# 1

# Introducción a los autómatas

La teoría de autómatas es el estudio de dispositivos de cálculo abstractos, es decir, de las "máquinas". Antes de que existieran las computadoras, en la década de los años treinta, A. Turing estudió una máquina abstracta que tenía todas las capacidades de las computadoras de hoy día, al menos en lo que respecta a lo que podían calcular. El objetivo de Turing era describir de forma precisa los límites entre lo que una máquina de cálculo podía y no podía hacer; estas conclusiones no sólo se aplican a las *máquinas abstractas de Turing*, sino a todas las máquinas reales actuales.

En las décadas de los años cuarenta y cincuenta, una serie de investigadores estudiaron las máquinas más simples, las cuales todavía hoy denominamos "autómatas finitos". Originalmente, estos autómatas se propusieron para modelar el funcionamiento del cerebro y, posteriormente, resultaron extremadamente útiles para muchos otros propósitos, como veremos en la Sección 1.1. También a finales de la década de los cincuenta, el lingüista N. Chomsky inició el estudio de las "gramáticas" formales. Aunque no son máquinas estrictamente, estas gramáticas están estrechamente relacionadas con los automátas abstractos y sirven actualmente como base de algunos importantes componentes de software, entre los que se incluyen componentes de los compiladores.

En 1969, S. Cook amplió el estudio realizado por Turing sobre lo que se podía y no se podía calcular. Cook fue capaz de separar aquellos problemas que se podían resolver de forma eficiente mediante computadora de aquellos problemas que, en principio, pueden resolverse, pero que en la práctica consumen tanto tiempo que las computadoras resultan inútiles para todo excepto para casos muy simples del problema. Este último tipo de problemas se denominan "insolubles" o "NP-difíciles ". Es extremadamente improbable que incluso la mejora de carácter exponencial en la velocidad de cálculo que el hardware de computadora ha experimentado ("Ley de Moore") tenga un impacto significativo sobre nuestra capacidad para resolver casos complejos de problemas insolubles.

Todos estos desarrollos teóricos afectan directamente a lo que los expertos en computadoras hacen. Algunos de los conceptos, como el de autómata finito y determinados tipos de gramáticas formales, se emplean en el diseño y la construcción de importantes clases de software. Otros conceptos, como la máquina de Turing, nos ayudan a comprender lo que podemos esperar de nuestro software. En particular, la teoría de los problemas intratables nos permite deducir si podremos enfrentarnos a un problema y escribir un programa para resolverlo (porque no pertenece a la clase de problemas intratables) o si tenemos que hallar alguna forma de salvar dicho

problema: hallar una aproximación, emplear un método heurístico o algún otro método para limitar el tiempo que el programa invertirá en resolver el problema.

En este capítulo de introducción, vamos a abordar la teoría de autómatas desde un punto de vista de alto nivel, así como sus usos. Gran parte del capítulo es una introducción a las técnicas de demostración y a los trucos que permiten llevar a cabo dichas demostraciones. Cubrimos también las demostraciones deductivas, la reformulación de proposiciones, las demostraciones por reducción al absurdo, las demostraciones por inducción y otros importantes conceptos. La última sección presenta los conceptos que dominan la teoría de autómatas: alfabetos, cadenas de caracteres y lenguajes.

## 1.1 ¿Por qué estudiar la teoría de autómatas?

Son varias las razones por las que el estudio de los autómatas y de la complejidad de cálculo constituyen una parte importante del núcleo de la Ciencias de la Computación. Esta sección presenta al lector estas razones, e introduce los temas más importantes que se cubren en este libro.

### 1.1.1 Introducción a los autómatas finitos

Los autómatas finitos constituyen un modelo útil para muchos tipos de hardware y software. A partir del Capítulo 2 veremos ejemplos de cómo se emplean estos conceptos. Por el momento, sólo enumeraremos algunos de los tipos más importantes:

1. Software para diseñar y probar el comportamiento de circuitos digitales.
2. El "analizador léxico" de un compilador típico, es decir, el componente del compilador que separa el texto de entrada en unidades lógicas, tal como identificadores, palabras clave y signos de puntuación.
3. Software para explorar cuerpos de texto largos, como colecciones de páginas web, o para determinar el número de apariciones de palabras, frases u otros patrones.
4. Software para verificar sistemas de todo tipo que tengan un número finito de estados diferentes, tales como protocolos de comunicaciones o protocolos para el intercambio seguro de información.

Aunque pronto veremos una definición precisa de los distintos tipos de autómatas, comenzaremos esta introducción informal con un boceto de lo que es y lo que hace un autómata finito. Existen muchos sistemas o componentes, como los que hemos enumerado anteriormente, que pueden encontrarse siempre en uno de una serie de "estados" finitos. El propósito de un estado es el de recordar la parte relevante del historial del sistema. Puesto que sólo existe un número finito de estados, generalmente, no es posible recordar el historial completo, por lo que el sistema debe diseñarse cuidadosamente, con el fin de recordar lo que es importante y olvidar lo que no lo es. La ventaja de disponer de sólo un número finito de estados es que podemos implementar el sistema mediante un conjunto fijo de recursos. Por ejemplo, podríamos implementarlo por hardware como un circuito, o como una forma simple de programa que puede tomar decisiones consultando sólo una cantidad limitada de datos o utilizando la posición del propio código para tomar la decisión.

**EJEMPLO 1.1**

Quizá el autómata finito no trivial más simple sea un interruptor de apagado/encendido (posiciones *on*/*off*). El dispositivo recuerda si está en el estado encendido ("*on*") o en el estado apagado ("*off*"), y permite al usuario pulsar un botón cuyo efecto es diferente dependiendo del estado del interruptor. Es decir, si el interruptor está en el estado *off*, entonces al pulsar el botón cambia al estado *on*, y si el interruptor está en el estado *on*, al pulsar el mismo botón pasa al estado *off*.

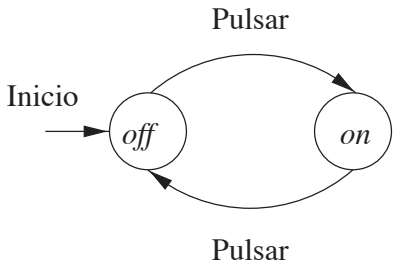


**Figura 1.1.** Modelo de un autómata finito de un interruptor de apagado/encendido (*on*/*off*).

En la Figura 1.1 se muestra el modelo de autómata finito para el interruptor. Como en todos los autómatas finitos, los estados están representados mediante círculos; en este ejemplo, hemos denominado a los estados *on* y *off*. Los arcos entre los estados están etiquetados con las "entradas", las cuales representan las influencias externas sobre el sistema. Aquí, ambos arcos se han etiquetado con la entrada *Pulsar*, que representa al usuario que pulsa el botón. Los dos arcos indican que, sea cual sea el estado en que se encuentra el sistema, cuando recibe la entrada *Pulsar* pasa al otro estado.

Uno de los estados se designa como el "estado inicial", el estado en el que el sistema se encuentra inicialmente. En nuestro ejemplo, el estado inicial es apagado (*off*) y, por conveniencia, hemos indicado el estado inicial mediante la palabra *Inicio* y una flecha que lleva al otro estado.

A menudo es necesario especificar uno o más estados como estado "final" o"de aceptación". Llegar a uno de estos estados después de una secuencia de entradas indica que dicha secuencia es correcta. Por ejemplo, podríamos establecer el estado *on* de la Figura 1.1 como estado de aceptación, ya que en dicho estado, el dispositivo que está siendo controlado por el interruptor funciona. Normalmente, los estados de aceptación se indican mediante un círculo doble, aunque en la Figura 1.1 no lo hemos hecho. □

**EJEMPLO 1.2**

En ocasiones, lo que recuerda un estado puede ser mucho más complejo que una elección entre las posiciones apagado/encendido (*on*/*off*). La Figura 1.2 muestra otro autómata finito que podría formar parte de un analizador léxico. El trabajo de este autómata consiste en reconocer la palabra clave `then`, por lo que necesita cinco estados, representando cada uno de ellos la posición dentro de dicha palabra que se ha alcanzado hasta el momento. Estas posiciones se corresponden con los prefijos de la palabra, desde la cadena de caracteres vacía (es decir, cuando no contiene ningún carácter) hasta la palabra completa.

En la Figura 1.2, los cinco estados se designan mediante el correspondiente prefijo de `then` visto hasta el momento. Las entradas se corresponden con las letras. Podemos imaginar que el analizador léxico examina un carácter del programa que se está compilando en un determinado instante, y que el siguiente carácter que se va a examinar es la entrada al autómata. El estado inicial se corresponde con la cadena vacía y cada uno de los estados tiene una transición a la siguiente letra de la palabra `then`, al estado que corresponde al siguiente prefijo más largo. El estado denominado `then` se alcanza cuando la entrada está formada por todas las letras de

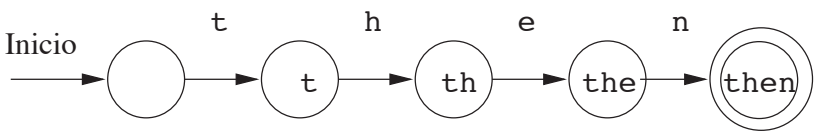


**Figura 1.2.** Modelo de autómata finito para el reconocimiento de la palabra `then`.

dicho término. Puesto que el trabajo de este autómata es indicar el reconocimiento de la palabra `then`, podemos considerar dicho estado como el único estado de aceptación. □

### 1.1.2 Representaciones estructurales

Existen dos importantes notaciones que no son las utilizadas normalmente con los autómatas, pero que desempeñan un importante papel en el estudio de los autómatas y sus aplicaciones.

1. Las *gramáticas* son modelos útiles en el diseño de software que sirve para procesar datos con una estructura recursiva. El ejemplo más conocido es el de un "analizador sintáctico" (*parser*), el componente de un compilador que se ocupa de las funciones anidadas recursivamente de los lenguajes de programación típicos, tales como expresiones aritméticas, condicionales, etc. Por ejemplo, una regla gramatical como $E \Rightarrow E + E$ establece que una expresión puede formarse tomando cualesquiera dos expresiones y conectándolas mediante un signo más; esta regla es típica de cómo se forman las expresiones en los lenguajes reales de programación. En el Capítulo 5 se presentan las gramáticas independientes del contexto, nombre con el que se conoce este tipo de gramáticas.

2. Las *expresiones regulares* también especifican la estructura de los datos, especialmente de las cadenas de texto. Como veremos en el Capítulo 3, los patrones de cadenas de caracteres que pueden describir expresiones regulares son los mismos que pueden ser descritos por los autómatas finitos. El estilo de estas expresiones difiere significativamente del de las gramáticas. Veamos a continuación un ejemplo simple de esto. La expresión regular estilo UNIX `'[A-Z][a-z]*[ ][A-Z][A-Z]'` representa palabras que comienzan por una letra mayúscula seguida de un espacio y de dos letras mayúsculas. Esta expresión representa patrones de texto que podrían corresponderse con el nombre de una ciudad y un estado, por ejemplo `Ithaca NY`. En cambio no reconocería nombres de ciudades formados por varias palabras, como por ejemplo `Palo Alto CA`, que sí podría ser reconocida por la expresión más compleja

   ```
   '[A-Z][a-z]*([ ][A-Z][a-z]*)*[ ][A-Z][A-Z]'
   ```

   Al interpretar dichas expresiones, sólo necesitamos saber que `[A-Z]` representa el rango de caracteres comprendido entre las letras mayúsculas "A" y "Z" (es decir, todas las letras mayúsculas) y que `[ ]` se utiliza para representar un único carácter en blanco. Además, el símbolo de asterisco (`*`) representa "cualquier número de" apariciones de la expresión anterior. Los paréntesis se emplean para agrupar componentes de la expresión; no representan caracteres del texto que se describe.

### 1.1.3 Autómatas y complejidad

Los autómatas son esenciales para el estudio de los límites de la computación. Como hemos indicado en la introducción del capítulo, existen dos factores importantes a este respecto:

1. ¿Qué puede hacer una computadora? Este área de estudio se conoce como "decidibilidad", y los problemas que una computadora puede resolver se dice que son "decidibles". Este tema se aborda en el Capítulo 9.

2. ¿Qué puede hacer una computadora de manera eficiente? Este área de estudio se conoce como "intratabilidad", y los problemas que una computadora puede resolver en un tiempo proporcional a alguna función que crezca lentamente con el tamaño de la entrada se dice que son "tratables". Habitualmente, se supone que todas las funciones polinómicas son de "crecimiento lento", mientras que se considera que las funciones que crecen más rápido que cualquier función polinómica crecen con demasiada rapidez. El tema se estudia en el Capítulo 10.

## 1.2 Introducción a las demostraciones formales

Si estudió en el instituto la geometría del plano antes de la década de los años noventa, probablemente habrá tenido que realizar algunas "demostraciones deductivas" detalladas, en las que se demostraba la veracidad de una proposición mediante una detallada secuencia de pasos y razonamientos. Aunque la geometría tiene su lado práctico (por ejemplo, se necesita conocer la fórmula que permite calcular el área de un rectángulo si se quiere comprar la cantidad correcta de moqueta necesaria para una habitación), el estudio de las metodologías de demostración formal fue, como mínimo, una importante razón para estudiar esta rama de las matemáticas en el instituto.

En Estados Unidos, en la década de los años noventa era normal enseñar los métodos de demostración como algo relacionado con la impresión personal acerca de las proposiciones. Aunque es conveniente establecer la veracidad de la proposición que se va a emplear, ya no se enseñan las técnicas de demostración en los colegios. Las demostraciones es algo que todos los informáticos necesitan conocer y comprender. Algunos de ellos aplican el estricto punto de vista de que una demostración formal de la corrección de un programa debe realizarse a la vez que se escribe el propio programa. Sin duda, este punto de vista no es productivo. Por otro lado, están aquellos que establecen que las demostraciones no tienen lugar en la disciplina de programación, en cuyo caso, suele aplicarse la afirmación "si no está seguro de que el programa sea correcto, ejecútelo y compruébelo".

Nuestra postura respecto a este tema se encuentra entre las dos anteriores. Probar los programas es fundamental. Sin embargo, la realización de pruebas sólo llega hasta cierto punto, ya que no es posible probar los programas para todas las posibles entradas. Aún más importante, si el programa es complejo, por ejemplo contiene recursiones o iteraciones, entonces si no se comprende qué es lo que ocurre al ejecutar un bucle o una llamada a función de forma recursiva, es poco probable que podamos escribir el código correctamente. Si al probar el código resulta ser incorrecto, será necesario corregirlo.

Para conseguir iteraciones o recursiones correctas, es necesario establecer hipótesis inductivas, y resulta útil razonar, formal o informalmente, que la hipótesis es coherente con la iteración o recursión. Este proceso sirve para comprender que el trabajo que realiza un programa correcto es esencialmente el mismo que el proceso de demostrar teoremas por inducción. Por tanto, además de propocionar modelos que resulten útiles para determinados tipos de software, es habitual en un curso sobre autómatas cubrir las metodologías para la realización de demostraciones formales. La teoría de autómatas, quizá más que cualquier otra materia de las que conforman las Ciencias de la Computación, se presta a demostraciones naturales e interesantes, tanto de tipo *deductivo* (una secuencia de pasos justificados) como *inductivo* (demostraciones recursivas de una proposición parametrizada que emplea la propia proposición para valores "más bajos" del parámetro).

### 1.2.1 Demostraciones deductivas

Como hemos dicho anteriormente, una demostración deductiva consta de una secuencia de proposiciones cuya veracidad se comprueba partiendo de una proposición inicial, conocida como *hipótesis* o de una serie de *proposiciones dadas*, hasta llegar a una *conclusión*. Cada uno de los pasos de la demostración, hay que deducirlo mediante algún principio lógico aceptado, bien a partir de los postulados o de algunas de las proposiciones anteriores de la demostración deductiva o de una combinación de éstas.

Las hipótesis pueden ser verdaderas o falsas, dependiendo normalmente de los valores de sus parámetros. A menudo, la hipótesis consta de varias proposiciones independientes conectadas por una operación AND lógica. En dichos casos, decimos que cada una de esas proposiciones es una hipótesis o un postulado.

El teorema que se demuestra partiendo de una hipótesis $H$ para llegar a una conclusión $C$ es la proposición "si $H$ entonces $C$". Decimos entonces que $C$ *se deduce* de $H$. A continuación proporcionamos un teroema de ejemplo de la forma "si $H$ entonces $C$".

**TEOREMA 1.3**

Si $x \geq 4$, entonces $2^x \geq x^2$. □

No es complicado ver informalmente que el Teorema 1.3 es cierto, aunque una demostración formal requiere aplicar el método de inducción, lo que veremos en el Ejemplo 1.4.1. En primer lugar, observe que la hipótesis $H$ es "$x \geq 4$". Esta hipótesis tiene un parámetro, $x$, por lo que no es ni verdadera ni falsa. Su validez depende del valor del parámetro $x$; por ejemplo, $H$ es verdadera para $x = 6$ y falsa para $x = 2$.

Por otro lado, la conclusión $C$ es "$2^x \geq x^2$". Esta proposición también utiliza el parámetro $x$ y es verdadera para ciertos valores de $x$ y falsa para otros. Por ejemplo, $C$ es falsa para $x = 3$, ya que $2^3 = 8$, que es menor que $3^2 = 9$. Por el contrario, $C$ es verdadera para $x = 4$, ya que $2^4 = 4^2 = 16$. Para $x = 5$, la proposición también es verdadera, ya que $2^5 = 32$ es al menos tan grande como $5^2 = 25$.

Quizá vea el argumento intutitivo de que la conclusión $2^x \geq x^2$ será verdadera cuando $x \geq 4$. Ya hemos visto que es verdadera para $x = 4$. Cuando $x$ es mayor que 4, el lado izquierdo de la ecuación, $2^x$, se duplica cada vez que $x$ aumenta en una unidad. Sin embargo, el lado derecho de la ecuación, $x^2$, aumenta según la relación $(\frac{x+1}{x})^2$. Si $x \geq 4$, entonces $(x+1)/x$ no puede ser mayor que 1,25 y, por tanto, $(\frac{x+1}{x})^2$ no puede ser mayor que 1,5625. Puesto que $1{,}5625 < 2$, cada vez que $x$ toma un valor mayor que 4, el lado izquierdo, $2^x$, se hace mayor que el lado derecho, $x^2$. Por tanto, siempre que partamos de un valor como $x = 4$, para el que se cumple la desigualdad $2^x \geq x^2$, podemos aumentar $x$ tanto como deseemos, y la desigualdad se satisfará.

Hemos completado así una demostración completa e informal del Teorema 1.3. En el Ejemplo 1.17 volveremos sobre esta demostración y la realizaremos de forma más precisa, después de explicar las demostraciones "inductivas".

El Teorema 1.3, como todos los teoremas interesantes, implica un número infinito de hechos relacionados, en este caso la proposición "si $x \geq 4$ entonces $2^x \geq x^2$" para todos los enteros $x$. En realidad, no necesitamos suponer que $x$ es un entero, pero la demostración habla de incrementar repetidamente $x$ en una unidad, comenzando en $x = 4$, por lo que sólo vamos a ocuparnos de la situación en que $x$ es un entero.

El Teorema 1.3 puede utilizarse para deducir otros teoremas. En el siguiente ejemplo abordamos una demostración deductiva completa de un teorema simple que emplea el Teorema 1.3.

**TEOREMA 1.4**

Si $x$ es la suma de los cuadrados de cuatro enteros positivos, entonces $2^x \geq x^2$.

**DEMOSTRACIÓN**. La idea intuitiva de la demostración es que si la hipótesis es verdadera para $x$, es decir, $x$ es la suma de los cuadrados de cuatro enteros positivos, entonces $x$ tiene que ser como mínimo igual a 4. Por tanto, la hipótesis del Teorema 1.3 se cumple y, por tanto, creemos dicho teorema, así podemos establecer que su conclusión también es verdadera para $x$. El razonamiento puede expresarse como una secuencia de pasos. Cada paso es bien la hipótesis del teorema que se va a demostrar, parte de dicha hipótesis o una proposición que se deduce de una o más proposiciones previas.

Por "deducir" queremos decir que si la hipótesis de algún teorema es una proposición previa, entonces la conclusión de dicho teorema es verdadera, y se puede escribir como una proposición de la demostración. Esta regla lógica a menudo se denomina *modus ponens*; es decir, si sabemos que $H$ es verdadera y sabemos que "si $H$ entonces $C$" es verdadero, podemos concluir que $C$ es verdadera. También podemos emplear otros pasos lógicos para crear una proposición que se deduzca a partir de una o más proposiciones anteriores. Por ejemplo, si $A$ y $B$ son dos proposiciones previas, entonces podemos deducir y escribir la proposición "$A$ y $B$".

La Figura 1.3 muestra la secuencia de proposiciones que necesitamos para demostrar el Teorema 1.4. Aunque por regla general no demostraremos los teoremas de manera tan elegante, esta forma de presentar la demostración ayuda a interpretar las demostraciones como listas muy precisas de proposiciones, justificándose cada una de ellas. En el paso (1), hemos repetido una de las proposiciones dadas del teorema: $x$ es la suma de los cuadrados de cuatro enteros. Suele resultar útil en las demostraciones nombrar las cantidades a las que se hace referencia, aunque no hayan sido identificadas previamente, y eso es lo que hemos hecho aquí asignando a los cuatro enteros los nombres $a$, $b$, $c$ y $d$.

| | Proposición | Justificación |
|---|---|---|
| 1. | $x = a^2 + b^2 + c^2 + d^2$ | Dado |
| 2. | $a \geq 1; b \geq 1; c \geq 1; d \geq 1$ | Dado |
| 3. | $a^2 \geq 1; b^2 \geq 1; c^2 \geq 1; d^2 \geq 1$ | (2) y propiedades aritméticas |
| 4. | $x \geq 4$ | (1), (3) y propiedades aritméticas |
| 5. | $2^x \geq x^2$ | (4) y Teorema 1.3 |

**Figura 1.3.** Demostración formal del Teorema 1.4.

En el paso (2), anotamos la otra parte de la hipótesis del teorema: los valores base de los cuadrados valen al menos 1. Técnicamente, esta proposición representa cuatro proposiciones diferentes, una para cada uno de los enteros implicados. Entonces, en el paso (3), observamos que si el valor de un número es como mínimo 1, entonces su cuadrado también es como mínimo 1. Utilizamos como justificación del hecho que la proposición (2) se cumple, así como las "propiedades aritméticas". Es decir, suponemos que el lector sabe, o es capaz de demostrar, proposiciones simples sobre desigualdades, como por ejemplo la proposición "si $y \geq 1$, entonces $y^2 \geq 1$".

El paso (4) utiliza las proposiciones (1) y (3). La primera proposición nos dice que $x$ es la suma de los cuatro cuadrados en cuestión y la proposición (3) nos dice que cada uno de los cuadrados es como mínimo 1. De nuevo, empleando las propiedades aritméticas, concluimos que $x$ es como mínimo $1+1+1+1$, es decir, 4.

En el paso final, el número (5), utilizamos la proposición (4), que es la hipótesis del Teorema 1.3. El propio teorema es la justificación que se emplea para obtener la conclusion, ya que su hipótesis es la proposición anterior. Dado que la proposición (5) es la conclusión del Teorema 1.3, también es la conclusión del Teorema 1.4, por tanto, el Teorema 1.4 queda demostrado. Es decir, hemos partido de la hipótesis de dicho teorema y hemos deducido su conclusión. □

### 1.2.2 Reducción a definiciones

En los dos teoremas anteriores, la hipótesis emplea términos con los que debería estar familiarizado: enteros, suma y multiplicación, por ejemplo. En muchos otros teoremas, incluyendo los de la teoría de autómatas, el término utilizado en la proposición puede tener implicaciones que son menos obvias. Una forma útil de proceder en muchas demostraciones es:

- Si no estamos seguros de cómo comenzar una demostración, convertimos todos los términos de la hipótesis a sus definiciones.

Veamos un ejemplo de un teorema que es sencillo de demostrar una vez que se ha expresado su proposición en términos elementales. Se utilizan las dos definiciones siguientes:

1. Un conjunto $S$ es *finito* si existe un entero $n$ tal que $S$ tiene exactamente $n$ elementos. Escribimos $\|S\| = n$, donde $\|S\|$ se utiliza para designar el número de elementos de un conjunto $S$. Si el conjunto $S$ no es finito, decimos que $S$ es *infinito*. Intuitivamente, un conjunto infinito es un conjunto que contiene más que cualquier número entero de elementos.

2. Si $S$ y $T$ son subconjuntos de algún conjunto $U$, entonces $T$ es el *complementario* de $S$ (con respecto a $U$) si $S \cup T = U$ y $S \cap T = \emptyset$. Es decir, cada elemento de $U$ es exactamente uno de $S$ y otro de $T$; dicho de otra manera, $T$ consta exactamente de aquellos elementos de $U$ que no pertenecen a $S$.

**TEOREMA 1.5**

Sea $S$ un subconjunto finito de un determinado conjunto infinito $U$. Sea $T$ el conjunto complementario de $S$ con respecto a $U$. Entonces $T$ es infinito.

**DEMOSTRACIÓN**. Intuitivamente, este teorema dice que si tenemos una cantidad infinita de algo ($U$), y tomamos una cantidad finita ($S$), entonces nos seguirá quedando una cantidad infinita. Comencemos redefiniendo los hechos del teorema como se muestra en la Figura 1.4.

Todavía no hemos avanzado de forma significativa, por lo que necesitamos utilizar una técnica de demostración común conocida como "demostración por reducción al absurdo". Con este método de demostración, que veremos más en detalle en la Sección 1.3.3, suponemos que la conclusión es falsa. Entonces utilizamos esta suposición, junto con partes de la hipótesis, para demostrar la afirmación opuesta de uno de los postulados dados de la hipótesis. Habremos demostrado entonces que es imposible que todas las partes de la hipótesis sean verdaderas y la conclusión sea al mismo tiempo falsa. La única posibilidad que queda entonces es que la conclusión sea verdadera cuando la hipótesis también lo sea. Es decir, el teorema es verdadero.

En el caso del Teorema 1.5, lo contrario de la conclusión es que "$T$ es finito". Supongamos que $T$ es finito, junto con la proposición de la hipótesis que dice que $S$ es finito; es decir, $\|S\| = n$ para algún entero $n$. De forma similar, podemos redefinir la suposición de que $T$ es finito como $\|T\| = m$ para algún entero $m$.

Ahora una de las proposiciones dadas nos dice que $S \cup T = U$ y $S \cap T = \emptyset$. Es decir, los elementos de $U$ son exactamente los elementos de $S$ y $T$. Por tanto, $U$ tiene $n+m$ elementos. Dado que $n+m$ es un entero y que hemos demostrado que $\|U\| = n+m$, se deduce que $U$ es finito. De forma más precisa, hemos demostrado que el número de elementos en $U$ es algún entero, que es la definición de "finito". Pero la proposición de que $U$ es finito contradice la proposición que establece que $U$ es infinito. Hemos empleado entonces la contradicción de nuestra conclusión para demostrar la contradicción de una de las proposiciones dadas de la hipótesis y, aplicando el principio de la "demostración por reducción al absurdo" podemos concluir que el teorema es verdadero. □

Las demostraciones no tienen que ser tan prolijas. Vistas las ideas que hay detrás de la demostración, vamos a demostrar una vez más el teorema, pero ahora en unas pocas líneas.

**DEMOSTRACIÓN**. (del Teorema 1.5) Sabemos que $S \cup T = U$ y que $S$ y $T$ son disjuntos, por lo que $\|S\| + \|T\| = \|U\|$. Dado que $S$ es finito, $\|S\| = n$ para algún entero $n$, y como $U$ es infinito, no existe ningún entero $p$ tal que $\|U\| = p$. Por tanto, suponemos que $T$ es finito; es decir, $\|T\| = m$ para algún $m$. Entonces $\|U\| = \|S\| + \|T\| = n+m$, lo que contradice la proposición dada de que no existe ningún entero $p$ que sea igual a $\|U\|$. □

## 1.2.3 Otras formas de teoremas

La forma "si-entonces" *(if-then)* del teorema es la más común en las áreas típicas de las matemáticas. Sin embargo, vamos a ver otros tipos de proposiciones que también resultan ser teoremas. En esta sección, examinaremos las formas más comunes de proposiciones y lo que normalmente tendremos que hacer para demostrarlas.

| Proposición original | Nueva proposición |
|---|---|
| $S$ es finito | Existe un entero $n$ tal que $\|S\| = n$ |
| $U$ es infinito | No existe un entero $p$ tal que $\|U\| = p$ |
| $T$ es el conjunto complementario de $S$ | $S \cup T = U$ y $S \cap T = \emptyset$ |

**Figura 1.4.** Redefinición de los postulados del Teorema 1.5.

### Proposiciones con cuantificadores

Muchos teoremas implican proposiciones que utilizan los *cuantificadores* "para todo" y "existe", o variaciones similares de estos como "para cada" en lugar de "para todo". El orden en el que aparecen estos cuantificadores afecta a lo que significa la proposición. Suele resultar útil interpretar las proposiciones con más de un cuantificador como un "partido" entre dos jugadores (para todo y existe) que especifican por turno los valores de los parámetros mencionados en el teorema. "Para todo" tiene que considerar todas las posibles opciones, por lo que generalmente las opciones de para todo se dejan como variables. Sin embargo, "existe" sólo tiene que elegir un valor, que puede depender de los valores elegidos por los jugadores anteriormente. El orden en que aparecen los cuantificadores en la proposición determina quién es el primero. Si el último jugador que hace una elección siempre puede encontrar algún valor permitido, entonces la proposición es verdadera.

Por ejemplo, consideremos una definición alternativa de "conjunto infinito": sea $S$ un conjunto *infinito* si y sólo si para todos los enteros $n$, existe un subconjunto $T$ de $S$ con exactamente $n$ elementos. En este caso, "para todo" precede a "existe", por lo que hay que considerar un entero arbitrario $n$. Ahora, "existe" elige un subconjunto $T$ y puede utilizar $n$ para ello. Por ejemplo, si $S$ fuera el conjunto de los números enteros, "existe" podría elegir el subconjunto $T = \{1, 2, \ldots, n\}$ y por tanto acertar para cualquier valor de $n$. Esto es una demostración de que el conjunto de los números enteros es infinito.

La siguiente proposición es parecida a la definición de "infinito", pero es *incorrecta* porque invierte el orden de los cuantificadores: "existe un subconjunto $T$ del conjunto $S$ tal que para todo $n$, el conjunto $T$ tiene exactamente $n$ elementos". Ahora, dado un conjunto $S$, como el de los enteros, el jugador "existe" puede elegir cualquier conjunto $T$; supongamos que selecciona $\{1, 2, 5\}$. En este caso, el jugador "para todo" tiene que demostrar que $T$ tiene $n$ elementos para *cada* posible $n$. Pero, "para todo" no puede hacerlo, ya que, por ejemplo, es falso para $n = 4$, de hecho, es falso para cualquier $n \neq 3$.

## Formas de "si-entonces"

En primer lugar, existen diversos tipos de enunciados de teoremas que parecen diferentes de la forma simple "si $H$ entonces $C$", pero de hecho expresan lo mismo: si la hipótesis $H$ es verdadera para un valor determinado del (de los) parámetro(s), entonces la conclusión $C$ es verdadera para el mismo valor. A continuación se enumeran varias formas en las que puede expresarse "si $H$ entonces $C$".

1. $H$ implica $C$.
2. $H$ sólo si $C$.
3. $C$ si $H$.
4. Si se cumple $H$, se cumple $C$.

También podemos encontrar muchas variantes de la forma (4), tales como "si $H$ se cumple, entonces $C$".

**EJEMPLO 1.6**

El enunciado del Teorema 1.3 podría expresarse de las cuatro formas siguientes:

1. $x \geq 4$ implica $2^x \geq x^2$.

### ¿Cómo de formales tienen que ser las demostraciones?

La respuesta a esta pregunta no es sencilla. El objetivo de una demostración es convencer a alguien, sea a un alumno o a nosotros mismos, acerca de la corrección de la estrategia que se está empleando en el código. Si es convincente, entonces es suficiente; si no convence al "destinatario" de la demostración, entonces es que deja mucho que desear.

Parte de la certidumbre respecto de las demostraciones depende de los conocimientos que tenga el destinatario. Por tanto, en el Teorema 1.4, hemos supuesto que el lector conoce todo lo relativo a las cuestiones aritméticas y debería comprender una proposición como "si $y \geq 1$ entonces $y^2 \geq 1$". Si no esta familiarizado con la aritmética, tendríamos que demostrar dicha proposición mediante algunos pasos adicionales en la demostración deductiva.

Sin embargo, hay algunas cuestiones que son necesarias en las demostraciones, y su omisión seguramente daría lugar a una demostración inadecuada. Por ejemplo, cualquier demostración deductiva que emplee proposiciones que no hayan sido justificadas por las proposiciones anteriores, no puede ser adecuada. Al demostrar una proposición "si y sólo si", hay que demostrar tanto la parte "si" como la parte "sólo-si". Como ejemplo adicional, las demostraciones adicionales (que se explican en la Sección 1.4) requieren demostraciones del caso básico y de la parte de inducción.

2. $x \geq 4$ sólo si $2^x \geq x^2$.
3. $2^x \geq x^2$ si $x \geq 4$.
4. Si $x \geq 4$, entonces $2^x \geq x^2$. □

Además, en la lógica formal, a menudo podremos ver el operador $\rightarrow$ en lugar de "si-entonces". Es decir, en algunos textos de matemáticas, podrá ver la proposición "si $H$ entonces $C$" escrita como $H \rightarrow C$; pero nosotros no vamos a emplear aquí esta notación.

## Proposiciones "Si y sólo si"

En ocasiones, encontraremos proposiciones de la forma "$A$ si y sólo si $B$"("$A$ if and only if $B$"). Otras formas de esta proposición son "$A$ iff $B$", [1]"$A$ es equivalente a $B$" o "$A$ exactamente si $B$". Realmente, esta proposición se corresponde con dos proposiciones si-entonces (if-then):"si $A$ entonces $B$" y "si $B$ entonces $A$". En estos casos, demostaremos la proposición "$A$ si y sólo si $B$" demostrando las dos proposiciones siguientes:

1. La *parte si*: "si $B$ entonces $A$" y
2. La *parte sólo si*: "si $A$ entonces $B$", lo que a menudo se escribe de la forma equivalente "$A$ sólo si $B$".

Las demostraciones pueden presentarse en cualquier orden. En muchos teoremas, una parte es claramente más fácil que la otra, por lo que es habitual abordar primero la parte más sencilla.

En lógica formal, se pueden emplear los operadores $\leftrightarrow$ o $\equiv$ para designar una proposición "si y sólo si". Es decir, $A \equiv B$ y $A \leftrightarrow B$ quieren decir lo mismo que "$A$ si y sólo si $B$".

[1] Iff, abreviatura de "if and only if" (si y sólo si), es una no-palabra que se emplea en algunos tratados de Matemáticas para abreviar.

Al demostrar una proposición si-y-sólo-si, es importante recordar que es necesario probar tanto la parte "si" como la parte "sólo-si". En ocasiones, resultará útil dividir una proposición si-y-sólo-si en una sucesión de varias equivalencias. Es decir, para demostrar "*A* si y sólo si *B*", primero hay que demostrar "*A* si y sólo si *C*", y luego demostrar "*C* si y sólo si *B*". Este método es adecuado, siempre y cuando se tenga en cuenta que cada paso si-y-sólo-si debe ser demostrado en ambas direcciones. La demostración de cualquier paso en una sola de las direcciones invalida la demostración completa.

A continuación se proporciona un ejemplo de una demostración sencilla de una proposición si-y-sólo-si. Emplearemos la siguiente notación:

1. $\lfloor x \rfloor$, el *suelo* del número real $x$, es el mayor entero igual o menor que $x$.
2. $\lceil x \rceil$, el *techo* del número real $x$, es el menor entero igual o mayor que $x$.

**TEOREMA 1.7**

Sea $x$ un número real. Entonces $\lfloor x \rfloor = \lceil x \rceil$ si y sólo si $x$ es un entero.

**DEMOSTRACIÓN**. *Parte sólo-si*. En esta parte, suponemos $\lfloor x \rfloor = \lceil x \rceil$ e intentamos demostrar que $x$ es un entero. Utilizando las definiciones de suelo y techo, observamos que $\lfloor x \rfloor \leq x$ y $\lceil x \rceil \geq x$. Sin embargo, se parte de que $\lfloor x \rfloor = \lceil x \rceil$. Luego podemos sustituir el suelo por el techo en la primera desigualdad para concluir que $\lceil x \rceil \leq x$. Puesto que $\lceil x \rceil \leq x$ y $\lceil x \rceil \geq x$ se cumplen, podemos concluir aplicando las propiedades de las desigualdades aritméticas que $\lceil x \rceil = x$. Dado que $\lceil x \rceil$ es siempre un entero, $x$ también tiene que ser un entero en este caso.
*Parte Si*. Supongamos ahora que $x$ es un entero y demostremos que $\lfloor x \rfloor = \lceil x \rceil$. Esta parte es sencilla. Por las definiciones de suelo y techo, cuando $x$ es un entero, tanto $\lfloor x \rfloor$ como $\lceil x \rceil$ son iguales a $x$ y, por tanto, iguales entre sí. □

### 1.2.4 Teoremas que parecen no ser proposiciones Si-entonces

En ocasiones, nos encontraremos con teoremas que no parecen contener una hipótesis. Un ejemplo muy conocido de esto lo encontramos en el campo de la trigonometría:

**TEOREMA 1.8**

$\text{sen}^2\,\theta + \cos^2\theta = 1$. □

Realmente, esta proposición *sí* contiene una hipotésis, la cual consta de todas las proposiciones necesarias para saber interpretar dicha proposición. En concreto, la hipótesis oculta es que $\theta$ es un ángulo y, por tanto, las funciones seno y coseno tienen sus significados habituales cuando se refieren a ángulos. A partir de las definiciones de estos términos y del Teorema de Pitágoras (en un triángulo rectángulo, el cuadrado de la hipotenusa es igual a la suma de los cuadrados de los catetos) es posible demostrar el teorema. En resumen, la forma si-entonces del teorema es: "si $\theta$ es un ángulo, entonces $\text{sen}^2\,\theta + \cos^2\theta = 1$".

## 1.3 Otras formas de demostración

En esta sección, vamos a abordar varios temas relacionados con la construcción de demostraciones:

1. Demostraciones empleando conjuntos.
2. Demostraciones por reducción al absurdo.
3. Demostraciones mediante contraejemplo.

### 1.3.1 Demostración de equivalencias entre conjuntos

En la teoría de autómatas, frecuentemente es necesario demostrar un teorema que establece que los conjuntos construidos de dos formas diferentes son el mismo conjunto. A menudo, se trata de conjuntos de cadenas de caracteres y se denominan "lenguajes", no obstante, en esta sección la naturaleza de los conjuntos no es importante. Si $E$ y $F$ son dos expresiones que representan conjuntos, la proposición $E = F$ quiere decir que los dos conjuntos representados son iguales. De forma más precisa, cada uno de los elementos del conjunto representado por $E$ está en el conjunto representado por $F$, y cada uno de los elementos del conjunto representado por $F$ está en el conjunto representado por $E$.

**EJEMPLO 1.9**

La *ley conmutativa de la unión* establece que podemos calcular la unión de dos conjuntos $R$ y $S$ en cualquier orden. Es decir, $R \cup S = S \cup R$. En este caso, $E$ es la expresión $R \cup S$ y $F$ es la expresión $S \cup R$. La ley conmutativa de la unión establece que $E = F$. □

Podemos expresar la igualdad de conjuntos $E = F$ como una proposición si-y-sólo-si: un elemento $x$ pertenece a $E$ si y sólo si $x$ pertenece a $F$. En consecuencia, veamos el esquema de una demostración de cualquier proposición que establezca la igualdad de dos conjuntos $E = F$; esta demostración tiene la misma forma que cualquier demostración del tipo si-y-sólo-si:

1. Demostrar que si $x$ pertenece a $E$, entonces $x$ pertenece a $F$.
2. Demostrar que si $x$ pertenece a $F$, entonces $x$ pertenece a $E$.

Como ejemplo de este proceso de demostración, probemos la *ley distributiva de la unión respecto de la intersección*:

**TEOREMA 1.10**

$R \cup (S \cap T) = (R \cup S) \cap (R \cup T)$.

**DEMOSTRACIÓN**. Las dos expresiones de conjuntos implicadas son $E = R \cup (S \cap T)$ y

$$F = (R \cup S) \cap (R \cup T)$$

Vamos a demostrar las dos partes del teorema de forma independiente. En la parte "si" suponemos que el elemento $x$ pertenece a $E$ y demostraremos que también pertenece a $F$. Esta parte, resumida en la Figura 1.5, utiliza las definiciones de unión e intersección, con las que suponemos que el lector estará familiarizado.

A continuación tenemos que demostrar la parte "sólo-si" del teorema. Aquí, suponemos que $x$ pertenece a $F$ y demostramos que también pertenece a $E$. Los pasos se resumen en la Figura 1.6. Puesto que hemos demostrado las dos partes de la proposición si-y-sólo-si, queda demostrada la ley distributiva de la unión respecto de la intersección. □

### 1.3.2 La conversión contradictoria

Toda proposición si-entonces tiene una forma equivalente que, en algunas circunstancias, es más fácil de demostrar. La *conversión contradictoria* de la proposición "si $H$ entonces $C$" es "si no $C$ entonces no $H$". Una proposición y su contradictoria son ambas verdaderas o ambas falsas, por lo que podemos demostrar una u otra. Para ver por qué "si $H$ entonces $C$" y "si no $C$ entonces no $H$" son lógicamente equivalentes, en primer lugar, observamos que hay que considerar cuatro casos:

| | Proposición | Justificación |
|---|---|---|
| 1. | $x$ pertenece a $R \cup (S \cap T)$ | Postulado |
| 2. | $x$ pertenece a $R$ o $x$ pertenece a $S \cap T$ | (1) y la definición de unión |
| 3. | $x$ pertenece a $R$ o $x$ pertenece a $S$ y $T$ | (2) y la definición de intersección |
| 4. | $x$ pertenece a $R \cup S$ | (3) y la definición de unión |
| 5. | $x$ pertenece a $R \cup T$ | (3) y la definición de unión |
| 6. | $x$ pertenece a $(R \cup S) \cap (R \cup T)$ | (4), (5), y la definición de intersección |

**Figura 1.5.** Pasos correspondientes a la parte "si" del Teorema 1.10.

| | Proposición | Justificación |
|---|---|---|
| 1. | $x$ pertenece a $(R \cup S) \cap (R \cup T)$ | Postulado |
| 2. | $x$ pertenece a $R \cup S$ | (1) y la definición de intersección |
| 3. | $x$ pertenece a $R \cup T$ | (1) y la definición de intersección |
| 4. | $x$ pertenece a $R$ o $x$ pertenece a $S$ y $T$ | (2), (3), y el razonamiento sobre la unión |
| 5. | $x$ pertenece a $R$ o $x$ pertenece a $S \cap T$ | (4) y la definición de intersección |
| 6. | $x$ pertenece a $R \cup (S \cap T)$ | (5) y la definición de unión |

**Figura 1.6.** Pasos correspondientes a la parte "sólo-si" del Teorema 1.10.

### Enunciados "Si-y-sólo-si" para conjuntos

Como hemos dicho, los teoremas que establecen equivalencias entre expresiones que implican conjuntos son proposiciones si-y-sólo-si. Por tanto, el Teorema 1.10 podría enunciarse del siguiente modo: un elemento $x$ pertenece a $R \cup (S \cap T)$ si y sólo si $x$ pertenece a

$$(R \cup S) \cap (R \cup T)$$

Otra expresión habitual de equivalencia entre conjuntos es la que emplea la forma "todos y sólo ellos". Por ejemplo, el Teorema 1.10 también se podría enunciar de la forma siguiente: "los elementos de $R \cup (S \cap T)$ son todos y sólo los elementos de

$$(R \cup S) \cap (R \cup T)$$

1. Tanto $H$ como $C$ son verdaderas.
2. $H$ es verdadera y $C$ es falsa.
3. $C$ es verdadera y $H$ es falsa.
4. Tanto $H$ como $C$ son falsas.

Sólo existe una manera de hacer que una proposición si-entonces sea falsa: la hipótesis tiene que ser verdadera y la conclusión falsa, lo que se corresponde con el caso (2). En los otros tres casos, incluyendo el número (4), en el que la conclusión es falsa, la proposición si-entonces es verdadera. Consideremos ahora en qué casos la conversión contradictoria "si no $C$ entonces no $H$" es falsa. Para que esta proposición sea falsa, su hipótesis (que es "no $C$") tiene que ser verdadera y su conclusión (que es "no $H$") tiene que ser falsa. Pero

### La inversa

No deben confundirse los términos "conversión contradictoria" e "inversa". La *inversa* de una proposición si-entonces se refiere al "otro sentido" de la proposición; es decir, la inversa de "si $H$ entonces $C$" es "si $C$ entonces $H$". A diferencia de la conversión contradictoria, que es la proposición lógicamente equivalente de la original, la inversa *no* es equivalente a la proposición original. De hecho, las dos partes de una demostración si-y-sólo-si siempre se corresponden con una proposición y su inversa.

"no $C$" es verdadera cuando $C$ es falsa y "no $H$" es falsa justamente cuando $H$ es verdadera. Estas dos condiciones se corresponden de nuevo con el caso (2), lo que demuestra que en cada uno de los cuatro casos, la proposición original y su conversión contradictoria son ambas verdaderas o falsas; es decir, son lógicamente equivalentes.

**EJEMPLO 1.11**

Recordemos el Teorema 1.3, cuyas proposiciones eran: "si $x \geq 4$, entonces $2^x \geq x^2$". La conversión contradictoria de esta proposición es "si no $2^x \geq x^2$ entonces no $x \geq 4$". En términos más coloquiales, utilizando el hecho de que "no $a \geq b$" es lo mismo que $a < b$, la conversión contradictoria es "si $2^x < x^2$ entonces $x < 4$". □

A la hora de tener que demostrar un teorema si-y-sólo-si, el uso de la conversión contradictoria en una de las partes nos proporciona varias opciones. Por ejemplo, suponga que deseamos demostrar la equivalencia entre conjuntos $E = F$. En lugar de demostrar que "Si $x$ pertenece a $E$ entonces $x$ pertenece a $F$ y si $x$ pertenece a $F$ entonces $x$ pertenece a $E$", podríamos también expresar uno de los sentidos en su forma contradictoria. Una forma de demostración equivalente es la siguiente:

- Si $x$ pertenece a $E$ entonces $x$ pertenece a $F$, y si $x$ no pertenece a $E$ entonces $x$ no pertenece a $F$.

También podríamos intercambiar $E$ y $F$ en la proposición anterior.

## 1.3.3 Demostración por reducción al absurdo

Otra forma de demostrar una proposición de la forma "si $H$ entonces $C$" consiste en demostrar la proposición:

- "$H$ y no $C$ implica falsedad".

Es decir, se comienza suponiendo que tanto la hipótesis $H$ como la negación de la conclusión $C$ son verdaderas. La demostración se completa probando que algo que se sabe que es falso se deduce lógicamente a partir de $H$ y $C$. Esta forma de demostración se conoce como *demostración por reducción al absurdo*.

**EJEMPLO 1.12**

Recordemos el Teorema 1.5, donde demostramos la proposición si-entonces con la hipótesis $H$ = "$U$ es un conjunto infinito, $S$ es un subconjunto finito de $U$ y $T$ es el complementario de $S$ con respecto a $U$". La conclusión $C$ fue que "$T$ era infinito". Demostremos ahora este teorema por reducción al absurdo. Hemos supuesto "no $C$"; es decir, que $T$ era finito.

La demostración nos llevó a deducir una falsedad a partir de $H$ y no $C$. En primer lugar, probamos a partir de las suposiciones de que $S$ y $T$ eran ambos finitos que $U$ también tenía que ser finito. Pero dado que en la hipótesis $H$ se ha establecido que $U$ es infinito y un conjunto no puede ser a la vez finito e infinito, hemos demostrado que la proposición lógica es "falsa". En términos lógicos, tenemos tanto la proposición $p$ ($U$ es finito) como

su negación, no $p$ ($U$ es infinito). Así podemos usar el hecho de que "$p$ y no $p$" es lógicamente equivalente a "falso". □

Para ver por qué las demostraciones por reducción al absurdo son lógicamente correctas, recordemos de la Sección 1.3.2 que existen cuatro combinaciones de valores de verdad para $H$ y $C$. Sólo en el segundo caso, cuando $H$ es verdadera y $C$ falsa, la proposición "si $H$ entonces $C$" es falsa. Comprobando que $H$ y no $C$ llevan a una falsedad, demostramos que el caso 2 no puede producirse. Por tanto, las únicas combinaciones posibles de valores de verdad para $H$ y $C$ son las tres combinaciones que hacen que "si $H$ entonces $C$" sea verdadera.

### 1.3.4 Contraejemplos

En la práctica, no se habla de demostrar un teorema, sino que tenemos que enfrentarnos a algo que parece que es cierto, por ejemplo, una estrategia para implementar un programa, y tenemos que decidir si el "teorema" es o no verdadero. Para resolver este problema, podemos intentar demostrar el teorema, y si no es posible, intentar demostrar que la proposición es falsa.

Generalmente, los teoremas son proposiciones que incluyen un número infinito de casos, quizá todos los valores de sus parámetros. En realidad, un convenio matemático estricto sólo dignificará a una proposición con el título de "teorema" si se cumple para un número infinito de casos; las proposiciones que no incluyen ningún parámetro o que sólo se aplican a un número finito de valores de su(s) parámetro(s) se conocen como *observaciones*. Basta con demostrar que un supuesto teorema es falso en un caso para que quede demostrado que no es un teorema. La situación es análoga a la de los programas, ya que, generalmente, se considera que un programa tiene un error si falla para una entrada para la que se pensaba que iba a funcionar correctamente.

Suele ser más fácil demostrar que una proposición no es un teorema que demostrar que *sí* lo es. Como hemos dicho, si $S$ es cualquier proposición, entonces la proposición "$S$ no es un un teorema" es propia de una instrucción sin parámetros y, por tanto, podemos considerarla como una observación en lugar de como un teorema. De los siguientes dos ejemplos, el primero obviamente no es un teorema y el segundo es una proposición que no está claro si es un teorema y requiere realizar ciertas investigaciones antes de decidir si se trata de un teorema o no.

**SUPUESTO TEOREMA 1.13**

Todos los número primos son impares. Más formalmente, podemos enunciar que: si un entero $x$ es un número primo, entonces $x$ es impar.

**REFUTACIÓN**. El entero 2 es primo, pero es par. □

Veamos ahora un "teorema" que implica aritmética modular. Existe una definición fundamental que es necesario que establezcamos antes de continuar. Si $a$ y $b$ son enteros positivos, entonces $a$ mod $b$ es el resto de dividir $a$ entre $b$, es decir, el único entero $r$ comprendido entre 0 y $b-1$, tal que $a = qb + r$ para algún entero $q$. Por ejemplo, $8 \bmod 3 = 2$ y $9 \bmod 3 = 0$. El primer supuesto teorema, que determinaremos que es falso, es:

**SUPUESTO TEOREMA 1.14**

No existe ninguna pareja de enteros $a$ y $b$ tal que $a \bmod b = b \bmod a$. □

Cuando hay que operar con parejas de objetos, tales como $a$ y $b$ en este caso, a menudo es posible simplificar la relación entre ambos aprovechando las relaciones de simetría. En este caso, podemos centrarnos en el caso en que $a < b$, ya que si $b < a$ podemos intercambiar $a$ y $b$ y obtener la misma ecuación que en el Supuesto Teorema 1.14. Sin embargo, no debemos olvidarnos del tercer caso en el que $a = b$, el cual puede frustrar nuestros intentos de llevar a cabo la demostración.

Supongamos que $a < b$. Luego $a \bmod b = a$, ya que por definición de $a \bmod b$, sabemos que $q = 0$ y $r = a$. Es decir, cuando $a < b$ tenemos $a = 0 \times b + a$. Pero $b \bmod a < a$, ya que cualquier mod $a$ está comprendido entre

0 y $a-1$. Por tanto, cuando $a < b$, $b \bmod a < a \bmod b$, por lo que $a \bmod b = b \bmod a$ es imposible. Utilizando el anterior argumento de simetría, sabemos también que $a \bmod b \neq b \bmod a$ si $b < a$.

Sin embargo, considere el tercer caso: $a = b$. Puesto que $x \bmod x = 0$ para cualquier entero $x$, *tenemos* que $a \bmod b = b \bmod a$ si $a = b$. Por tanto, disponemos de una refutación del supuesto teorema:

**REFUTACIÓN**. (del Supuesto Teorema 1.14) Sea $a = b = 2$. Luego $a \bmod b = b \bmod a = 0$

En el proceso de hallar el contraejemplo, hemos descubierto en realidad las condiciones exactas para las que se cumple el supuesto teorema. He aquí la versión correcta del teorema y su demostración.

**TEOREMA 1.15**

$$a \bmod b = b \bmod a \text{ si y sólo si } a = b$$

**DEMOSTRACIÓN**. *Parte Si*. Suponemos $a = b$. Luego como hemos visto anteriormente, $x \bmod x = 0$ para cualquier entero $x$. Por tanto, $a \bmod b = b \bmod a = 0$ cuando $a = b$.

*Parte Sólo-si*. Ahora suponemos que $a \bmod b = b \bmod a$. La mejor técnica es una demostración por reducción al absurdo, por lo que suponemos además la negación de la conclusión; es decir, suponemos $a \neq b$. Luego, puesto que hemos eliminado $a = b$, sólo tenemos que tener en cuenta los casos $a < b$ y $b < a$.

Ya hemos visto anteriormente que si $a < b$, tenemos $a \bmod b = a$ y $b \bmod a < a$. Por tanto, estas proposiciones, junto con la hipótesis $a \bmod b = b \bmod a$ nos lleva a una contradicción.

Teniendo en cuenta la simetría, si $b < a$ entonces $b \bmod a = b$ y $a \bmod b < b$. De nuevo llegamos a una contradicción de la hipotésis y podemos concluir que la parte sólo-si también es verdadera. Hemos demostrado la proposición en ambos sentidos y podemos concluir que el teorema es verdadero. □

## 1.4 Demostraciones inductivas

Existe una forma especial de demostración, denominada "inductiva", que es esencial a la hora de tratar con objetos definidos de forma recursiva. Muchas de las demostraciones inductivas más habituales trabajan con enteros, pero en la teoría de autómatas, también necesitamos demostraciones inductivas, por ejemplo, para conceptos definidos recursivamente como pueden ser árboles y expresiones de diversas clases, como expresiones regulares, las cuales hemos mencionado brevemente en la Sección 1.1.2. En esta sección, vamos a presentar el tema de las demostraciones inductivas mediante inducciones "sencillas" sobre enteros. A continuación, demostraremos cómo llevar a cabo inducciones "estructurales" sobre cualquier concepto definido de forma recursiva.

### 1.4.1 Inducciones sobre números enteros

Suponga que tenemos que demostrar una proposición $S(n)$ acerca de un número entero $n$. Un enfoque que se emplea habitualmente consiste en demostrar dos cosas:

1. El *caso base*, donde demostramos $S(i)$ para un determinado entero $i$. Normalmente, $i = 0$ o $i = 1$, pero habrá ejemplos en los que desearemos comenzar en cualquier valor mayor de $i$, quizá porque la proposición $S$ sea falsa para los enteros más pequeños.

2. El *paso de inducción*, donde suponemos $n \geq i$, siendo $i$ el entero empleado en el caso base, y demostramos que "si $S(n)$ entonces $S(n+1)$".

Intuitivamente, estas dos partes deberían convencernos de que $S(n)$ es verdadera para todo entero $n$ que sea igual o mayor que el entero de partida $i$. Podemos argumentar de la forma siguiente: supongamos que $S(n)$ es

falsa para uno o más enteros. Entonces debería existir un valor más pequeño que $n$, por ejemplo, $j$, para el que $S(j)$ fuera falsa, aún siendo $j \geq i$. Sin embargo, $j$ no podría ser $i$, porque hemos demostrado en el caso base que $S(i)$ es verdadera. Luego, $j$ tiene que ser mayor que $i$. Ahora sabemos que $j-1 \geq i$ y, por tanto, $S(j-1)$ es verdadera.

Sin embargo, en el paso de inducción, hemos demostrado que si $n \geq i$, entonces $S(n)$ implica $S(n+1)$. Suponga que ahora hacemos $n = j-1$. Luego por inducción sabemos que $S(j-1)$ implica $S(j)$. Dado que también sabemos que $S(j-1)$ es verdadero, podemos concluir que $S(j)$ lo es también.

Hemos partido de la negación de lo que deseábamos demostrar; es decir, hemos supuesto que $S(j)$ era falsa para algún $j \geq i$. En cada uno de los casos, hemos obtenido una contradicción, por lo que tenemos una "demostración por reducción al absurdo" de que $S(n)$ es verdadera para todo $n \geq i$.

Lamentablemente, existe un fallo sutil en el razonamiento anterior. La suposición de poder elegir un $j \geq i$ mínimo para el que $S(j)$ sea falsa depende de que conozcamos previamente el principio de induccción. Es decir, la única manera de demostrar que podemos hallar tal $j$ es hacerlo mediante un método que sea fundamentalmente una demostración inductiva. Sin embargo, la "demostración" anterior hace un buen uso del sentido intutitivo y encaja con muestra percepción del mundo real. Por tanto, en general, se considera una parte integrante de nuestro sistema de razonamiento lógico:

- *Principio de inducción*. Si demostramos $S(i)$ y demostramos que para todo $n \geq i$, $S(n)$ implica $S(n+1)$, entonces podemos concluir que se cumple $S(n)$ para todo $n \geq i$.

Los dos ejemplos siguientes ilustran el uso del principio de inducción para demostrar teoremas sobre números enteros.

**TEOREMA 1.16**

Para todo $n \geq 0$:

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6} \tag{1.1}$$

**DEMOSTRACIÓN**. La demostración se lleva a cabo en dos partes: el caso base y el paso de inducción. Veamos cada una de ellas.

**BASE**. Elegimos $n = 0$. Puede sorprender pensar que el teorema tenga sentido para $n = 0$, ya que el lado izquierdo de la Ecuación (1.1) es $\sum_{i=1}^{0}$ cuando $n = 0$. Sin embargo, existe un principio general que establece que cuando el límite superior de un sumatorio (0 en este caso) es menor que el límite inferior (1, en este caso), la suma no se realiza sobre ningún término y, por tanto, es igual a 0. Es decir, $\sum_{i=1}^{0} i^2 = 0$.

El lado derecho de la Ecuación (1.1) también es 0, ya que $0 \times (0+1) \times (2 \times 0+1)/6 = 0$. Por tanto, la Ecuación (1.1) se cumple cuando $n = 0$.

**PASO INDUCTIVO**. Supongamos ahora que $n \geq 0$. Tenemos que demostrar el paso de inducción, es decir, que la Ecuación (1.1) implica la misma fórmula si sustituimos $n$ por $n+1$. Esta fórmula es

$$\sum_{i=1}^{[n+1]} i^2 = \frac{[n+1]([n+1]+1)(2[n+1]+1)}{6} \tag{1.2}$$

Podemos simplificar las Ecuaciones (1.1) y (1.2) expandiendo las sumas y productos del lado derecho de dichas ecuaciones, con lo que:

$$\sum_{i=1}^{n} i^2 = (2n^3 + 3n^2 + n)/6 \tag{1.3}$$

$$\sum_{i=1}^{n+1} i^2 = (2n^3 + 9n^2 + 13n + 6)/6 \tag{1.4}$$

Tenemos que demostrar (1.4) utilizando (1.3), ya que en el principio de inducción estas proposiciones corresponden a $S(n+1)$ y $S(n)$, respectivamente. El "truco" está en descomponer la suma hasta $n+1$ del lado derecho de la Ecuación (1.4) en una suma hasta $n$ más el término correspondiente a $(n+1)$. De este modo, podemos reemplazar el sumatorio hasta $n$ por el lado izquierdo de la Ecuación (1.3) y demostrar que (1.4) es cierta. Estos pasos son los siguientes:

$$\left(\sum_{i=1}^{n} i^2\right) + (n+1)^2 = (2n^3 + 9n^2 + 13n + 6)/6 \tag{1.5}$$

$$(2n^3 + 3n^2 + n)/6 + (n^2 + 2n + 1) = (2n^3 + 9n^2 + 13n + 6)/6 \tag{1.6}$$

La verificación final de que (1.6) es verdadera sólo requiere aplicar al lado izquierdo de la ecuación algo de álgebra de polinomios para demostrar que es idéntido al lado derecho de la misma. □

**EJEMPLO 1.17**

En el siguiente ejemplo, vamos a demostrar el Teorema 1.3 de la Sección 1.2.1. Recuerde que este teorema establece que si $x \geq 4$, entonces $2^x \geq x^2$. Ya hemos proporcionado una demostración informal basada en la idea de que la relación $x^2/2^x$ disminuye cuando $x$ es mayor que 4. Podemos precisar esta idea si demostramos por inducción la proposición $2^x \geq x^2$ sobre $x$, partiendo de que $x = 4$. Observe que la proposición es falsa para $x < 4$.

**BASE**. Si $x = 4$, entonces $2^x$ y $x^2$ son iguales a 16. Por tanto, $2^4 \geq 4^2$ se cumple.

**PASO INDUCTIVO**. Supongamos que se cumple que $2^x \geq x^2$ para algún $x \geq 4$. Estableciendo esta proposición como hipótesis, tenemos que demostrar la misma proposición con $x+1$ en el lugar de $x$, es decir, $2^{[x+1]} \geq [x+1]^2$. Éstas son las proposiciones $S(x)$ y $S(x+1)$ en el principio de inducción; el hecho de que estemos utilizando $x$ en lugar de $n$ como parámetro no es importante; $x$ o $n$ simplemente designan una variable local.

Como en el Teorema 1.16, tenemos que escribir $S(x+1)$ de manera que podamos emplear $S(x)$. En este caso, podemos escribir $2^{[x+1]}$ como $2 \times 2^x$. Puesto que $S(x)$ nos dice que $2^x \geq x^2$, podemos concluir que $2^{x+1} = 2 \times 2^x \geq 2x^2$.

Pero necesitamos algo diferente; tenemos que demostrar que $2^{x+1} \geq (x+1)^2$. Una forma de hacerlo sería demostrando que $2x^2 \geq (x+1)^2$ y luego aplicar la transitividad de $\geq$ para demostrar que $2^{x+1} \geq 2x^2 \geq (x+1)^2$. En nuestra demostración de que:

$$2x^2 \geq (x+1)^2 \tag{1.7}$$

podemos emplear la suposición de que $x \geq 4$. Comenzamos simplificando la Ecuación (1.7):

$$x^2 \geq 2x + 1 \tag{1.8}$$

Dividiendo (1.8) entre $x$, obtenemos:

$$x \geq 2 + \frac{1}{x} \tag{1.9}$$

Dado que $x \geq 4$, sabemos que $1/x \leq 1/4$. Por tanto, el lado izquierdo de la Ecuación (1.9) es 4 como mínimo, y el lado derecho es 2.25 como máximo. Hemos demostrado por tanto que la Ecuación (1.9) es verdadera. Luego las Ecuaciones (1.8) y (1.7) también son ciertas. La Ecuación (1.7) a su vez nos dice que $2x^2 \geq (x+1)^2$ para $x \geq 4$ y nos permite demostrar la proposición $S(x+1)$, la cual recordemos que era $2^{x+1} \geq (x+1)^2$. □

### Números enteros como conceptos definidos recursivamente

Hemos mencionado que las demostraciones por inducción resultan útiles cuando el objeto en cuestión está definido de manera recursiva. Sin embargo, nuestros primeros ejemplos eran inducciones sobre números enteros, que normalmente no interpretamos como "definidos recursivamente". No obstante, existe una definición natural de carácter recursivo de los enteros no negativos, y esta definición se adapta a la forma en que se realizan las inducciones sobre los números enteros: desde objetos definidos en primer lugar a otros definidos con posterioridad.

**BASE**. 0 es un entero.

**PASO INDUCTIVO**. Si $n$ es un entero, entonces $n+1$ también lo es.

## 1.4.2 Formas más generales de inducción sobre enteros

A veces, una demostración inductiva sólo es posible empleando un esquema más general que el propuesto en la Sección 1.4.1, donde hemos demostrado una proposición $S$ para un valor base y luego hemos demostrado que "si $S(n)$ entonces $S(n+1)$". Dos generalizaciones importantes de este esquema son las siguientes:

1. Podemos utilizar varios casos base. Es decir, demostramos que $S(i), S(i+1), \ldots, S(j)$ para algunos valores $j > i$.
2. Para probar $S(n+1)$, podemos utilizar la validez de todas las proposiciones,

$$S(i), S(i+1), \ldots, S(n)$$

   en lugar de emplear únicamente $S(n)$. Además, si hemos demostrado los casos base hasta $S(j)$, entonces podemos suponer que $n \geq j$, en lugar de sólo $n \geq i$.

La conclusión que se obtendría a partir de estos casos base y del paso de inducción es que $S(n)$ es cierta para todo $n \geq i$.

**EJEMPLO 1.18**

El siguiente ejemplo ilustra el potencial de ambos principios. La proposición $S(n)$ que queremos demostrar es que si $n \geq 8$, entonces $n$ puede expresarse como una suma de treses y cincos. Observe que 7 no se puede expresar como una suma de treses y cincos.

**BASE**. Los casos base son $S(8)$, $S(9)$ y $S(10)$. Las demostraciones son $8 = 3+5$, $9 = 3+3+3$ y $10 = 5+5$, respectivamente.

**PASO INDUCTIVO**. Suponemos que $n \geq 10$ y que $S(8), S(9), \ldots, S(n)$ son verdaderas. Tenemos que demostrar $S(n+1)$ a partir de los postulados dados. Nuestra estrategia va a consistir en restar 3 de $n+1$, observar que este número tiene que poder escribirse como una suma de treses y cincos y sumar un 3 más a la suma para obtener una forma de escribir $n+1$.

Dicho de manera más formal, observe que $n-2 \geq 8$, por lo que podemos suponer que $S(n-2)$ es verdadera. Es decir, $n-2 = 3a+5b$ para los enteros $a$ y $b$. Luego, $n+1 = 3+3a+5b$, por lo que $n+1$ se puede escribir como la suma de $a+1$ treses y $b$ cincos. Esto demuestra $S(n+1)$ y concluye el paso de inducción. □

### 1.4.3 Inducciones estructurales

En la teoría de autómatas, existen varias estructuras definidas recursivamente sobre las que es necesario demostrar proposiciones. Los familiares conceptos sobre árboles y expresiones son ejemplos importantes de estas estructuras. Como las inducciones, todas las definiciones recursivas tienen un caso base, en el que se definen una o más estructuras elementales, y un paso de inducción, en el que se definen estructuras más complejas en función de las estructuras definidas previamente.

**EJEMPLO 1.19**

He aquí la definición recursiva de un árbol:

**BASE**. Un único nodo es un árbol y dicho nodo es la *raíz* del árbol.

**PASO INDUCTIVO**. Si $T_1, T_2, \ldots, T_k$ son árboles, entonces podemos formar un nuevo árbol de la forma siguiente:

1. Comenzamos con un nodo nuevo $N$, que es la raíz del árbol.
2. Añadimos copias de todos los árboles $T_1, T_2, \ldots, T_k$.
3. Añadimos arcos desde el nodo $N$ hasta las raíces de cada uno de los nodos $T_1, T_2, \ldots, T_k$.

La Figura 1.7 muestra la construcción inductiva de un árbol cuya raíz es $N$ a partir de $k$ árboles más pequeños. □

**EJEMPLO 1.20**

He aquí otra definición recursiva. En esta ocasión, definimos *expresiones* utilizando los operadores aritméticos $+$ y $*$, pudiendo ser los operandos tanto números como variables.

**BASE**. Cualquier número o letra (es decir, una variable) es una expresión.

**PASO INDUCTIVO**. Si $E$ y $F$ son expresiones, entonces $E+F$, $E*F$ y $(E)$ también lo son.
Por ejemplo, tanto 2 como $x$ son expresiones de acuerdo con el caso base. El paso inductivo nos dice que $x+2$, $(x+2)$ y $2*(x+2)$ también son expresiones. Observe que cada una de estas expresiones depende de que todas las anteriores sean expresiones. □

Cuando se tiene una definición recursiva, es posible demostrar teoremas acerca de ella utilizando la siguiente forma de demostración, conocida como *inducción estructural*. Sea $S(X)$ una proposición sobre las estructuras $X$ que están definidas mediante una determinada definición recursiva.

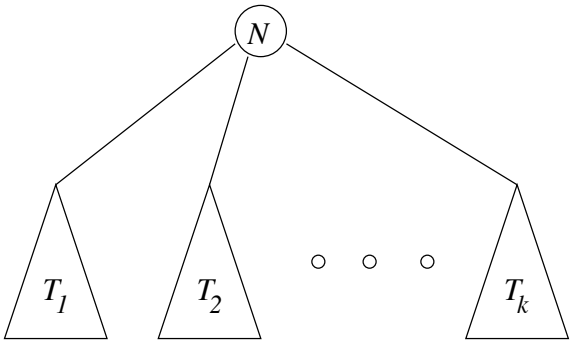


**Figura 1.7.** Construcción inductiva de un árbol.

## Explicación intuitiva de la inducción estructural

Informalmente, podemos sugerir por qué la inducción estructural es un método válido de demostración. Imagine la definición recursiva que establece, una a una, que determinadas estructuras $X_1, X_2, \ldots$ cumplen la definición. En primer lugar, están los elementos base y el postulado de que $X_i$ es el conjunto definido de estructuras que sólo puede depender del conjunto de miembros del conjunto definido de estructuras que precede a $X_i$ en la lista. Desde este punto de vista, una inducción estructural no es nada más que una inducción sobre el número entero $n$ de la proposición $S(X_n)$. Esta inducción puede ser de la forma generalizada vista en la Sección 1.4.2, con múltiples casos base y un paso inductivo que utiliza todas las instancias anteriores de la proposición. Sin embargo, recordemos que, como se ha explicado en la Sección 1.4.1, esta explicación intuitiva no es una demostración formal y, de hecho, tenemos que suponer la validez de este principio de inducción, tal y como se hizo con la validez del principio de inducción original en dicha sección.

1. Como caso base, probamos $S(X)$ para la(s) estructura(s) base $X$.
2. En el paso de inducción, tomamos una estructura $X$ que según la definición recursiva establece que se forma a partir de $Y_1, Y_2, \ldots, Y_k$. Suponemos que las proposiciones $S(Y_1), S(Y_2), \ldots, S(Y_k)$ son verdaderas y las utilizamos para probar $S(X)$.

La conclusión es que $S(X)$ es cierto para todo $X$. Los dos teoremas siguientes son ejemplos de hechos que pueden demostrarse para árboles y expresiones.

**TEOREMA 1.21**

El número de nodos de un árbol es superior al de arcos en una unidad.

**DEMOSTRACIÓN**. La proposición formal $S(T)$ que tenemos que demostrar por inducción estructural es: "si $T$ es un árbol y $T$ tiene $n$ nodos y $e$ arcos, entonces $n = e + 1$".

**BASE**. El caso base es en el que $T$ tiene un único nodo. Así, $n = 1$ y $e = 0$, por lo que la relación $n = e + 1$ se cumple.

**PASO INDUCTIVO**. Sea $T$ un árbol construido a partir del paso inductivo de la definición, a partir de $N$ nodos y $k$ árboles más pequeños $T_1, T_2, \ldots, T_k$. Podemos suponer que las proposiciones $S(T_i)$ se cumplen para $i = 1, 2, \ldots, k$. Es decir, $T_i$ tiene $n_i$ nodos y $e_i$ arcos; luego $n_i = e_i + 1$.

$T$ tiene $N$ nodos y son todos los nodos de los $T_i$ árboles. Por tanto, $T$ tiene $1 + n_1 + n_2 + \cdots + n_k$ nodos. Los arcos de $T$ son los $k$ arcos añadidos explícitamente en el paso de la definición inductiva más los arcos de los $T_i$. Por tanto, $T$ tiene

$$k + e_1 + e_2 + \cdots + e_k \tag{1.10}$$

arcos. Si sustituimos $n_i$ por $e_i + 1$ en la cuenta del número de nodos de $T$, vemos que $T$ tiene

$$1 + [e_1 + 1] + [e_2 + 1] + \cdots + [e_k + 1] \tag{1.11}$$

nodos. Luego como tenemos $k$ términos "+1" en (1.10), podemos reagrupar la Ecuación (1.11) de la forma siguiente:

$$k + 1 + e_1 + e_2 + \cdots + e_k \tag{1.12}$$

Esta expresión es exactamente más grande en una unidad que la expresión (1.10), la cual proporciona el número de arcos de $T$. Por tanto, el número de nodos de $T$ es superior en una unidad al número de arcos. □

**TEOREMA 1.22**

Todas las expresiones tienen el mismo número de paréntesis de apertura que de cierre.

**DEMOSTRACIÓN**. Formalmente, la proposición $S(G)$ se demuestra sobre cualquier expresión $G$ que esté definida mediante el proceso recursivo del Ejemplo 1.20:el número de paréntesis de apertura y de cierre de $G$ es el mismo.

**BASE**. Si $G$ se define a partir de la base, entonces $G$ es un número o una variable. Estas expresiones tienen cero paréntesis de apertura y cero paréntesis de cierre, luego tienen los mismos paréntesis de apertura que de cierre.

**PASO INDUCTIVO**. Hay tres reglas mediante las que se puede construir la expresión $G$ de acuerdo con el paso de inducción de la definición:

1. $G = E + F$.
2. $G = E * F$.
3. $G = (E)$.

Podemos suponer que $S(E)$ y $S(F)$ son verdaderas; es decir, $E$ tiene el mismo número de paréntesis de apertura que de cierre, por ejemplo, $n$ de cada clase, e igualmente $F$ tiene el mismo número de paréntesis de apertura que de cierre, por ejemplo, $m$ de cada clase. Entonces podemos calcular el número de paréntesis abiertos y cerrados de $G$ para cada uno de los tres casos siguientes:

1. Si $G = E + F$, entonces $G$ tiene $n + m$ paréntesis de apertura y $n + m$ paréntesis de cierre; $n$ de cada tipo procedentes de $E$ y $m$ de cada tipo procedentes de $F$.
2. Si $G = E * F$, la cantidad de paréntesis de $G$ es de nuevo $n + m$ de cada tipo por la misma razón que en el caso (1).
3. Si $G = (E)$, entonces habrá $n + 1$ paréntesis de apertura en $G$ (uno de los cuales aparece explícitamente y los otros $n$ proceden de $E$). Del mismo modo, hay $n + 1$ paréntesis de cierre en $G$; uno explícito y los otros $n$ procedentes de $E$.

En cada uno de los tres casos, vemos que el número de paréntesis de apertura y de cierre de $G$ es el mismo. Esta observación completa el paso de inducción y la demostración. □

## 1.4.4 Inducciones mutuas

En ocasiones, no se puede probar una sola proposición por inducción, y es necesario probar un grupo de proposiciones $S_1(n), S_2(n), \ldots, S_k(n)$ por inducción sobre $n$. En la teoría de autómatas se pueden encontrar muchas situaciones así. En el Ejemplo 1.23 veremos la situación habitual en la que es necesario explicar qué hace un autómata probando un grupo de proposiciones, una para cada estado. Estas proposiciones establecen bajo qué secuencias de entrada el autómata entra en cada uno de los estados.

En términos estrictos, probar un grupo de proposiciones no es diferente a probar la *conjunción* (AND lógico) de todas las proposiciones. Por ejemplo, el grupo de proposiciones $S_1(n), S_2(n), \ldots, S_k(n)$ podría reemplazarse por una sola proposición $S_1(n)$ AND $S_2(n)$ AND $\cdots$ AND $S_k(n)$. Sin embargo, cuando hay que demostrar realmente varias proposiciones independientes, generalmente, resulta menos confuso mantener las proposiciones separadas y demostrar cada de una ellas con su caso base y su paso de inducción. A este tipo de demostración se la denomina *inducción mutua*. A continuación presentamos un ejemplo que ilustra los pasos necesarios para llevar a cabo una inducción mutua.

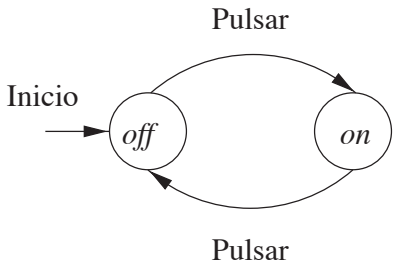


**Figura 1.8.** El mismo autómata de la Figura 1.1.

**EJEMPLO 1.23**

Volvamos al interruptor de encendido/apagado (on/off), que representamos como un autómata en el Ejemplo 1.1. El autómata correspondiente está representado en la Figura 1.8. Dado que al pulsar el botón se conmuta entre los estados *on* y *off*, y que el estado inicial del interruptor es *off*, suponemos que las siguientes proposiciones explican el modo de funcionamiento del interruptor:

> $S_1(n)$: el autómata está en el estado *off* después de haber pulsado el interruptor $n$ veces si y sólo si $n$ es par.
> $S_2(n)$: el autómata está en el estado *on* después de haber pulsado el interruptor $n$ veces si y sólo si $n$ es impar.

Podemos suponer que $S_1$ implica $S_2$ y viceversa, ya que sabemos que un número $n$ no puede ser a la vez par e impar. Sin embargo, lo que no siempre es cierto en un autómata es que esté en un estado y sólo en uno. El autómata de la Figura 1.8 siempre está exactamente en un estado, pero este hecho tiene que ser demostrado como parte de la inducción mutua.

A continuación, establecemos la base y el paso de inducción de las demostraciones de las proposiciones $S_1(n)$ y $S_2(n)$. Las demostraciones dependen de varios hechos que cumplen para los números enteros impares y pares: si sumamos o restamos 1 de un entero par, obtenemos un entero impar, y si sumamos o restamos 1 de un entero impar obtenemos un entero par.

**BASE**. Como caso base, elegimos $n = 0$. Dado que tenemos dos proposiciones, tenemos que demostrar ambas en ambos sentidos (porque $S_1$ y $S_2$ son proposiciones "si-y-sólo-si"); realmente tenemos cuatro casos base y cuatro pasos de inducción.

1. [$S_1$; parte Si] Puesto que 0 es par, tenemos que demostrar que después de pulsar 0 veces, el autómata de la Figura 1.8 se encuentra en el estado *off*. Como se trata del estado inicial, el autómata está de hecho en el estado *off* después de 0 pulsaciones.

2. [$S_1$; parte Sólo-si] El autómata se encuentra en el estado *off* después de 0 pulsaciones, por lo que tenemos que demostrar que 0 es par. Pero 0 es par por la definición de "par", por lo que no hay nada más que demostrar.

3. [$S_2$; parte Si] La hipótesis de la parte "si" de $S_2$ es que 0 es impar. Puesto que esta hipótesis $H$ es falsa, cualquier proposición de la forma "si $H$ entonces $C$" es verdadera, como se ha explicado en la Sección 1.3.2. Por tanto, esta parte del caso base también se cumple.

4. [$S_2$; parte Sólo-si] La hipótesis de que el autómata está en el estado *on* después de 0 pulsaciones también es falsa, ya que la única forma de llegar al estado *on* es siguiendo el arco etiquetado como *Pulsar*, lo que requiere que el interruptor sea pulsado al menos una vez. Puesto que la hipótesis es falsa, de nuevo podemos concluir que la proposición si-entonces es verdadera.

**PASO INDUCTIVO**. Ahora suponemos que $S_1(n)$ y $S_2(n)$ son verdaderas e intentamos demostrar $S_1(n+1)$ y $S_2(n+1)$. De nuevo, separamos la demostración en cuatro partes.

1. [$S_1(n+1)$; parte Si] La hipótesis para esta parte es que $n+1$ es par. Luego, $n$ es impar. La parte "si" de la proposición $S_2(n)$ dice que después de $n$ pulsaciones, el autómata se encuentra en el estado *on*. El arco que va desde el estado *on* al *off* etiquetado con *Pulsar* nos indica que la $(n+1)$ pulsación hará que el autómata entre en el estado *off*. Esto completa la demostración de la parte "si" de $S_1(n+1)$.

2. [$S_1(n+1)$; parte Sólo-si] La hipótesis es que el autómata se encuentra en el estado *off* después de $n+1$ pulsaciones. Inspeccionando el autómata de la Figura 1.8 vemos que la única forma de alcanzar el estado *off* después de una o más pulsaciones es estar en el estado *on* y recibir una entrada *Pulsar*. Por tanto, si estamos en el estado *off* después de $n+1$ pulsaciones, tenemos que haber estado en el estado *on* después de $n$ pulsaciones. Entonces, podemos utilizar la parte "sólo-si" de la proposición $S_2(n)$ para concluir que $n$ es impar. En consecuencia, $n+1$ es par, que es la conclusión deseada para la parte sólo-si de $S_1(n+1)$.

3. [$S_2(n+1)$; parte Si] Esta parte es prácticamente igual que el apartado (1), intercambiando los papeles de las proposiciones $S_1$ y $S_2$, y de "impar" y "par". El lector puede desarrollar fácilmente esta parte de la demostración.

4. [$S_2(n+1)$; parte Sólo-si] Esta parte es prácticamente igual que la (2), intercambiando los papeles de las proposiciones $S_1$ y $S_2$, y de "impar" y "par". □

Podemos abstraer del Ejemplo 1.23 el patrón de las inducciones mutuas:

- Cada una de las proposiciones tiene que demostrarse por separado para el caso base y el paso de inducción.
- Si las proposiciones son de tipo "si-y-sólo-si", entonces tienen que demostrarse ambos sentidos de cada proposición, tanto para el caso base como para el paso de inducción.

## 1.5 Conceptos fundamentales de la teoría de autómatas

En esta sección, vamos a presentar las definiciones de los términos más importantes empleados en la teoría de autómatas. Estos conceptos incluyen el de "alfabeto" (un conjunto de símbolos),"cadenas de caracteres" (una lista de símbolos de un alfabeto) y "lenguaje" (un conjunto de cadenas de caracteres de un mismo alfabeto).

### 1.5.1 Alfabetos

Un *alfabeto* es un conjunto de símbolos finito y no vacío. Convencionalmente, utilizamos el símbolo $\Sigma$ para designar un alfabeto. Entre los alfabetos más comunes se incluyen los siguientes:

1. $\Sigma = \{0,1\}$, el alfabeto *binario*.
2. $\Sigma = \{a,b,\ldots,z\}$, el conjunto de todas las letras minúsculas.
3. El conjunto de todos los caracteres ASCII o el conjunto de todos los caracteres ASCII imprimibles.

### 1.5.2 Cadenas de caracteres

Una *cadena de caracteres* (que también se denomina en ocasiones *palabra*) es una secuencia finita de símbolos seleccionados de algún alfabeto. Por ejemplo, 01101 es una cadena del alfabeto binario $\Sigma = \{0,1\}$. La cadena 111 es otra cadena de dicho alfabeto.

**Convenio de tipos para símbolos y cadenas**

Habitualmente, emplearemos las letras minúsculas del principio del alfabeto (o dígitos) para designar a los símbolos y las letras minúsculas del final del alfábeto, normalmente $w$, $x$, $y$ y $z$, para designar cadenas. Debe intentar utilizar este convenio con el fin de recordar el tipo de elementos con los que está trabajando.

## La cadena vacía

La *cadena vacía* es aquella cadena que presenta cero apariciones de símbolos. Esta cadena, designada por $\varepsilon$, es una cadena que puede construirse en cualquier alfabeto.

## Longitud de una cadena

Suele ser útil clasificar las cadenas por su *longitud*, es decir, el número de posiciones ocupadas por símbolos dentro de la cadena. Por ejemplo, 01101 tiene una longitud de 5. Es habitual decir que la longitud de una cadena es igual al "número de símbolos" que contiene; esta proposición está aceptada coloquialmente, sin embargo, no es estrictamente correcta. Así, en la cadena 01101 sólo hay dos símbolos, 0 y 1, aunque tiene cinco *posiciones* para los mismos y su longitus es igual a 5. Sin embargo, generalmente podremos utilizar la expresión "número de símbolos" cuando realmente a lo que se está haciendo referencia es al "número de posiciones".

La notación estándar para indicar la longitud de una cadena $w$ es $|w|$. Por ejemplo, $|011| = 3$ y $|\varepsilon| = 0$.

## Potencias de un alfabeto

Si $\Sigma$ es un alfabeto, podemos expresar el conjunto de todas las cadenas de una determinada longitud de dicho alfabeto utilizando una notación exponencial. Definimos $\Sigma^k$ para que sea el conjunto de las cadenas de longitud $k$, tales que cada uno de los símbolos de las mismas pertenece a $\Sigma$.

**EJEMPLO 1.24**

Observe que $\Sigma^0 = \{\varepsilon\}$, independientemente de cuál sea el alfabeto $\Sigma$. Es decir, $\varepsilon$ es la única cadena cuya longitud es 0.

Si $\Sigma = \{0,1\}$, entonces $\Sigma^1 = \{0,1\}$, $\Sigma^2 = \{00,01,10,11\}$, $\Sigma^3 = \{000,001,010,011,100,101,110,111\}$, etc.

Observe que existe una ligera confusión entre $\Sigma$ y $\Sigma^1$. Lo primero es un alfabeto; sus elementos 0 y 1 son los símbolos. Lo segundo es un conjunto de cadenas; sus elementos son las cadenas 0 y 1, cuya longitud es igual a 1. No vamos a utilizar notaciones diferentes para los dos conjuntos, confiando en que el contexto deje claro si $\{0,1\}$ o algún otro conjunto similar representa un alfabeto o un conjunto de cadenas. □

Por convenio, el conjunto de todas las cadenas de un alfabeto $\Sigma$ se designa mediante $\Sigma^*$. Por ejemplo, $\{0,1\}^* = \{\varepsilon,0,1,00,01,10,11,000,\ldots\}$. Expresado de otra forma,

$$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \cdots$$

En ocasiones, desearemos excluir la cadena vacía del conjunto de cadenas. El conjunto de cadenas no vacías del alfabeto $\Sigma$ se designa como $\Sigma^+$. Por tanto, dos equivalencias apropiadas son:

- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \cdots$.
- $\Sigma^* = \Sigma^+ \cup \{\varepsilon\}$.

### Concatenación de cadenas

Sean $x$ e $y$ dos cadenas. Entonces, $xy$ denota la *concatenación* de $x$ e $y$, es decir, la cadena formada por una copia de $x$ seguida de una copia de $y$. Dicho de manera más precisa, si $x$ es la cadena compuesta por $i$ símbolos $x = a_1a_2\cdots a_i$ e $y$ es la cadena compuesta por $j$ símbolos $y = b_1b_2\cdots b_j$, entonces $xy$ es la cadena de longitud $i+j$: $xy = a_1a_2\cdots a_ib_1b_2\cdots b_j$.

**EJEMPLO 1.25**

Sean $x = 01101$ e $y = 110$. Entonces $xy = 01101110$ e $yx = 11001101$. Para cualquier cadena $w$, tenemos las ecuaciones $\varepsilon w = w\varepsilon = w$. Es decir, $\varepsilon$ es el *elemento neutro de la concatenación*, dado que su concatenación con cualquier cadena proporciona dicha cadena como resultado (del mismo modo que 0, el elemento neutro de la suma, puede sumarse a cualquier número $x$ y proporciona $x$ como resultado). □

## 1.5.3 Lenguajes

Un conjunto de cadenas, todas ellas seleccionadas de un $\Sigma^*$, donde $\Sigma$ es un determinado alfabeto se denomina *lenguaje*. Si $\Sigma$ es un alfabeto y $L \subseteq \Sigma^*$, entonces $L$ es un *lenguaje de* $\Sigma$. Observe que un lenguaje de $\Sigma$ no necesita incluir cadenas con todos los símbolos de $\Sigma$, ya que una vez que hemos establecido que $L$ es un lenguaje de $\Sigma$, también sabemos que es un lenguaje de cualquier alfabeto que sea un superconjunto de $\Sigma$.

La elección del término "lenguaje" puede parecer extraña. Sin embargo, los lenguajes habituales pueden interpretarse como conjuntos de cadenas. Un ejemplo sería el inglés, donde la colección de las palabras correctas inglesas es un conjunto de cadenas del alfabeto que consta de todas las letras. Otro ejemplo es el lenguaje C, o cualquier otro lenguaje de programación, donde los programas correctos son un subconjunto de las posibles cadenas que pueden formarse a partir del alfabeto del lenguaje. Este alfabeto es un subconjunto de los caracteres ASCII. El alfabeto en concreto puede diferir ligeramente entre diferentes lenguajes de programación, aunque generalmente incluye las letras mayúsculas y minúsculas, los dígitos, los caracteres de puntuación y los símbolos matemáticos.

Sin embargo, existen también otros muchos lenguajes que veremos a lo largo del estudio de los autómatas. Algunos ejemplos son los siguientes:

1. El lenguaje de todas las cadenas que constan de $n$ ceros seguidos de $n$ unos para cualquier $n \geq 0$: $\{\varepsilon, 01, 0011, 000111, \ldots\}$.
2. El conjunto de cadenas formadas por el mismo número de ceros que de unos:

$$\{\varepsilon, 01, 10, 0011, 0101, 1001, \ldots\}$$

3. El conjunto de números binarios cuyo valor es un número primo:

$$\{10, 11, 101, 111, 1011, \ldots\}$$

4. $\Sigma^*$ es un lenguaje para cualquier alfabeto $\Sigma$.
5. $\emptyset$, el lenguaje vacío, es un lenguaje de cualquier alfabeto.
6. $\{\varepsilon\}$, el lenguaje que consta sólo de la cadena vacía, también es un lenguaje de cualquier alfabeto. Observe que $\emptyset \neq \{\varepsilon\}$; el primero no contiene ninguna cadena y el segundo sólo tiene una cadena.

La única restricción importante sobre lo que puede ser un lenguaje es que todos los alfabetos son finitos. De este modo, los lenguajes, aunque pueden tener un número infinito de cadenas, están restringidos a que dichas cadenas estén formadas por los símbolos que definen un alfabeto finito y prefijado.

### Definición de lenguajes mediante descripciones de conjuntos

Es habitual describir un lenguaje utilizando una "descripición de conjuntos":

$$\{w \mid \text{algo acerca de } w\}$$

Esta expresión se lee "el conjunto de palabras $w$ tal que (lo que se dice acerca de $w$ a la derecha de la barra vertical)". Algunos ejemplos son:

1. $\{w \mid w$ consta de un número igual de ceros que de unos $\}$.
2. $\{w \mid w$ es un entero binario que es primo $\}$.
3. $\{w \mid w$ es un programa C sintácticamente correcto $\}$.

También es habitual reemplazar $w$ por alguna expresión con parámetros y describir las cadenas del lenguaje estableciendo condiciones sobre los parámetros. He aquí algunos ejemplos; el primero con el parámetro $n$ y el segundo con los parámetros $i$ y $j$:

1. $\{0^n1^n \mid n \geq 1\}$. Esta expresión se lee: "El conjunto de 0 a la $n$ 1 a la $n$ tal que $n$ es mayor o igual que 1", este lenguaje consta de las cadenas $\{01, 0011, 000111, \ldots\}$. Observe que, como con los alfabetos, podemos elevar un solo símbolo a una potencia $n$ para representar $n$ copias de dicho símbolo.
2. $\{0^i1^j \mid 0 \leq i \leq j\}$. Este lenguaje consta de cadenas formadas por ceros (puede ser ninguno) seguidos de al menos el mismo número de unos.

## 1.5.4 Problemas

En la teoría de autómatas, un *problema* es la cuestión de decidir si una determinada cadena es un elemento de un determinado lenguaje. Como veremos, cualquier cosa que coloquialmente denominamos "problema" podemos expresarlo como elemento de un lenguaje. De manera más precisa, si $\Sigma$ es un alfabeto y $L$ es un lenguaje de $\Sigma$, entonces el problema $L$ es:

- Dada una cadena $w$ de $\Sigma^*$, decidir si $w$ pertenece o no a $L$.

**EJEMPLO 1.26**

El problema de comprobar si un número es primo se puede expresar mediante el lenguaje $L_p$, que consta de todas las cadenas binarias cuyo valor como número binario se corresponde con un número primo. Es decir, dada una cadena de 0s y 1s, decimos que "sí" si la cadena es la representación binaria de un número primo y decimos que "no" en caso contrario. Para ciertas cadenas, esta decisicón será sencilla. Por ejemplo, 0011101 no puede ser la representación de un número primero por la sencilla razón de que todos los enteros, excepto el 0, tienen una representación binaria que comienza por 1. Sin embargo, es menos obvio si la cadena 11101 pertenece al lenguaje $L_p$, por lo que cualquier solución a este problemas tendrá que emplear recursos de computación significativos de algún tipo: tiempo y/o espacio, por ejemplo. □

Un aspecto potencialmente insatisfactorio de nuestra definición de "problema" es que normalmente no se piensa en un problema como en una cuestión de toma de decisiones (¿es o no verdadero lo siguiente?) sino como en una solicitud de cálculo o de transformación de una entrada dada (hallar la mejor forma de llevar a

### ¿Es un lenguaje o un problema?

Lenguajes y problemas son realmente la misma cosa. El término que empleemos depende de nuestro punto de vista. Cuando sólo nos interesan las cadenas, por ejemplo, en el conjunto $\{0^n1^n \mid n \geq 1\}$, entonces tendemos a pensar en el conjunto de cadenas como en un lenguaje. En los últimos capítulos del libro, tenderemos a asignar "semánticas" a las cadenas, por ejemplo, pensaremos en ellas como en grafos codificados, expresiones lógicas o incluso enteros. En dichos casos, estaremos más interesados en lo que representan las cadenas que en las mismas, tenderemos a pensar que el conjunto de cadenas es un problema.

cabo una tarea). Por ejemplo, la tarea de analizar sintácticamente en un compilador C puede interpretarse como un problema en sentido formal, donde se proporciona una cadena ASCII y se solicita que se decida si la cadena es o no un elemento de $L_c$, el conjunto de programas C válidos. Sin embargo, el analizador hace más cosas además de tomar decisiones. Genera un árbol de análisis, anotaciones en una tabla de símbolos y posiblemente más cosas. Más aún, el compilador como un todo resuelve el problema de convertir un programa en C en código objeto de una máquina, lo que está muy lejos de simplemente responder "sí" o "no" acerca de la validez del programa.

No obstante, la definición de "problema" como lenguaje ha resistido la prueba del tiempo como la forma apropiada de tratar con la importante cuestión de la teoría de la complejidad. En esta teoría, lo que nos interesa es determinar los límites inferiores de la complejidad de determinados problemas. Especialmente importantes son las técnicas que demuestran que ciertos problemas no se pueden resolver en un periodo de tiempo menor que una exponencial del tamaño de la entrada. Resulta que la versión sí/no, basada en el lenguaje, de problemas conocidos resulta ser igual de difícil en este sentido que las versiones "resuélvelo".

Es decir, si podemos demostrar que es difícil decidir si una determinada cadena pertenece al lenguaje $L_X$ de cadenas válidas del lenguaje de programación $X$, entonces será lógico que no sea fácil traducir los programas en el lenguaje $X$ a código objeto. Si fuera sencillo generar el código correspondiente, entonces podríamos ejecutar el traductor y concluir que la entrada era un elemento válido de $L_X$, siempre que el traductor tuviera éxito en producir el código objeto. Dado que el paso final de determinar si se ha generado el código objeto no puede ser complicado, podemos utilizar el algoritmo rápido que genera el código objeto para decidir acerca de la pertenencia a $L_X$ de forma eficiente. De este modo se contradice la suposición de que es díficil demostrar la pertenencia a $L_X$. Disponemos de una demostración por reducción al absurdo de la proposición "si probar la pertenencia a $L_X$ es difícil, entonces compilar programas en el lenguaje de programación $X$ es difícil".

Esta técnica que demuestra que un problema es difícil utilizando su algoritmo supuestamente eficiente, para resolver otro problema que se sabe que es difícil se conoce como "reducción" del segundo problema al primero. Es una herramienta fundamental en el estudio de la complejidad de los problemas y se aplica con mucha más facilidad a partir de la idea de que los problemas son cuestiones acerca de la pertenencia a un lenguaje, en lugar de cuestiones de tipo más general.

## 1.6 Resumen del Capítulo 1

- ✦ *Autómatas finitos*. Los autómatas finitos utilizan estados y transiciones entre estados en respuesta a las entradas. Resultan útiles para construir diversos tipos de software, incluyendo el componente de análisis léxico de un compilador y los sistemas que permiten verificar la corrección de, por ejemplo, circuitos o protocolos.

- *Expresiones regulares*. Definen una notación estructural que permite describir los mismos patrones que se pueden representar mediante los autómatas finitos. Se emplean en muchos tipos comunes de software, incluyendo herramientas para la búsqueda de patrones, por ejemplo, en textos o en nombres de archivo.

- *Gramáticas independientes del contexto*. Definen una importante notación para describir la estructura de los lenguajes de programación y de los conjuntos relacionados de cadenas de caracteres; resultan útiles en la construcción del analizador sintáctico de un compilador.

- *Máquinas de Turing*. Son autómatas que modelan la potencia de las computadoras reales. Nos permiten estudiar la decidibilidad, es decir, el problema de qué puede o no puede hacer una computadora. También nos permiten distinguir los problemas tratables (aquellos que pueden resolverse en un tiempo polinómico) de los problemas intratables (los que no se pueden resolver en un tiempo polinómico).

- *Demostraciones deductivas*. Este método básico de demostración se basa en la construcción de listas de proposiciones que o bien son verdaderas o bien se deducen lógicamente de proposiciones anteriores.

- *Demostración de proposiciones Si-entonces*. Muchos teoremas son de la forma "si (algo) entonces (alguna otra cosa)". La proposición o proposiciones que siguen a la parte "si" son las hipótesis y las que siguen a la parte "entonces" es la conclusión. Las demostraciones deductivas de las proposiciones si-entonces comienzan con la hipótesis y continúan con proposiciones que se deducen lógicamente a partir de la hipótesis y de las proposiciones anteriores hasta que se demuestra la conclusión como una proposición más.

- *Demostración de proposiciones Si-y-sólo-si*. Existen otros teoremas de la forma "(algo) si y sólo si (alguna otra cosa)". Se demuestran probando las proposiciones si-entonces en ambos sentidos. Un tipo similar de teorema establece la igualdad de los conjuntos descritos de dos formas diferentes; se demuestran probando que cada uno de los dos conjuntos está contenido en el otro.

- *Demostración de la conversión contradictoria*. En ocasiones, es más fácil demostrar una proposición de la forma "si $H$ entonces $C$" demostrando la proposición equivalente: "si no $C$ entonces no $H$". Esta última se conoce como conversión contradictoria de la primera.

- *Demostración por reducción al absurdo*. En otros casos, es más conveniente demostrar la proposición "si $H$ entonces $C$" demostrando "si $H$ y no $C$ entonces (algo que sabemos que es falso)". Una demostración de este tipo se denomina demostración por reducción al absurdo.

- *Contraejemplos*. Algunas veces nos pedirán que demostremos que una determinada proposición no es verdadera. Si la proposición tiene uno o más parámetros, entonces podemos demostrar que es falsa proporcionando un único contraejemplo, es decir, una asignación de valores a los parámetros que hace que la proposición sea falsa.

- *Demostraciones inductivas*. Una proposición que tiene un parámetro entero $n$ a menudo puede demostrarse por inducción sobre $n$. Se demuestra que la proposición es verdadera para el caso base, un número finito de casos para valores concretos de $n$, y luego se demuestra el paso de inducción: si la proposición es verdadera para todos los valores hasta $n$, entonces es verdadera para $n + 1$.

- *Inducciones estructurales*. En algunas situaciones, incluyendo muchas de las de este libro, el teorema que se va a demostrar inductivamente es acerca de algo definido de forma recursiva, como en el caso de los árboles. Podemos demostrar un teorema acerca de objetos construidos por indución sobre el número de pasos utilizados en su construcción. Este tipo de inducción se conoce como estructural.

- *Alfabetos*. Un alfabeto es cualquier conjunto finito de símbolos.

- *Cadenas de caracteres*. Una cadena es una secuencia de símbolos de longitud finita.

- *Lenguajes y problemas*. Un lenguaje es un conjunto (posiblemente infinito) de cadenas, donde los símbolos símbolo de todas ellas se han seleccionado de un determinado alfabeto. Cuando las cadenas de un lenguaje se interpretan de alguna manera, la cuestión de si una cadena pertenece o no al lenguaje se dice, en ocasiones, que es un problema.

## 1.7 Referencias del Capítulo 1

Para ampliar el material de este capítulo, incluyendo los conceptos matemáticos que subyacen a las Ciencias de la Computación, recomendamos [1].

1. A. V. Aho y J. D. Ullman, *Foundations of Computer Science*, Computer Science Press, Nueva York, 1994.