, inicialmente tiene el valor 0) en r. Cuando el contador original alcanza el valor 0, tendremos ir en el tercer contador. Se copia entonces el tercer contador en el contador original y de nuevo el tercer contador se pone a 0, como se ha hecho en el punto (1). Por último, el contador original se incrementa en X.

Para completar la construcción, tenemos que inicializar los contadores con el fin de simular las pilas en sus condiciones iniciales: almacenando sólo el símbolo inicial de la máquina de dos pilas. Este paso se lleva a cabo incrementando los dos contadores en un entero pequeño, comprendido entre 1 y $r - 1$, que será el que se corresponde con el símbolo inicial. □

TEOREMA 8.15

Todo lenguaje recursivamente enumerable es aceptado por una máquina de dos contadores.

DEMOSTRACIÓN. Teniendo en cuenta el teorema anterior, sólo tenemos que demostrar cómo simular tres contadores mediante dos contadores. La idea consiste en representar los tres contadores, por ejemplo, i, j y k, mediante un único entero. El entero que seleccionamos es $m = 2^i 3^j 5^k$. Un contador almacenará este número, mientras que el otro se utiliza para multiplicar o dividir m por uno de los tres primeros números primos: 2, 3 y 5. Para simular la máquina de tres contadores, tenemos que realizar las siguientes operaciones:

1. Se incrementa i, j y/o k. Para incrementar i en 1, multiplicamos m por 2. Ya hemos visto en la demostración del Teorema 8.14 cómo multiplicar un contador por cualquier constante r utilizando un segundo contador. Del mismo modo, incrementamos j multiplicando m por 3, e incrementamos k multiplicando m por 5.

2. Se establece qué contador de entre i, j y k está a 0, si alguno de ellos lo está. Para establecer si $i = 0$, tenemos que determinar si m es divisible por 2. Se copia m en el segundo contador, utilizando el estado de la máquina contadora para recordar si hemos decrementado m un número par o impar de veces. Si hemos decrementado m un número impar de veces cuando ha llegado a 0, entonces $i = 0$. A continuación, restauramos m copiando el segundo contador en el primero. De forma similar, probamos si $j = 0$ determinando si m es divisible por 3 y si $k = 0$ determinando si m es divisible por 5.

Selección de constantes en la simulación de tres contadores mediante dos contadores

Observe la importancia en la demostración del Teorema 8.15 de que los números 2, 3 y 5 sean primos distintos. Si por ejemplo hemos elegido $m = 2^i 3^j 4^k$, entonces $m = 12$ podría representar $i = 0$, $j = 1$ y $k = 1$, o $i = 2$, $j = 1$ y $k = 0$. Por tanto, no podríamos establecer qué contador, i o k, está a 0, por lo que no podríamos simular la máquina de tres contadores de manera fiable.

3. Se decrementa i, j y/o k. Para ello, dividimos m entre 2, 3 o 5, respectivamente. La demostración del Teorema 8.14 nos dice cómo realizar dicha división entre cualquier constante utilizando un contador adicional. Dado que una máquina de tres contadores no puede decrementar un contador hasta un valor menor que 0, si m no es divisible equitativamente por la constante por la que estamos dividiendo, se producirá un error y la máquina de dos contadores que estamos simulando se detendrá sin aceptar. □

8.5.5 Ejercicios de la Sección 8.5

Ejercicio 8.5.1. De manera informal pero clara describa máquinas contadoras que acepten los siguientes lenguajes. En cada caso, utilice la menor cantidad posible de contadores, y nunca más de dos.

* a) $\{0^n 1^m \mid n \geq m \geq 1\}$.

b) $\{0^n 1^m \mid 1 \leq m \leq n\}$.

*! c) $\{a^i b^j c^k \mid i = j \text{ o } i = k\}$.

!! d) $\{a^i b^j c^k \mid i = j \text{ o } i = k \text{ or } j = k\}$.

!! **Ejercicio 8.5.2.** El propósito de este ejercicio es demostrar que una máquina de una pila con marcador de final en su entrada no es más potente que un autómata a pila determinista. $L\$$ es la concatenación del lenguaje L con el lenguaje que sólo contiene la cadena \$; es decir, $L\$$ es el conjunto de todas las cadenas $w\$$ tales que w pertenece a L. Demuestre que si $L\$$ es un lenguaje aceptado por un APD, donde \$ es el símbolo marcador de final que no aparece en ninguna cadena de L, entonces L también es aceptado por algún APD. *Consejo*: en realidad, esta pregunta consiste en demostrar que los lenguajes de los APD son cerrados para la operación L/a definida en el Ejercicio 4.2.2. Debe modificar el autómata a pila determinista P para $L\$$ reemplazando cada uno de los símbolos de la pila, X, por los pares (X,S), donde S es un conjunto de estados. Si la pila de P es $X_1X_2\cdots X_n$, entonces la pila del APD construido para L es $(X_1,S_1)(X_2,S_2)\cdots(X_n,S_n)$, donde cada S_i es el conjunto de estados q para el que P aceptará, partiendo de la configuración $(q,a,X_iX_{i+1}\cdots X_n)$.

8.6 Máquinas de Turing y computadoras

Ahora vamos a comparar la máquina de Turing con las computadoras que habitualmente utilizamos. Aunque estos modelos parecen bastante diferentes, pueden aceptar exactamente los mismos lenguajes: los lenguajes recursivamente enumerables. Dado que no hemos definido matemáticamente el concepto de "computadora común", los argumentos aplicados en esta sección son necesariamente informales. Tenemos que recurrir a la intuición en lo que se refiere a lo que las computadoras pueden hacer, especialmente cuando los números empleados exceden los límites normales que permite la arquitectura de estas máquinas (por ejemplo, espacios de direccionamiento de 32 bits). Los razonamientos aplicados en esta sección están basados en que:

1. Una computadora puede simular una máquina de Turing.
2. Una máquina de Turing puede simular una computadora y puede hacerlo en un periodo de tiempo que es, cómo máximo, polinómico en lo que respecta al número de pasos utilizados por la computadora.

8.6.1 Simulación de una máquina de Turing mediante una computadora

Examinemos en primer lugar cómo una computadora puede simular una máquina de Turing. Dada una MT concreta M, debemos escribir un programa que actúe como M. Un aspecto de M es su unidad de control. Dado que sólo existen un número finito de estados y un número finito de reglas de transición, nuestro programa puede codificar los estados como cadenas de caracteres y utilizar una tabla de transiciones, que consultará para determinar cada movimiento. Del mismo modo, los símbolos de cinta pueden codificarse como cadenas de caracteres de longitud fija, ya que el número de estos símbolos es finito.

Al estudiar cómo simulará el programa la cinta de la máquina de Turing surge una cuestión importante. Esta cinta puede crecer en longitud infinitamente, pero la memoria de una computadora (memoria principal, discos y otros dispositivos de almacenamiento) es finita. ¿Podemos simular una cinta infinita con una cantidad fija de memoria?

Si no existe la posibilidad de reeemplazar los dispositivos de almacenamiento, entonces realmente no podremos; la computadora entonces se comportaría como un autómata finito y los únicos lenguajes que aceptaría serían los regulares. Sin embargo, las computadoras comunes disponen de dispositivos de almacenamiento intercambiables, por ejemplo, discos "Zip". De hecho, el típico disco duro es un dispositivo extraible y puede reemplazarse por otro disco idéntico pero vacío.

Dado que no existe un límite evidente en lo que se refiere a cuántos discos podríamos utilizar, suponemos que podemos disponer de tantos discos como necesite la computadora. Podemos por tanto disponer que los discos se coloquen en dos pilas, como se muestra en la Figura 8.21. Una pila almacena los datos de las casillas de la cinta de la máquina de Turing que están situadas a la izquierda de la cabeza de la cinta y la otra pila almacena los datos almacenados a la derecha de la cabeza de la cinta. Cuanto más profundo se encuentre un símbolo en una pila, más alejado de la cabeza de la cinta se encontrará el dato.

Si la cabeza de la cinta de la MT se alejara hacia la izquierda lo suficiente como para llegar a casillas que no están representadas en el disco actualmente montado en la computadora, entonces se imprimirá el mensaje

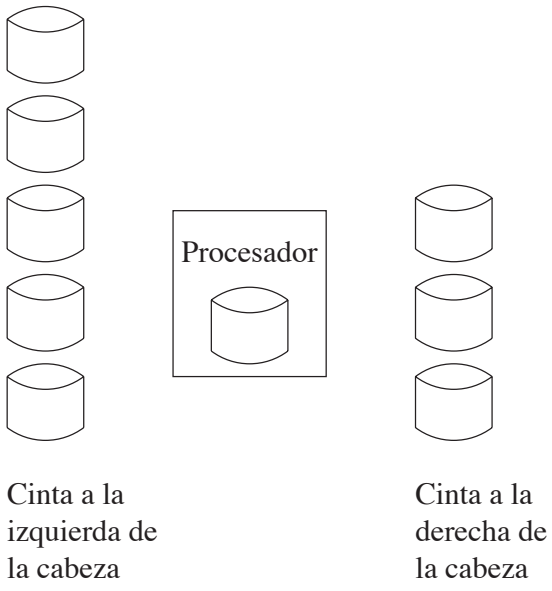

Figura 8.21. Simulación de una máquina de Turing mediante una computadora común.

El problema de los alfabetos de cinta muy grandes

El argumento de la Sección 8.6.1 comienza a ser cuestionable si el número de símbolos de cinta es tan grande que el código para uno de estos símbolos excede el espacio de almacenamiento de disco. Realmente, tendría que haber muchos símbolos de cinta, ya que un disco de 30 gigabytes, por ejemplo, puede representar cualquier símbolo de un alfabeto de $2^{240000000000}$ símbolos. Del mismo modo, el número de estados podría ser tan grande que no sea posible representar el estado utilizando el disco completo.

Una solución es limitar el número de símbolos de cinta que puede utilizar la MT. Siempre podemos codificar en binario un alfabeto de cinta arbitrario. Por tanto, cualquier máquina de Turing M puede ser simulada por otra máquina de Turing M' que sólo utilice los símbolos de cinta 0, 1 y B. Sin embargo, M' necesita muchos estados, ya que para simular un movimiento de M, la máquina M' tiene que explorar su cinta y recordar todos los bits que le indican qué símbolo está explorando la máquina M. De esta forma, los conjuntos de estados serán muy grandes y el PC que simula a la máquina de Turing M' puede tener que montar y desmontar varios discos en el momento de decidir cuál es el estado de M' y cuál debería ser el siguiente movimiento de la misma. Dado que no es de esperar que las computadoras realicen tareas de este tipo, los sistemas operativos típicos no dan soporte a programas de esta clase. Sin embargo, si lo deseáramos, podríamos programar la computadora en bruto y proporcionarle esta capacidad.

Afortunadamente, la simulación de una MT con un número muy grande de estados o símbolos de cinta puede ser más sutil. En la Sección 9.2.3 veremos que es posible diseñar una MT que sea realmente "programable". Esta MT, conocida como "universal", puede leer en su cinta la función de transición de cualquier MT codificada en binario y simular dicha MT. La MT universal tiene un número bastante razonable de estados y símbolos de cinta. Simulando la MT universal, se puede programar una computadora común para aceptar cualquier lenguaje recursivamente enumerable que deseemos, sin tener que recurrir a simular una cantidad de estados que alcance los límites de almacenamiento de un disco.

"cambiar izquierda". En este caso, un operador cambiaría el disco que está montado por otro y lo colocaría encima de la pila de la derecha. El disco situado en la parte superior de la pila de la izquierda se monta en la computadora y se reanuda el proceso.

De manera similar, si la cabeza de la cinta de la MT alcanza casillas suficientemente alejadas por la derecha que no están representadas por el disco montado actualmente, entonces se imprime el mensaje "cambiar derecha". El operador retirará entonces el disco actualmente montado en la parte superior de la pila izquierda y montará el disco en la parte superior de la pila derecha de la computadora. Si cualquiera de las pilas está vacía cuando la computadora solicita que se monte un disco de dicha pila, quiere decir que la MT ha entradao en una zona de todo blancos de la cinta. En este caso, el operador deberá acudir al almacén y adquirir un disco vacío para instalarlo.

8.6.2 Simulación de una computadora mediante un máquina de Turing

También tenemos que considerar la comparación contraria: hay cosas que una computadora común puede hacer que una máquina de Turing no puede. Una cuestión subordinada importante es si la computadora puede hacer determinadas operaciones más rápidamente que una máquina de Turing. En esta sección, afirmamos que una MT puede simular una computadora y en la Sección 8.6.3 veremos que la simulación puede realizarse de forma lo suficientemente rápida como para que los tiempos de ejecución de la computadora y de la MT "sólo" difieran polinómicamente para un problema dado. Debemos recordar de nuevo que existen importantes razones para pensar que las diferencias de orden polinómico en los tiempos de ejecución indican que los tiempos son similares,

mientras que las diferencias exponenciales son "excesivas". En el Capítulo 10 se aborda la teoría que compara los tiempos de ejecución polinómicos y exponenciales.

Para comenzar nuestro estudio acerca de cómo una MT simula una computadora, partimos de un modelo realista aunque informal del funcionamiento de una computadora típica.

a) En primer lugar, suponemos que el almacenamiento de una computadora consta de una secuencia infinitamente larga de *palabras*, cada una con una *dirección* asociada. En una computadora real, las palabras pueden tener 32 o 64 bits de longitud, aunque no vamos a imponer un límite a la longitud de una palabra dada. Supondremos que las direcciones están dadas por los enteros 0, 1, 2, etc. En una computadora real, los bytes individuales se numeran con enteros consecutivos, por lo que las palabras tienen direcciones que son múltiplos de 4 u 8, aunque esta diferencia no es importante. Además, en una computadora real, existiría un límite sobre el número de palabras en "memoria", pero dado que deseamos considerar el contenido de un número arbitrario de discos o de otros dispositivos de almacenamiento, supondremos que no existe ningún límite al número de palabras.

b) Suponemos que el programa de la computadora se almacena en algunas de las palabras de memoria. Cada una de estas palabras representan una instrucción, como en el lenguaje máquina o ensamblador de una computadora típica. Algunos ejemplos son instrucciones que mueven datos de una palabra a otra o que añaden una palabra a otra. Suponemos que el "direccionamiento indirecto" está permitido, por lo que una instrucción puede hacer referencia a otra palabra y utilizar el contenido de la misma como la dirección de la palabra a la que se aplica la operación. Esta capacidad, disponible en todas las computadoras actuales, es necesaria para llevar a cabo accesos a matrices, seguir vínculos de una lista o, en general, realizar operaciones con punteros.

c) Suponemos que cada instrucción implica un número limitado (finito) de palabras y que cada instrucción modifica el valor de como máximo una palabra.

d) Una computadora típica dispone de *registros*, que son palabras de memoria a las que se puede acceder especialmente rápido. A menudo, las operaciones como la suma están restringidas a llevarse a cabo en los registros. No aplicaremos ninguna de estas restricciones, sino que vamos a permitir que se realice cualquier operación sobre cualquier palabra. No tendremos en cuenta la velocidad relativa de las operaciones sobre diferentes palabras, ni será necesario si sólo estamos comparando las capacidades de reconocimiento de lenguajes de las computadoras y las máquinas de Turing. Aunque nos interesen los tiempos de ejecución polinómicos, la velocidades relativas del acceso a distintas palabras no es importante, ya que dichas diferencias son "sólo" un factor constante.

La Figura 8.22 muestra cómo diseñar la máquina de Turing para simular una computadora. Esta MT utiliza varias cintas, pero podría convertirse en una MT de una única cinta utilizando la construcción de la Sección 8.4.1. La primera cinta representa la memoria completa de la computadora. Hemos empleado un código en el que las direcciones de las palabras de memoria, en orden numérico, alternan con el contenido de dichas palabras de memoria. Tanto las direcciones como los contenidos están escritos en binario. Los marcadores `*` y `#` se utilizan para poder localizar más fácilmente el final de las direcciones y los contenidos, y para especificar si una cadena binaria es una dirección o un contenido. Otro marcador, `$`, indica el principio de la secuencia de direcciones y contenidos.

La segunda cinta es el "contador de instrucciones". Esta cinta almacena un entero en binario, que representa una de las posiciones de memoria sobre la cinta 1. El valor almacenado en esta posición se interpreta como la siguiente instrucción de la computadora que hay que ejecutar.

La tercera cinta almacena una "dirección de memoria" o el contenido de dicha dirección después de que ésta haya sido localizada en la cinta 1. Para ejecutar una instrucción, la MT tiene encontrar el contenido de una o más direcciones de memoria que almacenan los datos implicados en el cálculo. Primero se copia la dirección deseada

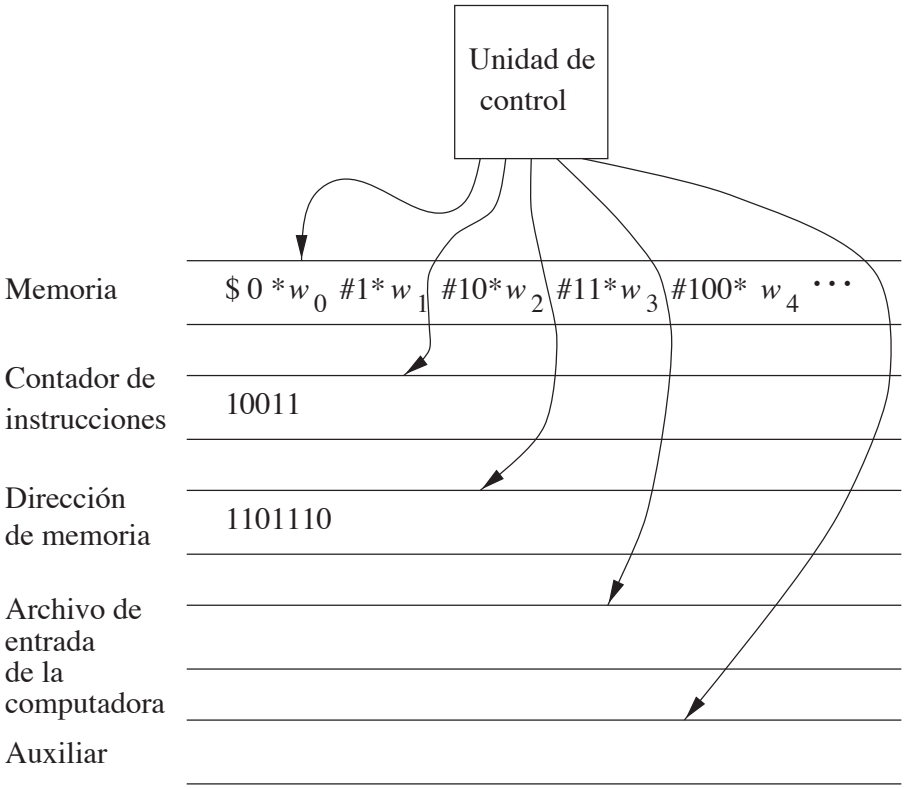

Figura 8.22. Máquina de Turing que simula una computadora típica.

en la cinta 3 y se compara con las direcciones de la cinta 1, hasta encontrarla. El contenido de esta dirección se copia en la tercera cinta y se mueve al lugar donde sea necesario, normalmente a una de las direcciones más bajas, que representan los registros de la computadora.

La máquina de Turing simulará el *ciclo de instrucción* de la computadora como sigue:

1. Busca en la primera cinta la dirección que se corresponde con el número de instrucción de la cinta 2. Partimos del símbolo $ de la primera cinta y nos movemos hacia la derecha comparando cada dirección con los contenidos de la cinta 2. Realizar la comparación de las direcciones de las dos cintas es sencillo, ya que sólo hay que mover las cabezas de las mismas hacia la derecha, en tándem, comprobando que los símbolos leídos siempre coinciden.

2. Cuando se encuentra la dirección de la instrucción, se examina su valor. Suponemos que cuando una palabra es una instrucción, sus primeros bits representan la acción que hay que llevar a cabo (por ejemplo, copiar, sumar, bifurcar), y los restantes bits corresponden a una dirección o a las direcciones implicadas en la acción.

3. Si la instrucción requiere el valor de alguna dirección, entonces dicha dirección será parte de la instrucción. Se copia dicha dirección en la tercera cinta y se marca la posición de la instrucción utilizando una segunda pista de la primera cinta (no se muestra en la Figura 8.22), con el fin de poder volver a la instrucción si fuera necesario. A continuación, se copia la dirección de memoria en la primera cinta y se copia su valor en la tercera cinta, la cinta que almacena la dirección de memoria.

4. Se ejecuta la instrucción, o la parte de la instrucción que implica a este valor. Aunque no podemos entrar en todas las posibles instrucciones de máquina, proporcionamos a continuación una muestra del tipo de cosas que podemos hacer con el nuevo valor:

a) Copiarlo en alguna otra dirección. Obtenemos la segunda dirección de la instrucción, la localizamos introduciéndola en la tercera cinta y buscamos la dirección en la cinta 1, como se ha explicado anteriormente. Una vez que se encuentra la segunda dirección, se copia el valor en el espacio reservado para el valor de dicha dirección. Si se necesita más espacio para el nuevo valor o éste utiliza menos espacio que el antiguo, se modifica el espacio disponible mediante la operación de *desplazamiento*. Es decir,

1) Se copia en la cinta auxiliar la parte ocupada de la cinta completa situada a la derecha del lugar que ocupa el nuevo valor.
2) Se escribe el nuevo valor utilizando la cantidad de espacio adecuada para el mismo.
3) Se copia de nuevo la cinta auxiliar en la cinta 1, inmediatamente a la derecha del nuevo valor.

Como caso especial, la dirección puede no aparecer todavía en la primera cinta, porque la computadora no la haya utilizado anteriormente. En este caso, se busca la posición en la primera cinta en la que debería estar, se hace la operación de desplazamiento para dejar el espacio necesario y se almacenan allí tanto la dirección como el nuevo valor.

b) Se suma el valor que se acaba de encontrar al valor de alguna otra dirección. Se vuelve a la instrucción para localizar la otra dirección, que estará en la cinta 1. Se realiza la suma binaria del valor de dicha dirección y el valor almacenado en la cinta 3. Explorando los dos valores de sus extremos derechos, una MT puede realizar muy fácilmente una suma con propagación de acarreo. Si el resultado necesitara más espacio, bastaría con utilizar la técnica de desplazamiento para crear más espacio en la cinta 1.

c) La instrucción puede ser un "salto"; es decir, una directiva que establece que la siguiente instrucción que se tiene que ejecutar es aquella cuya dirección es el valor almacenado ahora en la cinta 3. En este caso, basta con copiar la cinta 3 en la cinta 2 e iniciar de nuevo el ciclo de instrucción.

5. Después de ejecutar la instrucción y determiniar que la instrucción no es un salto, se suma 1 al contador de instrucciones de la cinta 2 y de nuevo se inicia el ciclo de instrucción.

Hay muchos otros detalles acerca de cómo la MT simula una computadora típica. En la Figura 8.22 se ha incluido una cuarta cinta que almacena la entrada simulada de la computadora, ya que ésta tiene que leer su entrada (la palabra cuya pertenencia a un lenguaje se está comprobando) de un archivo. En su lugar, la MT puede leer de su cinta.

También se ha incluido una cinta auxiliar. La simulación de algunas instrucciones de computadora puede utilizar una cinta auxiliar o varias para realizar operaciones aritméticas como la multiplicación.

Por último, suponemos que la computadora proporciona una salida que indica si acepta o no la entrada. Para traducir esta acción a términos que una máquina de Turing pueda ejecutar, suponemos que la computadora dispone de una instrucción de "aceptación", que podría corresponderse con una llamada a función que hace la computadora para escribir `sí` en un archivo de salida. Cuando la MT simula la ejecución de esta instrucción de la computadora, entra en uno de sus propios estados de aceptación y se detiene.

Aunque la explicación anterior no es una demostración completa y formal de que una MT pueda simular una computadora típica, debería ser suficiente para convencernos de que una MT es una representación válida de lo que puede hacer una computadora. Por tanto, de ahora en adelante, sólo utilizaremos la máquina de Turing como la representación formal de lo que se puede calcular con cualquier tipo de dispositivo computacional.

8.6.3 Comparación de los tiempos de ejecución de las computadoras y las máquinas de Turing

Ahora vamos a abordar el tema del tiempo de ejecución de una máquina de Turing que simula una computadora. Como hemos mencionado anteriormente:

- El tiempo de ejecución es importante porque utilizaremos la MT no sólo para examinar la cuestión de lo que se puede calcular, sino lo que se puede calcular con la eficiencia suficiente como para que resulte práctica una solución basada en computadora del problema.
- La línea divisoria entre los problemas *tratables* (aquellos que se pueden resolver de forma eficiente) y los *intratables* (aquellos que se pueden resolver, pero no lo suficientemente rápido como para que la solución pueda utilizarse) se encuentra, generalmente, entre lo que se puede calcular en un tiempo polinómico y lo que requiere un tiempo mayor que un tiempo de ejecución polinómico.
- Por tanto, necesitamos asegurarnos de que si un problema puede resolverse en un tiempo polinómico en una computadora típica, entonces también una MT puede resolverlo en un tiempo polinómico, y viceversa. Teniendo en cuenta esta equivalencia polinómica, las conclusiones a que se lleguen acerca de lo que una máquina de Turing puede o no puede hacer con la eficiencia adecuada pueden aplicarse también a una computadora.

Recuerde que en la Sección 8.4.3 hemos determinado que la diferencia entre los tiempos de ejecución de una MT de una cinta y una MT de varias cintas era polinómica, en concreto, cuadrática. Por tanto, basta con demostrar que cualquier cosa que pueda hacer una computadora, la MT de varias cintas descrita en la Sección 8.6.2 puede hacerlo en un periodo de tiempo que es polinómico respecto del tiempo que tarda la computadora. Sabremos entonces que esto mismo se cumple en una MT de una sola cinta.

Antes de proporcionar la demostración de que la máquina de Turing descrita anteriormente puede simular n pasos de una computadora en un tiempo $O(n^3)$, tenemos que abordar la cuestion de la multiplicación como instrucción de computadora. El problema es que no hemos establecido un límite para el número de bits que puede tener una palabra de computadora. Por ejemplo, si la computadora partiera de una palabra que almacenara el entero 2 y multiplicara dicha palabra por sí misma n veces consecutivas, entonces la palabra almacenaría el número 2^{2^n}. La representación de este número requiere $2^n + 1$, por lo que el tiempo que la máquina de Turing tarda en simular estas n instrucciones sería exponencial en función de n, como mínimo.

Una solución sería fijar la lóngitud máxima de palabra, por ejemplo, en 64 bits. Entonces, las multiplicaciones (y las demás operaciones) que produjeran una palabra demasiado larga harían que la computadora se detuviera, y la máquina de Turing no tendría que continuar con la simulación. Sin embargo, vamos a adoptar una postura más liberal: la computadora puede utilizar palabras de cualquier longitud, pero una instrucción de la computadora sólo puede generar una palabra cuya longitud sea de un bit más que la longitud de sus argumentos.

EJEMPLO 8.16

Teniendo en cuenta la restricción anterior, la operación suma puede realizarse, ya que la longitud de su resultado sólo puede tener un bit más que la longitud máxima de los sumandos. Sin embargo, la multiplicación no está permitida, ya que dos palabras de m bits pueden dar como resultado un producto de longitud $2m$. No obstante, podemos simular una multiplicación de enteros de m bits mediante una secuencia de m sumas, intercaladas con desplazamientos del multiplicando de un bit hacia la izquierda (que es otra operación que sólo incrementa la longitud de la palabra en 1). Por tanto, podemos multiplicar palabras arbitrariamente largas, aunque el tiempo que tarda la computadora es proporcional al cuadrado de la longitud de los operandos. □

Suponiendo un crecimiento máximo de un bit por instrucción de computadora ejecutada, podemos demostrar la relación polinómica existente entre los dos tiempos de ejecución. La idea en que se basa la demostración es la de observar que después de que se hayan ejecutado n instrucciones, el número de palabras a las que se habrá hecho referencia en la cinta de memoria de la MT será $O(n)$ y cada palabra de la computadora requiere $O(n)$ casillas de la máquina de Turing. Por tanto, la longitud de la cinta será de $O(n^2)$ casillas y la MT podrá almacenar el número finito de palabras que necesita una instrucción de computadora en un tiempo $O(n^2)$.

Sin embargo, es necesario que las instrucciones cumplan un requisito adicional. Incluso aunque la instrucción no genere como resultado una palabra larga, podría tardar una gran cantidad de tiempo en calcular dicho resultado. Por tanto, tenemos que hacer la suposición adicional de que la propia instrucción, aplicada a palabras de hasta longitud k, puede ejecutarse en $O(k^2)$ pasos en una máquina de Turing de varias cintas. Ciertamente, las operaciones típicas realizadas por una computadora, como la suma, el desplazamiento y la comparación de valores, pueden ser realizadas en $O(k)$ pasos de una MT de varias cintas, por lo que estamos siendo muy liberales en cuanto a lo que una computadora hace en una única instrucción.

TEOREMA 8.17

Si una computadora:

1. Tiene sólo instrucciones que incrementan la longitud máxima de palabra en, como máximo, 1, y
2. Tiene sólo instrucciones que una MT de varias cintas puede ejecutar sobre palabras de longitud k en $O(k^2)$ pasos o menos,

entonces la máquina de Turing descrita en la Sección 8.6.2 puede simular los n pasos de la computadora en $O(n^3)$ de sus propios pasos.

DEMOSTRACIÓN. Empezamos observando que la primera cinta (memoria) de la MT de la Figura 8.22 inicialmente sólo contiene el programa de la computadora. Dicho programa puede ser largo, pero su longitud es fija y constante, independientemente de n, el número de pasos de instrucción que ejecuta la computadora. Por tanto, existe una constante c que es la más larga de las palabras y direcciones que aparecen en el programa. Existe también una constante d que define el número de palabras que ocupa el programa.

Así, después de ejecutar n pasos, la computadora no puede haber creado ninguna palabra cuya longitud sea mayor que $c+n$ y, por tanto, no puede haber creado ni utilizado ninguna dirección cuya longitud sea mayor que $c+n$ bits. Cada instrucción crea como máximo una nueva dirección que contiene un valor, por lo que el número total de direcciones después de haber ejecutado n instrucciones es como máximo $d+n$. Dado que cada combinación dirección-palabra requiere a lo sumo $2(c+n)+2$ bits, incluyendo la dirección, el contenido y los dos marcadores que los separan, el número total de casillas de cinta de la MT ocupadas después de haber simulado n instrucciones es, como máximo, $2(d+n)(c+n+1)$. Como c y d son constantes, el número de casillas es $O(n^2)$.

Ahora sabemos que cada una de las búsquedas de direcciones, en número fijo, implicadas en una instrucción de computadora puede ejecutarse en un tiempo $O(n^2)$. Puesto que la longitud de las palabras es $O(n)$, la segundo suposición nos dice que una MT puede ejecutar las instrucciones en sí en un tiempo $O(n^2)$. El único coste significativo que queda en la ejecución de una instrucción es el tiempo que tarda la MT en crear más espacio en su cinta para almacenar una palabra nueva o ampliada. Sin embargo, el desplazamiento implica copiar, a lo sumo, $O(n^2)$ datos desde la cinta 1 a la cinta auxiliar, y de nuevo a la cinta 1. Por tanto, la operación de desplazamiento también requiere un tiempo de $O(n^2)$ por instrucción de computadora.

Podemos concluir que la MT simula un paso de la computadora en $O(n^2)$ de sus propios pasos. Por tanto, como hemos establecido en el enunciado del teorema, n pasos de la computadora pueden simularse en $O(n^3)$ pasos de la máquina de Turing. □

Una observación final. Hemos visto que el número de pasos que necesita una MT de varias cintas para simular una computadora es el cubo del número de pasos. También hemos visto en la Sección 8.4.3 que una MT de una cinta puede simular una MT de varias cintas en, como máximo, el cuadrado del número de pasos. Por tanto,

TEOREMA 8.18

Una computadora del tipo descrito en el Teorema 8.17 puede ser simulada en n pasos por una máquina de Turing de una única cinta, utilizando como máximo $O(n^6)$ pasos de la máquina de Turing. □

8.7 Resumen del Capítulo 8

- *Máquina de Turing*. La máquina de Turing es una máquina de computación abstracta con la potencia tanto de las computadoras reales como de otras definiciones matemáticas de lo que se puede calcular. La MT consta de una unidad de control y de una cinta infinita dividida en casillas. Cada casilla almacena uno de los símbolos del conjunto finito de símbolos de cinta y una casilla corresponde a la posición actual de la cabeza de la cinta. La MT realiza movimientos basados en su estado actual y en el símbolo de cinta de la casilla a la que apunta la cabeza de la cinta. En un movimiento, cambia el estado, sobreescribe la celda señalada por la cabeza con algún símbolo de cinta y mueve la cabeza una celda hacia la izquierda o hacia la derecha.

- *Aceptación en una máquina de Turing*. Inicialmente, la MT tiene en la cinta su entrada, una cadena de longitud finita de símbolos de cinta, y el resto de las casillas contienen un espacio en blanco. Este último es uno de los símbolos de cinta y la entrada se selecciona de un subconjunto de los símbolos de cinta que no incluye el espacio en blanco, y se conocen como símbolos de entrada. La MT acepta su entrada si llega a un estado de aceptación.

- *Lenguajes recursivamente enumerables*. Los lenguajes aceptados por la MT son los denominados lenguajes recursivamente enumerables (RE). Por tanto, los lenguajes RE son aquellos lenguajes que pueden ser reconocidos o aceptados por cualquier clase de dispositivos de computación.

- *Descripciones instantáneas o configuraciones de una máquina de Turing*. Podemos describir la configuración actual de una MT mediante una cadena de longitud finita que incluye todas las casillas de cinta situadas desde la posición más a la izquierda hasta el espacio en blanco más a la derecha. El estado y la posición de la cabeza se indican incluyendo el estado dentro de la secuencia de símbolos de cinta, justo a la izquierda de la casilla señalada por la cabeza de la cinta.

- *Almacenamiento en la unidad de control*. En ocasiones, al diseñar un MT para un determinado lenguaje resulta de ayuda imaginar que el estado tiene dos o más componentes. Una componente es la de control y actúa como lo hace normalmente un estado. Las restantes componentes almacenan los datos que la MT necesita recordar.

- *Pistas múltiples*. Con frecuencia, también resulta de ayuda pensar en los símbolos de la cinta como en vectores con un número fijo de componentes. De este modo, podemos ver cada componente como una pista separada de la cinta.

- *Máquinas de Turing de varias cintas*. Un modelo de MT extendido tiene una cierta cantidad fija de cintas mayor que uno. Un movimiento de esta MT se basa en el estado y en el vector definido por los símbolos señalados por la cabeza de cada una de las cintas. En un movimiento, la MT de varias cintas cambia de estado, sobreescribe los símbolos contenidos en las casillas señaladas por cada una de las cabezas de la cintas y mueve alguna o todas las cabezas de la cinta en determinada dirección. Aunque es capaz de reconocer ciertos lenguajes más rápido que la MT de una sola cinta convencional, la MT de varias cintas no puede reconocer ningún lenguaje que no sea recursivamente enumerable.

- *Máquinas de Turing no deterministas*. La MTN tiene un número finito de posibilidades para realizar el siguiente movimiento (estado, nuevo símbolo y movimiento de la cabeza) para cada estado y símbolo

señalado. Acepta una entrada si cualquier secuencia de elecciones lleva a una configuración que corresponde a un estado de aceptación. Aunque aparentemente más potente que la MT determinista, la MTN no es capaz de reconocer ningún lenguaje que no sea recursivamente enumerable.

- *Máquina de Turing con cinta semi-infinita*. Podemos restringir una máquina de Turing para que tenga una cinta que sólo es infinita por la derecha, sin casillas a la izquierda de la posición inicial de la cabeza. Tal MT puede aceptar cualquier lenguaje RE.

- *Máquinas multitarea*. Podemos restringir las cintas de una MT de varias cintas para que se comporten como una pila. La entrada se encuentra en una cinta separada, que se lee una vez de izquierda a derecha, del mismo modo que en un autómata finito o en un autómata a pila. Una máquina de una pila realmente es un autómata a pila determinista, mientras que una máquina con dos pilas puede aceptar cualquier lenguaje RE.

- *Máquinas contadoras*. Podemos restringir aún más las pilas de una máquina de varias pilas para que sólo contengan un símbolo distinto del marcador de fondo de pila. Así, cada pila se comporta como un contador, lo que nos permite almacenar un entero no negativo y comprobar si el entero almacenado es un 0, pero nada más. Basta una máquina con dos contadores para aceptar cualquier lenguaje RE.

- *Simulación de una máquina de Turing mediante una computadora real*. En principio, es posible, simular una máquina de Turing mediante una computadora real si aceptamos que existe un suministro potencialmente infinito de un dispositivo de almacenamiento extraíble, como por ejemplo, un disco, para simular la parte en que no hay espacios en blanco de la cinta de la MT. Puesto que los recursos físicos con los que se construyen los discos no son infinitos, este argumento es cuestionable. Sin embargo, dado que la cantidad de dispositivos de almacenamiento que existe en el universo es desconocida y, sin duda, muy grande, la suposición de que existen recursos infinitos, como en una cinta de una MT, es realista en la práctica y generalmente se acepta.

- *Simulación de una computadora mediante una máquina de Turing*. Una máquina de Turing puede simular el almacenamiento y la unidad de control de una computadora real empleando una cinta para almacenar todas las posiciones y los contenidos correspondientes a los registros, la memoria principal, los discos y otros dispositivos de almacenamiento. Por tanto, podemos estar seguros de que algo que una máquina de Turing no pueda hacer, tampoco lo podrá hacer una computadora real.

8.8 Referencias del Capítulo 8

La definición de la máquina de Turing se ha tomado de [8]. Prácticamente al mismo tiempo aparecieron varias propuestas que no empleaban máquinas con el fin de caracterizar lo que se puede calcular, entre las que se incluyen los trabajos de Church [1], Kleene [5] y Post [7]. El trabajo de Gödel [3] precedió a los anteriores y demostraba que no existía ninguna forma de que una computadora contestara a todas las cuestiones metemáticas.

El estudio de las máquinas de Turing de varias cintas, especialmente el de la comparación de su tiempo de ejecución con el de la máquina de una sola cinta fue iniciado por Hartmanis y Stearns [4]. El examen de las máquinas de varias pilas y contadoras se atribuye a [6], aunque la construcción que hemos proporcionado aquí se debe a [2].

El método de la Sección 8.1 de utilizar "hola, mundo" como sustituto de una máquina de Turing en lo que respecta a las cuestiones de aceptación y parada apareció en notas no publicadas de S. Rudich.

1. A. Church, "An undecidable problem in elementary number theory", *American J. Math*. **58** (1936), págs. 345–363.

2. P. C. Fischer, “Turing machines with restricted memory access”, *Information and Control* **9**:4 (1966), págs. 364–379.

3. K. Gödel, “Uber formal unentscheidbare Sätze der Principia Mathematica und verwander systeme”, *Monatschefte fur Mathematik und Physik* **38** (1931), págs. 173–198.

4. J. Hartmanis and R. E. Stearns, “On the computational complexity of algorithms”, *Transactions of the AMS* **117** (1965), págs. 285–306.

5. S. C. Kleene, “General recursive functions of natural numbers”, *Mathematische Annalen* **112** (1936), págs. 727–742.

6. M. L. Minsky, “Recursive unsolvability of Post’s problem of ‘tag’ and other topics in the theory of Turing machines”, *Annals of Mathematics* **74**:3 (1961), págs. 437–455.

7. E. Post, “Finite combinatory processes-formulation”, *J. Symbolic Logic* **1** (1936), págs. 103–105.

8. A. M. Turing, “On computable numbers with an application to the Entscheidungsproblem”, *Proc. London Math. Society* **2**:42 (1936), págs. 230–265. Véase también *ibid.* **2**:43, págs. 544–546.