

# Sistemas Expertos

## Principios y Programación



Tercera edición



Giarratano · Riley

# CAPÍTULO 1

## *Introducción a los sistemas expertos*

### 1.1 INTRODUCCIÓN

Este capítulo desarrolla una amplia introducción a los sistemas expertos, aquí se presentan sus principios fundamentales, se analizan sus ventajas y desventajas y se describen las áreas de aplicación apropiadas para ellos. También se analiza su relación con otros métodos de programación.

### 1.2 ¿QUÉ ES UN SISTEMA EXPERTO?

El primer paso para resolver cualquier problema es definir el área o **dominio** del problema que será resuelto; esta consideración es tan cierta para la inteligencia artificial (AI, artificial intelligence) como en la programación convencional. Sin embargo, a causa de la mística antes asociadas con la AI, hay una prolongada tendencia a seguir creyendo en el viejo adagio: "Si todavía no ha sido resuelto es un problema de AI"; otra definición popular es: "la AI está haciendo que las computadoras actúen como en las películas". Tal vez este tipo de cuadro mental haya sido común en los años setenta, cuando la AI se encontraba en plena fase de investigación, pero, hoy en día la AI resuelve muchos problemas reales y tiene muchas aplicaciones comerciales.

Aunque no se han encontrado soluciones generales a los problemas clásicos de la AI, como la traducción del lenguaje natural, la comprensión del habla y la visión, restringir el dominio del problema ayudaría a producir una solución útil. Por ejemplo, no es difícil construir sistemas sencillos de lenguaje natural si la entrada se restringe a oraciones de la forma sustantivo, verbo y predicado. Actualmente este tipo de sistemas funcionan bien para proporcionar una interfaz amigable con el usuario en muchos productos de software, como sistemas de base de datos y hojas de cálculo. De hecho, los analizadores gramaticales relacionados con los populares juegos de aventuras para computadora, basados en texto, exhiben un sorprendente grado de habilidad en la comprensión de lenguaje natural.

Como se muestra en la figura 1.1, la AI tiene muchas áreas de interés. El área de **sistemas expertos** es una aproximación muy exitosa a la solución de los problemas clásicos de AI en la programación de inteligencia. El profesor Edward Feigenbaum de la Universidad de Stanford, pionero en la tecnología de los sistemas expertos, los ha definido como "un programa de computación inteligente que usa el conocimiento y los procedimientos de inferencia para resolver problemas que son lo suficientemente difíciles como para requerir significativa ex-

periencia humana para su solución” (Feigenbaum 82). Es decir, un sistema experto es un sistema de cómputo que **emula** la habilidad de tomar decisiones de un especialista humano. El término *emular* significa que el sistema experto tiene el objetivo de actuar en todos los aspectos como un especialista humano. Una emulación es mucho más fuerte que una simulación, que en algunos aspectos sólo requiere que se actúe como en la realidad.

A pesar de que una solución a problemas de propósito general aún nos elude, los sistemas expertos funcionan bien en sus dominios restringidos. Como prueba de su éxito, sólo se necesita observar las muchas aplicaciones que tienen hoy en los negocios, la medicina, la ciencia y la ingeniería, además de todos los libros, diarios, conferencias y productos dedicados a ellos.

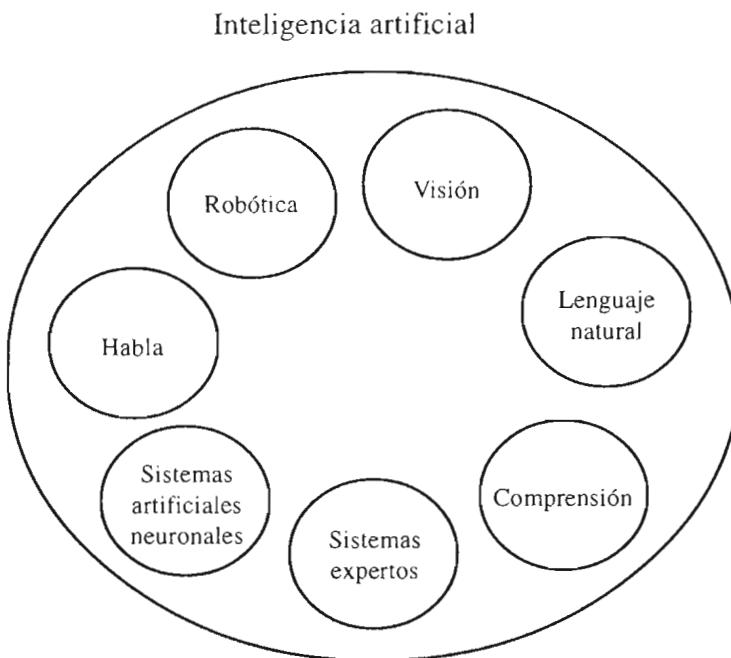
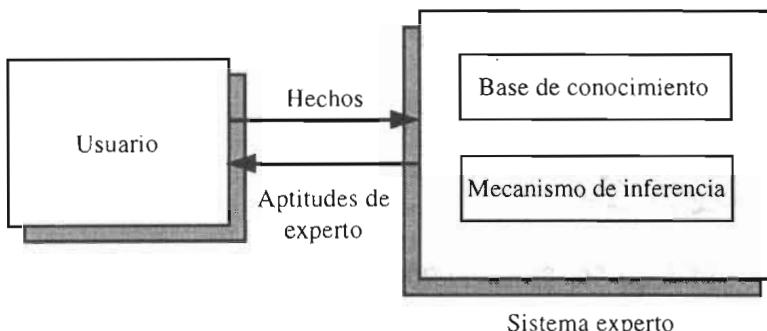


Figura 1.1 Algunas áreas de inteligencia artificial

Los sistemas expertos son una rama de la AI que hace un amplio uso del conocimiento especializado para resolver problemas como un **especialista** humano. Éste es una persona que tiene **experiencia desarrollada** en cierta área. Esto es, el especialista tiene conocimientos o habilidades especiales que la mayoría no conoce o de las que no dispone; puede resolver problemas que la mayoría no podría resolver, o los resuelve con mucha mayor eficiencia (y a otro costo). Cuando los sistemas expertos se desarrollaron por primera vez en los años setenta, contenían exclusivamente conocimiento experto; sin embargo, hoy en día a menudo se aplica el término *sistema experto* a cualquier sistema que utiliza tecnología de sistemas expertos. Esta tecnología puede incluir a los lenguajes y programas especiales de sistemas expertos, además del hardware diseñado para ayudar en su desarrollo y ejecución.

El conocimiento de los sistemas expertos puede obtenerse por experiencia o consulta de los conocimientos que suelen estar disponibles en libros, revistas y con personas capacitadas. Los términos *sistema experto*, *sistema basado en conocimiento*, o *sistema experto basado en conocimiento*, se usan como sinónimos. La mayoría utiliza el término sistema experto tan sólo porque es más corto, a pesar de que quizás no tengan experiencia en su sistema, sólo conocimiento general.

En la figura 1.2 se ilustra el concepto básico de un sistema experto basado en conocimiento. El usuario aporta los hechos u información al sistema experto y recibe consejo o experiencia como respuesta. En su interior, el sistema experto incluye dos componentes principales. La base de conocimiento contiene el conocimiento que le permite al **mecanismo de inferencia** sacar conclusiones; éstas son las respuestas del sistema experto a la consulta especializada del usuario.



**Figura 1.2 Conceptos básicos del funcionamiento de un sistema experto**

También se han diseñado excelentes sistemas basados en conocimientos que funcionan como el asistente inteligente de un especialista humano. Estos asistentes inteligentes están diseñados con tecnología de sistemas expertos debido a sus ventajas de desarrollo. Cuanto más conocimiento se añada a un asistente inteligente, actuará más como un especialista. Por tanto, desarrollar un asistente inteligente puede constituir un útil acontecimiento en la producción de un sistema experto completo. Además, es posible que al acelerar la solución de problemas proporcione más tiempo disponible a los especialistas. Los tutores inteligentes son otra nueva aplicación de la inteligencia artificial; a diferencia de los viejos sistemas de instrucción asistida por computadora, el nuevo sistema puede proporcionar instrucciones sensibles al contexto (Giarratano 91a).

En oposición a las técnicas de solución general de problemas el conocimiento de un especialista se centra específicamente en **dominio de problema**. Un dominio de problema es el área específica de problema, como medicina, finanzas, ciencias, ingeniería, etc., en el que un especialista puede resolver problemas con facilidad. Los sistemas expertos, como los especialistas humanos, suelen diseñarse especializados en un dominio de problemas. Por ejemplo, normalmente no se esperaría que un especialista en ajedrez tuviera conocimientos especializados en medicina. Las experiencias en un dominio de problema no llevan automáticamente a otro.

Al conocimiento del especialista para resolver problemas específicos se le llama **dominio de conocimiento** del experto. Por ejemplo, un sistema experto médico, diseñado para diagnosticar enfermedades infecciosas debe tener una gran cantidad de conocimiento acerca de los síntomas causados por este tipo de enfermedades. En este caso, el dominio del conocimiento es la medicina y consta del conocimiento de las enfermedades, sus síntomas y tratamientos. En la figura 1.3 se ilustra la relación entre el problema y el dominio de conocimiento. Observe que éste está completamente incluido dentro del dominio del problema. La porción exterior al dominio del conocimiento simboliza un área en que no existe conocimiento acerca de todos los problemas.

Un sistema experto médico normalmente no tiene conocimientos acerca de otras ramas de la medicina, como cirugía o pediatría. A pesar de que su conocimiento de las enfermedades

infecciosas es equivalente al de un especialista humano, el sistema experto no sabría nada de otros dominios de conocimiento a menos que estuviera programado con ellos.



**Figura 1.3 Un probable problema y su relación con el dominio del conocimiento**

En su dominio de conocimiento, el sistema experto razona o hace **inferencias** de la misma forma en que un especialista humano inferiría la solución de un problema. Esto es, dados algunos hechos, se infiere una conclusión. Por ejemplo, si su cónyuge no le ha hablado durante un mes, usted puede deducir que no ha tenido nada importante que decir, sin embargo, ésta es sólo una de las muchas inferencias posibles.

Como con cualquier nueva tecnología, existe todavía mucho que aprender acerca de los sistemas expertos. En la tabla 1.1 se resumen las diferentes perspectivas de quienes participan en una tecnología. En esta tabla, el tecnólogo puede ser un ingeniero o un diseñador de software y la tecnología puede ser hardware o software. Al resolver cualquier problema, éstas son preguntas que deben responderse o la tecnología no podrá usarse con éxito. Como cualquier otra herramienta, los sistemas expertos tienen aplicaciones apropiadas e inapropiadas. A medida que crezca nuestra experiencia al respecto, descubriremos cuáles son estas aplicaciones.

Persona	Pregunta
Gerente	¿Para qué puedo usarla?
Tecnólogo	¿Cómo puedo implantarla mejor?
Investigador	¿Cómo puedo ampliarla?
Consumidor	¿Cómo me ayudará? ¿Merece el trabajo y el costo? ¿Qué tan confiable es?

**Tabla 1.1 Diferentes visiones de la tecnología**

## 1.3 VENTAJAS DE LOS SISTEMAS EXPERTOS

Los sistemas expertos tiene varias características atractivas:

- *Mayor disponibilidad.* La experiencia está disponible para cualquier hardware de cómputo adecuado. En un sentido muy real, un sistema experto es la producción masiva de experiencia.

- *Costo reducido.* El costo de poner la experiencia a disposición del usuario se reduce enormemente.
- *Peligro reducido.* Los sistemas expertos pueden usarse en ambientes que podrían ser peligrosos para un ser humano.
- *Permanencia.* La experiencia es permanente. A diferencia de los especialistas humanos, que pueden retirarse, renunciar o morir, el conocimiento del sistema experto durará indefinidamente.
- *Experiencia múltiple.* El conocimiento de varios especialistas puede estar disponible para trabajar simultánea y continuamente en un problema, a cualquier hora del día o de la noche. El nivel de experiencia combinada de muchos sistemas expertos puede exceder el de un solo especialista humano (Harmon 85).
- *Mayor confiabilidad.* Al proporcionar una segunda opinión, los sistemas expertos incrementan la confianza en que un especialista ha tomado la decisión correcta o al dar un voto de calidad en caso de desacuerdos entre varios especialistas. Por supuesto, este método probablemente no funcionará si uno de ellos fue quien programó al sistema. Ambos deben coincidir siempre, a menos que el especialista haya cometido un error, lo que puede suceder si estaba cansado o bajo presión.
- *Explicación.* El sistema experto puede explicar clara y detalladamente el razonamiento que conduce a una conclusión, lo que aumenta la confianza en que se tomó la decisión correcta. Un ser humano puede estar demasiado cansado, mostrarse renuente o ser incapaz de hacerlo siempre.
- *Respuesta rápida.* Tal vez sea necesaria una respuesta rápida, o en tiempo real, para ciertas aplicaciones. Dependiendo del software y hardware usado, un sistema experto puede responder más rápido y estar más dispuesto que un especialista. Algunas situaciones de emergencia pueden exigir respuestas más rápidas que las de un humano, de modo que un sistema experto en tiempo real resulta una buena elección (Hugh 88; Ennis 86).
- *Respuestas sólidas, completas y sin emociones, en todo momento.* Esto puede ser muy importante en tiempo real y en situaciones de emergencia, cuando un especialista quizás no funcionaría a toda su capacidad a causa de la presión y la fatiga.
- *Tutoría inteligente.* El sistema experto puede actuar como un tutor inteligente, dejando que el estudiante ejecute programas de ejemplo y explicando el razonamiento del sistema.
- *Base de datos inteligente.* Los sistemas expertos pueden usarse para tener acceso a una base de datos en forma inteligente (Kerschberg 86; Schur 88).

El proceso de desarrollo de un sistema experto también tiene un beneficio indirecto, dado que el conocimiento de los especialistas humanos debe disponerse clara y formalmente para introducirlo en la computadora. Como se dispone explícitamente del conocimiento, en vez de tenerlo implícito en la mente del especialista, puede examinarse para corregirlo, darle más consistencia y completarlo. El conocimiento puede entonces ajustarse o reexaminarse, lo que aumenta su calidad.

## 1.4 CONCEPTOS GENERALES DE LOS SISTEMAS EXPERTOS

El conocimiento de un sistema experto puede representarse de varias maneras (puede estar encapsulado en reglas y objetos). Un método común de representar el conocimiento es en forma de **reglas** tipo SI...ENTONCES, como:

SI la luz es roja ENTONCES deténgase

Si existe el hecho de que la luz sea roja, esto concuerda con el patrón "la luz es roja", la regla se satisfizo y se ejecuta la acción "deténgase". Aunque se trata de un ejemplo muy simple, se han construido muchos sistemas expertos significativos expresando en **reglas** el conocimiento de los especialistas. En realidad, el método basado en el conocimiento para desarrollar sistemas expertos ha reemplazado completamente la antigua propuesta de AI de los años cincuenta y sesenta, que trataba de utilizar sofisticadas técnicas de razonamiento que no dependían del conocimiento. Algunas herramientas de los sistemas expertos, como CLIPS, permiten **objetos** además de reglas. Las reglas pueden corresponder a los objetos y también a los hechos mientras que, los objetos pueden operar independientemente de las reglas.

Hoy en día se ha construido una gran variedad de sistemas expertos basados en el conocimiento. Grandes sistemas, que contienen miles de reglas, tales como el sistema XCON/R1 de Digital Equipment Corporation, saben mucho más que cualquier especialista acerca de cómo configurar sistemas de computadora (McDermott 84). También se han construido muchos sistemas pequeños para tareas especializadas, con varios cientos de reglas. Quizá estos pequeños sistemas no operan al nivel de un especialista, pero están diseñados para aprovechar la tecnología de los sistemas expertos y ejecutar tareas intensivas de conocimiento. El conocimiento para estos pequeños sistemas puede estar en libros, revistas u otros documentos del dominio público.

Por el contrario un sistema experto clásico abarca conocimiento no escrito que debe obtenerse de un especialista a través de extensas entrevistas con un **ingeniero del conocimiento**, durante un largo periodo. Al proceso de construir un sistema experto se le llama **ingeniería del conocimiento** (Michie 73), y consiste en la adquisición de conocimiento a partir de un especialista humano o de otra fuente y su codificación en el sistema experto.

Las etapas generales en el desarrollo de un sistema experto se ilustran en la figura 1.4. El ingeniero del conocimiento establece primero un diálogo con el especialista, con el fin de obtener su conocimiento. Esta etapa es análoga a la de un diseñador de sistemas en programación convencionales, cuando analiza los requisitos del sistema con el cliente para el que está construyendo el programa. El ingeniero en conocimiento codifica explícitamente el conocimiento en la base de conocimiento. El especialista evalúa entonces al sistema experto y ofrece una crítica al ingeniero del conocimiento. Este proceso se repite hasta que el experto juzga la actuación del sistema como satisfactoria.

La expresión **sistema basado en el conocimiento** es un mejor término para la aplicación de tecnología basada en el conocimiento, que puede usarse para la creación de sistemas expertos o basados en el conocimiento. Sin embargo, tal y como ocurre con el término inteligencia artificial, es común usar el término sistemas expertos cuando se refieren a ambos tipos de sistemas, aunque el conocimiento no esté al nivel de un especialista humano.

Los sistemas expertos suelen estar diseñados de manera distinta a los programas convencionales porque los problemas no tienen generalmente una solución algorítmica y dependen de inferencias para obtener una solución razonable, considerando ésta como la mejor que se puede esperar si no hay un algoritmo disponible que ayude a obtener la solución óptima. Como el sistema experto depende de la inferencia, debe ser capaz de explicar su razonamiento para que éste pueda verificarse. La **facilidad de explicación** es parte integral de los sistemas expertos sofisticados. De hecho deben diseñarse elementos de explicación elaborada que permitan al usuario la exploración de múltiples líneas de conjeturas "¿Qué pasaría si...?", o preguntas de **razonamiento hipotético**, e incluso la traducción de lenguaje natural a reglas.

Algunos sistemas expertos permiten incluso que el sistema aprenda reglas mediante el ejemplo, a través de la **inducción de reglas**, en la que el sistema crea reglas a partir de tablas de datos. No es fácil dar forma de reglas al conocimiento de los especialistas sobre todo cuando nunca se le ha explorado sistemáticamente. Puede haber inconsistencias, ambigüedades, duplicaciones u otros problemas que no se notan hasta que se trata de representar formalmente el conocimiento del experto en un sistema experto.

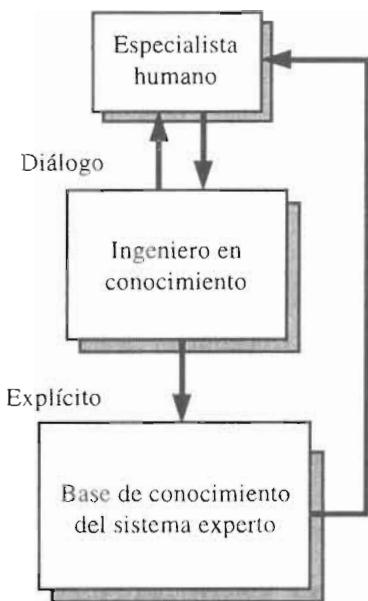


Figura 1.4 Desarrollo de un sistema experto

Los especialistas humanos también conocen el alcance de sus conocimientos y evalúan su consejo cuando el problema alcanza sus **límites de ignorancia**. Un especialista humano también sabe cuándo “romper las reglas”. A menos que los sistemas expertos se diseñen explícitamente para lidiar con la incertidumbre, harán recomendaciones con la misma confianza, aunque los datos con los que están tratando sean imprecisos o estén incompletos. El consejo de un sistema experto, como el de un especialista humano, se degrada suavemente en los límites de la ignorancia.

Hoy en día, una limitante práctica de muchos sistemas expertos es la carencia de **conocimiento causal**. Esto es, el sistema experto no comprende realmente las causas y los efectos fundamentales de un sistema. Es mucho más fácil programar sistemas expertos con **conocimiento superficial**, basado en el conocimiento empírico y heurístico, que con **conocimiento profundo**, basado en las estructuras básicas, funciones y comportamientos de los objetos. Por ejemplo, es mucho más fácil programar un sistema experto para que prescriba una aspirina para el dolor de cabeza de una persona, que programar todo el conocimiento bioquímico, fisiológico, anatómico y neurológico fundamentales acerca del cuerpo humano. La programación de un modelo causal del cuerpo humano sería una tarea enorme y, aunque tuviera éxito, probablemente la velocidad de respuesta sería demasiado lenta a causa de toda la información que tendría que procesar el sistema.

Una clase de conocimiento superficial es el **conocimiento heurístico** (*heurístico* viene del griego y significa “descubrir”). La heurística no es una garantía de éxito, como un algoritmo es una solución garantizada para un problema. En cambio, la heurística es un método práctico

o conocimiento empírico obtenido de la experiencia y que puede ayudar en la solución, pero que no garantiza su funcionamiento siempre. Sin embargo, en muchos campos como la medicina y la ingeniería, la heurística juega un papel esencial en algunos tipos de solución de problemas. Aunque se conozca una solución exacta, puede resultar poco práctica a causa de los costos o de las restricciones de tiempo. La heurística puede proporcionar atajos valiosos que reduzcan tiempo y dinero.

Otro problema con los sistemas expertos de hoy, es que su experiencia está limitada al dominio del conocimiento que maneja el sistema. Los sistemas expertos típicos no pueden generalizar su conocimiento usando la **analogía** para razonar acerca de nuevas situaciones, como lo hace la gente. A pesar de que la inducción de reglas ayuda, sólo algunos tipos limitados de conocimiento pueden agregarse al sistema de esta manera. La forma acostumbrada de construir un sistema experto, haciendo que el ingeniero en conocimiento repita el ciclo de entrevistar al especialista, construir un prototipo, probar, entrevistar, etcétera, es una tarea que requiere de mucho tiempo y trabajo. En realidad, este problema de transferir el conocimiento humano a un sistema experto es tan grande que se le llama el **cuello de botella de la adquisición del conocimiento**. Se trata de un término descriptivo porque el cuello de botella de la adquisición del conocimiento restringe la construcción de un sistema experto del mismo modo en que un cuello de botella restringe el flujo de líquido.

A pesar de sus limitaciones actuales, los sistemas expertos han tenido éxito al tratar con problemas del mundo real que los métodos de programación convencional habían sido incapaces de resolver, sobre todo aquellos que trataban con incertidumbre o con información incompleta. Lo más importante es estar informado de las ventajas y limitantes de esta nueva tecnología, para utilizarla apropiadamente.

## 1.5 CARACTERÍSTICAS DE UN SISTEMA EXPERTO

Un sistema experto suele diseñarse para que tenga las siguientes características generales:

- *Alto desempeño.* El sistema debe tener la capacidad de responder a un nivel de competencia igual o superior al de un especialista en el campo. Esto significa que la calidad del consejo dado por el sistema debe ser muy alta.
- *Tiempo de respuesta adecuado.* El sistema debe actuar en un tiempo razonable, comparable o mejor al tiempo requerido por un especialista, para alcanzar una decisión. Si un sistema experto necesita un año para tomar una decisión que un especialista tomaría en una hora, no sería muy útil. Las **restricciones de tiempo** en el desempeño de un sistema experto pueden ser especialmente severas en el caso de los sistemas en tiempo real, cuando una respuesta debe darse dentro de un intervalo.
- *Confiabilidad.* El sistema experto debe ser confiable y no propenso a “caídas”, o no será usado.
- *Comprendible.* El sistema debe ser capaz de explicar los pasos de su razonamiento mientras se ejecutan, de tal modo que sea comprendible. En lugar de ser sólo una “caja negra” que produce una respuesta milagrosa, el sistema debe tener capacidad de explicación, de la misma forma en que los especialistas pueden explicar su razonamiento. Este rasgo es muy importante por varias razones.

Una razón es que la vida humana puede depender de las respuestas del sistema experto. A causa de su gran potencial para perjudicar, todo sistema experto debe ser capaz de justificar sus conclusiones tal como un especialista humano puede explicar por qué se llegó a cierta

conclusión. Por tanto, una característica de explicación proporciona una revisión del razonamiento comprensible para los seres humanos.

Una segunda razón para tener un medio de explicación en un sistema experto se da durante la fase de desarrollo, para confirmar que el conocimiento ha sido adquirido y está siendo utilizado correctamente por el sistema. Esto es importante al depurar porque es posible que el conocimiento se introduzca con errores o que haya malentendidos entre el ingeniero del conocimiento y el especialista; un buen medio de explicación les permite verificar que el conocimiento sea correcto. Además, a causa de la forma en que están construidos los sistemas expertos comunes, es muy difícil leer un listado de un programa y entender su funcionamiento.

Otra fuente de errores pueden ser las interacciones imprevistas en el sistema experto, las que pueden detectarse ejecutando casos de prueba con razonamientos conocidos que el sistema debe seguir. Como se analizará posteriormente con más detalle, pueden aplicarse varias reglas a una situación dada sobre la cual está razonando el sistema. En un sistema experto el curso de la ejecución no es secuencial, de manera que no es posible simplemente leer su código línea tras línea y entender cómo opera. Es decir, el orden en que se introducen las reglas al sistema no es necesariamente el mismo en que se ejecutarán. El sistema experto actúa como un programa paralelo en que las reglas son procesadores de conocimiento independientes.

- *Flexibilidad.* Debido a la gran cantidad de conocimiento que un sistema experto puede tener, es importante contar con un mecanismo eficiente para añadir, modificar y eliminar conocimiento. Una razón de la popularidad de los sistemas basados en reglas es la capacidad de almacenaje eficiente y modular de las reglas.

Dependiendo del sistema, un mecanismo de explicación puede ser simple o elaborado. Uno simple, en un sistema basado en reglas, puede presentar una lista de todos los hechos que hicieron que la última regla se ejecutara. Los sistemas más elaborados pueden hacer lo siguiente:

- *Enumerar todas las razones a favor y en contra de una hipótesis en particular.* Una hipótesis es una proposición que debe probarse, tal como “el paciente tiene una infección por tétanos” en un sistema experto de diagnóstico médico. En un problema real puede haber varias hipótesis, de la misma manera en que el paciente puede tener varias enfermedades al mismo tiempo. Una hipótesis también puede verse como un hecho cuya verdad está en duda y debe probarse.
- *Enumerar todas las hipótesis que puedan explicar la evidencia observada.*
- *Explicar todas las consecuencias de una hipótesis.* Por ejemplo, suponiendo que el paciente tiene tétanos, debe haber también evidencia de fiebre a medida que la infección sigue su curso. Si este síntoma se observa, añade credibilidad a la hipótesis; si no se observa, reduce la credibilidad de la hipótesis.
- *Dar un pronóstico o predicción de lo que ocurrirá si la hipótesis es verdadera.*
- *Justificar las preguntas que el programa hace al usuario para obtener más información.* Estas preguntas pueden usarse para dirigir la línea de razonamiento hacia las rutas de diagnóstico probables. En la mayor parte de los problemas reales, es muy caro o toma mucho tiempo explorar todas las posibilidades y debe proporcionarse una guía para buscar la solución correcta. Por ejemplo, considerar el costo, el tiempo y el efecto de administrar todas las posibles pruebas médicas a un paciente que padece dolor de garganta.
- *Justificar el conocimiento del programa.* Por ejemplo, si el programa asegura que la hipótesis “el paciente tiene una infección por tétanos” es verdadera, el usu-

rio puede pedir una explicación. El programa puede justificar esta conclusión basándose en una regla que dice que si el paciente tiene una prueba de sangre positiva para tétanos, entonces el paciente tiene tétanos. Ahora, el usuario puede pedir al programa que justifique esta regla. El programa puede responder estableciendo que un examen de sangre positivo para una enfermedad es prueba de la enfermedad.

En este caso el programa está citando una **metarregla**, un conocimiento acerca de reglas (el prefijo *meta* significa “por encima” o “más allá”). Algunos programas, como Meta-DENDRAL, han sido creados específicamente para inferir nuevas reglas (Buchanan 78). El conocimiento justifica una hipótesis y éste, a su vez, se justifica con **una garantía** de que es correcto. Una garantía es, en esencia, una meta-explicación que esclarece la explicación dada por el sistema experto acerca de su razonamiento.

El conocimiento puede crecer fácilmente de manera acrecentada en un sistema basado en reglas. Esto quiere decir que la base de conocimiento puede crecer poco a poco a medida que se agregan reglas, de modo que pueden revisarse continuamente la ejecución y corrección del sistema. Si las reglas están diseñadas adecuadamente, las interacciones entre ellas se minimizarán o eliminarán como protección contra efectos inesperados. El crecimiento de conocimiento facilita la **construcción rápida de prototipos**, de modo que el ingeniero del conocimiento pueda mostrar pronto al especialista un prototipo de trabajo del sistema experto. Ésta es una característica importante porque mantiene el interés del especialista y de la administración en el proyecto. La construcción rápida de prototipos también muestra rápidamente las lagunas, inconsistencias o errores en el conocimiento del especialista o del sistema, de manera que puedan corregirse inmediatamente.

## 1.6 EL DESARROLLO DE LA TECNOLOGÍA DE SISTEMAS EXPERTOS

La AI tiene muchas ramas relacionadas con el habla, la visión, la robótica, la comprensión y aprendizaje del lenguaje natural y los sistemas expertos. Las raíces de los sistemas expertos abarcan muchas disciplinas; en particular, una de las raíces principales de los sistemas expertos es el área del procesamiento de la información humana, llamada **ciencia cognitiva**. La *cognición* es el estudio de la manera en que los humanos procesan la información. En otras palabras la cognición es el estudio de la manera en que piensa la gente, especialmente cuando resuelve problemas.

El estudio de la cognición es muy importante si se pretende lograr que las computadoras emulen a los especialistas humanos. A menudo, ellos no pueden explicar cómo resuelven problemas (simplemente las soluciones les llegan). Si el especialista no puede explicar cómo se soluciona un problema, no es posible codificar el conocimiento en un sistema experto basado en el conocimiento explícito. En este caso, la única posibilidad son los programas que aprenden por sí mismos a emular al especialista. Estos son programas basados en la inducción y en sistemas neuronales artificiales, que se analizarán más adelante.

### Solución humana de problemas y producciones

El desarrollo de la tecnología de sistemas expertos tiene amplios antecedentes. La tabla 1.2 es un breve resumen de algunos desarrollos importantes que han convergido en los modernos sistemas expertos. Cuando es posible, se dan las fechas de inicio de los proyectos, aun cuando muchos de ellos abarcaron varios años. Estos desarrollos se tratan con más detalle en este y

otros capítulos. La mejor referencia para todos los primeros sistemas la componen los tres volúmenes del *Manual de Inteligencia Artificial* (Feigenbaum 81).

Año	Evento
1943	Reglas de postproducción; Modelo neuronal de McCulloch y Pitts
1954	Algoritmo de Markov para controlar la ejecución de reglas
1956	Conferencia de Darmouth; Teórico lógico; Búsqueda Heurística; acuñación del término "AI"
1957	Perceptrón inventado por Rosenblatt; Inicia el GPS (General Problem Solver; Solucionador general de problemas) (Newell, Shaw y Simon)
1958	Lenguaje LISP de AI (McCarthy)
1962	Principios de neurodinámica ( <i>Principles of Neurodynamics</i> ) en la percepción, de Rosenblatt
1965	Método de solución para la comprobación automática de teoremas (Robinson)
	Lógica confusa para el razonamiento acerca de objetos confusos (Zadeh)
	Inicia el trabajo en DENDRAL, el primer sistema experto (Feigenbaum, Buchanan, et al.)
1968	Redes semánticas, modelo de memoria asociativa (Quillian)
1969	MACSYMA, sistema experto en matemáticas (Martin y Moses)
1970	Inicia el trabajo en PROLOG (Colmerauer, Rousell, et al.)
1971	HEARSAY I para reconocimiento del habla
	<i>Human Problem Solving</i> populariza las reglas (Newell y Simon)
1973	MYCIN, sistema experto para diagnóstico médico (Shortliffe, et. al.) que conduce a GUIDON, tutoría inteligente (Clancey)
	TEIRESIAS, concepto de mecanismo de explicación (Davis), y
	EMYCIN, primer shell (Van Melle, Shortliffe y Buchanan)
	HEARSAY II, modelo de pizarrón de cooperación múltiple entre especialistas.
1975	Marcos, representación del conocimiento (Minsky)
1976	AM (Artificial Mathematician, matemático artificial), descubrimiento creativo de conceptos matemáticos (Lenat)
	Teoría de la evidencia de Dempster-Shafer para el razonamiento bajo incertidumbre
	Inicia el trabajo en el sistema experto PROSPECTOR para la exploración mineral (Duda, Hart, et. al.)
1977	OPS shell (o armazón) de sistema experto (Forgy), usado en XCON/R1
1978	Inicia el trabajo en XCON/R1 (McDermott, DEC) para configurar los sistemas de computadora DEC
	Meta-DENDRAL, metarreglas e inducción de reglas (Buchanan)
1979	Algoritmo Rete para correspondencia rápida de patrones (Forgy).
	Inicia la comercialización de la AI
	Se forma Inference Corp. (libera ART, herramienta del sistema experto en 1985)
1980	Se funda Symbolics, LMI para fabricar máquinas LISP
1982	Sistemas expertos en matemáticas SMP; Hopfield Neural Net;
	Proyecto japonés de quinta generación para desarrollar computadoras inteligentes
1983	KEE, herramienta del sistema experto (InteliCorp)
1985	CLIPS herramienta del sistema experto (NASA)

Tabla 1.2 Algunos eventos importantes en la historia de los sistemas expertos

A finales de los años cincuenta y principios de los sesenta, se escribieron varios programas orientados a la solución de problemas en general. El más famoso de éstos fue el **Solucionador general de problemas**, creado por Newell y Simon y descrito en una serie de artículos que culminaron en el monumental trabajo de 920 páginas sobre la cognición, *Human Problem Solving* (Newell 72).

Uno de los resultados más significativos demostrados por Newell y Simon fue que gran parte de la solución humana de problemas o cognición puede expresarse con **reglas de pro-**

ducción del tipo SI...ENTONCES. Por ejemplo SI parece que va a llover ENTONCES lleve un paraguas, o SI su cónyuge está de mal humor ENTONCES no se muestre contento. Una regla corresponde a una pequeña colección modular de conocimiento llamada **fragmento**. Los fragmentos se organizan en una disposición suelta, con nexos hacia fragmentos de conocimiento relacionados. Una teoría sostiene que toda la memoria humana está organizada en fragmentos. Un ejemplo de una regla que representa un fragmento de conocimiento es:

SI el automóvil no avanza y  
el indicador de combustible está en vacío  
ENTONCES cargue gasolina

Newell y Simon popularizaron el uso de las reglas para representar al conocimiento humano y mostraron cómo puede razonarse con ellas. Los psicólogos cognitivos han usado reglas como modelos para explicar el procesamiento humano de la información. La idea básica es que la entrada sensorial proporciona estímulos al cerebro, el estímulo dispara las reglas apropiadas de la **memoria a largo plazo**, y ésta produce la respuesta oportuna. La memoria a largo plazo es dónde nuestra memoria está almacenada. Por ejemplo, todos tenemos reglas como:

SI hay una flama ENTONCES hay fuego  
SI hay humo ENTONCES puede haber fuego  
SI hay una sirena ENTONCES puede haber fuego

Observe que las últimas dos reglas no se expresan con total certeza, el fuego puede haberse apagado y aún haber humo en el aire. Del mismo modo, una sirena no prueba que haya fuego porque puede estar respondiendo a una falsa alarma. Los estímulos de ver las flamas, oler humo y escuchar una sirena dispararán éstos y otros tipos de reglas similares.

La memoria a largo plazo incluye muchas reglas que tienen la estructura simple SI...ENTONCES. De hecho, un gran maestro especialista en ajedrez puede tener 50,000 o más fragmentos de conocimiento acerca de los patrones del ajedrez. En contraste con la memoria a largo plazo, la **memoria a corto plazo** se utiliza para el almacenamiento temporal de conocimiento durante la solución del problema. A pesar de que la memoria a largo plazo puede guardar cientos de miles o más fragmentos, la capacidad de la memoria activa es sorprendentemente pequeña, de cuatro a siete fragmentos. Como un ejemplo sencillo de esto, trate de visualizar algunos números en su mente; la mayoría de las personas sólo pueden ver de cuatro a siete números al mismo tiempo, y también pueden memorizar mucho más de cuatro a siete números, sin embargo estos números se mantienen en la memoria a largo plazo.

Una teoría propone que la memoria a corto plazo representa el número de fragmentos que pueden estar activos de manera simultánea, y considera la solución humana de problemas como la diseminación de los fragmentos activados en la mente. Eventualmente, un fragmento puede activarse con tal intensidad que genera un pensamiento consciente y alguien se dice a sí mismo: "Hmm... algo se está quemando."

El otro elemento necesario para la solución humana de problemas es un **procesador cognitivo**, éste trata de encontrar las reglas que se activarán con el estímulo apropiado, pero no lo hará cualquier regla. Por ejemplo, usted no llenaría su tanque de gasolina cada vez que oye una sirena; solamente se activará una regla que corresponda al estímulo. Si se activan al mismo tiempo varias reglas, el procesador cognitivo debe ejecutar una solución de conflicto para decidir cuál tiene la prioridad más alta. Por ejemplo, esa regla se ejecutará si se activan estas dos:

SI hay fuego ENTONCES debo alejarme  
SI mis ropas se están quemando ENTONCES debo apagarlas

Entonces, las acciones de la regla con mayor prioridad, se ejecutarán antes que las otras. El **mecanismo de inferencia** de los sistemas expertos modernos corresponde al procesador cognitivo.

El modelo de Newell y Simon para la solución humana de problemas desde la perspectiva de la memoria a largo plazo (reglas), la memoria a corto plazo (memoria activa) y un procesador cognitivo (mecanismo de inferencia) es la base de los modernos sistemas expertos basados en reglas.

Reglas como éstas son una especie de **sistema de producción**. Hoy en día, los sistemas de producción basados en reglas son un método popular para implantar sistemas expertos. Las reglas individuales que comprenden un sistema de producción son las **reglas de producción**. Al diseñar un sistema experto, un factor importante es la cantidad de conocimiento o **granularidad** de las reglas. Muy poca granularidad dificulta la comprensión de una regla sin relación con otras. Demasiada granularidad hace que el sistema experto resulte muy difícil de modificar a causa de la gran cantidad de fragmentos entremezclados en una regla.

Hasta mediados de los sesenta, una de las principales búsquedas de la AI era producir sistemas inteligentes que dependieran poco del dominio de conocimiento y más de métodos de razonamiento poderosos. Incluso el nombre del Solucionador General de Problemas ilustra la concentración en máquinas que no estaban diseñadas para un dominio específico, sino que tenían el propósito de resolver muchas clases de problemas. A pesar de que los métodos de razonamiento usados por los solucionadores de problemas eran poderosos, las máquinas resultaban eternos principiantes. Cuando se presentaba un nuevo dominio, tenían que descubrir todo a partir de sus principios básicos y no eran tan buenos como los especialistas humanos, que dependían del dominio de conocimiento para acciones de alto nivel.

Un ejemplo del poder de conocimiento es el juego de ajedrez. A pesar de que las computadoras compiten ahora con los humanos, la gente juega bien, sin importar el hecho de que la computadora puede hacer cálculos millones de veces más rápido. Los estudios han demostrado (Chase 73) que los ajedrecistas expertos no tienen superpoderes de razonamiento, sino que dependen de su conocimiento de los patrones de las piezas de ajedrez adquirido con los años de juego. Como se mencionó anteriormente, se estima el conocimiento de un jugador experto de ajedrez en 50,000 patrones y los seres humanos son muy buenos para reconocer patrones como los de las piezas en un tablero de ajedrez. En lugar de tratar de razonar diez o veinte movimientos posibles para cada pieza por anticipado, como lo hace una computadora, el hombre analiza el juego a partir de patrones que revelan las amenazas a largo plazo mientras se mantiene alerta contra movimientos sorpresivos a corto plazo.

A pesar de que el conocimiento del dominio es poderoso, generalmente se limita al dominio. Por ejemplo, una persona que se convierte en ajedrecista experto, no se vuelve al mismo tiempo un experto para resolver problemas matemáticos, ni un experto en damas. Aunque parte del conocimiento se puede llevar a otro dominio, como la planeación cuidadosa de movimientos, ésta es una habilidad más que una pericia genuina.

A principios de los setenta se había hecho evidente que el conocimiento del dominio era la clave para construir máquinas que resolvieran problemas y que pudieran funcionar al nivel de los especialistas humanos. Aunque los métodos de razonamiento son importantes, los estudios han mostrado que los especialistas no dependen primordialmente del razonamiento para resolver un problema; en realidad, el razonamiento puede jugar un papel menor en la solución. En vez de esto, los expertos dependen de un vasto conocimiento de la heurística y de la experiencia adquirida con los años. Si un especialista no puede resolver un problema basándose en su pericia, entonces debe razonar partiendo de los principios básicos y la teoría (o, con mayor probabilidad, preguntar a otro perito). Generalmente la habilidad de razonamiento de un especialista no es mejor que la de la persona promedio al tratar con una situación que no le es familiar. Los primeros intentos por construir poderosos solucionadores de

problemas basados únicamente en el razonamiento, han demostrado que éstos están incompletos si dependen sólo del razonamiento.

El discernimiento de que el dominio del conocimiento era la clave para construir solucionadores de problemas reales, condujo al éxito de los sistemas expertos. Por tanto, los sistemas expertos que han tenido éxito hoy en día son sistemas basados en conocimiento, más que solucionadores de problemas generales. Además, la misma tecnología que condujo al desarrollo de los sistemas expertos, ha llevado también al desarrollo de sistemas basados en conocimiento que no necesariamente contienen experiencia humana.

Aunque a la experiencia se le considera conocimiento especializado y conocido sólo por unos pocos, el conocimiento suele encontrarse en libros, periódicos y otras recursos ampliamente disponibles. Por ejemplo, el conocimiento de cómo resolver una ecuación cuadrática o integrales y derivadas está ampliamente disponible. Se cuenta con programas de computadora basados en el conocimiento, como MACSYMA y SMP, para ejecutar de manera automática éstas y muchas otras operaciones matemáticas con operandos numéricos o simbólicos. Otros programas basados en conocimiento pueden ejecutar el proceso de control de plantas de manufactura. En la actualidad, los términos **programación basada en conocimiento** y *sistemas expertos* se utilizan como sinónimos. En realidad, los sistemas expertos se consideran ahora como un modelo de programación alternativo o **paradigma** para la programación algorítmica convencional.

## El ascenso de los sistemas basados en conocimiento

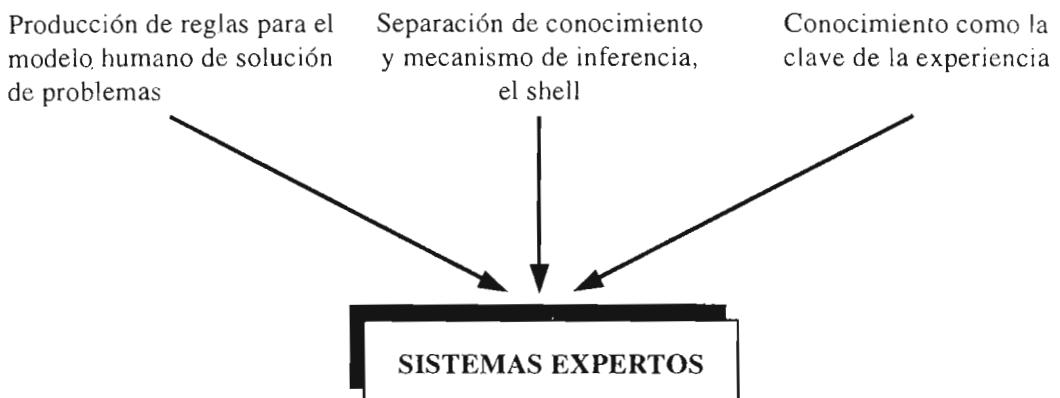
Con la aceptación del paradigma basado en conocimiento durante los setenta, se crearon con éxito varios prototipos de sistemas expertos. Estos sistemas podían interpretar espectrogramas de masas para identificar los componentes químicos (DENDRAL), diagnosticar enfermedades (MYCIN), analizar datos geológicos para la búsqueda de petróleo (DIPMETER) y minerales (PROSPECTOR), y configurar sistemas de cómputo (XCON/R1). La noticia de que PROSPECTOR descubrió un depósito de minerales con valor superior a los cien millones de dólares y de que XCON/R1 ahorraba a la Digital Equipment Corporation (DEC) millones de dólares al año, desató un sensacional interés en la nueva tecnología de los sistemas expertos en 1980. La rama de la AI que inició en los cincuenta como un estudio del procesamiento humano de la información, había crecido hasta alcanzar el éxito comercial por el desarrollo de programas prácticos para usarse en la vida real.

El sistema experto MYCIN fue importante por muchas razones. En primer lugar, demostró que la AI podía usarse en problemas prácticos reales. En segundo lugar, MYCIN fue el campo de pruebas para nuevos conceptos como el mecanismo de explicación, la adquisición automática de conocimientos y la tutoría inteligente, que se encuentran en muchos de los sistemas expertos actuales. La tercera razón por la que MYCIN fue importante es que demostró la viabilidad de la **shell** del sistema experto.

Los sistemas expertos previos, como el DENDRAL, eran sistemas únicos en los que la base de conocimiento estaba entremezclada con el software que aplicaba el conocimiento: el mecanismo de inferencia. MYCIN separó explícitamente la base de conocimiento del mecanismo de inferencia. Esto fue demasiado importante para el desarrollo de la tecnología de sistemas expertos porque significó que podía reutilizarse el núcleo esencial del sistema experto. O sea que un nuevo sistema experto podía construirse con mucha mayor rapidez que un sistema del tipo DENDRAL, vaciando el viejo conocimiento e introduciendo conocimiento acerca del nuevo dominio. La parte de MYCIN que trataba con la inferencia y la explicación, El shell, podía llenarse entonces con conocimiento acerca del nuevo sistema. A el shell obtenido al eliminar el conocimiento médico de MYCIN se le llamaba EMYCIN (MYCIN esencial o vacío).

Para finales de los setenta, habían convergido los tres conceptos básicos para la mayor parte de los sistemas expertos actuales, como se muestra en la figura 1.5. Estos conceptos son: las reglas, el shell y el conocimiento.

Para 1980, compañías nuevas comenzaron a sacar los sistemas expertos del laboratorio universitario y a fabricar productos comerciales; se introdujo nuevo software con múltiples opciones para el desarrollo de sistemas expertos, incluyendo la Herramienta Automática de Razonamiento (Automated Reasoning Tool, ART) de la Inference Corp., la Knowledge Engineering Tool (KEE) de IntelliCorp, y Rulemaster de Radian Corp. Además, se desarrolló nuevo hardware especializado para ejecutar el software con mayor velocidad que antes. Compañías como Symbolics y LMI introdujeron computadoras consideradas como máquinas LISP, porque estaban diseñadas para LISP, el lenguaje fundamental de base del software desarrollado para los sistemas expertos. En estas máquinas, el lenguaje ensamblador nativo, el sistema operativo y todos los demás códigos fundamentales se crearon en LISP.



**Figura 1.5**

**Convergencia de tres factores importantes en la creación del moderno sistema experto basado en reglas**

Desafortunadamente, esta alta tecnología también tiene un precio alto. Aunque una máquina LISP para un solo usuario era mucho más potente y productiva que una para propósitos generales, la máquina y la licencia de software para un solo usuario cuestan 100,000 dólares. Establecer un laboratorio de AI con media docena de programadores podría costar fácilmente 500,000 dólares.

Estas objeciones fueron sobrepasadas a mediados de los ochenta por la introducción del nuevo y potente software desarrollado, como CLIPS de la NASA. CLIPS está escrito en C para tener rapidez y movilidad, usa un poderoso acoplador de patrones llamado Algoritmo Rete, es gratuito para usuarios y contratistas gubernamentales; está disponible a un costo nominal para otros usuarios de COSMIC, que distribuye el software desarrollado por la NASA; las universidades pueden obtenerlo a mitad de precio. CLIPS está libre de derechos y cualquiera que obtenga una copia legítima de COSMIC puede distribuirlo gratuitamente.

CLIPS puede ser instalado en cualquier compilador C que soporte el lenguaje C de Kernigan y Richie. Ha sido instalado en PC de IBM y compatibles, VAX, Hewlett Packard, Sun, Cray y muchos otros fabricantes. COSMIC dispone de versiones para Windows de Macintosh y otras.

## 1.7 APPLICACIONES Y DOMINIOS DE LOS SISTEMAS EXPERTOS

Los programas de cómputo convencionales se utilizan para resolver muchos tipos de problemas. Por lo general, estos problemas tienen soluciones algorítmicas que se prestan bien a los programas y a los lenguajes de programación convencionales como FORTRAN, Pascal, Ada, etcétera. En muchas áreas de aplicación como los negocios y la ingeniería, los cálculos numé-

ricos son de importancia primordial. En contraste, los sistemas expertos están diseñados principalmente para el razonamiento simbólico.

Aunque lenguajes como LISP y PROLOG también se utilizan para el manejo simbólico, sirven a propósitos más generales que los shells de sistemas expertos. Esto no significa que no sea posible construir sistemas expertos en LISP y PROLOG, en realidad, muchos sistemas expertos han sido construidos con ellos. Sobre todo PROLOG tiene varias ventajas especiales para los sistemas de diagnóstico debido a su encadenamiento hacia atrás integrado. Sin embargo, es más conveniente y eficiente construir los grandes sistemas expertos con shells y programas con shell y equipo diseñados específicamente para ellos. En lugar de "reinventar la rueda" cada vez que se construye un nuevo sistema experto, es más eficiente utilizar las herramientas especializadas.

## Aplicaciones de los sistemas expertos

Los sistemas expertos se han aplicado casi a todos los campos del conocimiento. Algunos se han diseñado como herramientas de investigación, mientras que otros satisfacen importantes funciones de negocios e industriales. Un ejemplo de sistema experto usado en negocios de rutina es el sistema XCON de Digital Equipment Corporation (DEC). Este sistema llamado originalmente R1 fue desarrollado junto con John McDermott, de la Universidad Carnegie-Mellon. Y es un sistema experto para la configuración de sistemas de cómputo (McDermott 84).

La **configuración** de un sistema de cómputo significa que, cuando un consumidor hace un pedido, se suministran todas las partes correctas (software, hardware y documentación). Para los sistemas grandes, el sistema del consumidor se configura en la fábrica y se prueba para asegurar que cumple con los requisitos del cliente. A diferencia de la compra de un televisor o de una computadora para el hogar, un gran sistema de computo permite una gran cantidad de opciones e interconexiones. Al ensamblar un sistema grande, no basta con enviar el número solicitado de CPU, unidades de disco, terminales y demás; también deben proporcionarse las interconexiones y el cableado apropiados, además de revisar el sistema para verificar que funciona adecuadamente.

El sistema XCON es probablemente uno de los sistemas expertos de uso rutinario más exitoso y ahorra a la DEC millones de dólares al año, reduce el tiempo para configurar los pedidos y mejora su exactitud. XCON puede configurar una orden promedio en cerca de dos minutos, quince veces más rápido que un ser humano. Además, mientras que los humanos configuran las órdenes correctamente 70% de las veces, XCON tiene una eficiencia del 98%. Por esto, configurar un sistema de cómputo complejo en la fábrica constituye un gran esfuerzo. Es costoso configurar parcialmente un sistema para luego descubrir que no cumple con los requisitos del cliente, o que se necesitan otros componentes y que el envío debe retrasarse hasta que lleguen las piezas.

Cientos de sistemas expertos han sido construidos y comentados en publicaciones de cómputo, libros y conferencias. Tal vez esto sólo representa la punta del *iceberg*, porque muchas compañías y organizaciones militares no informarán de sus sistemas debido a las propiedades o secretos contenidos en el sistema. Basados en los sistemas descritos de la literatura pública, puede discernirse una amplia gama de aplicaciones para los sistemas expertos, como se muestra en la tabla 1.3.

De la tabla 1.4 a la 1.9 (Waterman 86), y en la bibliografía histórica al final de este capítulo, se muestran ejemplos de algunos sistemas expertos.

Clase	Área general
Configuración	Ensamblar correctamente los componentes apropiados de un sistema.
Diagnóstico	Inferir los problemas subyacentes basándose en la evidencia observada.
Instrucción	Enseñanza inteligente para que un estudiante pueda preguntar <i>Por qué, Cómo y Qué pasaría si</i> , como lo haría con un maestro humano.
Interpretación	Explicar los datos observados.
Supervisión	Comparar los datos observados con los esperados para juzgar el desempeño.
Planeación	Idear acciones para obtener el resultado deseado.
Pronóstico	Predecir el resultado de una situación dada.
Remedio	Prescribir tratamiento para un problema.
Control	Regular un proceso. Tal vez requiera de interpretación, diagnóstico, supervisión, planeación, pronóstico y remedios.

**Tabla 1.3 Gran cantidad de sistemas expertos**

Nombre	Química
CRYSTALIS	Interpretar la estructura tridimensional de una proteína.
DENDRAL	Interpretar la estructura molecular.
TQMSTUNE	Remediar el espectrómetro masivo triple y cuádruple Toms (mantenerlo afinado).
CLONER	Diseñar nuevas moléculas biológicas.
MOLGEN	Diseñar experimentos para clonación de genes.
SECS	Diseñar moléculas orgánicas complejas.
SPEX	Planear experimentos de biología molecular.

**Tabla 1.4 Sistemas expertos para la química**

Nombre	Electrónica
ACE	Diagnosticar fallas en las redes telefónicas.
JN-ATE	Diagnosticar fallas en el osciloscopio.
NDS	Diagnosticar la red de comunicaciones nacional.
EURISKO	Diseñar microelectrónica para tercera dimensión.
PALLADIO	Diseñar y probar nuevos circuitos VLSI.
REDESIGN	Rediseñar circuitos digitales.
CADHELP	Instruir para diseño apoyado por computadora.
SOPHIE	Instruir en diagnóstico de falla de circuitos.

**Tabla 1.5 Sistemas expertos en electrónica**

Nombre	Medicina
PUFF	Diagnosticar enfermedades de los pulmones.
VM	Supervisar pacientes en terapia intensiva.
ABEL	Diagnosticar electrolitos/ácido-base.
AI/COAG	Diagnosticar enfermedades de la sangre.
AI/REUHM	Diagnosticar enfermedades reumáticas.
CADUCEUS	Diagnosticar enfermedades de medicina interna
ANNA	Supervisar terapia para dedos.
BLUE BOX	Diagnosticar/remediar depresión.
MYCIN	Diagnosticar/remediar infecciones bacterianas.
ONCOCIN	Remediar/administrar pacientes de quimioterapia.
ATTENDING	Capacitar en administración anestésica.
GUIDON	Capacitar en infecciones bacterianas.

**Tabla 1.6 Sistemas expertos médicos**

Nombre	Ingeniería
REACTOR	Diagnosticar/remediar accidentes de reactor.
DELTA	Diagnosticar/remediar locomotoras GE.
STEAMER	Instruir en operación, planta de energía a vapor.

Tabla 1.7 Sistemas expertos en ingeniería

Nombre	Geología
DIPMETER	Interpretar los registros del medidor de profundidad.
LITHO	Interpretar los datos de registro de pozos petroleros.
MUD	Diagnosticar/remediar problemas de perforación.
PROSPECTOR	Interpretar datos geológicos para buscar minerales.

Tabla 1.8 Sistemas expertos en geología

Nombre	Sistemas de computadora
PTRANS	Dar pronóstico para la administración de computadoras. DEC
BDS	Diagnosticar partes deficientes en la red de conmutación.
XCON	Configurar sistemas de cómputo DEC.
XSEL	Configurar las órdenes de venta de las computadoras DEC.
XSITE	Configurar el sitio del cliente para las computadoras DEC
YES/MVS	Supervisar/controlar el sistema operativo MVS de IBM.
TIMM	Diagnosticar computadoras DEC.

Tabla 1.9 Sistemas expertos en computadoras

### Dominios apropiados para los sistemas expertos

Antes de empezar a construir un sistema experto, es esencial decidir si es el paradigma apropiado. Por ejemplo, una preocupación es si debe usarse un sistema experto en lugar de un paradigma alternativo, como la programación convencional. El dominio apropiado de un sistema experto depende de varios factores:

- ¿Puede solucionarse eficazmente el problema con programación convencional? Si la respuesta es sí, entonces un sistema experto no es la mejor elección. Por ejemplo, consideremos el problema de diagnosticar cierto equipo: si todos los síntomas de mal funcionamiento se conocen de antemano, entonces lo adecuado es una simple búsqueda de la falla en una tabla o árbol de decisiones. Los sistemas expertos son más adecuados para situaciones en las que no hay una solución algorítmica eficiente. A esos casos se les llama **problemas de estructura nociva** y sólo el razonamiento puede ofrecer esperanzas de una solución adecuada.

Tomemos como ejemplo de un problema de estructura nociva, el caso de una persona que está pensando en viajar y visita a un agente de viajes. Aunque la mayoría de las personas tiene un destino y planes elaborados, hay excepciones. En la tabla 1.10 se presenta una lista de algunas de estas características de un problema de estructura nociva, como lo indican las respuestas de esta persona a las preguntas del agente de viajes.

A pesar de que probablemente se trata de un caso exagerado, ilustra el concepto básico de un problema de estructura nociva. Como puede verse, este tipo de problema no se presta a una solución algorítmica porque hay demasiadas posibilidades. En este caso, se ejercería una op-

ción predeterminada cuando todo lo demás falle. Por ejemplo, el agente de viajes diría: “¡Ajá! Tengo el viaje perfecto para usted, un crucero alrededor del mundo. Por favor llene este formulario de tarjeta de crédito y nos ocuparemos de todo”.

Preguntas del agente de viajes	Respuestas
¿Puedo ayudarle?	No estoy seguro.
¿A dónde desea ir?	A algún lado.
¿Algún destino en particular?	Aquí y allá.
¿Qué tanto puede pagar?	No lo sé.
¿Puede conseguir algo de dinero?	No lo sé.
¿Cuándo quiere salir?	Tarde o temprano.

Tabla 1.10 Ejemplo de un problema de estructura nociva

Al tratar con problemas de estructura nociva, existe el peligro de que el sistema experto pueda accidentalmente dar una solución idéntica a una algorítmica; es decir, el sistema experto puede descubrir en forma inadvertida una solución algorítmica. Un indicador de que se ha dado esta situación es que se encuentre una solución que requiera una rígida **estructura de control**. O sea, el ingeniero del conocimiento obliga a que las reglas se ejecuten en una determinada secuencia, ajustando explícitamente las prioridades de muchas reglas. Al forzar una estructura de control rígida en el sistema experto, se cancela una de las principales ventajas de su tecnología, que es la de tratar con entradas inesperadas que no siguen un patrón predeterminado (Parrello 88). Es decir, los sistemas expertos reaccionan de manera oportuna ante su entrada, cualquiera que ésta sea. Los programas convencionales suelen esperar que la entrada siga cierta secuencia. Un sistema experto con demasiado control a menudo indica un algoritmo encubierto y puede ser un buen candidato para volver a codificarse como un programa convencional.

- *¿Está bien delimitado el dominio?* Es importante tener límites bien definidos en lo que se espera que sepa el sistema experto y en las aptitudes que debe tener. Por ejemplo, supongamos que se desea crear un sistema experto para diagnosticar dolores de cabeza. Es cierto que el conocimiento médico de un facultativo debería ponerse en la base de conocimientos. Sin embargo, para una comprensión más profunda de los dolores de cabeza, quizás también se deban insertar conocimiento de neuroquímica, después su área padre, bioquímica, después química, biofísica molecular y así sucesivamente hasta llegar a la física subnuclear. Otros dominios como retroalimentación biológica, psicología, psiquiatría, fisiología, ejercicio, yoga y manejo del estrés pueden contener también conocimiento pertinente acerca de los dolores de cabeza. El punto es: ¿cuándo dejar de agregar dominios? Cuantos más dominios haya, más complejo se volverá el sistema experto.

En particular, la tarea de coordinar toda la experiencia se vuelve sumamente importante. Sabemos por experiencia qué tan difícil es coordinar equipos conformados por especialistas sobre todo cuando ofrecen recomendaciones conflictivas. Si sabemos cómo programar adecuadamente la coordinación de la experiencia, entonces podemos tratar de programar un sistema experto para que tenga el conocimiento de varios especialistas. Se han hecho intentos de coordinar varios sistemas expertos en los sistemas HEARSAY II y HEARSAY III. Sin embargo, esta coordinación es una tarea compleja que debería ser vista más como una investigación que como la elaboración de un producto confiable para los sistemas expertos.

- *¿Hay la necesidad y el deseo de tener un sistema experto?* Aunque es una gran experiencia construir un sistema experto, no tiene caso si nadie está dispuesto a usarlo. Si hay muchos especialistas humanos, es difícil justificar un sistema experto con base en la escasez de experiencia humana. Además, si los expertos o los usuarios no quieren el sistema, éste no será aceptado aunque sea necesario.

En especial, la administración debe estar dispuesta a apoyar el sistema. Esto es aún más crítico para los sistemas expertos que para los programas convencionales, porque los sistemas expertos son una nueva tecnología, lo que significa que hay poca gente con experiencia en ella y mucha incertidumbre acerca de lo que puede lograrse. Sin embargo, el área de sistemas expertos requiere más apoyo, porque trata de resolver problemas que los programas convencionales no pueden. Los riesgos son mayores, pero también las recompensas.

- *¿Hay al menos un especialista que esté dispuesto a cooperar?* Debe haber un perito que esté dispuesto a aplicar y de preferencia entusiasmado, con el proyecto; pues, no todos los especialistas están dispuestos a que se examine su conocimiento en busca de fallas y luego sea colocado en una computadora. Aunque haya muchos dispuestos a cooperar en el desarrollo, quizás sea sabio limitar el número de especialistas que participan en él, ya que pueden tener distintas formas de resolver un problema, como la aplicación de diferentes pruebas diagnósticas, e incluso llegar a diferentes conclusiones. Tratar de codificar varios métodos para resolver problemas en una sola base de conocimiento puede crear conflictos internos e incompatibilidades.
- *¿Puede el experto explicar el conocimiento en forma comprensible al ingeniero del conocimiento?* Aunque el especialista esté dispuesto a cooperar, puede tener dificultades al expresar el conocimiento en términos explícitos. Como un ejemplo simple de esta dificultad, ¿podría usted explicar con palabras cómo mueve un dedo? Aun si pudiera decir que se hace contrayendo un músculo, la siguiente pregunta sería, ¿cómo contrae un músculo del dedo? La otra dificultad de comunicación entre el especialista y el ingeniero en conocimiento es que éste último no conoce la terminología técnica del especialista, problema que se acentúa con la terminología médica. Puede tomar un año o más para que el ingeniero del conocimiento entienda de qué está hablando el perito, sin contar con la traducción del conocimiento a código explícito de computadora.
- *¿El conocimiento para resolver problemas es principalmente heurístico e incierto?* Los sistemas expertos son apropiados cuando el conocimiento del especialista es muy heurístico e incierto. Es decir, el conocimiento puede estar basado en la experiencia, lo que se le llama **conocimiento empírico**, y el especialista puede probar varios métodos en caso de que uno no funcione. En otras palabras, su conocimiento puede ser un método de ensayo y error, en lugar de uno basado en lógica y algoritmos. Sin embargo, el especialista aún puede resolver el problema más rápido que alguien que no lo es. Ésta es una buena aplicación para los sistemas expertos. Si el problema puede resolverse únicamente con la lógica y los algoritmos, es mejor manejarlo con un programa convencional.

## 1.8 LENGUAJES, SHELLS Y HERRAMIENTAS

Una decisión fundamental al definir un problema es decidir qué tan bien modelarlo. Algunas veces se dispone de la experiencia que ayuda a escoger el mejor paradigma. Por ejemplo, la experiencia sugiere que una nómina se elabora mejor con un procedimiento de programación convencional; también sugiere que es preferible usar un paquete comercial, si está disponible, en lugar de escribir uno desde cero. Una guía general para seleccionar un paradigma es considerar primero el más tradicional: la programación convencional. La razón para esto es la vasta experiencia que tenemos con la programación convencional y la amplia variedad de paquetes comerciales disponibles. Si un problema no puede solucionarse eficazmente con la programación convencional, entonces hay que mirar hacia los paradigmas no convencionales, como la AI.

A pesar de que los sistemas expertos son una rama de la AI, existen lenguajes especializados para sistemas expertos que son completamente diferentes a los lenguajes comunes de la AI como LISP y PROLOG. Aunque se han desarrollado muchos otros, como IPL-II, SAIL, CONNIVER, KRL y Smalltalk, pocos se utilizan ampliamente, excepto con fines de investigación (Scown 85).

Un lenguaje para sistemas expertos es un lenguaje de orden más alto que los lenguajes como LISP o C porque permite hacer ciertas cosas con mayor facilidad, pero también permite atacar un rango más pequeño de problemas. Esto significa que la naturaleza especializada de estos lenguajes los hace adaptables para elaborar sistemas expertos, pero no para la programación en general. En muchos casos incluso es necesario salir temporalmente del lenguaje para sistemas expertos y ejecutar una función en el lenguaje de procesamiento.

La principal diferencia funcional entre estos lenguajes, es el enfoque de la representación. Los lenguajes de procedimiento se enfocan en proporcionar técnicas flexibles y robustas para representar datos. Por ejemplo, las estructuras de datos como matrices, registros, listas ligadas, pilas, colas y árboles son creadas y manipuladas fácilmente. Los lenguajes modernos como Modula-2 y Ada están diseñados para ayudar en la **abstracción de datos**, proporcionando estructuras para su encapsulamiento como módulos y paquetes. Esto proporciona un nivel de abstracción que después se implanta con métodos como operadores e instrucciones de control para producir un programa. Los datos están estrechamente entrelazados con los métodos para manipularlos. Por el contrario los lenguajes para los sistemas expertos se concentran en proporcionar formas robustas y flexibles de representar el conocimiento; el paradigma del sistema experto permite dos niveles de abstracción: abstracción de datos y **abstracción de conocimiento**. Los lenguajes para sistemas expertos separan específicamente los datos de los métodos para manipularlos. Un ejemplo de esta separación es la de los hechos (abstracción de datos) y de las reglas (**abstracción de conocimiento**) que emplea un lenguaje para el sistema experto basado en reglas.

Esta diferencia de enfoque también produce diferencias en la metodología de diseño del programa. A causa del estrecho entrelazamiento de los datos y el conocimiento en los programas orientados al procesamiento, los programadores deben describir con cuidado la secuencia de ejecución. Sin embargo, la separación explícita de datos y conocimiento en los lenguajes para sistemas expertos requiere un control mucho menos rígido de la secuencia de ejecución. Por lo general, se usa una pieza de código completamente separada, el mecanismo de inferencia, para aplicar el conocimiento a los datos; esta separación entre conocimiento y datos permite un mayor grado de paralelismo y modularidad.

Al escoger un lenguaje, una pregunta básica debe ser si el problema exige más conocimiento o inteligencia. Los sistemas expertos dependen de una gran cantidad de conocimiento especializado o experiencia para resolver un problema, mientras que la AI enfatiza un método para la solución de problema. Es común que los sistemas expertos dependan de la correspondencia de patrones en un dominio de conocimiento restringido para guiar su ejecución, mientras que la AI suele concentrarse en la búsqueda de paradigmas en dominios menos restringidos.

La manera acostumbrada para definir si hay necesidad de un sistema experto es decidir si se quiere programar la experiencia de un especialista humano, si éste existe y si va a cooperar, es entonces cuando una propuesta de sistema experto puede tener éxito.

El camino para seleccionar el lenguaje para el sistema experto está lleno de confusiones. Hace algunos años, la elección de un lenguaje de sistema experto era muy directa, solo había cerca de media docena de lenguajes disponibles y por lo general eran gratuitos o costaban una cantidad nominal que se pagaba a la universidad en la que se desarrollaban.

Sin embargo, con el explosivo crecimiento comercial en el campo de los sistemas expertos, desde los setenta, la selección de un lenguaje ya no es tan simple. En la actualidad hay docenas de lenguajes disponibles, con precios que llegan hasta los \$75,000 dólares. A pesar de que todavía es posible obtener gratis algunos de los viejos lenguajes como el OPS5, o a costo nominal en las universidades donde se desarrollaron, se debe pagar un precio en eficiencia, falta de funciones modernas y soporte.

Además de la confusión al elegir uno entre los muchos lenguajes disponibles hoy, la terminología usada para describirlos es confusa. Algunos vendedores aluden a sus productos como “herramientas”, mientras que otros hablan de shells y algunos otros de “ambientes integrados”. Para mantener la claridad en este libro los términos se definirán como sigue:

- **Lenguaje:** es un traductor de comandos escrito con una sintaxis específica. Un lenguaje para sistemas expertos también proporcionará un mecanismo de inferencia que ejecute las instrucciones del lenguaje. Dependiendo de la forma en que esté implantado, el mecanismo de inferencia puede proporcionar encadenamiento hacia atrás, hacia delante o ambos. Bajo esta definición de lenguaje, LISP no es un lenguaje para los sistemas expertos, pero PROLOG sí. Sin embargo, es posible escribir un lenguaje de éstos usando LISP e AI en PROLOG. Para el caso, incluso puede escribirse un lenguaje para sistemas expertos o AI en lenguaje ensamblador. Las preguntas acerca del tiempo de desarrollo, la conveniencia, la conservación, la eficiencia y la velocidad, determinan en qué lenguaje debe estar escrito el software.
- **Herramienta:** es un lenguaje adicionalmente asociado con programas de utilerías para facilitar el desarrollo, la depuración y el uso de los programas de aplicación. Los programas de utilerías pueden incluir editores de texto e imágenes, depuradores, administradores de archivos e incluso generadores de código. También pueden proporcionarse ensambladores de plataforma cruzada para portar el código de desarrollo a un hardware diferente. Por ejemplo, un sistema experto puede desarrollarse en un VAX de DEC y después ensamblarse para ejecutarse en un Motorola 68000. Algunas herramientas pueden incluso admitir el uso de paradigmas diferentes, como el encadenamiento hacia delante y hacia atrás en una aplicación.

En algunos casos, una herramienta puede integrarse con todos sus programas utilitarios en un solo ambiente, para presentar al usuario una interfaz común. Este método minimiza la necesidad de que el usuario abandone el ambiente para ejecutar una tarea. Por ejemplo, una herramienta simple tal vez no proporcione medios para la administración de archivos y, por tanto, el usuario tendrá que salir de la herramienta para aplicar comandos del sistema operativo. Un ambiente integrado permite un fácil intercambio de datos entre varios programas utilitarios dentro del mismo ambiente. Algunas herramientas ni siquiera requieren que el usuario escriba algún código; en cambio, la herramienta permite al usuario introducir conocimiento mediante el ejemplo, a partir de tablas u hojas de cálculo, y genera el código apropiado por sí misma.

- **Shell:** herramienta con propósitos especiales, diseñada para cierto tipo de aplicaciones en las que el usuario sólo debe proporcionar la base de conocimiento. El ejemplo clásico de esto es el shell de EMYCIN (MYCIN vacío), éste se hizo al eliminar la base del conocimiento médico del sistema experto MYCIN.

MYCIN estaba diseñado como un sistema de encadenamiento hacia atrás para diagnosticar enfermedades. EMYCIN se creó por la simple eliminación del conocimiento médico y pudo usarse como una shell que contenía conocimiento acerca de otros tipos de sistemas consultivos que utilizan el encadenamiento hacia atrás. El shell EMYCIN demostró que podía volver a utilizarse el software esencial de MICYN como mecanismo de inferencia y la interfaz del usuario. Este fue un paso muy importante en el desarrollo de la tecnología para los modernos sistemas expertos, porque significó que no era necesario construir desde cero cada nueva aplicación.

Hay muchas formas de caracterizar a los sistemas expertos: como representación del conocimiento, encadenamiento hacia atrás o hacia delante, manejo de la incertidumbre, razonamiento hipotético, medios de explicación, etcétera. A menos que una persona haya construido cierto número de sistemas expertos, es difícil que aprecie todas estas funciones, sobre todo aquellas que se encuentran en las herramientas más caras. La mejor manera de aprender la tecnología de los sistemas expertos es desarrollar varios sistemas con un lenguaje fácil de aprender y después, si usted necesita sus funciones, invertir en una herramienta más sofisticada.

## 1.9 ELEMENTOS DE UN SISTEMA EXPERTO

En la figura 1.6 se muestran los elementos de un sistema experto típico. En un sistema basado en reglas, la base de conocimientos contiene el conocimiento de dominio necesario para resolver los problemas codificados en forma de reglas; mientras que las reglas son un paradigma popular para representar conocimiento, otros tipos de sistemas expertos usan diferentes representaciones, como se analiza en el capítulo 2.

Un sistema experto consta de los siguientes componentes:

- **interfaz de usuario:** el mecanismo que permite la comunicación entre el usuario y el sistema experto.
- **medio de explicación:** explica al usuario el razonamiento del sistema.
- **memoria activa:** una **base de datos global** de los hechos usados por las reglas.
- **mecanismo de inferencia:** hace inferencias al decidir cuáles reglas satisfacen los hechos u objetos, da prioridad a las reglas satisfechas y ejecuta la regla con la prioridad más elevada.
- **agenda:** una lista con prioridades asignadas a las reglas, creada por el mecanismo de inferencia, cuyos patrones satisfacen los hechos u objetos de la memoria activa.
- **medio para la adquisición de conocimiento,** vía automática para que el usuario introduzca conocimientos en el sistema, sin tener al ingeniero del conocimiento para que codifique éste en forma explícita.

El medio para la adquisición del conocimiento es una función opcional en muchos sistemas. En algunas herramientas para los sistemas expertos como KEE y First Class, ésta puede aprender por inducción de reglas mediante ejemplos, y también generar reglas automáticamente. Sin embargo, los ejemplos provienen generalmente de datos de tipo tabular u hojas de cálculo, que se aplican mejor a los árboles de decisión. Las reglas generales que construye un

ingeniero en conocimiento pueden ser mucho más complejas que las reglas simples creadas mediante inducción.

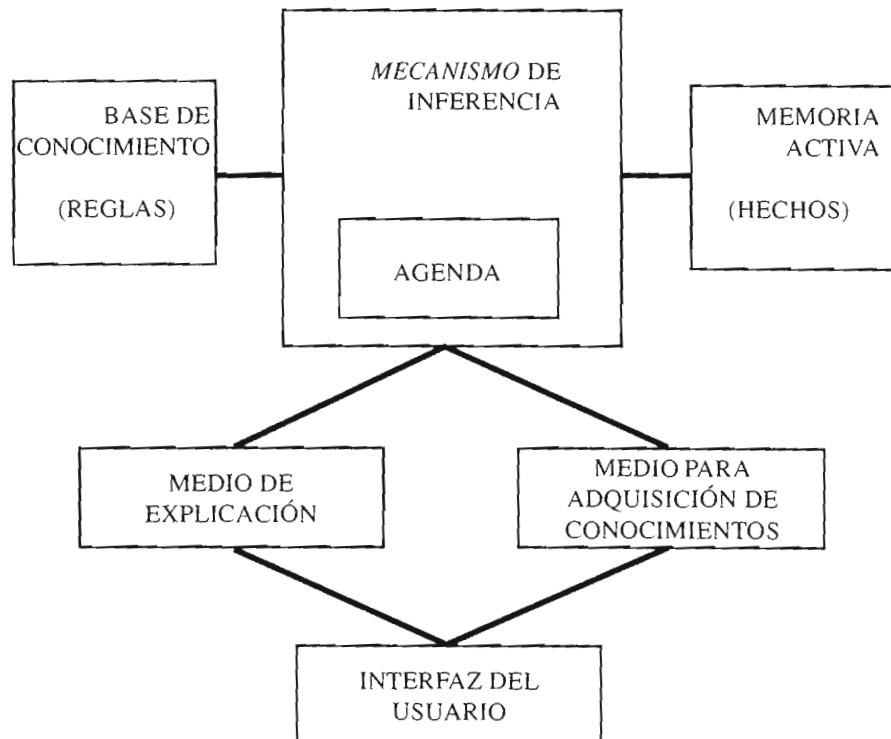


Figura 1.6 Estructura de un sistema experto basado en reglas

Dependiendo de la implantación del sistema, la interfaz de usuario puede ser una simple pantalla de texto o una pantalla más sofisticada, de alta resolución, con mapas de bits, que por lo común, se usa para simular un panel de control con botones y ventanas.

En un sistema experto basado en reglas, la base de conocimiento también recibe el nombre de **memoria de producción**. Tomemos como ejemplo el problema de decidir el cruce de una calle. La producción para las dos reglas es como sigue, donde las flechas significan que el sistema ejecutará las acciones que se encuentran a la derecha de la flecha si las condiciones de la izquierda son verdaderas:

La luz es roja → alto  
 La luz es verde → adelante

La producción de reglas puede expresarse en un pseudocódigo equivalente al formato SI...ENTONCES, como:

Regla: Luz\_roja  
 SI  
     la luz es roja  
 ENTONCES  
     alto

Regla: luz\_verde  
 SI  
     la luz es verde  
 ENTONCES  
     adelante

Cada regla se identifica con un nombre, seguido por la parte SI de la regla. A la sección entre las partes SI y ENTONCES de la regla se le da varios nombres como son **antecedente**, **parte condicional**, **patrón** o **lado izquierdo**. A la condición individual

la luz es verde

se le llama **elemento condicional** o **patrón**.

Algunos ejemplos de reglas de sistemas expertos reales son:

*Sistema MYCIN para diagnóstico de meningitis y bacteremia (infecciones bacterianas)*

SI  
     El medio de cultivo es la sangre, y  
     La identidad del organismo no es conocida con  
         certeza, y  
     La coloración del organismo es gramnegativa, y  
     La morfología del organismo es de bastoncillo, y  
     El paciente presenta elevación de la temperatura  
 ENTONCES  
     Hay una débil evidencia que sugiere(.4) que la  
         identidad del organismo es pseudomonas.

*XCON/R1 para configurar los sistemas de cómputo VAX de DEC*  
 SI

    El contexto actual es la asignación de dispositivos a  
         los módulos Unibus y  
     Hay una unidad de disco con puerto dual que no ha sido  
         asignada y  
     El tipo de controlador que se requiere es conocido y  
     Hay dos controladores y ninguno de ellos tiene un  
         dispositivo asignado, y  
     Se conoce el número de dispositivos que estos  
         controladores pueden soportar

## ENTONCES

Asignar la unidad de disco a cada uno de los controladores, y

Revisar que los dos controladores han sido asociados y que cada uno soporta un drive

En un sistema basado en reglas, el mecanismo de inferencia determina cuáles antecedentes de regla, si hay alguno, queda satisfecho por los hechos. Dos métodos generales de inferencia que se usan con frecuencia como estrategias para la solución de problemas con los sistemas expertos son: **encadenamiento hacia delante** y **encadenamiento hacia atrás**. Otros métodos usados para necesidades más específicas pueden incluir análisis de medios y fines, reducción del problema, localización en reversa, prueba de generación de planes, planeación jerárquica y principio mínimo de cometido, y manejo restringido.

El encadenamiento hacia adelante es el razonamiento desde los hechos hacia las conclusiones que resultan de ellos. Por ejemplo, si usted ve que está lloviendo antes de salir de casa (el hecho), entonces debe llevar un paraguas (la conclusión).

El encadenamiento hacia atrás implica el razonamiento en reversa desde una hipótesis, habrá de comprobarse una posible conclusión, a los hechos que la sustentan. Por ejemplo, si usted no ha mirado hacia fuera y alguien entra con los zapatos mojados y un paraguas, su hipótesis será que está lloviendo; para apoyarla podría preguntar a la persona si en verdad está lloviendo, si la respuesta es sí, entonces la hipótesis es verdadera y se convierte en un hecho. Como se dijo antes, una hipótesis puede verse como un hecho cuya veracidad está en duda y necesita establecerse. La hipótesis puede interpretarse entonces como un objetivo a probar.

Dependiendo del diseño, un mecanismo de inferencia llevará a cabo el encadenamiento hacia atrás o hacia delante. Por ejemplo, OPS5 y CLIPS están diseñados para el encadenamiento hacia delante, mientras que EMYCIN ejecuta el encadenamiento hacia atrás, y otros tipos de mecanismo de inferencia, como ART y KEE, ofrecen ambos. La elección de este mecanismo depende del tipo de problema. El diagnóstico de problemas se resuelve mejor con el encadenamiento hacia atrás, mientras que la pronóstico, la supervisión y el control se realizan mejor mediante el encadenamiento hacia delante.

La memoria activa puede contener hechos que contemplan el estado actual de la luz del semáforo, como "la luz es verde" o "la luz es roja". Uno de estos hechos o ambos pueden estar en la memoria activa al mismo tiempo. Si el semáforo está trabajando con normalidad, sólo uno de estos hechos estará en la memoria. Sin embargo, es posible que ambos hechos puedan estar en la memoria si hay un mal funcionamiento en el semáforo. Nótese la diferencia entre la base de conocimientos y la memoria activa. Los hechos no pueden interactuar entre sí, el hecho "la luz es verde" no tiene efecto sobre el hecho "la luz es roja"; en cambio, nuestro conocimiento del semáforo dice que si ambos hechos están presentes de manera simultánea, entonces hay una falla en el semáforo.

Si hay un hecho "la luz es verde" en la memoria activa, el mecanismo de inferencia se dará cuenta de que este hecho satisface la parte condicional de la regla de luz verde y pondrá esta regla en su agenda. Si una regla tiene varios patrones, entonces todos ellos deben satisfacerse de manera simultánea para que la regla pase a la agenda. Algunos patrones pueden satisfacerse especificando la ausencia de ciertos hechos en la memoria activa.

Cuando todos los patrones de una regla están satisfechos, se dice que está **activada** o **iniciada**. Muchas reglas activadas pueden estar en la agenda al mismo tiempo, en cuyo caso, el mecanismo de inferencia debe seleccionar una regla de **disparo**. El término *disparar* viene de la neurofisiología, el estudio del sistema nervioso. Una célula nerviosa individual o neurona emite una señal eléctrica cuando se estimula; la falta de más estímulos puede provocar que la neurona se dispare otra vez durante un breve periodo, a este fenómeno se le llama *refrac-*

ción. Los sistemas expertos basados en reglas están construidos usando la refracción con el propósito de prevenir enredos triviales. Es decir, que si la regla de la luz verde se mantiene disparando una y otra vez sobre el mismo hecho, el sistema experto no realizaría un trabajo útil.

Se han inventado muchos métodos para conseguir la refracción. En un lenguaje para sistemas expertos llamado OPS5, cada hecho recibe un identificador único, que se conoce como **etiqueta de tiempo**, cuando se introduce en la memoria activa. Después de que una regla disparó un hecho, el motor de inferencia no disparará otra vez sobre el mismo porque su etiqueta de tiempo ya se utilizó.

Después de la parte ENTONCES de una regla, hay una serie de **acciones** que se ejecutarán cuando la regla se dispara. A esta parte de la regla se le conoce como **la consecuencia o lado derecho (RHS)**. Cuando la regla de la luz roja se dispara, se ejecuta su acción, que es detenerse. Del mismo modo, cuando la regla de la luz verde se dispara, su acción es avanzar. Por lo general, las acciones específicas incluyen la adición o remoción de los hechos en la memoria activa o de los resultados de impresos. El formato de estas acciones depende de la sintaxis del lenguaje utilizado; por ejemplo, en OPS5, ART y CLIPS, la adición de un nuevo hecho llamado "ALTO" a la memoria activa sería (afirmar detener). Debido a su antecesor LISP, estos lenguajes se diseñaron para requerir paréntesis alrededor de los patrones y las acciones.

El mecanismo de inferencia opera en **ciclos**. Se han dado varios nombres para describir el ciclo, como: **ciclo acto-reconocimiento**, **ciclo selección-ejecución**, **ciclo situación-respuesta**, y **ciclo situación-acción**. Cualquiera que sea el nombre, el mecanismo de inferencia ejecutará un grupo de ciertas tareas repetidas hasta que ciertos criterios causen el cese de la ejecución. Las tareas de un ciclo para OPS5, un shell de sistema experto típico, se muestran en el siguiente pseudocódigo como **resolución de conflicto**, **acto**, **correspondencia** y **verificación de interrupciones**.

MIENTRAS no se haga

**Resolución de conflicto:** si hay activaciones, entonces seleccione aquella con la prioridad más alta.

**Acto:** ejecutar en forma secuencial las acciones en el lado derecho de la activación seleccionada. Aquellas que cambian la memoria activa tienen un efecto inmediato en este ciclo. Eliminar de la agenda la activación que se acaba de disparar.

**Correspondencia:** actualizar la agenda revisando si el lado izquierdo de cualquier regla están satisfechos. De ser así, activarlos. Eliminar acciones si el lado izquierdo de las reglas ya no se satisfacen.

**Verificación de interrupciones:** si se ejecuta una acción de interrupción o se da un comando de ruptura, entonces hacerlo.

FIN-MIENTRAS

**Aceptar las órdenes de un nuevo usuario**

Varias reglas pueden activarse y pasar a la agenda durante un ciclo, las activaciones de ciclos previos también se dejarán en la agenda a menos que hayan sido desactivadas porque su lado izquierdo ya no está satisfecho. Así, el número de activaciones en la agenda variará mientras procede la ejecución. Dependiendo del programa, una activación puede estar siempre en la agenda pero nunca seleccionada para dispararse; del mismo modo, algunas reglas quizás nunca se activen. En estos casos, los propósitos de estas reglas deberían volver a examinarse porque o son innecesarias, o sus patrones no se diseñaron bien.

El mecanismo de inferencia ejecuta las acciones con mayor prioridad de activación en la agenda, luego, la siguiente en grado de prioridad y así sucesivamente hasta que no quede ninguna activación. Se han diseñado varios esquemas de prioridad en los shells de sistemas

expertos. En general, todos los shells permiten que el ingeniero del conocimiento defina la prioridad de las reglas.

Los conflictos en la agenda suceden cuando diferentes activaciones tienen la misma prioridad y el mecanismo de inferencia debe decidir cuál regla disparar. Los distintos shells tienen formas diferentes de tratar este problema: en el paradigma original de Newell y Simon, aquellas reglas que se introdujeron primero en el sistema tienen la prioridad predeterminada más alta (Newell 72, p. 33); en OPS5, las reglas con patrones más complejos tienen la más alta prioridad; en ART y CLIPS, las reglas tienen la misma prioridad predeterminada a menos que el ingeniero en conocimiento les asigne prioridades distintas.

Para este momento, el control ha regresado al **nivel máximo** de intérprete de comandos para que el usuario dé más instrucciones al shell del sistema experto. El nivel máximo es el modo predeterminado en que el usuario se comunica con el sistema experto, y está indicado por la tarea “Aceptar nuevas órdenes del usuario”.

El nivel máximo es la interfaz del usuario con el shell mientras se está desarrollando una aplicación del sistema experto. Con regularidad, se diseñan interfaces de usuario más sofisticadas para facilitar la operación de los sistemas expertos. Por ejemplo, un sistema experto para controlar una planta de fabricación, puede tener una interfaz de usuario que muestra el diagrama de bloques de la planta, con despliegue de alta resolución de mapa de bits a colores. Las advertencias y los mensajes de estado pueden aparecer en colores brillantes, con botones y escalas simulados. En realidad, el mayor esfuerzo puede darse al diseñar e implementar la interfaz del usuario, y no en la base de conocimiento del sistema experto, sobre todo en un prototipo. Dependiendo de las capacidades del shell, la interfaz del usuario puede implementarse a partir de reglas o en otro lenguaje llamado por el sistema experto.

Un medio de explicación debe admitir que el usuario pregunte cómo llegó el sistema a cierta conclusión y por qué es necesaria cierta información. Para un sistema basado en reglas, la pregunta de cómo llegó el sistema a cierta conclusión es fácil de responder porque es posible mantener en un panel una historia de las reglas activadas y del contenido de la memoria activa. Los medios de explicación sofisticados permiten que el usuario haga preguntas del tipo “¿Qué pasa si?” para explorar las rutas de razonamiento que se alternan a través del razonamiento hipotético.

## 1.10 SISTEMAS DE PRODUCCIÓN

Uno de los tipos más populares de los sistemas expertos en la actualidad, es aquel basado en reglas. Las reglas son populares por varias razones.

- *Naturaleza modular.* Esto hace más fácil encapsular el conocimiento y expandir el sistema experto a partir de un desarrollo creciente.
- *Medios de explicación.* Es sencillo construir medios de explicación con reglas porque los antecedentes de una regla especifican con exactitud lo que es necesario para activarla. Al mantener el registro de las reglas que se dispararon, un medio de explicación puede presentar la cadena de razonamiento que condujo a cierta conclusión.
- *Semejanza con el proceso cognitivo humano.* Con base en el trabajo de Newell y Simon, las reglas aparecen como un modelo natural de la manera en que los humanos resuelven problemas. La representación SI...ENTONCES de las reglas facilita la explicación de los especialistas sobre la estructura del conocimiento que se trata de obtener de ellos. Otras ventajas de las reglas se describen en Hayes-Roth (85).

Las reglas son un tipo de producción cuyos orígenes se remontan a los cuarenta. Debido a la importancia de los sistemas basados en reglas, vale la pena examinar el desarrollo del concepto de regla. Esto dará una mejor idea de por qué los sistemas basados en reglas son tan útiles para los sistemas expertos.

## Sistemas de producción de Post

Los sistemas de producción fueron usados primero en lógica simbólica por Post, lo que dio origen a su nombre (Post 43). Él comprobó el importante y sorprendente resultado de que cualquier sistema de matemáticas o lógica podría escribirse como cierto tipo de sistema de reglas de producción. Este resultado establecía la gran capacidad de las reglas de producción para representar clases de conocimiento más grandes en lugar de estar limitadas a unos cuantos tipos. Bajo el término **reescribir reglas**, éstas también se usan en lingüística como una manera de definir la gramática de un lenguaje. Los lenguajes de computadora suelen definirse utilizando la Forma Backus-Naur (BNF) de reglas de producción.

La idea básica de Post era que cualquier sistema matemático o lógico representaba sólo un conjunto de reglas que especificaban cómo cambiar una hilera de símbolos por otra. Es decir que, dada una hilera de entrada, el antecedente, una regla de producción podría producir una hilera nueva, la consecuencia. Esta idea también es válida con programas y sistemas expertos en los que la hilera inicial de símbolos la constituyen los datos de entrada y la hilera de salida es una transformación de lo introducido.

Como un caso muy simple, supóngamos que la cadena de entrada es “el paciente tiene fiebre”, entonces la cadena de salida podría ser “tome una aspirina”. Nótese que no hay ningún significado que enlace a estas cadenas. Es decir, la manipulación de las cadenas se basa en la sintaxis y no en la semántica o en la comprensión de lo que significan *fiebre*, *paciente* o *aspirina*. Un humano sabe lo que estas cadenas significan en términos del mundo real, pero un sistema de producción Post es sólo una forma de transformar una cadena en otra. Una regla de producción para este ejemplo podría ser

*Antecedente → Consecuencia*

*La persona tiene fiebre → que tome una aspirina*

donde la flecha indica la transformación de una cadena en otra. Esta regla se puede interpretar en términos de la notación más familiar SI...ENTONCES de esta manera:

SI la persona tiene fiebre ENTONCES que tome un aspirina

Las reglas de producción también pueden tener múltiples antecedentes. Por ejemplo:

*La persona tiene fiebre Y*

*La fiebre es más alta que 39 → vea al doctor*

Nótese que el conectivo especial *Y* no es parte de la hilera, sencillamente indica que la regla tiene varios antecedentes.

Un sistema de producción de Post consta de un grupo de reglas de producción, como las siguientes (los números entre paréntesis son para propósitos de este análisis):

- (1) el carro no arranca → revisar batería
- (2) el carro no arranca → revisar gasolina
- (3) revisar batería Y si la batería está mal → reemplazarla
- (4) revisar gasolina Y si no hay gasolina → llenar el tanque de gasolina

Si hay una cadena “el carro no arranca”, es posible usar las reglas (1) y (2) para generar las cadenas “revisar la batería” y “revisar gasolina”, sin embargo, no hay un mecanismo de control que aplique ambas reglas a la cadena. Sólo puede aplicarse una regla, o ambas o ninguna. Si hay otra cadena “la batería está mal” y una cadena “revisar batería”, entonces la regla (3) puede aplicarse para generar la cadena “reemplazar batería”.

El orden con el que se escriban las reglas escritas no tiene un significado especial. Las de nuestro ejemplo también podrían escribirse en el siguiente orden y el sistema seguiría siendo el mismo:

- (4) revisar gasolina Y si no hay → llenar el tanque
- (2) el carro no arranca → revisar gasolina
- (1) el carro no arranca → revisar batería
- (3) revisar batería Y si la batería está mal → reemplazar la batería

A pesar de que las reglas de producción de Post eran útiles en la colocación de la parte fundamental de los sistemas expertos, no eran adecuadas para escribir los programas prácticos. La limitante básica de las reglas de producción de Post para la programación es la falta de una **estrategia de control** para guiar la aplicación de las reglas. Un sistema Post permite que las reglas se apliquen de cualquier manera en las hileras porque no hay una especificación dada sobre cómo deberían aplicarse.

Como una analogía, supóngamos que alguien va a la biblioteca para buscar un libro acerca de sistemas expertos. Una vez en ella, comienza a mirar aleatoriamente los libros; si la biblioteca es muy grande, encontrar el libro que necesita puede tomar mucho tiempo. Aun si encuentra la sección de libros en sistemas expertos, su siguiente decisión puede llevarlo por accidente a una sección completamente distinta, como la de cocina francesa. La situación empeora aún más si necesita material del primer libro para ayudarle a determinar el segundo libro que requiere. Una búsqueda al azar del segundo libro también tomará mucho tiempo.

## Algoritmos de Markov

El siguiente adelanto en la aplicación de las reglas de producción fue hecho por Markov, quien especificó una estructura de control para los sistemas de producción (Markov 54). Un **algoritmo de Markov** es un grupo ordenado de producciones que se aplican en orden de prioridad a una hilera de entrada. Si la regla de prioridad más alta no es aplicable, entonces se aplica la siguiente y así sucesivamente. El algoritmo de Markov termina si (1) la última producción no es aplicable a una hilera o (2) se aplica una producción que termina con un punto.

Los algoritmos de Markov también pueden aplicarse a las subhileras de una hilera, comenzando de la izquierda. Por ejemplo, el sistema de producción que sólo incluye la regla

$AB \rightarrow HIJ$

cuando se aplica a la hilera de entrada GABKAB produce la nueva hilera GHJKAB. Puesto que la producción ahora se aplica a la nueva hilera, el resultado final es GHJKHIJ.

El símbolo especial  $\wedge$  representa la **hilera nula** sin caracteres. Por ejemplo, la producción

$A \rightarrow \wedge$

borra las apariciones del carácter A en una hilera.

Otros símbolos especiales representan cualquier símbolo individual y están indicados por las letras minúsculas a, b, c, etcétera. Estos símbolos representan variables de caracteres individuales y son parte importante de los lenguajes para los sistemas expertos modernos. Por ejemplo, la regla

$$AxB \rightarrow BxA$$

invertirá los caracteres A y B.

Las letras griegas  $\alpha$ ,  $\beta$ , y las siguientes se usan para la puntuación especial de las cadenas. Se usan las letras griegas porque son distintas del alfabeto ordinario.

Un ejemplo de un algoritmo de Markov que mueve la primera letra de una hilera de entrada al último sitio es como a continuación lo muestra Elson (73). Las reglas están ordenadas en términos de la prioridad más alta (1), la siguiente (2) y así en sucesión. A las reglas se les da prioridad en el orden en el que se introdujeron.

$$(1) \quad \alpha xy \rightarrow y\alpha x$$

$$(2) \quad \alpha \rightarrow ^$$

$$(3) \quad ^ \rightarrow \alpha$$

El trazo de la ejecución se muestra en la tabla 1.11 para la cadena de entrada ABC.

Regla	Éxito o fracaso	Hilera
1	F	ABC
2	F	ABC
3	E	$\alpha ABC$
1	E	B $\alpha AC$
1	E	BC $\alpha A$
1	F	BC $\alpha A$
2	E	BCA

Tabla 1.11 Trazo de la ejecución de un algoritmo de Markov

Nótese que el símbolo  $\alpha$  actúa de manera análoga a la de una variable temporal en un lenguaje de programación convencional. Sin embargo, en lugar de mantener un valor, el símbolo  $\alpha$  se usa para apartar lugares y marcar la progresión de los cambios en la cadena de entrada. Una vez que el trabajo está hecho, la regla 2 elimina el símbolo  $\alpha$ . El programa termina entonces cuando la regla 2 se aplica, porque hay un periodo después de la regla 2.

## Algoritmo Rete

Puede notarse que hay una estrategia de control definitiva para los algoritmos de Markov, con las reglas de más alta prioridad ordenadas en primer lugar. Mientras procede la regla de mayor prioridad, ésta se utiliza. Si no, el algoritmo de Markov prueba reglas de prioridad más baja. A pesar de que el algoritmo de Markov puede utilizarse como la base de un sistema experto, es muy poco eficaz en sistemas con muchas reglas. El problema de la eficiencia es de gran importancia si queremos crear sistemas expertos para problemas reales que contengan cientos o miles de reglas. Sin importar qué tan bueno sea todo lo demás en un sistema, si el usuario tiene que esperar mucho tiempo para una respuesta, el sistema no se usará. Lo que en verdad es necesario es un algoritmo que conozca todas las reglas y pueda aplicar cualquier regla sin tener que probar cada una en forma secuencial.

Una solución para este problema es el **algoritmo Rete** desarrollado en 1979 por Charles L. Forgy en la Universidad Carnegie-Mellon para su tesis de doctorado en filosofía, en el shell del sistema experto OPS (sistema de producción oficial) de esa institución. El algoritmo Rete es un rápido igualador de patrones que obtiene su velocidad del almacenamiento de información sobre las reglas de una red. En lugar de tener que igualar los hechos con todas las reglas en cualquier ciclo-acto reconocimiento, el algoritmo Rete sólo busca los cambios en las correspondencias de cada ciclo. Esto acelera en gran medida la correspondencia de los hechos con los antecedentes, porque los datos estáticos que no cambiaron de un ciclo a otro pueden pasarse por alto. (Este tema se analizará con mayor amplitud en los capítulos sobre CLIPS.) Los algoritmos de igualado rápido como Rete completaron las bases para la aplicación práctica de los sistemas expertos. En la figura 1.7 se resumen los fundamentos de la tecnología de los sistemas expertos modernos basados en reglas.

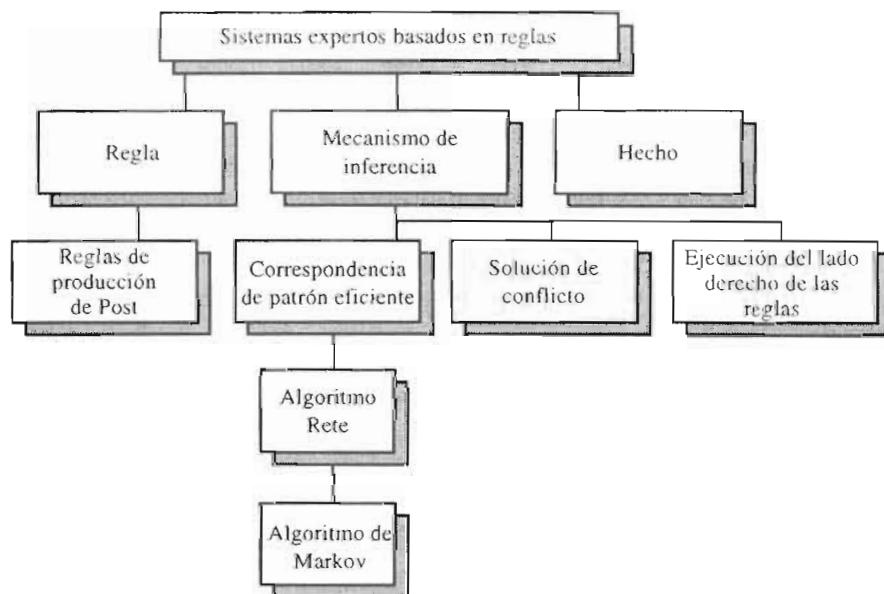


Figura 1.7 Fundamentos de los modernos sistemas expertos basados en reglas

Se han desarrollado diferentes versiones del lenguaje OPS y de su shell, incluyendo OPS2, OPS4 y OPS5. Una versión comercial más reciente desarrollada por Forgy es OPS83, un shell muy veloz. Sin embargo, ésta tiene diferencias significativas de sintaxis con el estilo OPS5 y depende, en parte, de los procedimientos para una ejecución más rápida.

## 1.11 PARADIGMAS DE PROCEDIMIENTOS

Los paradigmas de programación pueden clasificarse como procedurales y no procedurales. En la figura 1.8 se muestra una **taxonomía** o clasificación de los paradigmas de procedimientos en términos de lenguajes. En la figura 1.9 se muestra una taxonomía de paradigmas que no son de procedimientos. En estas figuras se ilustra la relación de los sistemas expertos con otros paradigmas y sólo deben considerarse como guías generales, no como definiciones estrictas. Algunos de los paradigmas y lenguajes tienen características que los sitúan en más de

una clase. Por ejemplo, algunos consideran que la programación funcional es de procedimientos, mientras que otros la consideran declarativa (Ghezzi 87).

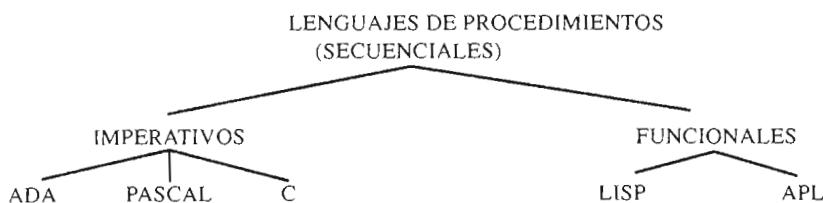


Figura 1.8 Lenguajes de procedimientos

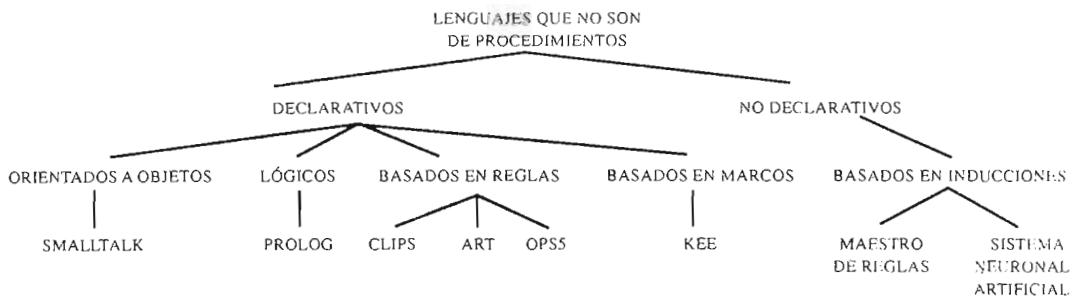


Figura 1.9 Lenguajes que no son de procedimientos

Un **algoritmo** es un método de solución de problemas en un número finito de pasos. Implementar un algoritmo en un programa es un **programa de procedimientos**. Los términos **programación algorítmica**, **programación de procedimientos** y **programación convencional** suelen usarse como conceptos intercambiables que se refieren a programas que no son del tipo AI. Una concepción frecuente del programa de procedimientos es que procede en forma secuencial, instrucción por instrucción, hasta que se encuentra una instrucción de ramificación. Otro sinónimo común para estos programas es el de programas **secuenciales**. Sin embargo, el término **programación secuencial** implica demasiada restricción si consideramos que todos los lenguajes de programación modernos apoyan la repetición y, por tanto, no son secuenciales en el sentido estricto.

El rasgo que distingue a un paradigma de procedimientos es que el programador debe especificar con exactitud *cómo* debe codificarse la solución de un problema. Incluso los generadores de código deben producir código de procedimientos. En cierto sentido, el uso de generadores de código pertenece a la **programación que no es de procedimientos**, porque remueve todos o la mayor parte del código de procedimientos que escribe el programador. La meta de esta clase de programación es hacer que el programador especifique cuál es el objetivo y luego deje que el sistema determine la manera en que lo alcanzará.

## Programación imperativa

Los términos **imperativa** y **orientada a las afirmaciones** se usan como sinónimos. Los lenguajes como FORTRAN, Ada, Pascal, Modula 2, COBOL y BASIC tienen la característica predominante de que las afirmaciones son imperativos u órdenes, que indican a la computadora lo que tiene que hacer. Los lenguajes imperativos se desarrollaron como una forma de liberar al programador de la necesidad de codificar el lenguaje ensamblador en la arquitectura de Von Neumann. En consecuencia, los lenguajes imperativos ofrecen gran soporte a varia-

bles, asignación de operaciones y repetición. Todas éstas son operaciones de bajo nivel que los lenguajes modernos tratan de sustituir proporcionando funciones como reiteración, procedimientos, módulos, paquetes, etcétera. Los lenguajes imperativos también se caracterizan por su énfasis en las estructuras de control rígidas y sus diseños de programa **inferiores**.

Un problema serio en todos los lenguajes es la dificultad de probar si los programas son correctos. Desde el punto de vista de la AI, otro problema serio es que los lenguajes imperativos no son muy eficientes para manipular símbolos. Como la arquitectura del lenguaje imperativo se moldeó para ajustarse a la arquitectura de la computadora de von Neumann, hay lenguajes que pueden soportar sin problemas el procesamiento de números, pero no las manipulaciones simbólicas. Sin embargo, los lenguajes imperativos como C y Ada se han usado como el lenguaje básico subyacente para escribir shells de los sistemas expertos. Estos lenguajes y las shells construidas a partir de ellos se ejecutan con mayor eficiencia y rapidez en las computadoras comunes para propósitos generales que los shells antiguos que se construyeron usando LISP.

Debido a su naturaleza secuencial, los lenguajes imperativos no son muy eficientes para implementar directamente sistemas expertos, sobre todo los basados en reglas. Como una ilustración de este problema, consideremos el conflicto de codificar la información de un problema del mundo real con cientos o miles de reglas, por ejemplo, el sistema XCON, utilizado por DEC para configurar sistemas de computadora, tiene en la actualidad cerca de 7000 reglas en su base de conocimiento. Previamente, se hicieron intentos infructuosos para codificar este programa en FORTRAN y BASIC, antes de tomar con éxito el método del sistema experto. El modo directo para codificar este conocimiento en un lenguaje imperativo hubiera requerido 7000 afirmaciones SI...ENTONCES o un CASE muy largo; pues este estilo de codificar presentaría grandes problemas de eficiencia si consideramos que las 7000 reglas necesitan revisarse en busca de los patrones de correspondencia en cada ciclo de acto de reconocimiento. Nótese que el mecanismo de inferencia y su ciclo acto reconocimiento tendrían que estar codificados también en lenguaje imperativo.

La eficiencia del programa podría aumentar si las reglas se ordenaran de manera que se pusiera al principio aquella con mayor probabilidad de ejecutarse. Sin embargo, esto requeriría una sintonización considerable del sistema y cambiaría cuando se añadieran nuevas reglas o cuando se borraran o modificaran las anteriores. Un mejor método para aumentar la eficiencia podría ser la construcción de un árbol de los patrones de reglas, para reducir el tiempo de búsqueda al determinar cuáles reglas habrían de activarse. En lugar de hacer que el programador construya en forma manual el árbol, sería preferible hacer que la computadora lo construya automáticamente, basándose en el patrón y en la sintaxis de acción de las reglas SI...ENTONCES. También sería útil tener una sintaxis SI...ENTONCES que fuera más conducive a representar el conocimiento y tuviera pruebas versátiles de correspondencia de patrones, pero esto requiere el desarrollo de un analizador que estudie la estructura de entrada y un intérprete o compilador para ejecutar la nueva sintaxis SI...ENTONCES.

Cuando todas estas técnicas se implementan para aumentar la eficiencia, el resultado es un sistema experto calificado. Si el mecanismo de inferencia, el analizador y el intérprete se remueven para facilitar el desarrollo de otros sistemas expertos, éstos constan de una shell de sistema experto. Por supuesto, en lugar de hacer todo este desarrollo desde el principio, en la actualidad es mucho más fácil usar un shell existente que esté documentado y que haya sido probado varias veces.

## Programación funcional

La naturaleza de la **programación funcional**, como se exemplificó mediante lenguajes como LISP y APL, es diferente a los lenguajes orientados a las afirmaciones, con su gran dependencia en elaboradas estructuras de control y en los diseños inferiores. La idea fundamental de la

programación funcional es combinar las funciones simples para producir funciones más potentes. En esencia, se trata de un diseño **superior**, comparado con los diseños comunes de los lenguajes imperativos.

La programación funcional se centra en las **funciones**. En matemáticas, una función es una **asociación** o regla que define la posición de los miembros de un conjunto, el **dominio**, en otro conjunto, el **codominio**. Un ejemplo de la definición de una función es:

cubo ( $x$ )  $\equiv x^*x^*x$ , donde  $x$  es un número real y cubo es una función con valores reales

Las tres partes de la definición de la función son:

- (1) la asociación,  $x^*x^*x$
- (2) el dominio, números reales
- (3) el codominio, números reales

de la función cubo. El símbolo  $\equiv$  significa “es equivalente a” o “se define como”. La siguiente notación es una manera más corta de escribir que la definición de posiciones al cubo es del dominio de números reales, simbolizado como  $\mathbb{R}$ , al codominio de números reales.

cubo:  $\mathbb{R} \rightarrow \mathbb{R}$

Una notación general para una función  $f$  que asigna las posiciones de un dominio  $S$  a un dominio  $T$  es  $f: S \rightarrow T$  (Gersting 82). El **rango** de la función  $f$  es el grupo de todas las imágenes  $f(s)$ , donde  $s$  es un elemento de  $S$ . En el caso de la función cúbica, las imágenes de  $s$  son  $s*s*s$  y el rango es el conjunto de todos los números reales. El rango y el codominio son el mismo para la función cúbica. Sin embargo, esto puede no ser cierto para otras funciones como la función al cuadrado,  $x*x$ , con dominio y codominio de números reales. Como el rango de la función al cuadrado es sólo el de los números reales no negativos, el rango y el codominio no son el mismo.

Usando notación de conjuntos, el rango de una función puede escribirse como:

$R \equiv \{f(s) \mid s \in S\}$

Las **llaves** ( $\{\}$ ) denotan un **conjunto**. La barra ( $|$ ) se lee como “tal que”. La afirmación anterior puede leerse: el rango  $R$  es equivalente al conjunto de valores  $f(s)$ , tal que todos los elementos  $s$  se encuentran en el conjunto  $S$ . La asociación es un conjunto de pares ordenados  $(s, t)$ , donde  $s \in S$ ,  $t \in T$ , y  $t = f(s)$ . Cada miembro de  $S$  debe tener uno y sólo un elemento de  $T$  asociado con él. Sin embargo, varios valores de  $t$  pueden estar asociados con un solo  $s$ . Como ejemplo, cada número positivo  $n$  tiene dos raíces cuadradas,  $\pm\sqrt{n}$ .

Las funciones también pueden definirse en forma recursiva como en:

factorial ( $n$ )  $\equiv n * \text{factorial } (n-1)$

Donde  $n$  es un número entero y  
factorial es una función entera

Las funciones recursivas se utilizan con frecuencia en lenguajes funcionales como LISP.

Los conceptos y las expresiones matemáticas son **referencias transparentes**, porque el significado del todo está determinado en su totalidad por sus partes. No hay sinergismo implicado entre las partes. Por ejemplo, consideremos la expresión funcional  $x + (2*x)$ . Ob-

viamente, el resultado es  $3*x$ . Ambos,  $x+(2*x)$  y  $3*x$ , dan el mismo resultado sin importar los valores que se sustituyan por  $x$ . Incluso cuando otras funciones se sustituyen por  $x$ , el resultado es el mismo. Por ejemplo, considerando que  $h(y)$  es una función arbitraria. Entonces  $h(y)+(2*h(y))$  sería equivalente a  $3*h(y)$ .

Ahora, consideremos la siguiente afirmación de tarea en un lenguaje de computadora imperativo como Pascal:

```
sum := f(x) + x
```

Si el parámetro  $x$  pasa por referencia y su valor se cambia en la llamada de función,  $f(x)$ , ¿Cuál valor se usará para  $x$ ? Dependiendo de la manera en que esté escrito el compilador, el valor de  $x$  puede ser el valor original si se guardó en una pila, o el nuevo valor si  $x$  no se guardó. Puede surgir más confusión si un compilador evalúa las expresiones de derecha a izquierda mientras que otro lo hace de izquierda a derecha. En este caso,  $f(x)+x$  no se evaluaría a igual que  $x+f(x)$  en diferentes compiladores; incluso cuando se usa el mismo lenguaje pueden ocurrir otros efectos secundarios debido a variables globales. Por tanto, a diferencia de las funciones matemáticas, las funciones de los programas no son referencias transparentes.

Los lenguajes de programación funcionales se crearon para ser referencias transparentes. Un lenguaje funcional se compone de cinco partes:

- **objetos de datos** sobre los que operan las funciones del lenguaje
- **funciones primitivas** para operar sobre los objetos de datos
- **formas funcionales** para sintetizar nuevas funciones a partir de otras funciones
- **operaciones de aplicación** sobre funciones que devuelven un valor
- **procedimientos de nominación** para identificar nuevas funciones.

Por lo general, los lenguajes funcionales se implementan para facilitar la construcción y la respuesta inmediata del usuario.

En **LISP** (procesamiento de listas), los objetos de datos son **expresiones simbólicas** (expresiones-S) que pueden ser **listas** o **átomos**, mientras que en **APL** (lenguaje de programación A) los objetos son matrices. Algunos ejemplos de listas son:

```
(leche huevos queso)
(compras (abarrotes (leche huevos queso) ropa
(pantalones)))
()
```

Las listas siempre están encerradas entre paréntesis de correspondencia, con espacios que separan los elementos. Los elementos de la lista pueden ser átomos, como leche, huevos y queso, o listas insertadas como (leche huevos queso) y (pantalones). Las listas pueden separarse pero los átomos no. La **lista vacía**, (), no contiene ningún elemento y se le llama **nula**.

Debe haber algunas funciones primitivas proporcionadas por el lenguaje como una base para la construcción de funciones más complejas. En la versión original de LISP, creada por John McCarthy en 1960, había pocas funciones primitivas, como se muestra en la tabla 1.12. Las funciones primitivas CAR, CDR, CPR y CTR son acrónimos que deben su nombre a los registros específicos del hardware en la primera máquina en que se corrió LISP; CPR y CTR ahora son obsoletos, pero los otros se han mantenido. También se muestran los **predicados** de LISP, que son funciones especiales que devuelven valores que representan verdadero o falso.

La versión original de LISP se llamó LISP “puro” porque era funcional en su totalidad. Sin embargo, no era muy eficiente para escribir programas. Las adiciones no funcionales han

hecho que LISP incremente su eficiencia en la escritura de programas. Por ejemplo, SET actúa como el operador de asignación, mientras que LET y PROG pueden usarse para crear variables locales y para ejecutar una secuencia de expresiones S. Aunque éstos actúan como funciones, no son funcionales en el sentido matemático original.

Función	Predicados
QUOTE	ATOM
CAR	EQ
CDR	NULL
CPR	
CTR	
CONS	
EVAL	
COND	
LAMBDA	
DEFINE	
LABEL	

Tabla 1. 12 Primitivos y funciones del LISP original

Desde su creación, LISP ha sido el lenguaje para AI líder en los Estados Unidos. Muchos de los shells originales de los sistemas expertos se escribieron en LISP porque es muy fácil experimentar con él. No obstante, las computadoras convencionales no ejecutan LISP con mucha eficiencia e incluso son peores al ejecutar los shells construidos usando LISP. Para evitar este problema, varias compañías ofrecen máquinas diseñadas de manera específica para ejecutar código LISP, éstas lo usan a plenitud, incluso como lenguaje ensamblador, sin embargo, cuestan mucho más que las máquinas convencionales y son para usuarios individuales.

Este problema de alto costo tiene un impacto tanto en el desarrollo como en el **problema de distribución**. No es suficiente desarrollar un programa estupendo si éste no puede distribuirse para su uso debido a su alto costo. Una buena estación de trabajo de desarrollo no es necesariamente un buen vehículo de distribución debido a su velocidad, poder, tamaño, peso, ambiente o restricciones de costo. Algunas aplicaciones incluso pueden requerir que el código final se sitúe en ROM por razones de costo y duración. Poner el código en ROM puede ser un problema con algunas herramientas de la AI y los sistemas expertos que requieren hardware especial para ejecutarse. Es mejor considerar esta posibilidad por adelantado, en vez de tener que volver a codificar un programa.

Un problema adicional es el de insertar la AI con los lenguajes de programación convencional como C, Ada, Pascal y FORTRAN. Por ejemplo, algunas aplicaciones que requieren procesamiento extensivo de números se hacen mejor en los lenguajes convencionales, o en los lenguajes para sistemas expertos escritos en LISP o PROLOG.

A menos que se tomen previsiones especiales, los sistemas expertos escritos en LISP a menudo son difíciles de insertar en cualquier programa diferente. Una consideración importante al seleccionar un lenguaje de AI debe ser el lenguaje en el que la herramienta está escrita. Por razones de transporte, eficiencia y velocidad, muchas herramientas de los sistemas expertos en la actualidad se escriben en C o se convierten a él. Esto también elimina el problema de necesitar costoso hardware especial para las aplicaciones basadas en LISP.

## 1.12 PARADIGMAS QUE NO SON DE PROCEDIMIENTOS

Los paradigmas que no son de procedimientos no dependen de que el programador proporcione los detalles exactos sobre cómo solucionar un problema. Esto es lo opuesto a los paradigmas de procedimientos, que especifican la manera en que se computa una función o una secuencia de afirmaciones. Hay que recordar que en los paradigmas que no son de procedimientos, el énfasis está en especificar lo que se va a realizar y en permitir que el sistema determine cómo va a realizarlo.

### Programación declarativa

El **paradigma declarativo** separa la **meta** de los métodos utilizados para alcanzarla. El usuario especifica la meta mientras que el mecanismo subyacente de implementación trata de satisfacerla. Se han creado varios paradigmas y lenguajes de programación asociados para implementar el modelo declarativo.

### Programación orientada a objetos

El paradigma **orientado a objetos** es otro caso de paradigma que puede considerarse en parte imperativo y en parte declarativo. Hoy en día, el término *orientado a objetos* se usa en dos formas diferentes. La expresión **diseño orientado a objetos** está creciendo en popularidad como una metodología de programación para los lenguajes de programación imperativa como Ada y Modula-2. La idea básica es diseñar un programa considerando los datos usados en él como objetos y después implementando operaciones en esos objetos. Esto es lo opuesto a los diseños inferiores, que proceden a partir del refinamiento gradual de una estructura de control del programa. En realidad, el diseño orientado a objetos es, en esencia, lo que solía llamarse diseño superior; por desgracia, el término *superior* nunca sonó tan impresionante como *orientado a objetos*.

Como un ejemplo del diseño orientado a objetos, consideremos la tarea de escribir un programa para administrar una cuenta de cobros con menú interactivo (Riley 87). Los objetos de datos importantes son el saldo actual, el monto del cobro y el monto del pago. Es posible definir varias operaciones para actuar en los objetos de datos; estas operaciones serían añadir cobro, hacer pago y añadir interés mensual. Una vez que todos los objetos de datos, las operaciones y la interfaz del menú están definidos, puede empezar la codificación. Esta metodología de diseño orientado a objetos se adapta de manera adecuada a un programa con una estructura de control débil. No sería tan adaptable en un programa que requiere una estructura de control fuerte, como una aplicación de nómina de pago. Para el caso de la nómina de pago hay una secuencia definida de pasos a seguir.

1. Obtener los datos de la tarjeta checadora
2. Contabilizar las incapacidades y las vacaciones
3. Multiplicar las horas trabajadas por la tasa de pago
4. Multiplicar las horas de tiempo extra por la tasa de tiempo extra
5. Añadir bonos o comisiones de venta, si son aplicables
6. Sustraer las deducciones por impuestos, seguro, cuotas y retiro
7. Emitir cheque de pago

El diseño orientado a objetos no requiere funciones especiales de lenguaje, puede hacerse en FORTRAN, BÁSICO, C, etcétera. Los lenguajes como Ada y Modula-2 apoyan el diseño orientado a objetos porque estos lenguajes cuentan con funciones para encapsular los datos en módulos y paquetes.

El término **programación orientada a objetos** se usó por primera vez para referirse a lenguajes como Smalltalk, que estaba diseñado de manera específica para objetos. En la actualidad, el término se usa con frecuencia para referirse a la programación de un diseño orientado a objetos, incluso en un lenguaje que no tiene soporte de objetos verdadero.

Smalltalk tiene funciones para soportar objetos construidos dentro del lenguaje, de hecho, tiene un ambiente de programación construido en su totalidad mediante objetos. Smalltalk desciende de SIMULA 67, un lenguaje desarrollado para simulación que introdujo el concepto de **clase**, que lleva al concepto de información oculta en módulos. Una clase es en esencia un tipo, que es una plantilla que define la estructura de los datos. En realidad, los conceptos de tipo en Pascal y Modula-2 viene del de clase. Un **caso** de una clase es un objeto de datos que puede manipularse; este término ha pasado a los sistemas expertos, donde denota un hecho que corresponde a un patrón. Del mismo modo, se dice que una regla ha sido requerida si su lado izquierdo está satisfecho. En los sistemas basados en reglas, los términos *activada* y *requerida* son sinónimos.

Otro concepto significativo que proviene de SIMULA 67 es el de **herencia**, en este sistema podía definirse una **subclase** para heredar las propiedades de las clases. Por ejemplo, una clase puede estar definida para componerse de objetos que pueden usarse en una pila y otra clase definida para integrarse de números complejos. Una subclase puede definirse con facilidad como objetos que son números complejos usados en una pila. Es decir, estos objetos han heredado propiedades de las dos clases superiores, llamadas **superclases**. El concepto de herencia puede extenderse para organizar los objetos en una jerarquía en la que los objetos puedan heredarse de sus clases, que a su vez puedan heredarse de sus clases y así sucesivamente. La herencia es útil porque los objetos pueden heredar propiedades de sus clases sin que el programador tenga que especificar cada propiedad. Muchas de las herramientas de los sistemas expertos que se usan hoy en día, como ART y KEE, permiten la herencia porque es una herramienta versátil en la construcción de grandes grupos de hechos.

## Programación lógica

Una de las primeras aplicaciones de la inteligencia artificial en las computadoras, fue la de probar teoremas lógicos con el programa Logic Theorist de Newell y Simon. Se informó por primera vez sobre este programa en la conferencia de Dartmouth, sobre AI en 1956 y causó sensación porque las computadoras electrónicas sólo se habían usado antes para cálculos numéricos. Ahora una computadora estaba razonando de verdad las pruebas de los teoremas matemáticos, una tarea que se pensaba que sólo los matemáticos eran capaces de realizar.

En el Logic Theorist y su sucesor, el General Problem Solver (GPS), Newell y Simon se concentraron en tratar de implementar algoritmos avanzados que pudieran resolver cualquier problema. Mientras que el Logic Theorist estaba pensado sólo para probar teoremas matemáticos, el GPS estaba diseñado para resolver cualquier clase de problema lógico, incluyendo juegos y acertijos como los del ajedrez, la Torre de Hanoi, Misioneros y caníbales y criptaritmético. Un ejemplo de sus famosos acertijos criptaritméticos (aritmética secreta) es

DONALD  
+ GERALD  
ROBERT

donde se sabe que D = 5. El objetivo es descubrir los valores aritméticos de las otras letras en el rango de 0 a 9.

El GPS fue el primer programa para solución de problemas que separó con claridad el conocimiento para resolver problemas del dominio de conocimiento. Este paradigma de separar en forma explícita el conocimiento para resolver problemas del dominio de conocimiento se usa

hoy en día como la base de los sistemas expertos. En los sistemas expertos actuales, el mecanismo de inferencia decide cuál conocimiento debe usarse y cómo debe aplicarse.

Los esfuerzos para mejorar la comprobación de los teoremas mecánicos continuaron. A principios de los setenta, se descubrió que la computación es un caso especial de deducción mecánica lógica (Kowalski 1985). Cuando el encadenamiento hacia atrás se aplicó a enunciados con la forma “conclusión si condiciones”, éste fue bastante eficaz para la comprobación significativa de teoremas. Las condiciones pueden considerarse como patrones representativos que serán igualados, como en las reglas de producción que ya se analizaron. A los enunciados expresados en esta forma, se les llama cláusulas de Horn en honor a Alfred Horn, quien fue el primero en investigarlas. En 1972, Kowalski, Colmerauer y Roussel crearon el lenguaje PROLOG, para implementar la programación lógica por encadenamiento hacia atrás, usando las cláusulas de Horn.

El encadenamiento hacia atrás puede usarse para representar el conocimiento en una representación declarativa y para controlar el proceso de razonamiento; por lo general, procede definiendo **submetas** más pequeñas que deben satisfacerse si la meta inicial va a satisfacerse. Estas submetas a su vez se dividen en submetas más pequeñas y así sucesivamente.

Un ejemplo de conocimiento declarativo es el siguiente ejemplo clásico:

```
Todos los hombres son mortales
Sócrates es un hombre
```

lo que podría expresarse en cláusulas de Horn:

```
alguien es mortal
  si alguien es un hombre
Sócrates es un hombre
  si (en todos los casos)
```

En el enunciado sobre Sócrates, la condición *si* es verdadera en todos los casos. En otras palabras, el conocimiento sobre Sócrates no requiere correspondencia con ningún patrón. Comparando esto con el caso mortal en el que alguien debe ser un hombre para satisfacer el patrón de la condición *si*, nótese que la cláusula de Horn puede interpretarse como un procedimiento que indica cómo satisfacer un objetivo. Esto quiere decir que para determinar si alguien es mortal es necesario determinar si alguien es un hombre. A continuación se muestra un ejemplo algo más complejo:

```
Un automóvil necesita gasolina, aceite y las llantas
infladas para correr
```

que puede expresarse en una cláusula de Horn como:

```
x es un automóvil y corre
  si x tiene gasolina y
  si x tiene aceite y
  si x tiene llantas infladas
```

Nótese que el problema de determinar si un automóvil correrá ha sido reducido a tres subproblemas o submetas más simples. Ahora supongamos que hay algún conocimiento declarativo adicional como el siguiente:

```
El medidor de combustible muestra cargado si un automóvil
tiene gasolina
La varilla del aceite muestra cargado si un automóvil
tiene aceite
```

El medidor de presión de aire señala al menos 20 si un automóvil tiene las llantas infladas  
 El medidor de combustible muestra cargado  
 La varilla del aceite indica vacío  
 El medidor de presión de aire muestra cuando menos 15

Esto puede traducirse en las siguientes cláusulas de Horn:

- x tiene gasolina
  - si el medidor de combustible muestra cargado
- x tiene aceite
  - si la varilla del aceite muestra cargado
- x tiene las llantas infladas
  - si el medidor de presión de aire muestra cuando menos 20
- El medidor de combustible no está vacío
  - si (en todos los casos)
- La varilla del aceite indica vacío
- La presión de aire es al menos de 15
  - si (en todos los casos)

A partir de estas cláusulas, un comprobador de teoremas mecánicos puede probar que el automóvil no correrá porque no hay aceite y la presión de aire es insuficiente.

Una de las ventajas de los sistemas de encadenamiento hacia atrás es que su ejecución puede proceder en paralelo. Esto quiere decir que si están disponibles varios procesadores, podrían trabajar de manera simultánea en la satisfacción de submetas. Esta operación paralela acelera en gran medida la ejecución de los programas y es una de las razones por las que los japoneses escogieron PROLOG como el lenguaje de programación para su proyecto de Computadoras de quinta generación (Moto-Oka 1982). Éste fue un proyecto ambicioso para desarrollar la siguiente generación de computadoras para los noventa, con funciones que sobrepasaran con mucho las de los modelos actuales. Estas máquinas iban a programarse con técnicas declarativas, pero este método ha tenido dificultades. Los japoneses anunciaron el proyecto de Quinta generación en 1981, con un presupuesto de 850 millones de dólares; las 26 metas iniciales del proyecto incluían la traducción de lenguaje natural del japonés a inglés y viceversa, la solución de problemas y la inferencia mil veces más rápida que las máquinas actuales; además de las funciones para leer texto manuscrito, entender imágenes, introducir enormes bases de datos y conversar en lenguaje natural.

PROLOG es más que sólo un lenguaje. A un nivel primario, PROLOG es un shell porque requiere

- un intérprete o mecanismo de inferencia
- una base de datos (hechos y reglas)
- un formulario de correspondencia de patrones llamado **unificación**
- un mecanismo de **rastreo hacia atrás** para perseguir submetas alternas, si fracasa la búsqueda para satisfacer una meta.

Versiones más elaboradas de PROLOG, como TurboPROLOG de Borland, son herramientas debido a todas las mejoras que tienen.

Como ejemplo del encadenamiento hacia atrás, supongamos que si tiene cambio o una tarjeta de crédito, puede pagar por el aceite para hacer que el automóvil corra, una submeta es revisar si tiene cambio. Si no existe el hecho de traer cambio, el mecanismo de encadenamiento hacia atrás explorará la otra submeta para ver si trae una tarjeta de crédito; si es así, la

meta de pagar por el aceite puede satisfacerse. Nótese que la ausencia de un hecho para comprobar una meta es tan efectivo, aunque quizás menos eficiente, como un hecho negativo como "la varilla del aceite indica vacío". Tanto los hechos negativos como los faltantes pueden provocar que no se satisfaga una meta.

Si el encadenamiento hacia atrás y los mecanismos de correspondencia de patrones no son necesarios para el problema, entonces el programador puede trabajar en torno a ellos o codificar en un lenguaje diferente. Una de las ventajas de la programación lógica son las especificaciones ejecutables. Es decir, especificar los requerimientos de un problema con cláusulas Horn produce un programa ejecutable, diferente a la programación convencional, donde el documento de necesidades no tiene un aspecto del todo parecido al código ejecutable final. Los lenguajes imperativos más recientes, como Ada, intentan evitar este problema ofreciendo la capacidad de usar Ada mismo como el lenguaje de descripción del programa (PDL). El PDL puede usarse como un esqueleto sobre el cual construir el resto del programa, pero si el PDL no trabaja correctamente, tampoco lo hará el programa.

A diferencia de los sistemas de reglas de producción, el orden en que se introducen las submetas, los hechos y las reglas en un programa PROLOG tiene efectos significativos. La eficiencia (y por consiguiente la velocidad) se ven afectadas por la manera en que PROLOG busca en su base de datos, sin embargo, hay algunos programas que se ejecutan correctamente si las submetas, los hechos y las reglas se introducen de una manera, pero van en un ciclo infinito o tienen un mensaje de error de tiempo de ejecución si la orden cambia (Ghezzi 87, pp. 304-306).

## Sistemas expertos

Los sistemas expertos pueden considerarse programación declarativa porque el programador no especifica cómo es que el programa va a alcanzar su objetivo al nivel de un algoritmo. Por ejemplo, en un sistema experto basado en reglas, cualquier regla puede activarse y pasar a la agenda si su lado izquierdo corresponde a los hechos. El orden en que las reglas se introdujeron no afecta cuáles reglas se activan, de este modo, la orden de enunciado del programa no especifica un flujo de control rígido. Otros tipos de sistemas expertos se basan en **marcos**, que se analizarán en el capítulo 2, y **redes de inferencia**, que se comentan en el capítulo 4.

Hay muchas diferencias entre los sistemas expertos y los programas convencionales. En la tabla 1.3 se muestran algunas de las diferencias típicas.

Característica	Programa convencional	Sistema experto
Controlado por...	Orden de enunciado	Mecanismo de inferencia
Control y datos	Integración implícita	Separación explícita
Fuerza de control	Fuerte	Débil
Solución por...	Algoritmo	Reglas e inferencia
Búsqueda de solución	Pequeña o nula	Grande
Solución del problema	El algoritmo es correcto	Reglas
Entrada	Se asume como correcta	Incompleta, incorrecta
Entrada inesperada	Diffícil de tratar	Gran capacidad de respuesta
Salida	Siempre correcta	Varía con el problema
Explicación	Ninguna	De manera habitual
Aplicaciones	Numérica, archivo y texto	Razonamiento simbólico
Ejecución	Por lo general, secuencial	Reglas oportunistas
Diseño del programa	Diseño estructurado	Poca o ninguna estructura
Modificable	Con dificultad	Razonable
Expansión	Hecha a grandes saltos	Creciente

Tabla 1.13

Algunas diferencias comunes entre los programas convencionales y los sistemas expertos

Los sistemas expertos a menudo están diseñados para tratar con la incertidumbre debido a que el razonamiento es una de las mejores herramientas que hemos descubierto para hacerlo. La incertidumbre puede surgir en los datos que se introducen al sistema experto e incluso en la base de conocimiento misma; al principio esto puede parecer sorprendente a la gente acostumbrada a la programación convencional. Sin embargo, gran parte del conocimiento humano es heurístico, lo que significa que sólo puede trabajar de manera adecuada parte del tiempo. Además, los datos de entrada pueden ser incorrectos, estar incompletos, resultar inconsistentes, o tener otros errores. Las soluciones algorítmicas no son capaces de tratar con estas situaciones porque un algoritmo garantiza la solución de un problema en una serie finita de pasos.

Dependiendo de los datos de entrada y de la base de conocimiento, el sistema experto puede llegar a la respuesta correcta, a una buena respuesta, a una mala respuesta o a una falta de respuesta. Aunque esto pudiera parecer perturbador al principio, la alternativa sería no tener respuesta en ningún caso. De nuevo, lo importante, y que debe tenerse en mente es que un sistema experto no actuará peor que el mejor para resolver este problema (un especialista humano), y quizás lo hará mejor. Si conocieramos un método algorítmico que fuera superior a un sistema experto, lo usaríamos. Lo importante es usar la mejor, y quizás única, herramienta para el trabajo.

### Programación no declarativa

Los paradigmas no declarativos se están volviendo populares. Estos nuevos paradigmas se utilizan cada vez más para una amplia variedad de aplicaciones. Pueden usarse en aplicaciones individuales o junto con otros paradigmas.

### Programación basada en la inducción

Una aplicación de la AI que está despertando interés es la programación **basada en inducción**, en este paradigma el programa aprende a partir de ejemplos. Una aplicación a la que se ha aplicado este paradigma es el acceso a bases de datos; en lugar de que el usuario tenga que escribir los valores específicos de uno o más campos para una búsqueda, basta con seleccionar de los ejemplos apropiados uno o más campos con esas características. El programa infiere las características de los datos y busca su correspondencia en la base de datos.

Las herramientas para los sistemas expertos, como 1st-Class y KDS, ofrecen aprendizaje por inducción, por lo que aceptan ejemplos y estudios de caso, y generan reglas automáticamente.

## 1.13 SISTEMAS NEURONALES ARTIFICIALES

Durante los ochenta surgió un nuevo desarrollo en la programación de paradigmas, los **sistemas neuronales artificiales (ANS, artificial neural systems)**, basados en la forma en que el cerebro procesa la información. A este paradigma a veces se le llama conexiónismo, porque modela las soluciones a los problemas entrenando neuronas simuladas, conectadas en una red de trabajo. Muchos investigadores están estudiando las redes neuronales, pues las redes mantienen un gran potencial como frente de los sistemas expertos que requieren cantidades masivas de entrada mediante sensores, además de respuestas en tiempo real.

### El problema del agente viajero

ANS ha tenido un éxito notable al proporcionar respuestas en tiempo real a complejos problemas de reconocimiento de patrones. En una ocasión, una red neuronal ejecutándose en una microcomputadora ordinaria obtuvo una muy buena solución al problema del agente viajero

en 0.1 segundos, comparada con la solución óptima que requiere una hora de tiempo de CPU en un mainframe (Port 86). El problema del agente viajero es importante porque se trata del problema clásico que se enfrenta con optimizar la ruta de señales en un sistema de telecomunicaciones. Optimizar la ruta es importante para minimizar el tiempo de viaje y, por tanto, la eficiencia y velocidad.

El problema básico del agente viajero es el de calcular las rutas más cortas a través de una lista de ciudades dada. En la tabla 1.14 se muestran las rutas posibles de una a cuatro ciudades. Observe que el número de rutas es proporcional al factorial del número de ciudades menos uno, ( $N - 1$ )!

Número de ciudades	Rutas
1	1
2	1-2-1
3	1-2-3-1 1-3-2-1
4	1-2-3-4-1 1-2-4-3-1 1-3-2-4-1 1-3-4-2-1 1-4-2-3-1 1-4-3-2-1

Tabla 1.14 Rutas del problema del agente viajero

Mientras que hay  $9! = 362880$  rutas para diez ciudades, hay  $29! = 8.8E30$  rutas posibles para treinta ciudades. Este problema es un ejemplo clásico de la **explosión combinatoria**, porque el número de rutas posibles se incrementa tan rápido que no hay soluciones prácticas para una cantidad realista de ciudades. Si toma una hora del CPU de un mainframe resolver este problema para 30 ciudades, tomaría 30 horas para 31 ciudades y 330 horas para 32 ciudades. Estos son números muy pequeños en realidad cuando se comparan con los miles de interruptores de telecomunicaciones y las ciudades que se utilizan en la asignación de rutas para paquetes de datos y artículos reales.

Una red neuronal puede resolver el caso de 10 ciudades con la misma rapidez que para 30, mientras que a una computadora convencional le toma mucho más. Para el caso de las 10 ciudades, la red neuronal dio con una de las dos mejores rutas y para el caso de las 30 ciudades, dio con una de las 100,000,000 mejores rutas. Esto es más impresionante si uno se da cuenta de que esta solución se encuentra entre las 1E-22 mejores soluciones. A pesar de que las redes neuronales no siempre obtienen respuesta óptima, pueden proporcionar la mejor conjectura en tiempo real. En muchos casos, es mejor tener un 99.999999999999999999 por ciento de respuestas correctas en un milisegundo, que una respuesta cien por ciento correcta en 30 horas.

## Elementos de un sistema neuronal artificial

Un ANS es básicamente una computadora análoga que usa elementos de procesamiento simples, conectados de una manera altamente paralela. Los elementos de procesamiento ejecutan en sus entradas funciones booleanas o aritméticas simples. La llave para el funcionamiento de un ANS son los **pesos** asociados con cada elemento, son éstos los que representan la información almacenada en el sistema.

En la figura 1.10 se muestra una neurona artificial típica, puede tener múltiples entradas pero solo una salida. El cerebro humano contiene cerca de  $10^{10}$  neuronas y una neurona puede tener miles de conexiones con otra. Las señales de entrada a la neurona se multiplican por los

pesos y se suman para obtener la entrada total de la neurona,  $I$ . Los pesos pueden representarse como una matriz e identificarse con subíndices.

$$I \equiv \text{Entrada de la neurona}_j = \sum_j W_{ij} I_j$$

$$\text{Salida de la neurona} = \frac{1}{1 + e^{-(I - \theta)}}$$

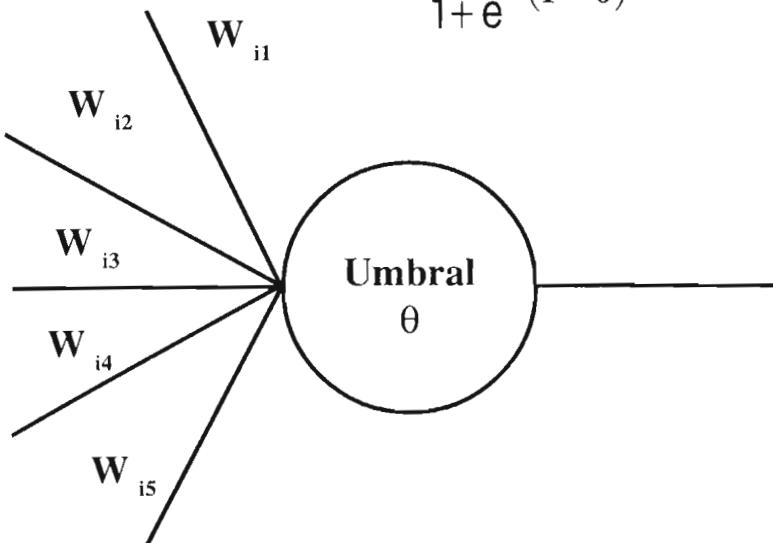


Figura 1.10 Elemento de proceso de una neurona

A menudo, la salida de la neurona se toma como una **función sigmoidea** de la entrada. La sigmoidea es representativa de las neuronas reales, que se acercan a los límites para entradas muy pequeñas o muy grandes, y se le llama **función de activación**, y una función de uso muy común es  $(1 + e^{-x})^{-1}$ . Cada neurona tiene también un **valor de entrada** asociado,  $\theta$ , que se resta a la entrada total,  $I$ . En la figura 1.11 se muestra un ANS que puede calcular la **O exclusiva (XOR)** de sus entradas, usando una técnica llamada propagación hacia atrás. La XOR sólo da una salida verdadera cuando sus entradas no son todas verdaderas ni todas falsas. El número de nodos en el estrato escondido variará dependiendo de la aplicación y el diseño.

Las redes neuronales no están programadas en el sentido convencional: Se conocen cerca de 13 algoritmos de aprendizaje para las redes neuronales, como **propagación de contador** y propagación hacia atrás para entrenar redes. El programador “programa” la red con sólo sustituir la entrada y los datos de salida correspondientes. La red aprende ajustando automáticamente los pesos en la red de trabajo que conecta las neuronas. Los pesos y los valores de umbral de las neuronas determinan la propagación de los datos a través de la red y, por tanto, su respuesta correcta a los datos de entrenamiento. Entrenar la red para que obtenga las respuestas correctas puede tomar horas o días, dependiendo del número de patrones que la red deba aprender, el hardware y el software. Sin embargo, una vez que se realiza el aprendizaje, la red responde con rapidez.

Si la simulación del software no es lo bastante rápida, ANS puede fabricarse en chips para que responda en tiempo real; una vez que se ha entrenado a la red de trabajo y que se han determinado los pesos, se puede construir un chip. AT&T y otras compañías están fabricando chips experimentales para redes neuronales que contienen cientos de neuronas, es muy probable que en los próximos años se fabriquen chips que contengan miles de ellas.

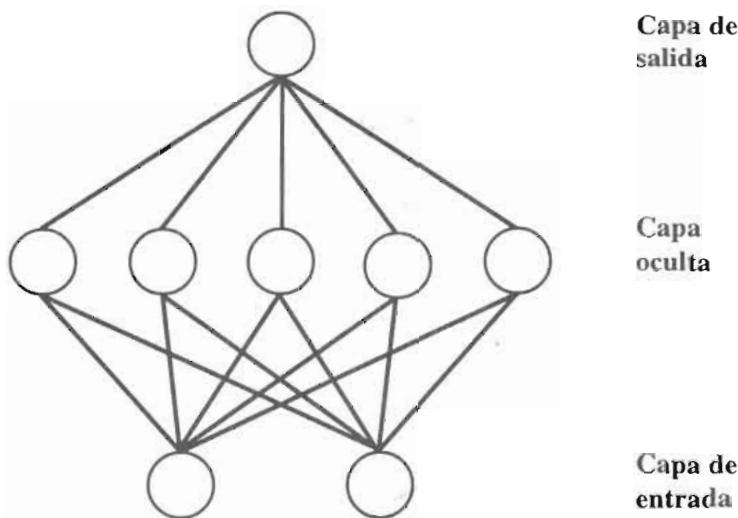


Figura 1.11 Red de propagación hacia atrás

### Características de los Sistemas Neuronales Artificiales

La arquitectura de los ANS es muy diferente de la arquitectura de las computadoras convencionales. En una computadora convencional es posible correlacionar información discontinua con las celdas de memoria. Por ejemplo, un número de seguro social puede almacenarse como código ASCII en un grupo contiguo de celdas de memoria, al examinar el contenido de este grupo contiguo, puede reconstruirse directamente el número de seguro social; esta reconstrucción es posible porque hay una relación de uno a uno entre cada carácter del número de seguro social y la celda de memoria que contiene el código ASCII de ese carácter.

Los ANS están modelados a partir de las teorías actuales del cerebro, donde la información se representa por los pesos. Sin embargo, no existe correlación directa entre un peso y un artículo específicos de la información almacenada. Esta representación distributiva de la información es similar a la de un holograma, en el que las líneas actúan como una rejilla de difracción para reconstruir la imagen almacenada cuando la luz del láser la atraviesa.

Una red neuronal es una buena opción cuando hay una cantidad considerable de datos empíricos y no hay un algoritmo que proporcione velocidad y precisión suficientes. Los ANS tienen muchas ventajas en comparación con el almacenamiento de las computadoras convencionales.

- *El almacenamiento es tolerante a fallas.* Pueden eliminarse partes de la red y sólo hay una degradación en la calidad de los datos almacenados. Esto ocurre porque la información está almacenada de manera distribuida.
- *La calidad de la imagen almacenada se degrada poco a poco en proporción a la cantidad de red que se elimina.* No hay pérdida catastrófica de la información. El almacenamiento y la calidad de las funciones también son características de los hologramas.
- *Los datos se almacenan con naturalidad en forma de memoria asociativa.* Una memoria asociativa es aquella en la que datos parciales bastan para recordar la información completa almacenada. Esto difiere de la memoria convencional, en

la que los datos se recuerdan especificando su dirección. Una entrada parcial o con ruido puede seguir produciendo la información original completa.

- *Las redes pueden extrapolarse e interpolarse a partir de su información almacenada.* El entrenamiento enseña a una red a buscar rasgos significativos o relaciones entre los datos; después la red puede extraer para sugerir relaciones con nuevos datos. En un experimento (Hinton 86), se entrenó a una red neuronal sobre las relaciones familiares de 24 personas hipotéticas; luego la red también pudo responder correctamente acerca de relaciones sobre las que no había sido entrenada.
- *Las redes tienen plasticidad.* Aunque se eliminan varias neuronas, la red puede volver a entrenarse a su nivel original de habilidad, si quedan las neuronas suficientes. Ésta es también una característica del cerebro, en el que, si se destruyen algunas partes, pueden reaprenderse los niveles originales de habilidad.

Estas características hacen muy atractivos a los ANS para las naves espaciales robotizadas, equipos de campos petroleros, dispositivos submarinos, controles de procesamiento y otras aplicaciones que necesitan funcionar largo tiempo en ambientes hostiles sin reparaciones. Además de la confiabilidad, los ANS ofrecen la posibilidad de tener bajos costos de mantenimiento debido a su plasticidad. Aunque se pueden hacer reparaciones de hardware, probablemente sea más rentable reprogramar la red neuronal en lugar de reemplazarla.

Los ANS no suelen ser adecuados para aplicaciones que requieren procesamiento de números o una solución óptima. Además, si existe una solución algorítmica práctica, un ANS no es una buena elección.

## Desarrollo de la tecnología de los ANS

Los orígenes de ANS se encuentran en el modelo matemático de las neuronas hecho por McCulloch y Pitts en 1943 (McCulloch 43). Hebb explicó en 1949 cómo aprenden las neuronas (Hebb 49). En el aprendizaje hebbiano, la eficiencia de una neurona para estimular a otra se incrementa con el **disparo**. El término *disparo* significa que una neurona emite un impulso electroquímico que puede estimular a otras neuronas conectadas a ella. Hay evidencia de que la conductividad de las conexiones entre neuronas, llamada **sinapsis**, se incrementa con los disparos. En los ANS, el peso de las conexiones entre neuronas cambia para simular la conductividad cambiante de las neuronas naturales.

En 1961, Rosenblatt publicó un libro de gran influencia que trataba con el nuevo tipo de sistema de neuronas artificiales que investigaba, llamado **perceptrón** (Rosenblatt 61). El perceptrón era un dispositivo notable que mostraba aptitudes para el aprendizaje y reconocimiento de patrones; consistía básicamente de dos capas de neuronas y un algoritmo de aprendizaje simple. Los pesos tenían que asignarse manualmente, en contraste con los modernos ANS, que asignan los pesos por sí solos, basados en su entrenamiento. Muchos investigadores entraron al campo de los ANS y comenzaron a estudiar los perceptrones durante los sesenta.

La era inicial del perceptrón terminó en 1960, cuando Minsky y Papert publicaron un libro, *Perceptrons*, que mostraba las limitaciones teóricas de los perceptrones como máquinas de cálculo general (Minsky 69). Señalaron una deficiencia del perceptrón: sólo era capaz de calcular 14 de las 16 funciones lógicas básicas, lo que significa que un perceptrón no es un dispositivo de cálculo para propósitos generales; en particular, probaron que un perceptrón no puede reconocer el O exclusivo. Aunque no habían investigado seriamente los ANS de múltiples capas, expresaron la visión pesimista de que las múltiples capas probablemente no lograrían resolver el problema XOR. Los fondos gubernamentales para la investigación de

los ANS pasaron a favorecer al método simbólico de AI utilizando lenguajes como LISP y algoritmos. Durante los años setenta, se popularizaron nuevos métodos para representar información simbólica en la AI a través de marcos, inventados por Minsky. Se ha seguido trabajando en los perceptrones, y nuevos tipos de éstos han logrado superar las objeciones de Minsky (Reece 87). Debido a su simplicidad, es fácil construir perceptrones y otros sistemas ANS con la tecnología moderna de circuitos integrados.

La investigación de los ANS continuó en pequeña escala durante los setenta. Sin embargo, el campo entró finalmente en un renacimiento que se inició con el trabajo de Hopfield en 1982 (Hopfield 82). Él dio a los ANS una firme base teórica con la red Hopfield de dos capas, y demostró cómo los ANS pueden resolver una amplia variedad de problemas. La estructura general de una red Hopfield se muestra en la figura 1.12. En particular, Hopfield mostró cómo un ANS podía resolver el problema del agente viajero en un tiempo constante, comparado con la explosión combinatoria encontrada por las soluciones algorítmicas convencionales. Una forma de circuito electrónico de un ANS pudo resolver el problema en un microsegundo. Los ANS pueden resolver fácilmente otros problemas de optimización combinatoria, como el mapa de los cuatro colores, la correspondencia euclíadiana (Hopfield 86b) y el código de transposición (Tank 85).

Un ANS que puede resolver fácilmente el problema del XOR es la red de **propagación hacia atrás**, también conocida como la **regla delta generalizada** (Rumelhart 86). La red de propagación hacia atrás suele implantarse como una red de tres capas, aunque pueden especificarse capas adicionales. A las capas que se encuentran entre las capas de entrada y salida se les llama **capas ocultas**, porque sólo las capas de entrada y de salida están visibles para el mundo exterior. Otro tipo popular de ANS es el de propagación de conteo, inventado por Hetch-Nielsen en 1986 (Hetch-Nielsen 90). Un importante resultado teórico de las matemáticas, el teorema de Kolmogorov, puede interpretarse como prueba de que una red de trabajo con tres capas,  $n$  entradas y  $2n + 1$  neuronas en la capa oculta, puede crear un mapa de cualquier función continua (Hetch-Nielsen 87).

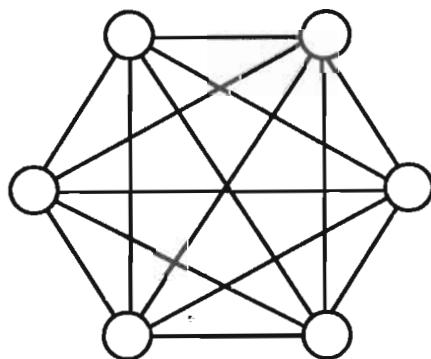


Figura 1.12 Red artificial neuronal de Hopfield

### Aplicaciones de la tecnología de ANS

Un ejemplo significativo del aprendizaje por propagación hacia atrás se demostró con una red neuronal que aprendió la pronunciación correcta de palabras tomadas de un texto (Sejnowski 86). El ANS fue entrenado corrigiendo su salida con el uso de un dispositivo de texto a pronunciación de DEC llamado DECTalk. Se requirieron 20 años de investigación lingüística para determinar las reglas para pronunciación correcta que utilizó DECTalk. El ANS aprendió las habilidades de pronunciación equivalentes en una sola noche, escuchando únicamente la pronunciación correcta del habla a partir del texto; no se programó ninguna habilidad lingüística en el ANS.

Existen investigaciones de los ANS encaminadas al reconocimiento de blancos de radar, a través de computadoras electrónicas y ópticas (Farhat 86). Las nuevas implantaciones de redes neuronales, utilizando componentes ópticos, prometen computadoras ópticas con velocidades millones de veces superiores a las electrónicas. La implementación óptica de los ANS es atractiva a causa del paralelismo inherente de la luz. Es decir, los rayos de luz no interfieren entre sí mientras viajan; es posible generar y manipular fácilmente grandes cantidades de fotones con componentes ópticos como espejos, lentes, moduladores de luz espacial de alta velocidad programables, series de dispositivos ópticos biestables que pueden funcionar como neuronas ópticas, y rejillas de difracción. Las computadoras ópticas diseñadas como ANS parecen complementarse; los avances futuros de los ANS parecen promisorios (Giarratano 90 a). En particular, los ANS son útiles para controlar sistemas en los que las propuestas convencionales no resultan satisfactorias (Giarratano 91 b).

### Desarrollos comerciales de los ANS

Varias nuevas compañías y otros ya existentes se han organizado para desarrollar tecnología y productos para los ANS. Nestor comercializa un producto de ANS llamado NestorWriter, que puede reconocer entradas manuscritas y convertirlas en texto usando una PC. Otras compañías como TRW, SAIC, HNC, Synaptics, Neural Tech, Revelations Research y Texas Instruments, comercializan diversos simuladores ANS y tarjetas aceleradoras de hardware para acelerar el aprendizaje. Una de las mejores gangas para entrar en la computación neuronal es el volumen tres de las obras de Rumelhart sobre procesamiento distribuido en paralelo (editorial MIT Press). El libro describe media docena de simuladores ANS y también incluye un disquete con software para una máquina compatible con IBM-PC por \$27.50. Adquiriendo una tarjeta aceleradora, el usuario puede tener un potente sistema para ANS a bajo costo.

## 1.14 SISTEMAS EXPERTOS INTERCONECTADOS Y APRENDIZAJE INDUCTIVO

Es posible construir sistemas expertos usando los ANS. En un sistema, el ANS es la base de conocimiento, construida mediante ejemplos de entrenamiento, tomados de la medicina (Gallant 88). En este caso, los sistemas expertos tratan de clasificar una enfermedad, por medio de sus síntomas, dentro de una de las enfermedades conocidas en que el sistema ha sido entrenado. Un mecanismo de inferencia conocido como MACIE (Matrix Controlled Inference Engine: mecanismo de inferencia controlado por matriz) fue diseñado para usar la base de conocimiento ANS. El sistema usa el encadenamiento hacia delante para hacer inferencias y el encadenamiento hacia atrás para solicitar al usuario cualquier dato adicional necesario para llegar a una respuesta. A pesar de que los ANS no pueden explicar por sí solos la razón por la que se le asignaron ciertos valores a sus pesos, MACIE puede interpretar los ANS y generar reglas SI...ENTONCES para explicar su conocimiento.

Un sistema experto de ANS como éste utiliza el **aprendizaje inductivo**; es decir, el sistema **induce** la información en su base de conocimiento a través de ejemplos. La inducción es el proceso de inferir casos generales a partir de casos específicos. Además de los ANS, hay en el mercado varias estructuras de los sistemas expertos que, de manera explícita, generan reglas a partir de ejemplos. La meta del aprendizaje inductivo es reducir o eliminar el cuello de botella que representa la adquisición de conocimiento. Al colocar el esfuerzo de adquisición de conocimiento en el sistema experto, es posible reducir el tiempo de desarrollo y aumentar la confiabilidad si el sistema induce reglas desconocidas para un ser humano. Los sistemas expertos se han combinado con los ANS (Giarratano 90b).

## 1.15 RESUMEN

En este capítulo se han revisado los problemas y desarrollos que han conducido hacia los sistemas expertos. Por lo general, con la programación convencional no es posible resolver los problemas para los que se usan los sistemas expertos, porque ésta carece de conocimiento o de un algoritmo eficiente. Como los sistemas expertos están basados en el conocimiento, pueden usarse con efectividad en problemas del mundo real, de estructura nociva y difíciles de resolver por otros medios.

También se analizaron las ventajas y desventajas de los sistemas expertos, en el contexto de la selección de un dominio de problema apropiado para su aplicación y se describieron los criterios para seleccionar las aplicaciones apropiadas.

Se analizaron las bases de una shell para los sistemas expertos con referencia a los basados en reglas. Se describió el ciclo básico de mecanismo de inferencia acto reconocimiento y se ilustró con un ejemplo de regla simple. Finalmente, se describió la relación entre los sistemas expertos y otros paradigmas de programación a partir del dominio apropiado de cada paradigma. Lo importante es el concepto de que los sistemas expertos deben ser vistos como otra herramienta de programación que resulta adecuada para algunas aplicaciones y no lo es para otras. En capítulos posteriores se describirán con más detalle las características y la adaptabilidad de los sistemas expertos.

La mejor fuente de información actual acerca de la inteligencia artificial es el grupo de noticias comp.ai de Internet y Usenet. Allí está disponible gran cantidad de información y software. Otros grupos de noticias están interesados en lógica confusa, redes neuronales, estructuras de sistema experto y muchos otros temas. Además de la información y el software, estos grupos de noticias permiten a los usuarios enviar preguntas y recibir respuestas de otros.

## PROBLEMAS

- 1.1 Identifique a una persona, aparte de usted, que esté considerada como un especialista o como alguien con mucho conocimiento. Entrevístela y analice qué tan bien podría modelar un sistema experto la experiencia de esta persona, a partir de cada criterio de la sección “Ventajas de los sistemas expertos”, de las páginas 4 y 5.
  
- 1.2
  - a) Escriba diez reglas no triviales que expresen el conocimiento del especialista del problema 1.
  - b) Escriba un programa que le dará consejo experto. Incluya prueba de resultados para demostrar que cada una de las diez reglas ofrece el consejo correcto. Para facilitar la programación usted puede permitir que el usuario proporcione las entradas en un menú.
  
- 1.3
  - a) En la obra de Newell y Simon, *Human Problem Solving*, mencionan el problema de los nueve puntos. Si se dan nueve puntos organizados como se muestra, ¿Cómo dibujaría usted cuatro líneas que unan todos los puntos sin a) separar su lápiz del papel y b) sin pasar dos veces por algún punto. (Sugerencia: puede extender la línea más allá de los puntos.)  
 ...
 ...
 ...
  - b) Explique su razonamiento (si lo hay) para encontrar la solución y analice si un sistema experto o algún otro tipo de programa podría ser un buen paradigma para resolver este problema.

- 1.4 Escriba un programa que pueda resolver problemas criptoaritméticos. Muestre el resultado para el siguiente problema, en el que  $D = 5$ .
- DONALD  
+ GERALD  
ROBERT
- 1.5 Escriba un conjunto de reglas de producción para extinguir cinco tipos diferentes de fuego, como el provocado por petróleo, productos químicos, y otros, dando el tipo de fuego.
- 1.6 a) Escriba un conjunto de reglas de producción para diagnosticar tres tipos de veneno, basándose en los síntomas.  
b) Modifique el programa para que éste recomiende un tratamiento, una vez que se ha identificado el veneno.
- 1.7 Escriba un informe sobre tres diferentes aplicaciones de los sistemas expertos que no se hayan analizado en este capítulo. Incluya sus fuentes de información.
- 1.8 Elabore diez reglas heurísticas del tipo SI...ENTONCES que le ahorren tiempo.
- 1.9 Elabore diez reglas heurísticas del tipo SI...ENTONCES para comprar un automóvil usado.
- 1.10 Elabore diez reglas heurísticas del tipo SI...ENTONCES para planear su horario de clases.
- 1.11 Elabore diez reglas heurísticas del tipo SI...ENTONCES para comprar una motocicleta.
- 1.12 Elabore diez reglas heurísticas del tipo SI...ENTONCES para comprar un vehículo de tres ruedas.
- 1.13 Elabore diez reglas heurísticas del tipo SI...ENTONCES para dar excusas por un trabajo no entregado a tiempo.
- 1.14 Escriba un informe en XCON, XSEL y todos los otros sistemas expertos para ventas y manufactura usados por Digital Equipment Corp. En particular, analice cómo es que los sistemas expertos cooperan para trabajar juntos. Use los dos artículos más recientes como referencia e incluya una copia de los artículos con su informe.
- 1.15 Responda las siguientes preguntas acerca de uno de los sistemas expertos clásicos a los que se hizo referencia en la bibliografía histórica, buscando por lo menos tres textos apropiados u otras referencias. Incluya una copia de todas sus referencias con sus respuestas.
- ¿Cuáles fueron los propósitos de su desarrollo?
  - ¿Qué tanto éxito tuvo para propósitos de investigación?
  - ¿Tuvo alguna vez éxito comercial o se usó rutinariamente?
  - ¿Qué otros sistemas expertos, si los hubo, se debieron a éste?
  - ¿Cuáles nuevos conceptos desarrolló este sistema?

## BIBLIOGRAFÍA

(Buchanan 78). Buchanan, B.G. y Mitchell, T., "Model-directed Learning of Production Rules," Waterman, D. A., and Hayes-Roth, F., eds., *Pattern Directed Inference Systems*, Academic Press, pp. 297-312, 1978.

(Buchanan 78). Bruce G. Buchanan y Edward A. Feigenbaum, "Dendral and Meta-Dendral: Their Applications Dimension," *Artificial Ingelligence*, 11, (1), pp. 5-24, 1978.

# CAPÍTULO 2

## *La representación del conocimiento*

### 2.1 INTRODUCCIÓN

En este capítulo se analizan algunas de las representaciones más comunes del conocimiento para los sistemas expertos. La representación del conocimiento es de gran importancia en estos sistemas por dos razones. En primer lugar, los shells de los sistemas expertos están diseñados para cierto tipo de representación del conocimiento, como las reglas o la lógica. En segundo lugar, la forma en que un sistema experto representa al conocimiento afecta su desarrollo, eficiencia, velocidad y mantenimiento. En el capítulo 3 analizaremos cómo es que se hacen las inferencias.

### 2.2 EL SIGNIFICADO DEL CONOCIMIENTO

*Conocimiento*, como *amor*, es una de esas palabras de las que todos sabemos el significado, aunque resulta difícil definirlo: ambas tienen también muchos significados. Otras palabras como *datos*, *hechos* e *información* se utilizan para sustituir *conocimiento*.

La **epistemología** es el estudio del conocimiento (Angeles 81); se ocupa de su naturaleza, su estructura y sus orígenes. En la figura 2.1 se ilustran algunas de las categorías de la epistemología. Además de los tipos de conocimiento filosófico que expresaron Aristóteles, Platón, Descartes, Hume, Kant y otros, hay dos tipos especiales llamados **a priori** y **a posteriori**. El primer término viene del latín y significa “lo que precede”. Un conocimiento *a priori* precede y es independiente del conocimiento obtenido por medio de los sentidos; por ejemplo, las frases “todo tiene una causa” y “la suma de los ángulos de todos los triángulos en un plano es de 180 grados”. El conocimiento *a priori* se considera verdad universal y no puede negarse sin contradicción. Las frases lógicas, las leyes matemáticas y el conocimiento de los adolescentes son otros ejemplos de conocimiento *a priori*.

Lo opuesto al conocimiento *a priori* es el que se deriva de los sentidos, o conocimiento *a posteriori*. La verdad o falsedad de este conocimiento puede verificarse mediante la experiencia de los sentidos, como en la frase “la luz es verde”. Sin embargo, como la experiencia sensorial no siempre es confiable, un conocimiento *a posteriori*, puede rechazarse con base en nuevo conocimiento, sin necesidad de caer en contradicciones. Por ejemplo, si vemos a alguien de ojos café, creeríamos que de ese color son sus ojos. Sin embargo, si más tarde

observamos que esa persona se quita los lentes de contacto de color café y tiene los ojos azules, tendríamos que revisar nuestro conocimiento.

El conocimiento puede clasificarse aún más como **conocimiento por procedimientos**, **conocimiento declarativo** y **conocimiento tácito**. Los primeros dos tipos corresponden a los paradigmas de procedimientos y declarativo abordados en el capítulo 1.

A menudo el conocimiento por procedimientos se refiere a la forma en que sabemos hacer algo, un buen ejemplo sería saber cómo hervir agua en un pote. El conocimiento declarativo alude a la capacidad de saber que algo es verdadero o falso, que puede expresarse en forma de frases declarativas como “no toques con los dedos un pote con agua hirviendo”.

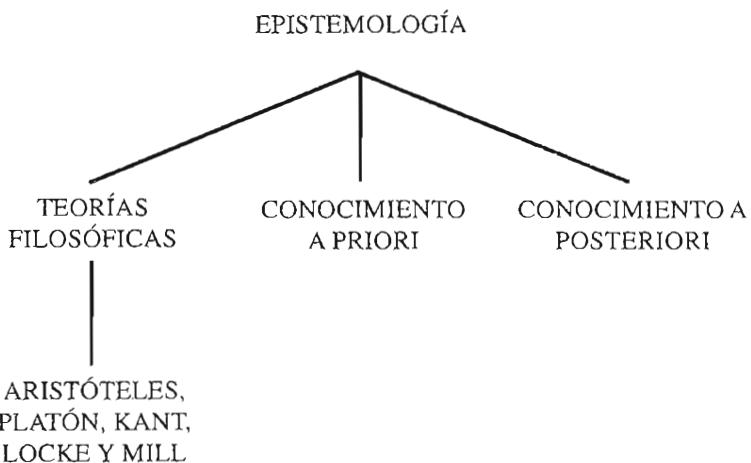


Figura 2.1 Algunas categorías de la epistemología

Al conocimiento tácito en ocasiones se le llama **conocimiento inconsciente**, porque no puede expresarse mediante el lenguaje. Un ejemplo es saber cómo mover la mano, en gran medida, se podría decir que el movimiento proviene de tensar o relajar ciertos músculos y tendones. Pero en el siguiente nivel inferior, ¿cómo saber tensar o relajar los músculos y tendones? Otros ejemplos son caminar o andar en bicicleta. Los sistemas de computación ANS se relacionan con este tipo de conocimiento porque normalmente la red neuronal no puede explicar su conocimiento de manera directa, pero podría hacerlo si se tiene un programa apropiado (consulte la sección 1.14).

El conocimiento es de vital importancia en los sistemas expertos. En efecto, una analogía con la expresión clásica de Wirth

$$\text{Algoritmos} + \text{Estructura de datos} = \text{Programas}$$

para sistemas expertos es

$$\text{Conocimiento} + \text{Inferencia} = \text{Sistemas expertos}$$

En este libro se utiliza el conocimiento como parte de una jerarquía, lo que se ilustra en la figura 2.2. En la base se encuentra el ruido, formado por elementos que son de poco interés y que ocultan datos. El siguiente nivel de la jerarquía son los datos, elementos de interés poten-

cial. En el tercer nivel hay información, o datos procesados que resultan de interés. El siguiente nivel es el conocimiento, que representa información muy especializada. En el capítulo 1, el conocimiento de los sistemas expertos basados en reglas se definió como las reglas activadas por hechos para producir nuevos hechos o conclusiones. Este proceso de *inferencia* es la segunda parte esencial de un sistema experto. El término inferencia se usa generalmente para sistemas mecánicos como los sistemas expertos. Para el pensamiento humano, generalmente se utiliza *razonamiento*.



**Figura 2.2 La jerarquía del conocimiento**

El término *hechos* puede significar datos o información. Dependiendo de la forma en que estén escritos, los sistemas expertos pueden formular inferencias utilizando estos hechos. También pueden (1) separar los datos del ruido, (2) transformar datos en información o (3) transformar la información en conocimiento.

Como ejemplo de estos conceptos, observe la siguiente secuencia de 24 números:

137178766832525156430015

Sin conocimiento, toda esta secuencia aparecería como ruido. Sin embargo, si se sabe que esta secuencia tiene un significado, entonces se convierte en datos. Determinar lo que son datos y lo que es ruido es parecido al viejo dicho relacionado con la jardinería, "hierba es cualquier cosa que crece y que no es lo que usted quiere".

Quizá exista cierto conocimiento que permita convertir estos datos en información. Por ejemplo, el siguiente algoritmo procesa los datos para producir información.

Agrupe los números en pares.

Ignore cualquier número de dos dígitos menor a 32.

Sustituya los caracteres ASCII por los números de dos dígitos.

La aplicación de este algoritmo a los anteriores 24 dígitos produce la siguiente información

ORO 438+

Ahora el conocimiento puede aplicarse a esta información. Por ejemplo, tal vez haya una regla

```

SI oro es menor a 500
y el precio está subiendo (+)
ENTONCES
compra oro

```

Aunque no se muestra explícitamente en la figura 2.2, *la experiencia* es un tipo de conocimiento que poseen los especialistas. Ésta no se encuentra comúnmente en fuentes públicas de información, como libros y artículos; es el conocimiento implícito de los especialistas que debe extraerse y hacerse explícito para que pueda codificarse en un sistema experto. Por encima del conocimiento está el **metaconocimiento**, el significado del prefijo **meta** es “más allá de”. El metaconocimiento es conocimiento acerca del conocimiento y la experiencia. Como es posible diseñar un sistema experto con conocimiento de varios dominios diferentes, el metaconocimiento especificaría cuál base de conocimiento es la aplicable. Por ejemplo, un sistema experto puede tener bases de conocimiento acerca de reparación de autos Chevrolet 1988, Ford 1985 y Cadillac 1989; se emplearía la base apropiada dependiendo del automóvil que se necesite reparar. Esto no es eficiente para la memoria y rapidez, porque todas las bases del conocimiento estarían trabajando al mismo tiempo. Además, pueden surgir conflictos cuando el sistema experto trate de decidir las reglas aplicables de todas las bases del conocimiento a la vez. El metaconocimiento podría utilizarse también dentro de un dominio para decidir cuál grupo de reglas es el más aplicable.

En un sentido filosófico, la **sabiduría** es la cumbre del conocimiento. La sabiduría es el metaconocimiento de determinar los mejores objetivos en la vida y cómo lograrlos. Una regla de sabiduría sería

```

SI tengo suficiente dinero para mantener feliz a mi esposa
ENTONCES me retiraré y disfrutaré de la vida

```

Sin embargo, debido a la extrema escasez de sabiduría en el mundo, debemos limitarnos a sistemas basados en el conocimiento y dejarle los basados en la sabiduría a los políticos.

## 2.3 PRODUCCIONES

Se han concebido varias técnicas diferentes de representación del conocimiento. Entre éstas se incluyen reglas, redes semánticas, marcos, guiones, lenguajes de representación del conocimiento como KL-1 (Woods 83) y KRYPTON (Brachman 83), gráficas conceptuales (Sowa 84) y otros (Brachman 80). Como se describió en el capítulo 1, las reglas de producción suelen utilizarse como base de conocimiento para los sistemas expertos, porque sus ventajas sobrepasan en mucho a sus desventajas.

Una notación formal para definir producciones es la forma Backus-Naur (BNF; McGetrick 80). Esta notación es un **metalingüaje** para definir la **sintaxis** de un lenguaje. La sintaxis define la forma, mientras que la **semántica** se refiere al significado. Un metalingüaje es un lenguaje para describir otros. El prefijo *meta* significa “más allá de”, por lo tanto un metalingüaje va “más allá de” un lenguaje normal.

Hay muchos tipos de lenguajes: naturales, lógicos, matemáticos y de cómputo. La notación de BNF para la simple regla de una oración formada por un sujeto y un verbo seguidos por un signo de puntuación, es la siguiente regla de producción:

```
<oración> ::= <sujeto> <verbo> <signo final>
```

Los **paréntesis en ángulo**, `<>` y `::=` son símbolos del metalenguaje y no del lenguaje que se especifica. El símbolo “`::=`” significa “se define como” y es el equivalente en BNF de la flecha,  $\rightarrow$ , utilizada en el capítulo 1 con las reglas de producción. Para evitar confusión con el operador de asignación de Pascal, `:=`, utilizaremos la flecha.

A los términos dentro de los paréntesis en ángulo se les llama **símbolos no terminales** o simplemente no terminales. Un símbolo no terminal es una variable en que se representa otro término. Éste puede ser no terminal o **terminal**. Un terminal no se puede reemplazar con algo más y, por tanto, es una constante.

La `<frase>` no terminal resulta especial porque se trata del único **símbolo de inicio** a partir del cual se definen los otros. En la definición de lenguajes de programación, al símbolo de inicio a menudo se le llama `<programa>`. La siguiente regla de producción

```
<oración> → <sujeto> <verbo> <signo final>
```

establece que una oración se integra con un sujeto seguido por un verbo y luego un signo final. Las siguientes reglas completan las no terminales especificando sus terminales posibles. La **barra** significa “o” en el metalenguaje.

```
<sujeto> → Yo | Tú | Nosotros
<verbo> → fui | vine | fuiste | vinistes | fuimos |
           vinimos
<signo final> → . | ? | !
```

Todas las oraciones posibles en el lenguaje, las producciones, pueden producirse reemplazando sucesivamente cada uno de los no terminales con sus no terminales o terminales del lado derecho, hasta eliminar todos los no terminales. Las siguientes son algunas producciones:

```
Yo fui.
¿Yo fui?
¡Yo fui!
Tú fuiste.
¿Tú fuiste?
¡Tú fuiste!
Nosotros fuimos.
¿Nosotros fuimos?
¡Nosotros fuimos!
```

A un conjunto de terminales se le llama **cadena** del lenguaje. Si la cadena puede derivarse del símbolo de inicio, mediante el reemplazo de no terminales, con sus reglas de definición, entonces se le llama **oración válida**. Por ejemplo, “Nosotros”, “NosotrosNosotros”, y “fuifustefuimos” son cadenas válidas del lenguaje, pero no son oraciones válidas.

Una **gramática** es un conjunto completo de reglas de producción que definen un lenguaje sin ambigüedades. Aunque las reglas anteriores definen una gramática, es un conjunto muy limitado porque hay pocas producciones posibles. Por ejemplo, una gramática más elaborada también podría incluir objetos directos, como en las producciones

```
<oración> → <sujeto> <verbo> <objeto> <signo final>
<objeto> → casa | trabajo | escuela
```

Aunque ésta es una gramática válida, resulta demasiado simple para el uso práctico. Una gramática más práctica es la siguiente, en la que se ha omitido el signo final para simplificar:

```

<oración> → <frase de sujeto> <verbo> <frase de objeto>
<frase de sujeto> → <determinador> <sustantivo>
<frase de objeto> → <determinador> <adjetivo> sustantivo>
<determinador> → un | uno | unos | unas | el | la | los |
| las | este | esta | estos | estas | esos | esas
<sustantivo> → hombre | devorador
<verbo> → es | está | fue, estuvo
<adjetivo> → postres | gran
  
```

El <determinador> se utiliza para indicar un elemento específico. Utilizando esta gramática, pueden generarse oraciones como:

el hombre fue un devorador de postres  
un devorador fue el gran hombre

Un **árbol de análisis gramatical** o **árbol de derivación** es una representación gráfica de una oración, descompuesta en todos los terminales y no terminales utilizados para derivarla. En la figura 2.3 se muestra el árbol de análisis gramatical para la frase “el hombre fue un gran devorador”. Sin embargo la cadena “hombre fue un gran devorador” no es una oración válida porque carece del determinador en la frase de sujeto. Un compilador crea un árbol de análisis gramatical cuando trata de determinar si las oraciones de un programa se adecuan a la sintaxis válida de un lenguaje.

El árbol de la figura 2.3 muestra que la oración “el hombre fue un gran devorador” puede

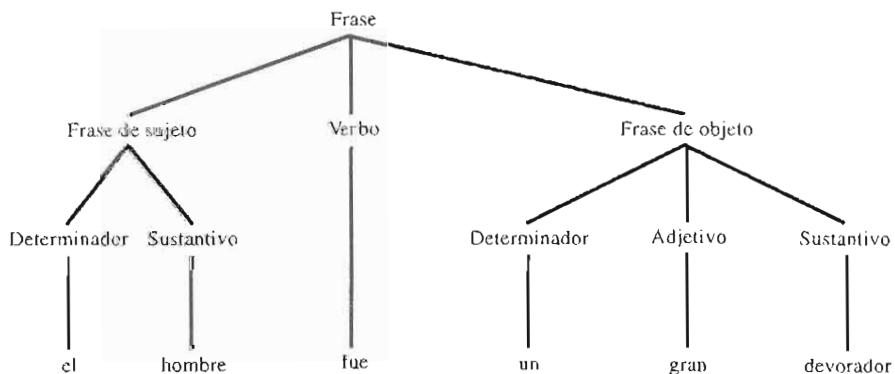


Figura 2.3 Árbol de análisis gramatical de una oración

derivarse del símbolo de inicio mediante la aplicación de las producciones apropiadas. A continuación se explican los pasos de este proceso; la **flecha doble**, =>, significa aplicar las producciones mostradas.

```

<frase> => <frase de sujeto> <verbo> <frase de objeto>
<frase de sujeto> => <determinador> <sustantivo>
<determinador> => el
<sustantivo> => hombre
<verbo> => fue
<frase de objeto> => <determinador> <adjetivo> sustantivo>
<determinador> => un
<adjetivo> => gran
<sustantivo> => devorador

```

Una forma alterna de utilizar las producciones consiste en generar frases válidas sustituyendo todas las terminales apropiadas por no terminales. Por supuesto, no todas las producciones, como “el hombre fue el postre”, tendrán sentido.

## 2.4 REDES SEMÁNTICAS

Una **red semántica**, o red, es una técnica clásica de representación de la AI utilizada para información relativa a las proposiciones (Stillings 87). A una red semántica también se le llama **red de proposiciones**. Como se analizó antes, una proposición es una frase verdadera o falsa, como “todos los perros son mamíferos” y “un triángulo tiene tres lados”. Son formas de conocimiento declarativo porque establecen hechos. En términos matemáticos, una red semántica es una gráfica rotulada y con dirección. Una proposición siempre es verdadera o falsa y se le llama **atómica** porque su veracidad ya no puede dividirse más, el término *atómica* se emplea en el sentido clásico de objeto indivisible. En contraste, las proposiciones que trataremos en el capítulo 5 normalmente sólo necesitan ser verdaderas o falsas.

Las redes semánticas se desarrollaron para la AI como una forma de representar la memoria y la comprensión del lenguaje del ser humano (Quillian 68). Quillian las utilizó para analizar el significado de palabras en frases. Desde entonces se han aplicado a muchos problemas relacionados con la representación del conocimiento.

La estructura de una red semántica se muestra gráficamente en términos de **nodos** y los **arcos** que los conectan. A los nodos suele denominárseles como objetos y a los arcos como **vínculos** o **bordes**.

Los vínculos de una red semántica se utilizan para expresar relaciones. Por lo general, los nodos se utilizan para representar objetos físicos, conceptos o situaciones. En la figura 2.4 (a) se muestra una red ordinaria, en realidad una gráfica con dirección, donde los vínculos indican las rutas de una línea aérea entre ciudades. Los nodos son los círculos, los vínculos, las líneas que los conectan y las flechas muestran la dirección en que pueden volar los aviones, de aquí el término gráfica con dirección. En la figura 2.4 (b) los vínculos muestran las relaciones entre los miembros de una familia. Éstas son de gran importancia en las redes semánticas porque proporcionan la estructura básica para la organización del conocimiento. Sin relaciones, el conocimiento es una simple colección de hechos inconexos. Con ellas, el conocimiento es una estructura cohesiva a partir de la cual puede inferirse conocimiento adicional. Por ejemplo, en la figura 2.4 (b) se puede inferir que Ann y Bill son los abuelos de John, aunque no haya un vínculo explícito rotulado “abuelo de”.

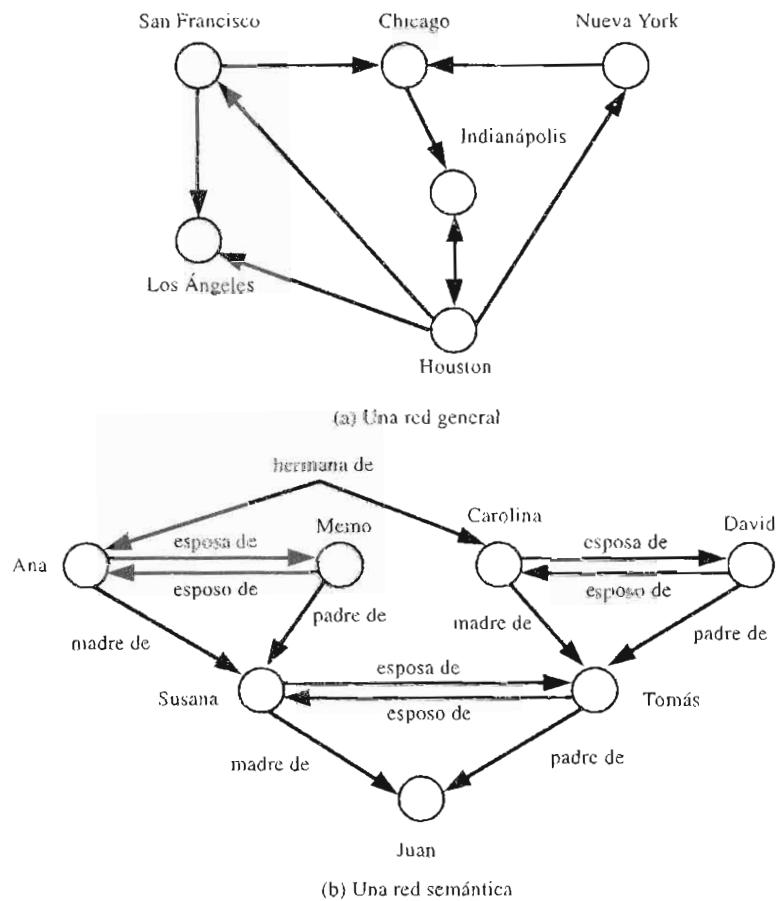


Figura 2.4 Dos tipos de red

A las redes semánticas a veces se les denomina **redes asociativas** porque los nodos están asociados o relacionados con otros nodos. De hecho, el trabajo original de Quillian modeló la memoria humana como una red asociativa en la cual los conceptos eran los nodos y los vínculos formaban las conexiones entre ellos. De acuerdo con este modelo, cuando un nodo de concepto se estimula al leer las palabras de una frase, sus vínculos con otros nodos se activan en un patrón de dispersión. Si otro nodo recibe activación suficiente, el concepto será forzado dentro de la mente consciente. Por ejemplo, aunque se conozcan miles de palabras, sólo se está pensando en las palabras específicas de la frase que se lee.

Ciertos tipos de relaciones han probado ser muy útiles en una amplia variedad de representaciones del conocimiento. En lugar de definir nuevas relaciones para problemas diferentes, es habitual usar estos tipos estándar. El uso de tipos comunes facilita la comprensión de una red no muy conocida.

Dos tipos de vínculos utilizados con frecuencia son **ES-UN** y **UN TIPO DE** (Winston 84). La figura 2.5 es un ejemplo de una red semántica que utiliza estos vínculos; en esta figura, **ES-UN** significa “es un caso de” y se refiere a un miembro específico de una clase. Una clase está relacionada con el concepto matemático de conjunto, que alude a un grupo de objetos.

Aunque un conjunto puede tener elementos de cualquier tipo, los objetos de una clase tienen alguna relación entre sí. Por ejemplo, es posible definir un conjunto formado por

{ 3, huevos, azul, llantas, arte }

Sin embargo, los miembros de este conjunto no tienen relación entre sí. Por el contrario, aviones, trenes y automóviles se relacionan en una clase porque todos son tipos de transporte.

El vínculo UN-TIPO-DE se utiliza aquí para relacionar una clase con otra. No se usa para relacionar un individuo específico porque esa es la función de ES-UN. UN-TIPO-DE relaciona una clase individual con una clase padre a la que pertenece una clase hijo.

Desde otro punto de vista, el UN-TIPO-DE relaciona nodos **genéricos** con nodos genéricos, mientras que ES-UN relaciona un caso o un **individuo** con una clase genérica. Observe en la figura 2.5 que las clases más generales se encuentran en la parte superior y las más específicas en la base. La clase más general, a la que señala una flecha UN-TIPO-DE, se le denomina **superclase**; si ésta tiene un vínculo UN-TIPO-DE apuntando a otro nodo, entonces también es una clase de la superclase a la que señala el vínculo UN-TIPO-DE. Otra forma de expresarlo es que un vínculo UN-TIPO-DE señale de una **subclase** a una clase. El vínculo SON se usa algunas veces para UN-TIPO-DE y se lee como el verbo "son".

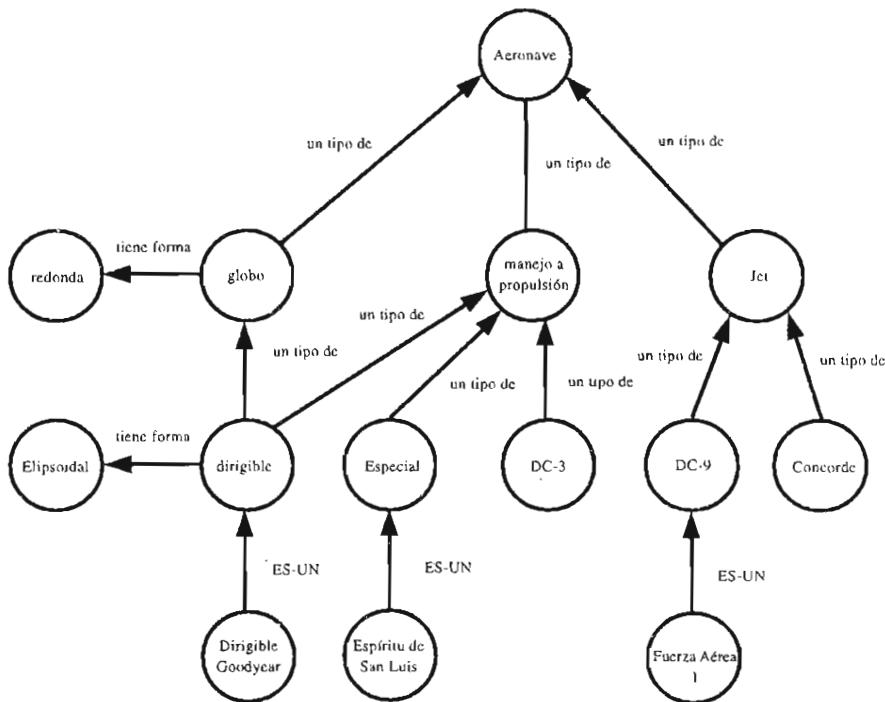


Figura 2.5 Red semántica con vínculos ES-UN y UN-TIPO-DE

Los objetos de una clase tienen uno o más **atributos** en común y cada atributo tiene un **valor**. La combinación de atributo y valor es una **propiedad**. Por ejemplo, un dirigible tiene

atributos de tamaño, peso, forma, color, etcétera. El valor del atributo de forma es elipsoidal. En otras palabras, un dirigible tiene la forma elipsoidal como una de sus propiedades. Como el Goodyear es un dirigible y los dirigibles tienen forma elipsoidal, entonces el Goodyear es elipsoidal. También se pueden encontrar otros tipos de vínculos en las redes semánticas. El vínculo **ES** define un valor. Por ejemplo, cualquier aeronave presidencial de Estados Unidos es el Fuerza Aérea 1. Si el presidente está en un DC-9, entonces el Fuerza Aérea 1 es un DC-9. El vínculo **CAUSA** expresa conocimiento causal. Por ejemplo, el aire caliente CAUSA que un globo se eleve.

A la duplicación de las características de un nodo por parte de un descendiente se le llama **herencia**. A menos que haya más evidencias específicas de lo contrario, se supone que todos los miembros de una clase heredarán todas las propiedades de sus superclases. Por ejemplo, los globos tienen una forma redonda. Sin embargo, como la clase dirigible tiene un vínculo que señala a la forma elipsoidal, ésta tiene precedencia. La herencia es una herramienta muy útil en la representación del conocimiento porque elimina la necesidad de repetir características comunes. Los vínculos y la herencia proporcionan medios muy eficientes de representación del conocimiento porque muchas relaciones complejas pueden mostrarse con pocos nodos y vínculos.

## 2.5 TRIPLETA OBJETO-ATRIBUTO-VALOR

Un problema con el uso de las redes semánticas es que no hay definiciones estándar para nombrar los vínculos. Por ejemplo, algunos libros utilizan **ES-UN** para relaciones genéricas e individuales (Barr 81; Staugaard 87). **ES-UN** se utiliza entonces en el sentido ordinario de las palabras "es un" y también en el sentido de **UN-TIPO-DE**. En la herramienta para sistemas expertos ART, **IS-A** (**ES-UN**) relaciona dos clases de objetos e **INSTANCE-OF** (**CASO-DE**) relaciona a un individuo con una clase.

Otro vínculo muy utilizado es **TIENE-UN**, que relaciona una clase con una subclase. **TIENE-UN** señala a la inversa de **UN-TIPO-DE** y suele utilizarse para relacionar un objeto con una parte del objeto. Por ejemplo,

```
el automóvil TIENE-UN motor
el automóvil TIENE-UNAS llantas
el automóvil ES-UN ford
```

Más específicamente, **ES-UN** relaciona un valor con un atributo mientras que **TIENE-UN** relaciona un objeto con un atributo.

Los elementos objeto, atributo y valor ocurren tan frecuentemente que es posible construir una red semántica simplificada utilizando únicamente estos elementos. Una **tripleta objeto-atributo-valor (OAV)**, puede utilizarse para caracterizar todo el conocimiento de una red semántica y se usó en el sistema experto **MYCIN** para diagnosticar enfermedades infecciosas. La representación del trío OAV es conveniente para ordenar el conocimiento en forma de tabla y luego traducirla al código de la computadora mediante inducción de reglas. En la tabla 2.1 se muestra el ejemplo de una tabla con el trío OAV.

Los tríos OAV son especialmente útiles para representar los hechos y sus patrones para llevarlos al antecedente de una regla. La red semántica para este tipo de sistema consta de nodos para objetos, atributos y valores, conectados por vínculos **TIENE-UN** y **ES-UN**. Si sólo habrá de representarse un objeto y no se requiere de herencia, tal vez bastaría una simple representación llamada **pares atributo-valor**, o simplemente **AV**.

Objeto	Atributo	Valor
manzana	color	roja
manzana	tipo	mcintosh
manzana	cantidad	100
uvas	color	rojas
uvas	tipo	sin semilla
uvas	cantidad	500

Tabla 2.1 Tabla OAV

## 2.6 PROLOG Y REDES SEMÁNTICAS

Las redes semánticas son fáciles de traducir a PROLOG. Por ejemplo,

```
es un (dirigible_godyear, dirigible).
es un (espíritu_de_san_luis, especial).
tiene forma (dirigible, elipsoidal).
tiene forma (globo, redondo).
```

son instrucciones de PROLOG que expresan algunas de las relaciones de la red semántica de la figura 2.5. Un punto marca el final de una frase.

### Elementos esenciales de PROLOG

Cada una de las frases anteriores es una **expresión de un predicado** de PROLOG, o simplemente un predicado, porque está basado en lógica de predicados. Sin embargo, PROLOG no es un verdadero lenguaje de lógica de predicados porque es un lenguaje computacional con instrucciones ejecutables. En PROLOG, un predicado está formado por el nombre de predicado, como es un, seguido de cero o más argumentos encerrados entre paréntesis y separados por comas. A continuación se presentan algunos ejemplos de predicados y comentarios elaborados en PROLOG precedidos por punto y coma:

```
color(rojo). ; rojo es un color
padre_de(tom, john). ; tom es el padre de john
madre_de(susan, john). ; susan es la madre de john
padres(tom, susan, john). ; tom y susan son
; padres de john
```

Los predicados con dos argumentos son más fáciles de entender si coloca el nombre de predicado siguiendo al primer argumento. Los significados de estos predicados con más de dos argumentos deben establecerse explícitamente, como lo ilustra el predicado padres. Otra dificultad es que las redes semánticas se utilizan primordialmente para representar relaciones binarias, porque una línea sólo tiene dos extremos, por supuesto. No es posible poner la afirmación padres como un borde con una sola dirección porque hay tres argumentos. La idea de colocar a Tom y Susan juntos en un nodo de padres y a John en las otras direcciones lleva a una nueva complicación. Entonces es imposible utilizar el nodo de los padres para otra relación binaria, como madre-de, porque también se incluiría a Tom.

Los predicados pueden expresarse también con relaciones IS-A (ES-UN) y HAS-A (TIENE-UN).

```
is_a(rojo,color).
has_a(john,padre).
has_a(john,madre).
has_a(john,padres).
```

Observe que el predicado `has_a` no expresa el mismo significado que antes, porque el padre, la madre y los padres de John no son nombrados explícitamente. Para ello, se deben agregar algunos predicados adicionales.

```
is_a(tom,padre).
is_a(susan,madre).
is_a(tom,padre).
is_a(susan,padre).
```

Incluso estos predicados adicionales no expresan el mismo sentido de los originales. Por ejemplo, sabemos que John tiene un parente y que Tom es un parente, pero esto no informa que Tom sea el parente de John.

Todas las frases anteriores describen realmente hechos de PROLOG. Los programas de PROLOG están formados por hechos y reglas en la forma general de **metas**:

$p :- p_1, p_2, \dots, p_n.$

en los que  $p$  es la cabeza de la regla y  $p_i$  son las **submetas**. Normalmente esta expresión es una cláusula de Horn, que establece que la meta principal,  $p$ , se satisface si y sólo si son satisfechas todas las submetas. La excepción se presenta cuando se utiliza el predicado especial para fallas, que resulta sumamente conveniente porque no es fácil probar una negación en lógica clásica y PROLOG está basado en la lógica clásica. Una negación es vista como la falla al tratar de encontrar una prueba, esto puede ser un proceso muy largo y complicado si hay muchas coincidencias probables. El predicado de **corte** y de falla especial hace que el proceso de negación sea más eficiente reduciendo la búsqueda de una prueba.

Las comas que separan las submetas representan una disyunción Y lógica. El símbolo,  $:-$ , se interpreta como un SI. Si sólo existe la cabeza y no hay lado derecho, como en

$p.$

entonces la cabeza se considera verdadera. Esto se debe a que predicados como los siguientes se consideran hechos y, por tanto, deben ser verdaderos.

```
color(rojo).
has_a(john,padre).
```

Otra forma de pensar en los hechos es considerándolos una conclusión incondicional que no depende de nada más y, por lo tanto, el SI,  $:-$ , no es necesario. Por el contrario, las reglas de PROLOG necesitan el SI porque son conclusiones condicionales que en verdad dependen de una o más condiciones. Como ejemplo, las reglas de los padres quedan así:

```
padre(X,Y) :- papá (X,Y).
padre(X,Y) :- mamá (X,Y).
```

lo que significa que X es un padre de Y, si X es el papá de Y o si X es la mamá de Y. Del mismo modo, un abuelo puede definirse como:

```
abuelo(X, Y) :- padre(X, Z), padre(Z, Y).
```

y un antepasado puede definirse como:

- (1) ancestro(X, Y) :- padre(X, Y).
- (2) ancestro(X, Y) :- ancestro(X, Z), ancestro(Z, Y).

en donde (1) y (2) sólo se utilizan con propósitos de identificación.

## Búsqueda en PROLOG

Un sistema para ejecución de instrucciones de PROLOG es generalmente un intérprete, aunque algunos sistemas pueden generar código compilado. En la figura 2.6 se muestra la forma general de un sistema PROLOG. El usuario interactúa con PROLOG mediante la introducción de consultas y la recepción de respuestas. La **base de datos de los predicados** contiene a los predicados de regla y de hecho que se han introducido, y con ellos forma la base del conocimiento. El intérprete trata de determinar si un predicado de consulta que el usuario introdujo se encuentra en la base de datos. La respuesta devuelta es afirmativa en caso de que se encuentre en la base de datos; de lo contrario, es negativa. Si la consulta es una regla, el intérprete tratará de satisfacer los subobjetivos conduciendo primero una búsqueda a fondo, como se muestra en la figura 2.7. Por el contrario, también se muestra una búsqueda primero a lo ancho, aunque éste no es el modo normal de PROLOG. En una **búsqueda primero a fondo**, la búsqueda desciende lo más posible y luego regresa. En PROLOG también va de izquierda a derecha. Una **búsqueda primero a lo ancho** pasa de nivel en nivel antes de descender al siguiente nivel inferior.

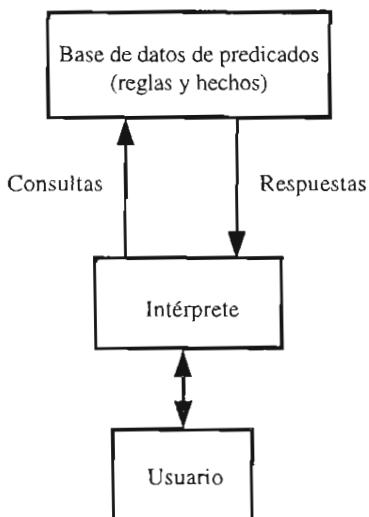


Figura 2.6 Organización general de un sistema PROLOG

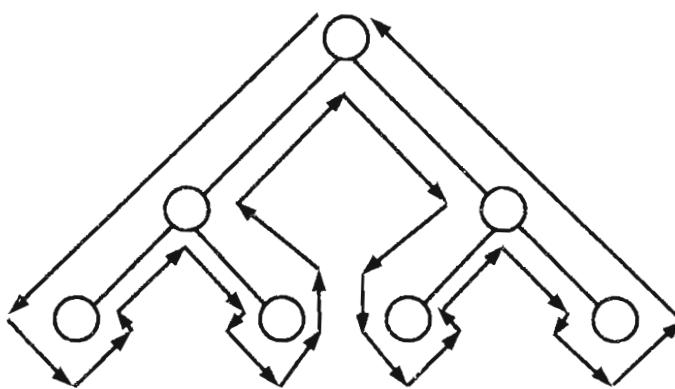
Consideremos las reglas de los antepasados, definidas como (1) y (2), como ejemplo de búsqueda de objetivos en PROLOG. Tomemos los siguientes hechos como entrada.

- (3) `padre(ann,mary)` .
- (4) `padre(ann,susan)` .
- (5) `padre(mary,bob)` .
- (6) `padre(susan,john)` .

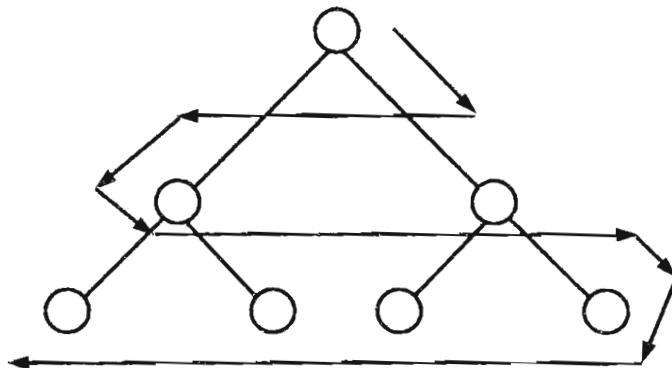
Ahora supongamos que se le consulta a PROLOG si Ann es antepasado de Susan

```
:-ancestro(ann,susan).
```

donde la ausencia de una cabeza indica que se trata de una **consulta**, que es una condición que habrá de probar PROLOG. Hechos, reglas y consultas son los tres tipos de cláusula de Horn aplicables a PROLOG. Se puede probar una condición, si es la conclusión del caso de una cláusula. Por supuesto, la propia cláusula debe poderse probar por sí misma, lo que se hace mediante la demostración de las condiciones de la cláusula.



(a) Búsqueda primero a fondo



(b) Búsqueda primero a lo ancho

Figura 2.7 Búsquedas primero a fondo y primero a lo ancho para un árbol arbitrario

Al recibir esta entrada, PROLOG empezaría a buscar una instrucción cuya cabeza coincida con el patrón de entrada `ancestro(ann,susan)`. A esto se le llama **coincidencia de patrones**, de manera parecida a la coincidencia de patrones de hechos en el antecedente de una regla de producción. La búsqueda empieza desde la primera instrucción que se introdujo (1), la **superior**, y sigue hasta la última instrucción (6), la **inferior**. Hay una posible coincidencia con la primera regla de ancestro (1). La variable X coincide con ann y la variable Y con susan. Como la cabeza coincide, ahora PROLOG tratará de equiparar el cuerpo de (1), produciendo la submeta `padre(ann,susan)`. PROLOG trata de que esta submeta coincida con las cláusulas y en algún momento coincidirá con el hecho (4) `padre(ann,susan)`. No hay más objetivos que coincidan y PROLOG responde "sí" porque la consulta original es verdadera.

Otro ejemplo, supongamos que la consulta es

```
: - ancestro(ann, john).
```

La primera regla de ancestros (1) coincide y X toma el valor de ann y Y el de john. PROLOG ahora trata de hacer que coincida el cuerpo de (1), `padre(ann,john)`, con todas las instrucciones de padre. Ninguna coincide y, por tanto, el cuerpo de (1) no puede ser cierto. Como el cuerpo de (1) no es cierto, la cabeza no puede ser cierta.

Como (1) no puede ser verdadero, PROLOG prueba la segunda instrucción de ancestros (2). X toma el valor de ann y Y el de john. PROLOG trata de probar que la cabeza de (2) es verdadera probando que ambas submetas de (2) son verdaderas. Es decir, debe probarse que `ancestro(ann,Z)` y `ancestro(Z,john)` son verdaderos. PROLOG trata de equiparar las submetas de izquierda a derecha a partir de `ancestro(ann,Z)`. Empezando desde la parte superior, coincide primero con la instrucción (1), por lo tanto PROLOG trata de probar el cuerpo de (1), `padre(ann,Z)`. Empezando a buscar nuevamente desde la parte superior, este cuerpo coincide primero con la instrucción (3) y por ello PROLOG asigna a Z el valor de mary. Ahora PROLOG trata de equiparar la segunda submeta de (2) como ancestro (mary,john). Esto coincide con (1) y, por tanto, PROLOG trata de satisfacer su cuerpo, `padre(mary,john)`. Sin embargo, no hay `padre(mary,john)` en la instrucción (3) a la (6) y, por tanto, esta búsqueda falla.

Entonces PROLOG examina de nuevo su suposición de que Z debe ser mary. Como esta opción no funcionó, trata de encontrar otro valor que funcione. Otra posibilidad surge de (4) al asignar a Z el valor de susan. A esta técnica de retroceso para probar una ruta de búsqueda diferente cuando otra falla, se le denomina rastreo hacia atrás o **backtrack** y se utiliza a menudo para resolver problemas.

Con la elección de Z como susan, PROLOG intenta probar que `ancestro(susan,john)` es verdadero. Mediante (1), debe probarse que el cuerpo `padre(susan,john)` es verdadero. Efectivamente, el hecho (3) coincide y entonces se prueba que la consulta

```
: - ancestro(ann, john).
```

es verdadera porque se ha probado que sus condiciones son verdaderas.

Observe que la estructura de control de PROLOG tiene el tipo del algoritmo de Markov, en el que la búsqueda de patrones coincidentes se determina normalmente por el orden en que se introducen las cláusulas de Horn. Esto difiere con los sistemas expertos basados en reglas, en que suelen seguir el paradigma de Post, donde el orden en que se introducen las reglas no afecta la búsqueda.

PROLOG tiene muchas otras funciones y capacidades que no se mencionan en esta breve introducción. Desde el punto de vista de un sistema experto, el backtrack y la equiparación de patrones son muy útiles, como también lo es la naturaleza declarativa de PROLOG, porque la especificación de un programa es un programa ejecutable.

## 2.7 DIFICULTADES CON REDES SEMÁNTICAS

Aunque las redes semánticas pueden ser muy útiles en la representación del conocimiento, tienen limitantes, como la falta de estándares para asignación de nombres a los vínculos, que ya se explicó. Esto hace difícil entender para qué se diseñó realmente la red y si fue diseñada de una forma congruente. Un problema complementario al asignar nombre a los vínculos es el nombre de los nodos. Si un nodo se rotula “silla”, ¿Representa

- una silla específica
- la clase de todas las sillas
- el concepto de una silla
- el nombre del cerro de la Silla

o algo más? Para que una red semántica represente **conocimiento definitivo** (es decir, conocimiento que puede definirse), deben definirse con rigor los nombres de vínculos y nodos. Claro que los mismos problemas pueden ocurrir en los lenguajes de programación.

Otro problema es la explosión combinatoria en la búsqueda de nodos, sobre todo si la respuesta a una consulta es negativa. Para que una consulta produzca un resultado negativo, tendrían que buscarse muchos o todos los vínculos de la red. Como se mostró en el problema del agente viajero del capítulo 1, el número de vínculos es el factorial del número de nodos menos uno, si están todos conectados. Aunque no todas las representaciones necesitarán este grado de conectividad, existe la posibilidad de que surja una explosión combinatoria.

Las redes semánticas fueron propuestas originalmente como modelos de la memoria asociativa humana, en la que un nodo tiene vínculos con otros y la recuperación de la información ocurre debido a una dispersión de la activación de los nodos. No obstante, el cerebro humano también debe disponer de otros mecanismos, porque no le toma mucho tiempo responder la consulta: ¿hay un equipo de fútbol en Plutón? Hay alrededor de  $10^{10}$  neuronas en el cerebro humano y alrededor de  $10^{15}$  vínculos. Si todo el conocimiento estuviera representado por una red semántica, tomaría demasiado tiempo responder consultas negativas como en el caso de la consulta del fútbol, porque todas las búsquedas incluyen  $10^{15}$  vínculos.

Las redes semánticas son lógicamente inadecuadas porque no pueden definir el conocimiento de la forma en que lo hace la lógica. Una representación lógica puede especificar cierta silla, algunas sillas, todas las sillas, ninguna silla, etcétera, como se analizará más adelante en este capítulo. Otro problema es que las redes semánticas también son heurísticamente inadecuadas porque no hay forma de insertar en la red información heurística relacionada con la manera de explorarla de forma eficiente. Una regla **heurística** es una regla práctica que puede ayudar a encontrar una solución, pero que no está garantizada de la misma forma que un algoritmo. Las reglas heurísticas son muy importantes en la AI porque los problemas típicos en ello son tan difíciles que no existe una solución de algoritmo o, si la hay, no resulta eficiente para uso práctico. La herencia es la única estrategia estándar de control integrada en una red que puede ayudar, pero no todos los problemas pueden tener esta estructura.

Se han probado varios métodos para corregir los problemas inherentes a las redes semánticas; se han hecho mejoras lógicas y se han probado mejoras heurísticas al adjuntar procedimientos a los nodos, que serán ejecutados cuando el nodo se active. Sin embargo, los sistemas resultantes mejoraron poco su capacidad a costa de la fuerza de expresión natural de las redes semánticas. La conclusión de toda este esfuerzo es que, como cualquier herramienta, las redes semánticas deben utilizarse para aquello que hacen mejor, mostrar relaciones binarias, y no considerarse una herramienta universal.

## 2.8 ESQUEMAS

Una red semántica es un ejemplo de una **estructura de conocimiento superficial**. La superficialidad se encuentra presente porque todo el conocimiento de la red semántica se encuentra en los vínculos y nodos. El término *estructura del conocimiento* es análogo a una estructura de datos en la que más que sólo datos se representa una colección ordenada de conocimiento. Una estructura de conocimiento **profundo** tiene conocimiento causal que explica por qué ocurre algo. Por ejemplo, es posible construir un sistema experto médico con conocimiento superficial como sigue:

```
SI una persona tiene fiebre
ENTONCES toma una aspirina
```

Pero estos sistemas no conocen la bioquímica fundamental del cuerpo ni la razón por la que la fiebre disminuye con la aspirina. La regla podría haberse definido:

```
SI una persona tiene un mono rosa
ENTONCES toma un refrigerador
```

En otras palabras, el conocimiento del sistema experto es superficial porque está basado en la sintaxis más que en la semántica, donde dos palabras cualquiera podrían ser sustituidas por X y Y en la siguiente regla

```
SI una persona tiene un (X)
ENTONCES toma un (Y)
```

Observe que (X) y (Y) no son variables en esta regla, sino que representan dos palabras cualquiera. Los médicos tienen conocimiento causal porque han tomado muchos cursos y han adquirido experiencia en el tratamiento de personas enfermas. Si un tratamiento no está funcionando correctamente, los médicos pueden razonar acerca de él para encontrar una alternativa. En otras palabras, un especialista sabe cuándo romper las reglas.

Muchos tipos de conocimientos del mundo real no pueden representarse con la simple estructura de una red semántica. Se necesitan estructuras más complejas para representar mejor complejas estructuras del conocimiento. En la AI, el término *esquema* se utiliza para describir una estructura de conocimiento más compleja que la red semántica. Este término viene de la psicología, donde alude a la organización continua del conocimiento o la respuesta de una criatura debida a un estímulo. Esto es, a medida que las criaturas aprenden la relación entre una causa y su efecto, tratarán de repetir la causa si el efecto es agradable, o evitarla, si no lo es.

Por ejemplo, los actos de comer y beber son esquemas **sensomotores** agradables que involucran información de coordinación de los sentidos con los movimientos motores (músculos) necesarios para realizarlos. Una persona no tiene que pensar en este conocimiento tácito para saber cómo realizar estos actos y es muy difícil explicar exactamente cómo desciende al nivel de controlar los músculos. Un esquema aún más difícil de explicar es cómo andar en bicicleta. ¿Cómo tratar de explicar el sentido del equilibrio?

Otro tipo de esquema es el **esquema de concepto**. Por ejemplo, todos tienen un concepto de *animal*. La mayoría de las personas a las que se les pida que expliquen cómo es un animal probablemente lo describirían como algo que tiene cuatro patas y pelo. Por supuesto, el concepto de *animal* será diferente si la persona creció en una granja, en una ciudad, cerca de un río, etcétera. Sin embargo, todos tenemos **estereotipos** de conceptos en nuestras mentes. Aun-

que el término *estereotipo* quizá posea un significado negativo en el lenguaje coloquial, para la AI significa un ejemplo típico. Para mucha gente un estereotipo de un animal podría ser algo como un perro.

Un esquema conceptual es una abstracción en la cual se clasifican objetos específicos, de acuerdo con sus propiedades generales. Por ejemplo, si ve un pequeño objeto rojo, redondo con tallo verde, sobre un letrero que dice "Fruta artificial", probablemente lo reconocerá como una manzana artificial. El objeto tiene las propiedades que usted asocia con el esquema conceptual de manzana, salvo que sea una manzana real, por supuesto.

El esquema conceptual de una manzana real incluirá propiedades generales de las manzanas como tamaños, colores, sabores, usos, etcétera; pero no incluirá detalles acerca de dónde se cosechó exactamente, el medio por el que se transportó hasta el supermercado, el nombre de la persona que la colocó en el anaquel, etcétera. Estos detalles no son importantes para las propiedades que conforman su concepto abstracto de una manzana. Además, tome en consideración que un ciego quizás tenga un esquema de concepto muy diferente, en el que la textura sea más importante.

Al concentrarse en las propiedades generales de un objeto, resulta más fácil razonar acerca de ellos sin distraerse con detalles poco importantes. Habitualmente los esquemas tienen una estructura interna en sus nodos, mientras que las redes semánticas no, en ellas el rótulo es todo el conocimiento acerca del nodo. Una red semántica es como un estructura de datos de la ciencia de la computación, en que la clave de búsqueda corresponde también a los datos almacenados en el nodo. Un esquema es como una estructura de datos donde los nodos contienen registros. Cada registro puede incluir datos, registros o apuntadores a otros nodos.

## 9 MARCOS

Un tipo de esquema que se ha utilizado en muchas aplicaciones de la AI es el **marco** (Minsky 75). Otro es el **guión**, que es en esencia una secuencia de marcos ordenada temporalmente (Schank 77). Propuestos como un método para comprender la visión, el lenguaje natural y otras áreas, los marcos proporcionan una estructura conveniente para representar objetos que son comunes a una situación dada, como los estereotipos. En particular, los marcos son útiles para simular conocimiento de sentido común, que es un área que le resulta muy difícil dominar a las computadoras. Las redes semánticas son básicamente representaciones bidimensionales del conocimiento; los marcos agregan una tercera dimensión para permitir que los nodos tengan estructuras, que pueden ser valores simples u otros marcos.

La característica básica de un marco, es que representa conocimiento relacionado con un tema concreto que cuenta con mucho conocimiento predeterminado. Un sistema de marcos sería una buena elección para describir un dispositivo mecánico como un automóvil; porque sus componentes, como el motor, la carrocería, los frenos, etcétera, se relacionarían entre sí para dar una visión general de sus vínculos. Sería posible obtener aún más detalles de los componentes si se examina la estructura de los marcos. Aunque las marcas individuales de automóviles variarán, muchos autos tienen características comunes como llantas, motor, carrocería, transmisión, etcétera. El marco difiere de la red semántica, que suele utilizarse para una amplia representación del conocimiento. Del mismo modo que con las redes semánticas, no hay estándares para definir los sistemas basados en marcos. Se han diseñado varios lenguajes específicos para los marcos, como FRL, SRL, KRL, KEE, HP-RL y funciones para acrecentarlos como LISP tales como LOOPS y FLAVORS (Finin 86).

Un marco es análogo a la estructura de registro en un lenguaje de alto nivel como Pascal o un átomo con su lista de propiedades en LISP. Correspondiendo a los campos y los valores de un registro son las **ranuras** y sus **rellenos** de un marco. Un marco es básicamente un grupo de ranuras y rellenos que definen un objeto estereotípico. En la figura 2.8 se muestra un ejemplo

Figura 2.8

Figura 2

de un marco para un automóvil. Desde el punto de vista de OAV, el automóvil es un objeto, el nombre de la ranura es el atributo y el relleno es el valor.

---

Ranuras	Rellenos
fabricante	General Motors
modelo	Chevrolet Caprice
año	1979
transmisión	automática
motor	a gasolina
ruedas	4
color	azul

---

Figura 2.8 Marco de automóvil

Casi ningún marco es tan simple como el que se muestra en la figura 2.8. La utilidad de los marcos recae en los sistemas jerárquicos de marcos y en la herencia. Al utilizar marcos en las ranuras de relleno y en la herencia, pueden construirse sistemas de representación del conocimiento muy versátiles. En particular, los sistemas expertos basados en marcos son muy útiles para representar al conocimiento causal, porque su información está organizada en causas y efectos. Por el contrario, los sistemas expertos basados en reglas dependen generalmente del conocimiento no organizado, que no es causal.

Algunas herramientas basadas en marcos como KEE permiten un amplio rango de elementos para almacenar en las ranuras. Éstas pueden contener reglas, imágenes, comentarios, depuración de información, preguntas a los usuarios, hipótesis relacionadas con una situación u otros marcos.

Los marcos se diseñan generalmente para representar conocimiento genérico o específico. En la figura 2.9 se ilustra un marco genérico para el concepto de propiedad.

---

Espacios	Rellenos
nombre	propiedad
especialización_de_tipos	un_tipo_de_objeto (automóvil, bote, casa)
propietario	si-se-agrega: Procedimiento AGREGAR_PROPRIEDAD predeterminado: gobierno si-se-necesita: Procedimiento BUSCAR_PROPIETARIO
ubicación	(casa, trabajo, móvil)
estado	(falta, malo, bueno)
bajo_garantía	(sí, no)

---

Figura 2.9 Marco genérico para propiedad

Los rellenos pueden ser valores, como una propiedad en la ranura del nombre, o un rango de valores, como en la ranura de tipos. Las ranuras también pueden contener procedimientos adjuntos, llamados **anexos de procedimiento**. Por lo general son de tres tipos: los de tipo **si-es-necesario** son procedimientos que se ejecutarán cuando se necesita un valor de relleno pero no hay ninguno presente o el valor **predeterminado** no es adecuado. Los valores predeterminados son de importancia fundamental en los marcos, porque son modelos de algunos aspectos del cerebro. Corresponden a las expectativas de una situación que se construye a partir de la experiencia. Cuando encontramos una nueva situación, se modifica el marco más cercano para ayudarnos a ajustar la situación, así las personas no empiezan desde cero en cada situación nueva; solamente, se modifican las opciones predeterminadas o los otros rellenos. A menudo se utilizan las opciones predeterminadas para representar el conocimiento de

sentido común. Éste puede considerarse como el conocimiento que se conoce de manera general. Utilizamos el sentido común cuando no se dispone de más conocimiento específico para la situación.

El tipo **si-se-agrega** se ejecuta cuando se agrega un valor de una ranura. En la ranura de tipos, el procedimiento si-se-agrega ejecuta un procedimiento llamado AGREGAR-PROPIEDAD para añadir un nuevo tipo de propiedad, si es necesario. Por ejemplo, este procedimiento se ejecutaría para accesorios, TV, estéreo, etcétera, porque la ranura de tipos no contiene estos valores.

Un tipo **si-se-elimina** se ejecuta cada vez que se elimina un valor de una ranura. Este tipo de procedimiento se ejecuta cuando un valor es obsoleto.

Los rellenos de ranura también pueden contener relaciones, como en las ranuras especialización\_de. Las relaciones un-tipo-de y es-un se utilizaron en las figuras 2.9, 2.10 y 2.11 para mostrar la manera en que estos marcos están relacionados jerárquicamente. Los marcos de las figuras 2.9 y 2.10 son genéricos mientras que el de la figura 2.11 es un marco específico porque se trata de un caso del marco del automóvil. Supongamos aquí que las relaciones un-tipo-de son genéricas y las es-un son específicas.

Ranuras	Rellenos
nombre	automóvil
especialización_de	propiedad un-tipo-de
tipos	(sedan, deportivo, convertible)
fabricante	(GM, Ford, Chrysler)
ubicación	móvil
ruedas	4
transmisión	(manual, automática)
motor	(gasolina, diesel)

Figura 2.10 Marco de automóvil: un submarco genérico de propiedad

Los sistemas de marcos están diseñados para que los marcos más genéricos se encuentren en la parte superior de la jerarquía. Se supone que los marcos pueden personalizarse al modificar los casos predeterminados y crear marcos más específicos. Los marcos tratan de modelar objetos del mundo real utilizando el conocimiento genérico para la mayor parte de los atributos de un objeto, y el conocimiento específico para casos especiales. Por ejemplo, generalmente consideramos a las aves como criaturas que pueden volar, pero ciertos tipos de aves, como los pingüinos y los avestruces, no pueden hacerlo. Estos tipos representan clases más específicas de aves y, en la jerarquía de un marco, sus características se encontrarían en niveles inferiores a los de aves como los canarios o petirrojos. En otras palabras, la parte superior de la jerarquía del marco para aves especifica cosas que son más ciertas para todas las aves y los niveles inferiores reflejarían los límites confusos entre los objetos reales. Al objeto que tiene todas las características comunes se le llama **prototipo**, que significa literalmente el primer tipo.

Ranuras	Rellenos
nombre	automóvil de John
especialización_de	es_un automóvil
fabricante	GM
propietario	Juan Pérez
transmisión	automática
motor	a gasolina
estado	bueno
bajo_garantía	sí

Figura 2.11 Ejemplo de un marco de automóvil

Los marcos también pueden clasificarse por sus aplicaciones. Un **marco situacional** contiene conocimiento sobre lo que se espera en una situación determinada, como una fiesta de cumpleaños. Un **marco de acción** contiene ranuras que especifican las acciones que se realizarán en una situación determinada. Es decir, los rellenos son procedimientos para realizar algunas acciones, como eliminar una parte defectuosa de un cinturón de seguridad. Un marco de acción representa conocimiento del procedimiento. La combinación de marcos de situación y de acción puede utilizarse para describir relaciones de causa y efecto, en la forma de **marcos de conocimiento causal**.

Se han construido sistemas de marcos muy sofisticados para diversas tareas. Dos de los sistemas más impresionantes, han demostrado la capacidad de los marcos para descubrir creativamente conceptos matemáticos (Lenat 77) y describir comprensión matemática en dominios como el álgebra lineal (Rissland 78). En estos sistemas, las ranuras se usaron como generadores de tareas, tales como respuestas a preguntas. El sistema AM de Lenat haría conjetas de interesantes conceptos novedosos y los exploraría.

## 2.10 DIFICULTADES CON LOS MARCOS

Los marcos se concibieron originalmente como un paradigma para representar conocimiento estereotipado. La característica más importante de un estereotipo es que cuenta con funciones bien definidas, de modo que muchas de sus ranuras tienen valores predeterminados. Los conceptos matemáticos son buenos ejemplos de estereotipos bien adecuados a los marcos. El paradigma de los marcos tiene un aspecto intuitivo, porque la representación organizada de marcos de conocimiento suele ser más fácil de comprender que cualquier sistema lógico o de producción con muchas reglas (Jackson 86). Sin embargo, han aparecido problemas importantes en los sistemas de marcos que parecen una modificación o cancelación de las ranuras sin restricciones. El ejemplo clásico de este problema se ilustra en la figura 2.12 con un marco que describe elefantes.

nombre	elefante
especialización_de	un-tipo-de mamífero
color	gris
patas	4
tronco	cilíndrico

Figura 2.12 Marco de elefante

A primera vista, el marco para el elefante parece una descripción genérica razonable de los elefantes. Sin embargo, si suponemos que existe un elefante de tres patas llamado Clyde, que ha perdido una pata debido a un accidente de cacería o que simplemente ha mentido sobre el hecho de perder una pata para que su nombre apareciera en este libro. Los importantes es que el marco de elefante asegura que un elefante tiene cuatro patas, no tres. Por tanto, no podemos considerar que el marco para el elefante sea una definición de elefante.

Por supuesto, el marco puede modificarse para permitir elefantes con tres patas o menos. Sin embargo, esto no proporciona una buena definición, por que pueden surgir problemas adicionales con otras ranuras. Suponiendo que Clyde adquiere ictericia y su piel se vuelve amarilla, ¿deja de ser un elefante?

Una alternativa para observar un marco como una definición, es considerarlo como descripción de un elefante típico. Sin embargo, esto conduce a otros problemas debido a la herencia. Observe que el marco indica que un elefante es un-tipo-de mamífero. Como estamos

interpretando los marcos como típicos, nuestro sistema de marco indica que un elefante típico es un mamífero típico; aunque un elefante es un mamífero, probablemente no sea un mamífero típico. Sobre la base de la cantidad, los seres humanos, las vacas, las ovejas o las ratas probablemente sean más representativos de los mamíferos típicos.

Casi ninguno de los sistemas de marcos proporciona una manera de definir ranuras inalterables. Cualquier ranura puede cambiarse, de modo que pueden modificarse o cancelarse las propiedades que un marco hereda hacia cualquier lugar de la jerarquía. Esto significa que cada marco es, en realidad, un marco primitivo y como no hay garantía de que las propiedades sean comunes, cada marco crea sus propias reglas. Nada es realmente cierto en un sistema sin restricciones como éste. Por lo tanto, es imposible hacer frases universales como las creadas en la definición de un elefante. Además, a partir de definiciones más simples como las del elefante, es imposibles construir de manera confiable objetos compuestos, como en el caso del elefante con tres patas. Los mismos tipos de problema se aplican a las redes semánticas con la herencia. Si las propiedades de cualquier nodo pueden cambiarse, entonces nada es cierto.

## 2.11 LÓGICA Y CONJUNTOS

Además de reglas, marcos y redes semánticas, el conocimiento puede representarse mediante los símbolos de la lógica, que es el estudio de las reglas del razonamiento exacto. Una parte importante del razonamiento es la inferencia de conclusiones a partir de premisas. La aplicación de computadoras para realizar razonamientos ha tenido como resultado la programación lógica y el desarrollo de lenguajes basados en la lógica como PROLOG. La lógica también tiene importancia primordial en los sistemas expertos, donde el mecanismo de inferencia razona a partir de los hechos para extraer conclusiones. En realidad, un término descriptivo para la programación lógica y los sistemas expertos es **sistemas de razonamiento automatizados**.

En el siglo IV antes de nuestra era, el filósofo griego Aristóteles desarrolló la lógica formal. Esta lógica aristotélica se basaba en los silogismos, de los que inventó 14 tipos, cinco más se inventaron en la Edad Media. Los silogismos tienen dos **premisas** y una **conclusión**, que se infiere a partir de aquéllas. El siguiente es el ejemplo clásico de un silogismo:

Premisa:	Todos los hombres son mortales
Premisa:	Sócrates es un hombre
Conclusión:	Sócrates es mortal

En un silogismo, la **premisa** proporciona la evidencia a partir de la cual surgirá la conclusión. Un silogismo es una manera de representar el conocimiento. Otra manera es un **diagrama de Venn**, como el que se muestra en la figura 2.13.

El círculo exterior representa a todas las criaturas mortales. El círculo interno representa a un hombre dibujado por completo en el interior del círculo mortal, para indicar que todos los hombres son mortales; como Sócrates es un hombre, el círculo que lo representa se dibuja por completo dentro del segundo círculo. Hablando estrictamente, el círculo que representa a Sócrates debe ser un punto, porque un círculo implica una clase, pero por razones de legibilidad utilizaremos círculos para todo. La conclusión de que Sócrates es mortal es una consecuencia del hecho de que su círculo se encuentra dentro del círculo de las criaturas mortales y, por tanto, debe ser mortal.

En términos matemáticos, un círculo del diagrama de Venn representa un conjunto, que es una colección de objetos. A los objetos de un conjunto se les llama **elementos**. Algunos ejemplos de conjuntos son

- $A = (1, 3, 5)$   
 $B = (1, 2, 3, 4, 5)$   
 $C = (0, 2, 4, \dots)$   
 $D = (\dots, -4, -2, 0, 2, 4, \dots)$   
 $E = (\text{aviones}, \text{globos}, \text{dirigibles}, \text{jets})$   
 $F = (\text{aviones}, \text{globos})$

donde los puntos suspensivos, ..., llamados elipses, indican que los términos continúan indefinidamente.

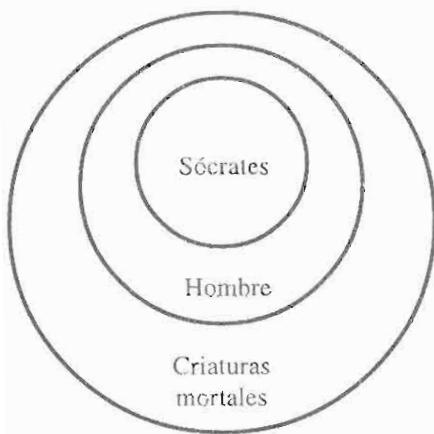


Figura 2.13 Diagrama de Venn

La letra griega épsilon,  $\in$ , indica que un elemento es un miembro de un conjunto. Por ejemplo,  $1 \in A$  significa que el número 1 es un elemento del conjunto A previamente definido. Si un elemento no es miembro de un conjunto, se utiliza el símbolo  $\notin$ , como en  $2 \notin A$ .

Si dos conjuntos arbitrarios, como X y Y se definen para que cada elemento de X sea un elemento de Y, entonces X es un **subconjunto** de Y y se escribe en la forma matemática  $X \subseteq Y$  o  $Y \supseteq X$ . A partir de la definición de un subconjunto, se desprende que cada conjunto es un subconjunto de sí mismo. A un subconjunto que no es el propio conjunto se le llama **subconjunto propio**. Por ejemplo, el conjunto X definido previamente es un subconjunto propio de Y. Al analizar los conjuntos, es útil considerarlos como subconjuntos de un **conjunto universal**. Éste cambia a medida que cambia el tema de análisis. En la figura 2.14 se ilustra un subconjunto formado como la **intersección** de dos conjuntos en el universo de todos los automóviles. Si se trata de números, el universo sería todos los números. El universo se dibuja como un rectángulo que rodea a sus subconjuntos. Los conjuntos universales no se usan sólo por conveniencia. El uso indiscriminado de condiciones para definir conjuntos puede dar como resultado paradojas lógicas.

Si dejamos que A sea el conjunto de todos los automóviles azules, B el conjunto de todos los automóviles con transmisión manual y C el conjunto de todos los automóviles azules con transmisión manual, escribiríamos:

$$C = A \cap B$$

En que el símbolo  $\cap$  representa la intersección de los conjuntos. Otra manera de escribir esto es a partir de elementos x, como se muestra a continuación

$$C = \{x \in U \mid (x \in A) \wedge (x \in B)\}$$

donde

U representa el conjunto universal

$\mid$  se lee como “tal que” (a veces se utilizan dos puntos, :, en lugar de  $\mid$ )

$\wedge$  es el **Y lógico**

La expresión para C se lee como C contiene aquellos elementos x que se encuentran en el universo tal que x es un elemento de A y x es un elemento de B. El Y lógico proviene del álgebra booleana. Una expresión integrada por dos operandos conectados por el operador lógico Y, es verdadero si y sólo si ambos operandos son verdaderos. Si A y B no tienen elementos en común, entonces  $A \cap B = \emptyset$ , donde la letra griega fi,  $\emptyset$ , representa el **conjunto vacío o conjunto nulo**, {}, que es el que no tiene elementos. aunque veces se utiliza la letra griega lambda,  $\lambda$ , para representarlo. A veces se utilizan otras notaciones para el conjunto universal: el número 1 (y el número 0 para el conjunto vacío). Aunque el conjunto vacío no tiene elementos, aún es un conjunto. Una analogía es un restaurante con un conjunto de clientes. Si nadie viene, el conjunto de clientes está vacío.

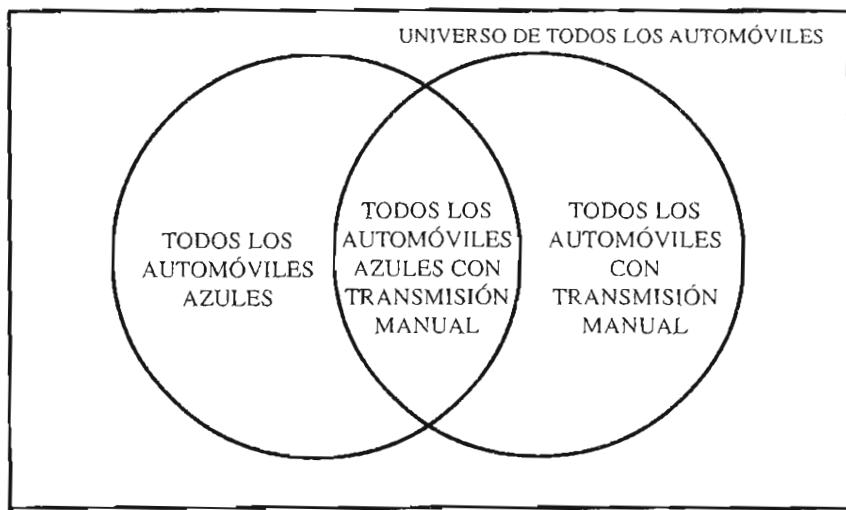


Figura 2.14 Intersección de conjuntos

Otra operación de conjunto es la **unión**, que es el conjunto de todos los elementos en A o B. Se representa de la siguiente forma

$$A \cup B = \{x \in U \mid (x \in A) \vee (x \in B)\}$$

donde

$\cup$  es el operador de conjunto para **unión**

$\vee$  es el operador lógico **O**.

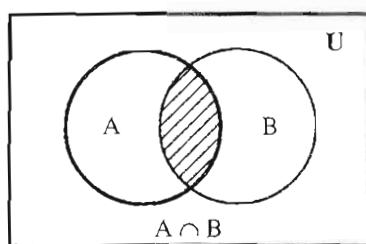
El **complemento** del conjunto A es el conjunto de todos los elementos que no están en A.

$$A' = \{x \in U \mid \sim (x \in A)\}$$

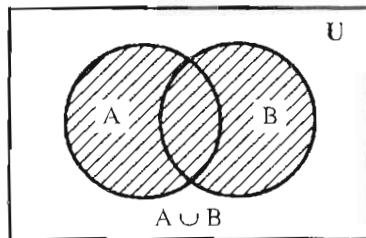
donde

el apóstrofe, ', significa complemento de un conjunto  
la tilde,  $\sim$ , es el operador **NO lógico** es decir negación

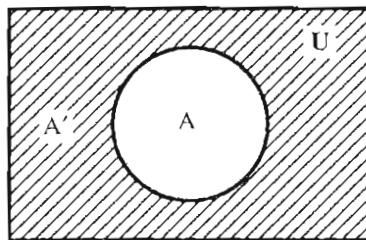
Los diagramas de Venn para estas operaciones básicas se muestran en la figura 2.15.



(a) INTERSECCIÓN DE CONJUNTOS



(b) UNIÓN DE CONJUNTOS



(c) COMPLEMENTO DE UN CONJUNTO

Figura 2.15 Operaciones básicas con conjuntos

## 2.12 LÓGICA DE PROPOSICIONES

La más antigua y uno de los tipos más simples de **lógica formal** es el silogismo. El término *formal* significa que la lógica se relaciona con la forma de las frases lógicas más que con su

significado. En otras palabras, tiene que ver con la sintaxis más que con la semántica de las frases. Aunque el término *lógica formal* puede sonar intimidatorio, no es más difícil que el álgebra. En realidad, el álgebra es realmente la lógica formal para números. Por ejemplo, supongamos le pidieron resolver el siguiente problema: una escuela tiene 25 computadoras con un total de 60 tarjetas de memoria. Algunas de ellas tienen dos y algunas cuatro tarjetas. ¿Cuántas computadoras hay de cada tipo? La solución puede escribirse algebraicamente como sigue:

$$\begin{aligned} 25 &= X + Y \\ 60 &= 2X + 4Y \end{aligned}$$

que puede resolverse fácilmente:  $X = 20$  y  $Y = 5$ .

Ahora considere este problema. Hay 25 animales con un total de 60 patas en un corral. Algunos tienen dos patas y otros cuatro. (Nota: Clyde, nuestro elefante de tres patas está en África, no en el corral). ¿Cuántos animales hay de cada tipo? Como puede ver, las mismas ecuaciones algebraicas se aplican si hablamos de computadoras, animales o cualquier otra cosa. De la misma forma que las ecuaciones algebraicas nos permiten concentrarnos en la manipulación matemática de símbolos sin considerar lo que representan, la lógica formal nos permite concentrarnos en el razonamiento sin confundirnos con los objetos sobre los que razonamos.

Como un ejemplo de lógica formal, tomaremos en cuenta el siguiente silogismo con las palabras sin sentido *squeeg* y *moof*.

Premisa:	Todos los squeegs son moofs
Premisa:	John es un squeeg
Conclusión:	John es un moof

Aunque las palabras *squeeg* y *moof* no tienen sentido ni significado, la *forma* de este argumento sigue siendo correcta. Es decir, el argumento es **válido** sin importar cuáles palabras se usen, porque el silogismo tiene una forma válida. En realidad, cualquier silogismo con la forma

Premisa:	Todas las X son Y
Premisa:	Z es una X
Conclusión:	Z es una Y

es válido sin importar qué se sustituye con X, Y y Z. Este ejemplo ilustra esos significados, sin importar la lógica formal. Sólo es importante la forma o apariencia. Este concepto de separar la forma del significado o semántica, es lo que convierte a la lógica en una herramienta poderosa. También hace que la validez de un argumento pueda considerarse objetivamente, sin el prejuicio causado por la semántica. Esto se parece al álgebra en que la corrección de expresiones como  $X + X = 2X$  se mantiene sin importar si X representa un entero, manzanas, aviones o cualquier cosa.

Los silogismos aristotélicos fueron el fundamento de la lógica hasta 1847, cuando el matemático inglés George Boole publicó la primer obra que describía la **lógica simbólica**. Aunque Leibnitz había desarrollado su propia versión en el siglo XVII, nunca llegó a ser de uso general. Uno de los nuevos conceptos que Boole propuso, fue una modificación de la idea aristotélica de que el sujeto tiene existencia, llamada **importancia existencial**. De acuerdo con la concepción clásica de Aristóteles, una proposición como "todas las sirenas nadan bien" no se podría utilizar como premisa o conclusión porque las sirenas no existen. La idea de Boole, ahora llamada **concepción moderna**, hace más relajada esta restricción; su importancia se debe a

que ahora se puede razonar acerca de clases vacías. Por ejemplo, una proposición como “todos los discos que fallan son baratos” no podría utilizarse en un silogismo aristotélico, a menos que hubiera por lo menos un disco que falló.

Otra contribución de Boole fue la definición de un conjunto de **axiomas** integrado por símbolos que representan tanto objetos como clases, y operaciones algebraicas para manipular los símbolos. Son las definiciones fundamentales a partir de las cuales se construyen sistemas lógicos como las matemáticas y la propia lógica. Utilizando únicamente axiomas se pueden elaborar teoremas. Un teorema es una afirmación que se puede probar mostrando cómo se deriva de los axiomas. Entre 1910 y 1913, Whitehead y Russell publicaron su monumental obra de tres volúmenes y 2000 páginas, *Principios matemáticos*, que mostró una lógica formal como base de las matemáticas. Fue recibido como un gran hito porque, aparentemente, pone a las matemáticas sobre bases firmes en lugar de apelar a la intuición, eliminando totalmente el significado de la aritmética y concentrándose, en cambio, en las formas y su manipulación. Debido a esto, las matemáticas se han descrito como una colección de marcas sin sentido en el papel. Sin embargo, en 1931 Gödel probó que los sistemas formales basados en axiomas no siempre resultaban internamente congruentes y, por tanto, libres de contradicciones.

La lógica de proposiciones, también llamada *cálculo de proposiciones*, es una lógica simbólica para la manipulación de proposiciones. En particular, se ocupa de la manipulación de las **variables lógicas** que representan proposiciones. Aunque muchas personas consideran al cálculo dentro de la perspectiva que inventaron Newton y Leibnitz, la palabra tiene un significado más general; procede de la palabra latina *calculus*, pequeña piedra utilizada para realizar cálculos. Su significado general es un sistema especial para la manipulación de símbolos. Otros términos utilizados para la lógica de proposiciones son **cálculo de afirmaciones** y **cálculo de frases**. Las afirmaciones se clasifican generalmente en cuatro tipos, como se ilustra en la tabla 2.2.

Tipo	Ejemplo
Imperativo	¡Qué te digo!
Interrogativo	¿Qué es eso?
Admirativo	¡Eso es grandioso!
Declarativo	Un cuadrado tiene cuatro lados iguales.

Tabla 2.2 Tipos de afirmaciones

La lógica de proposiciones tiene que ver con el subconjunto de oraciones declarativas que pueden clasificarse como verdaderas o falsas. Una afirmación como “Un cuadrado tiene cuatro lados iguales”, tiene un valor de verdad definido como verdadero, mientras que la afirmación “George Washington fue el segundo presidente de Estados Unidos” tiene un valor de verdad falso. A una afirmación en la que es posible determinar su valor de verdad, se le llama **frase o proposición**. A una frase también se le llama **afirmación cerrada**, porque su valor de verdad no está abierto a cuestionamientos.

Si en el prefacio de este libro se afirma “todo en estas páginas es una mentira”, esa frase no puede clasificarse como verdadera o falsa. Si es verdadera, entonces dije la verdad en el prefacio, cosa que no puedo hacer; si no es verdad, entonces cada palabra escrita debe ser verdadera, por lo tanto yo miento, lo que tampoco puede ser verdad. Frases de este tipo son conocidas como la Paradoja de Liar. A las afirmaciones que no se puede responder de manera absoluta se les llama **afirmaciones abiertas**. La afirmación “Las espinacas son deliciosas” es también una frase abierta, porque es cierto para unas personas y no lo es para otras. “Él es alto” es una afirmación abierta porque tiene una variable “Él”. No es posible asignar valores

verdaderos para abrir afirmaciones hasta que conozcamos a la persona o el caso específico al que se refiere la variable. Otra dificultad con esta afirmación es el significado de la palabra "alto", pues lo que parece alto para algunos no lo parece para otros. Aunque la ambigüedad de "alto" no se puede manejar en el cálculo de proposiciones o de afirmaciones, puede resolverse fácilmente con la lógica de confusión, descrita en el capítulo 5.

Una **afirmación compuesta** está formada por el uso de conectores lógicos o frases individuales. Los conectores comunes de la lógica se muestran en la tabla 2.3.

Conector	Significado
$\wedge$	Y; conjunción
$\vee$	O; disyunción
$\sim$	NO; negación
$\rightarrow$	si...entonces; condicional
$\leftrightarrow$	si y sólo si; doble condicional

Tabla 2.3 Algunos conectores lógicos

Hablando estrictamente, la negación no es un conector porque es una operación unitaria que aplica al siguiente operando, de modo que en realidad no conecta nada. Tiene mayor prioridad que los otros operadores, así que no es necesario colocarla entre paréntesis. Una frase como  $\sim p \wedge q$  significa lo mismo que  $(\sim p) \wedge q$ .

El condicional es análogo a la flecha de las reglas de producción en las que se expresa como una forma si...entonces. Por ejemplo

si está lloviendo entonces lleva una sombrilla

se puede poner en la forma

$p \rightarrow q$

donde

$p = \text{está lloviendo}$   
 $q = \text{lleva una sombrilla}$

Algunas veces se utiliza  $\supset$  por  $\rightarrow$ . Otro término para el condicional es **implicación material**. El condicional doble,  $p \leftrightarrow q$ , es equivalente a

$(p \rightarrow q) \wedge (q \rightarrow p)$

y es verdadero sólo cuando  $p$  y  $q$  tienen los mismos valores de verdad. Esto es,  $p \leftrightarrow q$  sólo es verdadero cuando ambos son verdaderos o cuando ambos son falsos. El condicional doble tiene los siguientes significados:

$p$  si y sólo si  $q$   
 $q$  si y sólo si  $p$   
 si  $p$  entonces  $q$ , si  $q$  entonces  $p$

Una **tautología** es una frase compuesta que siempre es verdadera, sin importar si sus frases individuales son verdaderas o falsas. Una **contradicción** es una frase compuesta que siempre es falsa. Una frase **condicional** no es una tautología ni una contradicción. A las tautologías y a las contradicciones se les denomina analíticamente verdaderas o analíticamente falsas, respectivamente, porque sus valores de verdad pueden determinarse a partir de sus simples formas. Por ejemplo, la tabla de verdad de  $p \vee \neg p$  muestra que es una tautología, mientras que  $p \wedge \neg p$  es una contradicción.

Si un condicional es también una tautología entonces se le llama *implicación* y tiene el símbolo  $\Rightarrow$  en lugar de  $\rightarrow$ . A un condicional doble que también es una tautología se le llama *equivalencia lógica* o *equivalencia de material* y se simboliza con  $\Leftrightarrow$  o  $\equiv$ . Dos frases lógicamente equivalentes siempre tienen los mismos valores de verdad. Por ejemplo,  $p \equiv \neg \neg p$ .

Desafortunadamente, ésta no es la única definición posible para una implicación, ya que hay 16 posibles tablas de verdad para dos variables que pueden tomar valores de verdadero o falso. En efecto, en un artículo se han revisado 11 definiciones diferentes del operador de implicación utilizado en los sistemas expertos (Whalen 85).

El condicional no significa exactamente lo mismo que SI ... ENTONCES en un lenguaje de procedimientos o un sistema experto basado en reglas. En los sistemas de procedimientos y experto, SI ... ENTONCES significa ejecutar las acciones siguiendo el ENTONCES si las condiciones del SI son verdaderas. En la lógica, el condicional se define por su tabla de verdad y su significado puede traducirse al lenguaje natural de varias maneras. Por ejemplo, si

$$p \rightarrow q$$

donde  $p$  y  $q$  son cualquier afirmación, esto puede traducirse como

p implica q  
 si p entonces q  
 p, sólo si q  
 p es suficiente para q  
 q si p  
 q es necesario para p

Como ejemplo, hagamos que  $p$  represente "usted tiene 18 años o más" y que  $q$  represente "usted puede votar". La  $p \rightarrow q$  condicional puede significar

usted tiene 18 años o más implica que puede votar  
 si tiene 18 años o más entonces puede votar  
 usted tiene 18 años o más, sólo si puede votar  
 usted tiene 18 años o más, es suficiente para votar  
 usted puede votar si tiene 18 años o más  
 usted puede votar si es necesario porque tiene 18 años o más

En algunos casos, es necesario un cambio en el orden de las palabras para que sean frases gramaticalmente correctas. El último ejemplo dice que si no ocurre  $q$  entonces  $p$  tampoco. Se expresa de manera apropiada como "La posibilidad de votar requiere que usted tenga 18 años o más".

En la tabla 2.4 se muestran los valores para los conectores binarios lógicos. Son binarios porque requieren dos operandos. El conector de negación,  $\neg$ , es un operador unitario en el operando que le sigue, como se muestra en la tabla 2.5.

Un conjunto de conectores lógicos es **adecuado** si cada función de verdad puede representarse utilizando sólo la conectividad del conjunto adecuado. Ejemplos de conjuntos adecuados son  $\{\sim, \wedge, \vee\}$ ,  $\{\sim, \wedge\}$ ,  $\{\sim, \vee\}$  y  $\{\sim, \rightarrow\}$ .

A un conjunto de un solo elemento se le llama **unitario**. Hay dos conjuntos de unitarios adecuados. Son NO-O y NO-Y. El conjunto NO-O es  $\{\downarrow\}$  y el conjunto NO-Y es  $\{\mid\}$ . Al operador “ $\mid$ ” se le llama **negación alternativa**. Se utiliza para negar que  $p$  y  $q$  son verdaderos. Es decir  $p \mid q$  afirma que por lo menos una de las afirmaciones,  $p$  o  $q$ , es verdadera. El **operador de negación de unión**, “ $\downarrow$ ” niega que  $p$  o  $q$  sean verdaderos. Es decir,  $p \downarrow q$  afirma que tanto  $p$  como  $q$  son falsos.

$p$	$q$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
V	V	V	V	V	V
V	F	F	V	F	F
F	V	F	V	V	F
F	F	F	F	V	V

Tabla 2.4 Tabla de verdad de los conectores lógicos binarios

$p$	$\neg p$
V	F
F	V

Tabla 2.5 Tabla de verdad de la negación

## 2.13 LA LÓGICA DE PREDICADO DE PRIMER ORDEN

Aunque la lógica de proposiciones es útil, tiene limitaciones. El principal problema es que la lógica de proposiciones sólo puede tratar con afirmaciones completas. Es decir, no puede examinar la estructura interna de una afirmación. La lógica de proposiciones ni siquiera puede probar la validez de un silogismo como

Todos los seres humanos son mortales  
 Todas las mujeres son seres humanos  
 Por tanto, todas las mujeres son mortales

La **lógica de predicado** se desarrolló con el fin de analizar casos más generales. Su forma más simple es la lógica de predicados de **primer orden**, la base de lenguajes de programación lógicos como PROLOG. En esta sección utilizaremos el término *lógica de predicado* para aludir a la lógica de predicado de primer orden. La lógica de proposiciones es un subconjunto de la lógica de predicado.

La lógica de predicado se relaciona con la estructura interna de las afirmaciones, sobre todo, se relaciona con el uso de palabras especiales llamadas **cuantificadores**, como “todo”, “algo” y “no”. Estas palabras son muy importantes porque cuantifican explícitamente otras palabras y hacen más exactas las afirmaciones. Todos los cuantificadores se relacionan con “cuánto” y, por tanto, permiten un alcance más amplio de la expresión que la lógica de proposiciones.

## 2.14 EL CUANTIFICADOR UNIVERSAL

Una afirmación cuantificada universalmente tiene el mismo valor de verdad para todos los reemplazos en el mismo dominio. La **variable de dominio** el **cuantificador universal** se representa con el símbolo  $\forall$  seguido por uno o más argumentos. El símbolo  $\forall$  se interpreta como “para cada uno” o “para todos”. Por ejemplo, en el dominio de los números

$$(\forall x) (x + x = 2x)$$

establece que para cada  $x$  (donde  $x$  es un número), la afirmación  $x + x = 2x$  es verdadera. Si representamos esta frase con el símbolo  $p$ , entonces puede expresarse aún más brevemente como:

$$(\forall x) (p)$$

Como otro ejemplo, dejemos que  $p$  represente la afirmación “todos los perros son animales”, como en

$$(\forall x) (p) \equiv (\forall x) (\text{si } x \text{ es un perro} \rightarrow x \text{ es un animal})$$

La afirmación opuesta es “los perros no son animales” y se escribe como

$$(\forall x) (\text{si } x \text{ es un perro} \rightarrow \neg x \text{ es un animal})$$

También puede leerse como

Cada perro no es un animal  
Todos los perros no son animales

Otro ejemplo sería “todos los triángulos son polígonos”, y se escribe como sigue:

$$(\forall x) (x \text{ es un triángulo} \rightarrow x \text{ es un polígono})$$

y se lee “para todo  $x$ , si  $x$  es un triángulo, entonces  $x$  es un polígono”. Una manera más corta de escribir afirmaciones lógicas que incluyen predicados es utilizando las **funciones de predicado** para describir las propiedades del sujeto. La afirmación lógica anterior también puede escribirse

$$(\forall x) (\text{triángulo}(x) \rightarrow \text{polígono}(x))$$

Las funciones de predicado suelen escribirse en notación más breve utilizando mayúsculas para representar los predicados. Por ejemplo, hagamos que  $T$  = triángulo y  $P$  = polígono. Entonces la afirmación de triángulo puede escribirse de manera aún más breve como

$$(\forall x) (T(x) \rightarrow P(x)) \circ (\forall y) (T(y) \rightarrow P(y))$$

Observe que cualquier variable puede utilizarse en lugar de las variables de modelo  $x$  y  $y$ . Un ejemplo más, hagamos que  $H$  sea la función de predicado de *ser humano* y  $M$  sea la función de *mortal*. Entonces la afirmación de que todos los seres humanos son mortales puede escribirse como:

$$(\forall x) (H(x) \rightarrow M(x))$$

y se lee que para toda  $x$ , si  $x$  es ser humano, entonces  $x$  es mortal. Esta afirmación de lógica de predicado también puede representarse como una red semántica, tal como se muestra en la figura 2.16. También puede expresarse con reglas como

SI  $x$  es un ser humano  
ENTONCES  $x$  es mortal

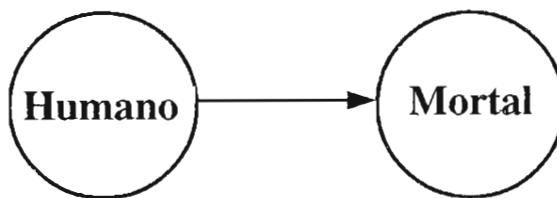


Figura 2.16 Representación de la red semántica de una afirmación de lógica de predicado

El cuantificador universal también puede interpretarse como un conjunto de predicados acerca de casos. Por ejemplo, un perro llamado Sparkler es un caso particular de la clase perros y puede escribirse como

$Pero(Sparkler)$

Donde Perro es la función de predicado y Sparkler es un caso.

Una frase de lógica de predicado como

$(\forall x) P(x)$

puede interpretarse casos de  $a_i$  como

$P(a_1) \wedge P(a_2) \wedge P(a_3) \wedge \dots \wedge P(a_n)$

donde los puntos suspensivos indican que el predicado se extiende a todos los miembros de la clase. Esta afirmación indica que el predicado se aplica a todos los casos de la clase.

Es posible utilizar cuantificadores múltiples. Por ejemplo, la ley conmutativa de la adición requiere dos cuantificadores, como en

$(\forall x) (\forall y) (x + y = y + x)$

que establece que “para todo  $x$  y para todo  $y$ , la suma de  $x$  y  $y$  es igual a la suma de  $y$  y  $x$ ”.

## 2.15 EL CUANTIFICADOR EXISTENCIAL

Otro tipo de cuantificador es el **cuantificador existencial**. Éste describe una afirmación como verdadera por lo menos para un miembro del dominio. Esto es una forma restringida del cuantificador universal que indica que una instrucción es verdadera para todos los miembros del dominio. El cuantificador existencial se escribe como  $\exists$  seguido por uno o más archivos. Por ejemplo

$$\begin{aligned}(\exists x) & \ (x \cdot x = 1) \\(\exists x) & \ (\text{elefante}(x) \wedge \text{nombre}(\text{Clyde}))\end{aligned}$$

La primera afirmación establece que hay algún  $x$  cuyo producto al multiplicarse por sí mismo es igual a 1. La segunda instrucción indica que hay algún elefante con el nombre de Clyde.

El cuantificador existencial puede leerse de varias maneras, como

existe  
por lo menos un  
para algún  
hay un  
algún

Por ejemplo:

$$(\forall x) \ (\text{elefante}(x) \rightarrow \text{cuatro-patas}(x))$$

dice que todos los elefantes tienen cuatro patas. Sin embargo, la afirmación de que algunos elementos tienen tres patas se escribe con un  $\vee$  lógico y con cuantificadores universales como sigue

$$(\exists x) \ (\text{elefante}(x) \wedge \text{tres-patas}(x))$$

Así como el cuantificador universal puede expresarse como conjunción, el cuantificador existencial puede expresarse como una disyunción de casos,  $a_i$ .

$$P(a_1) \vee P(a_2) \vee P(a_3) \vee \dots \vee P(a_n)$$

Las afirmaciones cuantificadas y sus negaciones para el ejemplo en que  $P$  representa "elefantes y mamíferos" se muestra en la tabla 2.6. Los números entre paréntesis sólo sirven para identificar los ejemplos para el análisis siguiente.

Ejemplo	Significado
(1a) $(\forall x)(P)$	Todos los elefantes son mamíferos
(1b) $(\exists x)(-P)$	Algunos elefantes no son mamíferos
(2a) $(\exists x)(P)$	Algunos elefantes son mamíferos
(2b) $(\forall x)(-P)$	Ningún elefante es mamífero

Tabla 2.6 Ejemplos de cuantificadores negados

Los ejemplos (1a) y (1b) son negaciones entre sí, como (2a) y (2b). Observe que la negación de una afirmación cuantificada universalmente de (1a) es una afirmación cuantificada existencialmente de la negación de  $P$ , como se muestra en (1b). De igual manera, la negación de una afirmación cuantificada existencialmente de (2a) es la afirmación cuantificada universalmente de la negación de  $P$ , como se muestra en (2 b).

## 2.16 CUANTIFICADORES Y CONJUNTOS

Los cuantificadores pueden utilizarse para definir conjuntos en un universo, U, como se muestra en la tabla 2.7.

La relación de que A es un subconjunto propio de B, escrito como  $A \subset B$ , significa que a pesar de que todos los elementos de A están en B, hay por lo menos un elemento de B que no se encuentra en A. Si E representa elefantes y M representa mamíferos, la relación de conjunto

$$E \subset M$$

establece que todos los elefantes son mamíferos, pero no todos los mamíferos son elefantes. Si G = gris y C = cuatro-patas, la afirmación de que todos los elefantes grises con cuatro patas son mamíferos se escribe

$$(E \cap G \cap C) \subset M$$

Expresión de conjunto	Equivalente lógico
$A = B$	$\forall x (x \in A \leftrightarrow x \in B)$
$A \subseteq B$	$\forall x (x \in A \rightarrow x \in B)$
$A \cap B$	$\forall x (x \in A \wedge x \in B)$
$A \cup B$	$\forall x (x \in A \vee x \in B)$
$A'$	$\forall x (x \in U \mid \neg (x \in A))$
$U$ (Universo)	V (Verdadero)
$\emptyset$ (conjunto vacío)	F (Falso)

Tabla 2.7 Algunas expresiones de conjuntos y sus equivalentes lógicos

Utilizando las siguientes definiciones, se muestran algunos ejemplos de afirmaciones cuantificadas

E = elefantes

R = reptiles

G = gris

C = cuatro-patas

P = perros

M = mamíferos

Ningún elefante es reptil

$E \cap R = \emptyset$

Algunos elefantes son grises

$E \cap G \neq \emptyset$

Ningún elefante es gris

$E \cap G = \emptyset$

Algunos elefantes no son grises

$E \cap G' \neq \emptyset$

Todos los elefantes son grises y de cuatro patas

$E \subset (G \cap C)$

Todos los elefantes y perros son mamíferos

$(E \cup P) \subset M$

Algunos elefantes tienen cuatro patas y son grises

$(E \cap F \cap G) \neq \emptyset$

Como otra analogía de los conjuntos y formas lógicas, las leyes de De Morgan se muestran en la tabla 2.8, donde el símbolo de equivalencia,  $\equiv$ , el (bicondicional) significa que la afirmación de la izquierda tiene el mismo valor de verdad que la de la derecha. Es decir, las afirmaciones son equivalentes.

Conjunto	Lógica
$(A \cup B)' \equiv A' \cap B'$	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
$(A \cap B)' \equiv A' \cup B'$	$\neg(p \vee q) \equiv \neg p \wedge \neg q$

Tabla 2.8 Conjunto y formas lógicas de las leyes de De Morgan

## 2.17 LIMITACIONES DE LA LÓGICA DE PREDICADO

Aunque esta lógica es muy útil en muchas situaciones, hay algunos tipos de afirmaciones que no pueden expresarse en ella utilizando los cuantificadores universal y existencial (Rescher 64). Por ejemplo, la siguiente afirmación no puede expresarse con lógica de predicado:

La mayoría de la clase recibió dieces

En esta afirmación, el cuantificador *mayoría* significa más de la mitad, pero no puede expresarse con los cuantificadores universal y existencial. Para implantar *mayoría* una lógica debe proporcionar algunas predicados para conteo, como la lógica borrosa, descrita en el capítulo 5. Otra limitante de esta lógica es en la expresión de cosas que a veces son verdaderas, pero no siempre. Este problema también puede resolverse con lógica borrosa. Sin embargo, la introducción de un conteo también produce más complicaciones en el sistema lógico y la hace de manera más parecida a las matemáticas.

## 2.18 RESUMEN

En este capítulo revisamos los elementos de la teoría del conocimiento y las técnicas para representarlo. La representación del conocimiento es de la mayor importancia para los sistemas expertos. El conocimiento puede clasificarse de varias maneras, como *a priori*, *a posteriori*, de procedimiento, declarativo y tácito. Las reglas de producción, las redes semánticas, los esquemas, los marcos y la lógica son métodos comunes para la representación del conocimiento en los sistemas expertos. Cada uno de estos paradigmas tiene ventajas y desventajas. Antes de diseñar un sistema experto, debe decidir cuál es el mejor paradigma para el problema que habrá de resolverse. En lugar de tratar de usar una herramienta para todos los problemas, elija la mejor herramienta para el problema particular.

## PROBLEMAS

- 2.1 Dibuje una red semántica para computadoras utilizando vínculos UN-TIPO-DE y ES-UN. Considere las clases microcomputadora, minicomputadora, supercomputadora, sistema de cómputo, dedicado, de propósito general, a nivel de tarjeta, computadora-en-un-chip, de un solo procesador y de varios procesadores. Incluya casos específicos.

# CAPÍTULO 3

## *Métodos de inferencia*

### 3.1 INTRODUCCIÓN

En este capítulo analizaremos varios métodos de razonamiento o inferencia, tema que reviste particular importancia en los sistemas expertos porque el razonamiento es la técnica normal que utilizan los sistemas expertos para resolver problemas. Es decir, se usan regularmente cuando no existe una solución algorítmica o cuando ésta es inadecuada y el razonamiento ofrece la única posibilidad de obtener una solución.

### 3.2 ÁRBOLES, REJILLAS Y GRÁFICAS

Un **árbol** es un estructura jerárquica de datos conformada por **nodos** que almacenan información o conocimiento, y por **ramas**, que conectan a los nodos. A veces a las ramas se les denomina **vínculos** o **bordes**, y a los nodos **vértices**. En la figura 3.1 se muestra un árbol binario general que tiene cero, una o dos ramas por nodo. En un **árbol orientado**, el **nodo raíz** ocupa el lugar más alto de la **jerarquía**, y las **hojas**, el más bajo. Un árbol puede considerarse como una clase especial de red semántica en la que cualquier nodo, excepto el raíz, tiene exactamente un nodo **padre** y cero o más nodos **hijo**. Para el tipo usual de árbol binario hay un máximo de dos hijos por nodo y los nodos hijo de la derecha son diferentes a los de la izquierda.

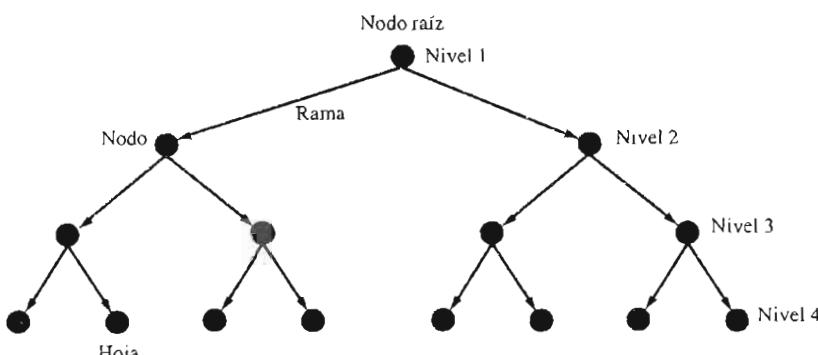


Figura 3.1 Árbol binario

Si un nodo tiene más de un parente, se encuentra en una red. En la figura 3.1, se observa que sólo hay una secuencia de bordes, o **ruta**, de la raíz a cualquier nodo; porque no es posible moverse contra una flecha. En los árboles orientados todas las flechas apuntan hacia abajo.

Los árboles son un caso especial de una estructura general matemática llamada **gráfica**. El término *red* suele utilizarse como sinónimo de *gráfica* cuando se describe un ejemplo particular de gráfica como una red telefónica. Una gráfica puede tener cero o más vínculos entre los nodos y ninguna distinción entre padres e hijos. Un ejemplo sencillo de gráfica es un mapa: las ciudades son los nodos y los vínculos son los caminos. Los vínculos pueden tener flechas o direcciones asociadas con ellos y un **peso** para diferenciarse; se puede hacer una analogía con las calles de un solo sentido que tienen límites de peso para los camiones que circulan por ellas. Los pesos de una gráfica pueden representar cualquier tipo de información. Si la gráfica es la ruta de una línea aérea, los pesos pueden ser los kilómetros entre las ciudades, el costo del vuelo, el consumo de combustible, etcétera.

Un sistema neuronal artificial es otro ejemplo de una gráfica con ciclos, porque durante el entrenamiento hay retroalimentación de la información de una capa a la otra de la red, lo que modifica los pesos. Una gráfica simple no tiene vínculos que se encuentren inmediatamente detrás del propio nodo, como se muestra en la figura 3.2(a). Un **circuito** o **ciclo** es una ruta que recorre una gráfica y que inicia y termina en el mismo nodo, como la ruta ABCA de la figura 3.2(a). Como su nombre lo indica, una gráfica **acíclica** no tiene ciclos, mientras que una **gráfica conectada** tiene vínculos con todos sus nodos. En la figura 3.2(c) se muestra una gráfica con vínculos dirigidos, llamada **digráfica**, y un **autolazo**. Una gráfica acíclica con dirección es una **rejilla**, y se ejemplifica en la figura 3(d); y un árbol con una sola ruta desde la raíz hasta su única hoja es un **árbol degenerado**. Los árboles degenerados binarios de tres nodos se muestran en la figura 3.2(e). Por lo general, las flechas *no* se muestran explícitamente en un árbol, porque se supone que éstas apuntan hacia abajo.

Árboles y rejillas son útiles para clasificar objetos a causa de su naturaleza jerárquica. Un ejemplo es un árbol familiar que muestra las relaciones y la ascendencia de las personas relacionadas. Otra aplicación de los árboles y las rejillas es la toma de decisiones, en estos casos se les llama **árboles** o **rejillas de decisión**; en lo sucesivo se utilizará el término **estructura** para denominarlos a ambos. Una estructura de decisión es tanto un esquema de representación del conocimiento como un método de razonamiento sobre su conocimiento. En la figura 3.3, se muestra un ejemplo de árbol de decisión para clasificar animales. Este ejemplo sirve para el juego clásico de las 20 preguntas: los nodos contienen preguntas, las ramas “si” o “no” las respuestas y las hojas las suposiciones de cuál animal se trata.

A diferencia de los árboles de ciencias de la computación, los árboles de clasificación pueden dibujarse con la raíz hacia abajo. En la figura 3.4 se muestra una pequeña parte de un árbol de decisión para clasificar fresas, en él no se muestra la raíz, que tiene una rama hacia el nodo “Hojas simples” y otra hacia el nodo “Hojas compuestas”. El proceso de decisión inicia en la parte inferior, al identificar los rasgos más notorios, como si las hojas son simples o compuestas, los detalles más específicos, que requieren una observación más cercana, se usan a medida que subimos por el árbol. Es decir, los conjuntos más grandes de alternativas se examinan primero y después el proceso de decisión comienza a reducir las posibilidades a grupos más pequeños. Ésta es una buena forma de organizar las decisiones tomando en consideración el tiempo y el esfuerzo necesarios para llevar a cabo observaciones más detalladas.

Si las decisiones son binarias, un árbol de decisión binario es a la vez fácil de construir y muy eficiente. Cualquier pregunta baja un nivel en el árbol: una pregunta puede decidir una de dos posibles respuestas, dos preguntas pueden decidir una de cuatro posibles respuestas, tres preguntas pueden decidir una de ocho posibles respuestas, etc. Si se construye un árbol binario de modo tal que todas las hojas sean respuestas y todos los nodos dirigidos hacia

abajo sean preguntas, puede haber un máximo de  $2^N$  respuestas para N preguntas. Por ejemplo, diez preguntas pueden clasificar uno de 1,024 animales, mientras que 20 preguntas pueden clasificar una de 1,048,576 posibles respuestas.

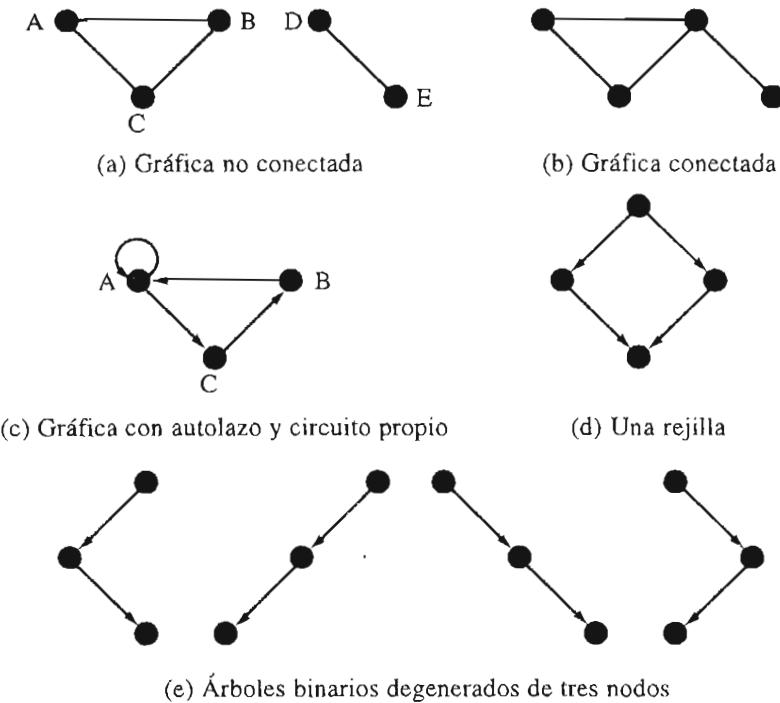


Figura 3.2 Gráficas simples

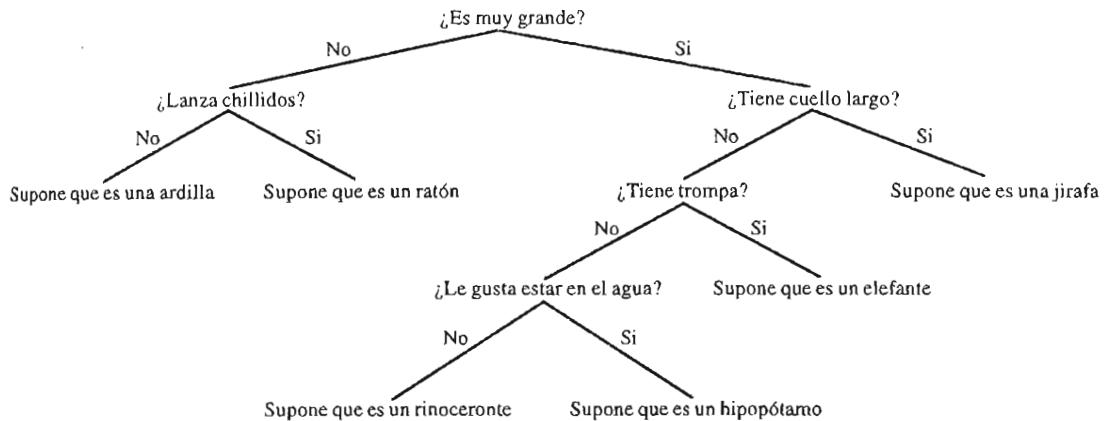


Figura 3.3 Árbol de decisión mostrando conocimiento acerca de animales

Otra característica útil de los árboles de decisión es que pueden realizar un **autoaprendizaje**. Si la suposición es errónea, puede llamarse a un procedimiento para pedir al usuario una nueva pregunta de clasificación correcta y las respuestas a las opciones "si" y "no". Deben

crearse dinámicamente nuevos nodos, ramas y hojas y añadirse al árbol. En el programa original Animales, escrito en BASIC, el conocimiento estaba almacenado en instrucciones DATA; cuando el usuario enseñaba al programa un nuevo animal, tenía lugar el aprendizaje automático mientras el programa generaba nuevas instrucciones DATA que contenían información acerca del nuevo animal. En Pascal y otros lenguajes con soporte de apuntadores, el conocimiento sobre el animal puede almacenarse en árboles. Con la herramienta CLIPS para sistemas expertos, pueden construirse automáticamente nuevas reglas mientras el programa aprende nuevo conocimiento (Giarratano 92). Esta **adquisición automatizada de conocimiento** es muy útil porque puede evitar el cuello de botella de la adquisición de conocimiento descrito en el capítulo 1.

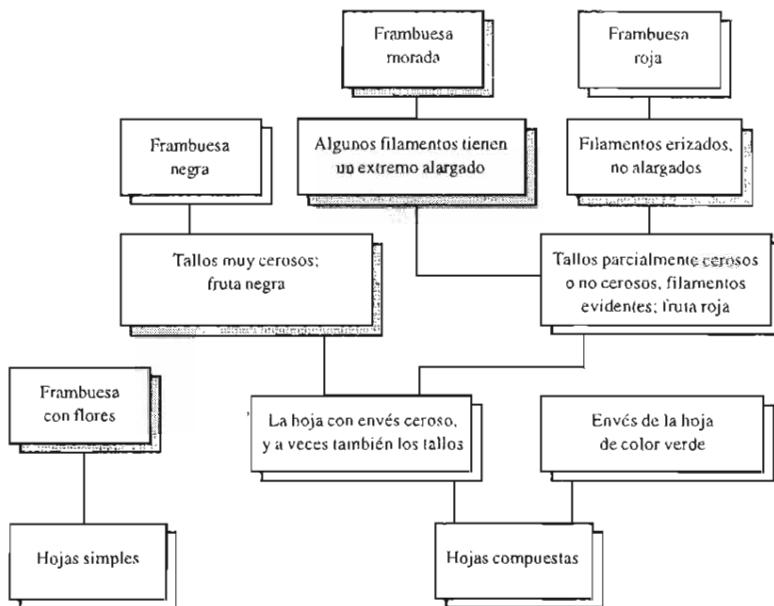


Figura 3.4 Fragmento de un árbol de decisión para clasificar varias especies de frambuesas

Las estructuras de decisión pueden traducirse a reglas de producción en forma mecánica. Esto puede lograrse fácilmente a partir de una búsqueda primero a lo ancho de la estructura y de la generación de reglas SI... ENTONCES en cada nodo. Por ejemplo, el árbol de decisión de la figura 3.3 podría traducirse a reglas de la siguiente manera:

```

SI PREGUNTA = "¿ES MUY GRANDE?" Y RESPUESTA = "NO"
ENTONCES PREGUNTA := "¿LANZA CHILLIDOS?"
  
```

```

SI PREGUNTA = "¿ES MUY GRANDE?" Y RESPUESTA = "SÍ"
ENTONCES PREGUNTA := "¿TIENE UN CUELLO LARGO?"
  
```

y así para los otros nodos. Un nodo de hoja podría generar una RESPUESTA en lugar de una pregunta. Los procedimientos apropiados también consultarían al usuario para obtener una entrada y podrían construir nuevos nodos si ésta es errónea.

A pesar de que las estructuras de decisión son importantes herramientas de clasificación, están limitadas porque no pueden tratar con variables, mientras que un sistema experto sí lo

puede hacer. Los sistemas expertos, más que simples clasificadores, son herramientas de propósito general.

### 3.3 ESTADO Y ESPACIOS DE PROBLEMA

Los diagramas pueden aplicarse a muchos problemas prácticos. Un método útil para describir el comportamiento de un objeto, es definir un gráfico llamada **espacio de estado**. Un *estado* es una colección de características que pueden usarse para definir el status o **estado** de un objeto. El espacio de estado es el conjunto de estados que muestran las **transiciones** entre los estados que puede experimentar el objeto. Una transición lleva un objeto de un estado a otro.

#### Ejemplos de espacios de estado

Como un simple ejemplo de espacios de estado, consideremos la compra de una bebida gaseosa en una máquina expendedora: cuando se depositan monedas en la máquina, ésta hace una transición de un estado a otro. En la figura 3.5 se ilustra el espacio de estado, suponiendo que sólo están disponibles monedas de 25 y de 5 centavos, y que se requieren 55 centavos para obtener una bebida. Si consideramos monedas de otras denominaciones, como de 10 y 50 centavos, se complica más la gráfica y no se muestra aquí.

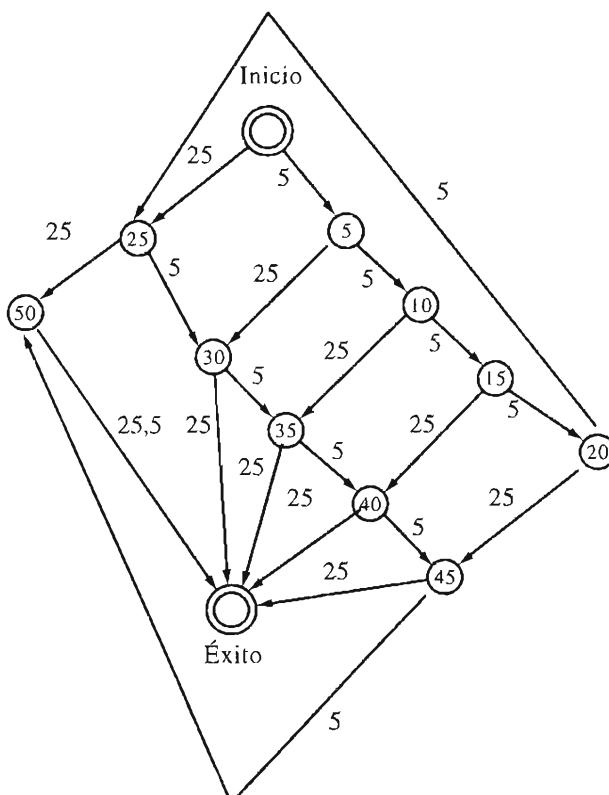


Figura 3.5

Gráfica de estado para una máquina expendedora de bebidas gaseosas que acepta monedas de 25 y 5 centavos

Los estados de inicio y éxito se dibujan como círculos dobles para facilitar su identificación. Los estados se muestran como círculos y las transiciones posibles a otros estados se muestran como flechas. Observe que esta gráfica es una gráfica con pesos, donde éstos son las monedas que pueden entrar a la máquina en cada estado.

A esta gráfica también se le llama gráfica de **máquina de estado finito**, porque describe el número finito de *estados* de una máquina. El término *máquina* se usa en un sentido muy general, puede ser un objeto real, un algoritmo, un concepto, etc. Asociadas con cada *estado* hay acciones que la conducen a otro. En cualquier momento, la máquina puede estar en un solo estado. A medida que acepta la entrada a un *estado*, pasa de ese estado a otro. Si se dan las entradas correctas, pasará del inicio al éxito o estado final. Si un estado no está diseñado para aceptar cierta entrada, la máquina se detendrá en él. Por ejemplo, la máquina de bebidas gaseosas no tiene la capacidad de aceptar monedas de diez centavos, si alguien pone una de estas monedas en la máquina, la respuesta no está definida. Un buen diseño incluirá la posibilidad de entradas no válidas para cada estado y proporcionará transiciones a un estado de error. Éste es diseñado para proporcionar mensajes de error apropiados y llevar a cabo cualquier acción necesaria.

Las máquinas de estado finito se usan con frecuencia en compiladores y otros programas para determinar la validez de una entrada. Por ejemplo, la figura 3.6 muestra parte de una máquina de estado finito para probar la validez de cadenas de entrada. Los caracteres de la entrada son examinados uno por uno y sólo serán aceptadas las cadenas de caracteres MIENTRAS, ESCRIBIR y EMPEZAR. Las flechas se muestran desde el estado EMPEZAR para una entrada exitosa y también para una entrada errónea; en este caso, avanza hacia el estado de error. Por eficiencia, algunos estados como el señalado por "L" y "T" se usan para probar a MIENTRAS y a ESCRIBIR.

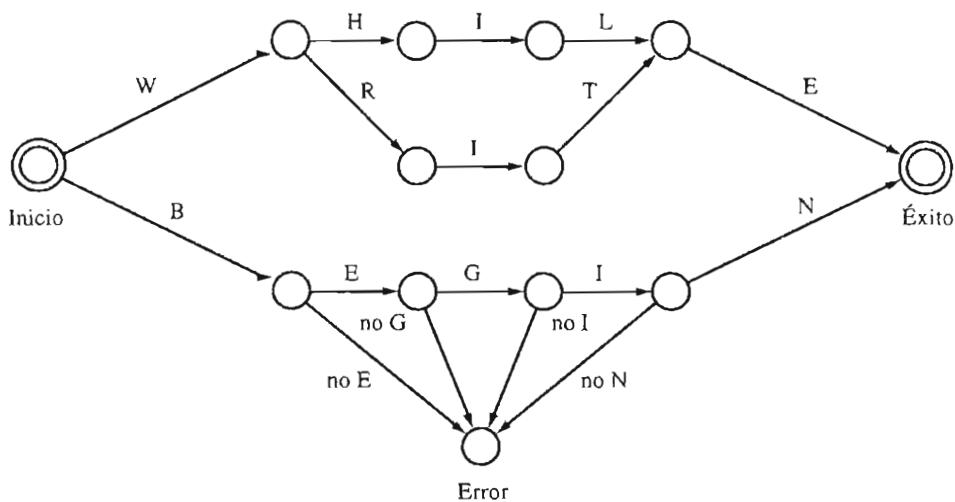


Figura 3.6 Parte de un máquima de estado finito para determinar cadenas válidas MIENTRAS, ESCRIBIR y EMPEZAR.

**Nota:** Sólo se muestran algunas transiciones al estado de error.

Los diagramas de estado también son útiles en la descripción de soluciones a problemas. En este tipo de aplicaciones se puede considerar al espacio de estado como un **espacio de problema**, en el que algunos estados corresponden a etapas intermedias de la solución de

problemas y algunos estados corresponden a las respuestas. En un espacio de problema puede haber varios estados de éxito que correspondan a soluciones posibles. Para encontrar la solución a un problema dentro de su espacio de problema, es necesario encontrar una ruta válida desde el inicio (afirmación del problema) hasta el éxito (respuesta). El árbol de decisión de animales puede verse como un espacio de problema donde las respuestas sí/no a las preguntas determinan el estado de transición.

Otro ejemplo de espacio de problema sucede en el problema clásico del mono y los plátanos que se muestra en la figura 3.7. El problema consiste en dar instrucciones a un mono para decirle cómo alcanzar algunos plátanos que cuelgan del techo. Como los frutos están fuera de su alcance, pero dentro del cuarto hay un sofá y una escalera. La configuración inicial típica es la del mono en el sofá. Las instrucciones pueden ser:

```

salta del sofá
ve a la escalera
pon la escalera bajo los plátanos
trepa la escalera
agarra los plátanos

```

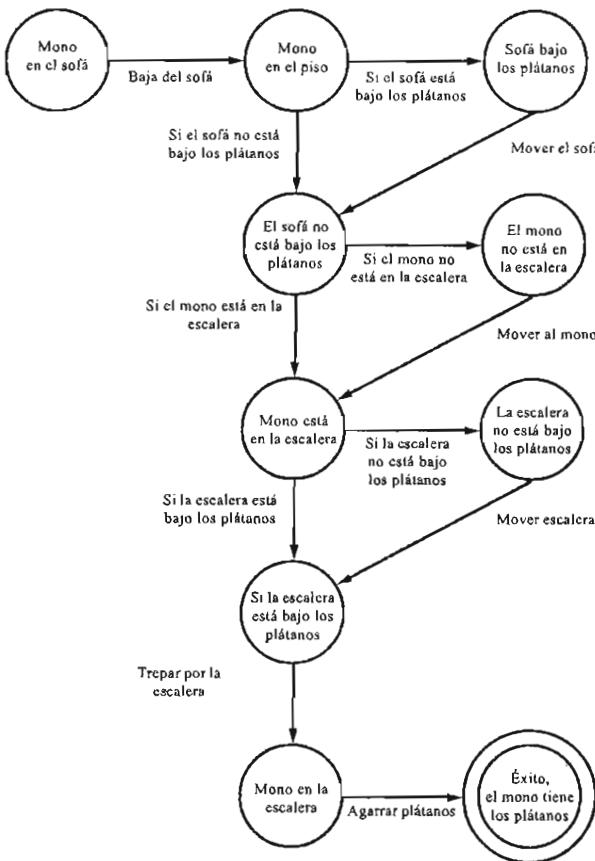
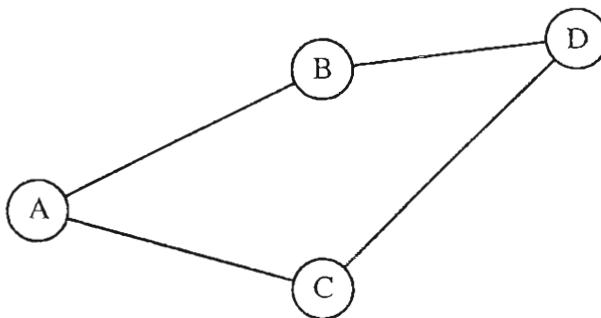


Figura 3.7 El espacio de estado para el problema del mono y los plátanos

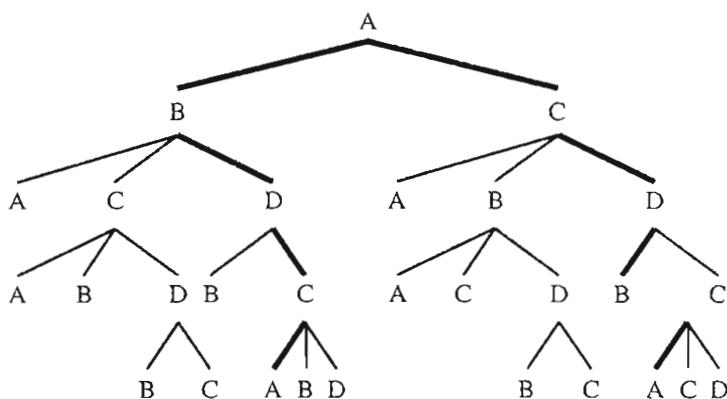
Las instrucciones variarán de acuerdo con las posiciones iniciales del mono, el sofá y la escalera. Como hay varios estados iniciales, no se muestra aquí el doble círculo especial de inicio. Por ejemplo, otro estado de inicio posible es con el mono en el sofá debajo de los plátanos, de esta manera, el mono tendrá entonces que hacer el sillón a un lado antes de poner la escalera debajo de los plátanos. En el estado inicial más simple, el mono ya se encuentra en la escalera debajo de las bananas.

Aunque este problema parece obvio para un ser humano, requiere una considerable cantidad de razonamiento. Una aplicación práctica de un sistema de razonamiento como éste, es la de dar a un robot instrucciones relacionadas con la realización de una tarea. En lugar de suponer que todos los objetos del ambiente se mantienen fijos en un lugar, una solución general es un sistema de razonamiento que puede tratar con diversas situaciones.

Otra aplicación útil de los diagramas es la exploración de rutas para encontrar la solución a un problema. En la figura 3.8(a) se muestra una red simple para el problema del agente viajero; en este ejemplo, se supone que el problema está en encontrar una ruta completa para visitar todos los nodos desde el nodo A. Como es común en estos casos, ningún nodo puede visitarse dos veces. En la figura 3.8(b) se muestran todas las rutas posibles que empiezan desde el nodo A, en forma de un árbol. Las rutas correctas, ABDCA y ACDBA, se muestran con bordes gruesos en esta gráfica.



(a) Gráfica de un problema del agente viajero



(b) Rutas de búsqueda (las rutas óptimas se muestran con bordes gruesos)

**Figura 3.8 Problema del vendedor viajero**

Dependiendo del algoritmo de búsqueda, es probable que la exploración de rutas para encontrar la correcta requiera una cantidad considerable de rastreo hacia atrás. Por ejemplo, puede investigarse primero la ruta ABA sin éxito y después rastrear hacia atrás hasta B. Desde B se buscará sin éxito en las rutas CA, CB, CDB y CDC. A continuación se buscará sin éxito en la ruta BDB hasta que se encuentre la primera ruta correcta, ABDCA.

### Espacios de problemas con estructura nociva

Una aplicación útil de los espacios de estado se encuentra al distinguir los problemas con estructura nociva (Pople 82). En el capítulo 1 se definió que un problema con estructura nociva es aquel asociado con la incertidumbre. Ésta puede especificarse con mayor precisión usando un espacio de problema.

Como ejemplo de un problema con estructura nociva, consideremos otra vez el caso de una persona que piensa viajar y visita a un agente de viajes, que se analizó en el capítulo 1. En la tabla 3.1 se enumeran algunas características de este problema con estructura nociva como un espacio de problema, indicado por las respuestas de la persona a las preguntas del agente de viajes.

Comparando la tabla 3.1 con la 1.10 (página 19), se verá que el concepto de un espacio de problema permite especificar con mayor precisión las características de un problema con estructura nociva. Es esencial diferenciar estos parámetros con precisión para determinar si una solución es factible y, si lo es, qué es lo que requiere. Un problema no tiene necesariamente una estructura nociva sólo por presentar una, varias o incluso todas estas características, porque depende mucho del rigor. Por ejemplo, todos los problemas de comprobación de teoremas tienen un número infinito de posibles soluciones, pero esto no hace que la comprobación de teoremas sea un problema de estructura nociva.

Como puede verse en la tabla 3.1, existen muchas incertidumbres, y aún así los agentes de viajes tratan con ellas todos los días. Aunque no todos los casos pueden ser tan malos como éste, indica por qué sería muy difícil una solución algorítmica.

Característica	Respuesta
Objetivo no explícito	Estoy pensando en ir a <u>alguna</u> parte
Espacio de problema no delimitado	No estoy seguro de <u>adónde</u> ir
Estados de problema no discretos	Solo quiero viajar; el destino no es importante
Estados intermedios difíciles de alcanzar	No tengo suficiente dinero para ir
Operadores de estado desconocidos	No sé cómo conseguir el dinero
Restricciones de tiempo	Debo ir pronto

Tabla 3.1 Ejemplo de un problema con estructura nociva para un viaje

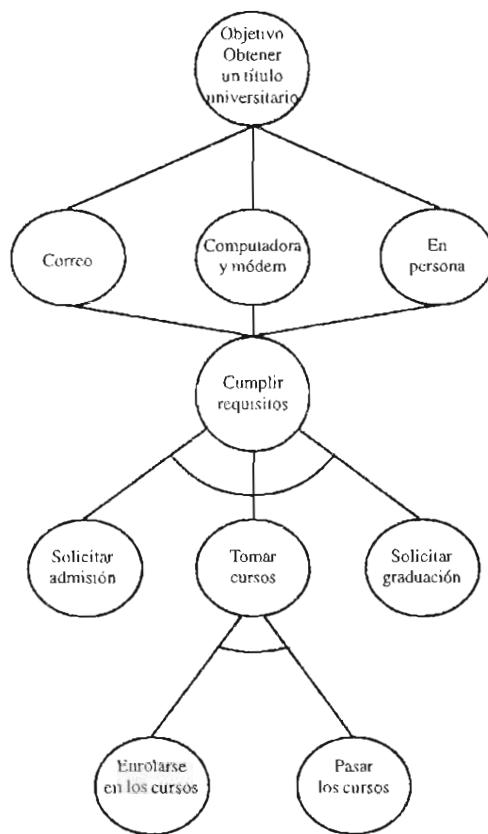
Un problema bien formado es en el que conocemos el problema explícito, el objetivo y los operadores que pueden aplicarse para ir de un estado al otro; y se le considera **determinístico** porque al aplicar un operador a un estado, estamos seguros del estado siguiente, el espacio de problema está delimitado y los estados son discretos. Esto significa que hay un número finito de estados y que cada uno está bien definido.

En el problema del viaje los estados no están delimitados porque los destinos posibles son infinitos. Una situación análoga puede suceder con un medidor analógico, que indica un número infinito de lecturas posibles. Si consideramos que cada lectura del medidor es un estado, entonces hay un número infinito de estados y éstos no están bien definidos porque corresponden a los números reales. Como hay un número infinito de números reales entre dos números reales cualesquiera, los estados no son discretos porque el siguiente estado sólo difiere infinitesimalmente. Por el contrario, las lecturas de un medidor digital están delimitadas y son discretas.

### 3.4 ÁRBOLES Y OBJETIVOS Y-O

Muchos tipos de sistemas expertos usan el encadenamiento hacia atrás para encontrar soluciones. PROLOG es un buen ejemplo de un sistema de encadenamiento hacia atrás que trata de resolver un problema dividiéndolo en subproblemas más pequeños y resolviéndolos individualmente. Por su parte, los optimistas consideran la solución de un problema como una meta que debe alcanzarse. Con el propósito de alcanzar una meta, deben ejecutarse cero o más submetas.

Un tipo de árbol o rejilla que es útil en la representación de problemas de encadenamiento hacia atrás, es el árbol Y-O. En la figura 3.9 se muestra un ejemplo sencillo de una rejilla Y-O para alcanzar la meta de obtener un título universitario. Para cumplir este objetivo, puede asistir en persona a una universidad o tomar cursos por correspondencia; con éstos el trabajo puede realizarse ya sea con tarjetas por correo o vía electrónica, usando una computadora casera y un módem.



**Figura 3.9 Rejilla Y-O que muestra cómo obtener un título universitario**

Con el fin de satisfacer los requisitos para el título, deben completarse tres subobjetivos: 1) solicitar la admisión, 2) tomar los cursos y 3) solicitar la graduación. Observe que bajo el círculo de satisfacer los requisitos, hay un arco que une los bordes para unir estos tres subobjetivos. El arco indica que satisfacer los requisitos es un nodo Y que sólo puede satisfa-

cerse si se satisfacen sus tres subobjetivos. Los objetivos sin el arco, como computadora y módem, correo, y en persona, son nodos O en los que la satisfacción de cualquiera de estos subobjetivos satisface el objetivo padre de obtener un título universitario.

Esta gráfica es una rejilla porque el subobjetivo de satisfacer los requisitos tiene tres nodos padre: 1) correo, 2) computadora y módem y 3) en persona. Observe que sería posible dibujar esta gráfica como un árbol con la simple duplicación del subobjetivo de satisfacer requisitos y de su subárbol de objetivos para los objetivos correo, computadora y módem, y en persona. Sin embargo, como el subobjetivo Satisfacer requisitos es el mismo para cada padre, no hay una ventaja real y sólo se usa más papel para dibujar el árbol.

Como otro ejemplo sencillo, en la figura 3.10 se muestra un árbol Y-O para el problema de llegar al trabajo por diferentes medios. Para hacerlo más completo, podría convertirse en una rejilla; por ejemplo, podría añadirse un borde desde el nodo manejar a la estación del tren hasta el nodo automóvil y otro desde caminar a la estación del tren hasta el nodo caminar. En la figura 3.11 se muestra una rejilla exclusiva del tipo Y-O.

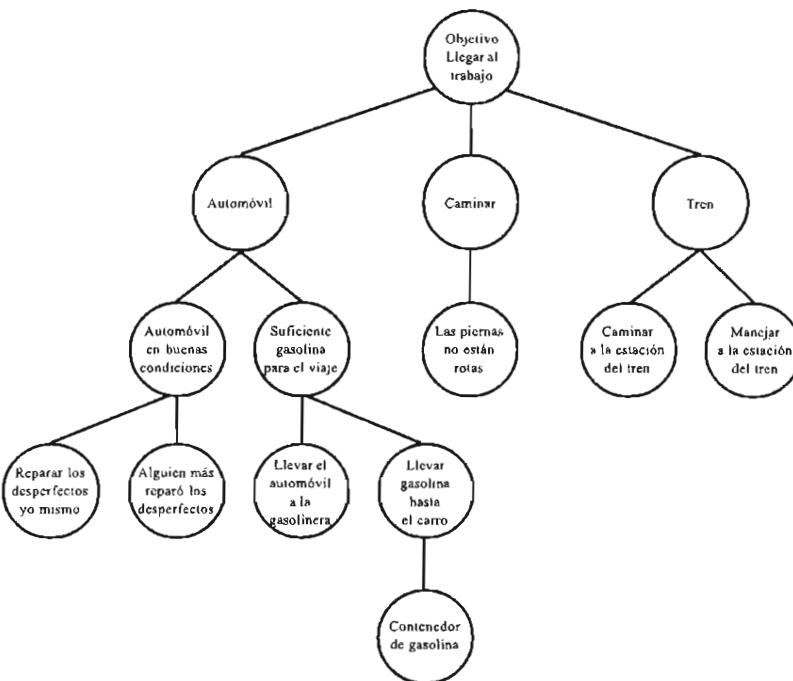


Figura 3.10 Árbol Y-O simple que muestra métodos para llegar al trabajo.

Otra forma de describir las soluciones a un problema es la rejilla Y-O-NO, que usa símbolos de compuerta lógica en lugar de la notación de los árboles Y-O. En la figura 3.12 se muestran los símbolos de entradas lógicas para Y, O y NO. Estas compuertas implantan las tablas de verdad para Y, O y NO que se analizaron en el capítulo 2. En la figura 3.13 se muestra la figura 3.9 implantada con entradas Y y O.

Los árboles Y-O y los árboles de decisión tienen las mismas ventajas y desventajas básicas. La ventaja principal de las rejillas Y-O-NO es su posible implantación en hardware para altas velocidades de procesamiento, ya que pueden diseñarse a la medida para su fabricación como circuitos integrados. En la práctica, un tipo de entrada lógica como la NO-Y se usa por

razones de economía en la fabricación más que para separar compuertas Y, O y NO. A partir de la lógica, puede probarse que es posible implantar cualquier función lógica mediante una compuerta NO-Y. Un circuito integrado con un sólo tipo de dispositivo es más barato de fabricar que uno con varios tipos de compuerta lógica.

Un chip que utiliza el encadenamiento hacia delante puede calcular rápidamente la respuesta como una función de sus entradas porque el procesamiento se desarrolla en paralelo. Chips como éstos pueden utilizarse para la supervisión de los datos en tiempo real de sensores y genera una respuesta apropiada, dependiendo de las entradas. La principal desventaja es que, como otras estructuras de decisión, un chip diseñado para lógica no puede manejar situaciones para las que no fue diseñado. Sin embargo, un ANS implantado en un chip puede manejar entradas inesperadas.

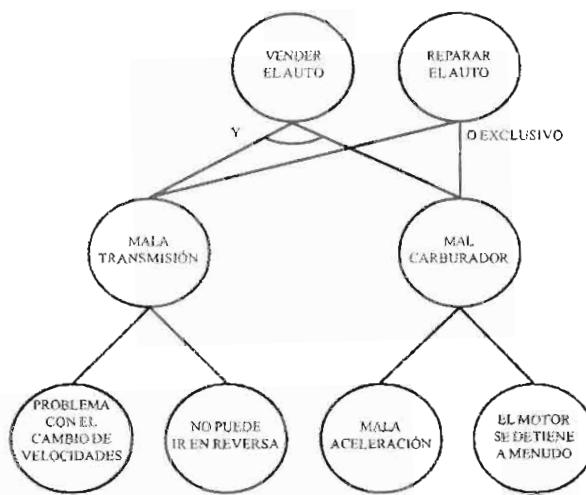


Figura 3.11 Rejilla Y-O para decidir si vender o reparar un automóvil

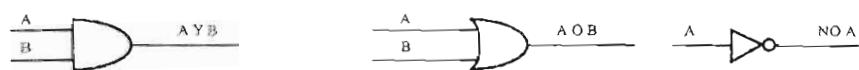


Figura 3.12 Símbolos de compuerta lógica Y, O y NO



Figura 3.13 Representación de la compuerta lógica Y-O para la figura 3.9

## 3.5 LÓGICA DEDUCTIVA Y SILOGISMOS

En el capítulo 2 analizamos la representación del conocimiento por medio de la lógica. Ahora se verá cómo hacer inferencias para generar nuevos conocimientos o información. En el resto de este capítulo se explican diferentes métodos de inferencia. La figura 3.14 muestra un panorama general de los métodos de inferencia, que a continuación se resumen:

- *Deducción*. Razonamiento lógico en el que las conclusiones deben desprenderse de sus premisas.
- *Inducción*. Inferencia pasando del caso específico al general.
- *Intuición*. Teoría no comprobada. La respuesta simplemente aparece, tal vez por reconocimiento inconsciente de un patrón subyacente. Los sistemas expertos no realizan todavía este tipo de inferencia. Los ANS permanecerían como una promesa para este tipo de inferencia porque pueden extraer su enseñanza en lugar de simplemente proporcionar una respuesta condicionada o una interpolación. Es decir, una red neuronal siempre ofrecerá su mejor suposición como respuesta.
- *Heurística*. Reglas empíricas basadas en la experiencia.
- *Ensayo y error*. A menudo se utiliza con la planeación por razones de eficiencia.
- *Abducción*. Razonamiento hacia atrás desde una conclusión verdadera hacia las premisas que habrían causado la conclusión.
- *Predeterminación*. En la ausencia de conocimiento específico, adopta conocimiento general o común como opción predeterminada.
- *Autoepistemología*. Conocimiento de sí mismo.
- *No monótono*. El conocimiento previo podría ser incorrecto cuando se obtiene nueva evidencia.
- *Analogía*. Inferencia de una conclusión con base en las similitudes con otra situación.

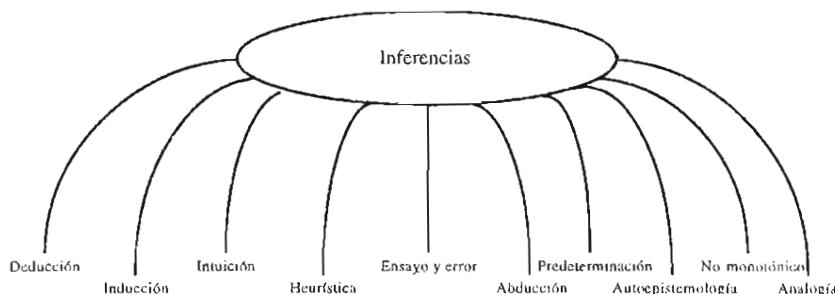


Figura 3.14 Tipos de inferencia

Aunque no se muestra explícitamente en la figura 3.14, el **conocimiento por sentido común** puede ser una combinación de cualquiera de estos tipos. El razonamiento del sentido común es el que las personas utilizan en situaciones ordinarias y es muy difícil que las computadoras lo dominen. La aplicación de la lógica de confusión para razonamientos de este tipo se analiza en el capítulo 5.

Uno de los métodos más comunes para formular inferencias es la **lógica deductiva**, que se ha utilizado desde tiempos remotos para determinar la validez de un **argumento**. Un argu-

mento lógico es un grupo de afirmaciones en el que la última de ellas está justificada con base en las anteriores de la **cadena de razonamiento**. Un tipo de argumento lógico es el silogismo que abordamos brevemente en el capítulo 2. Como ejemplo de un silogismo:

Premisa: Cualquiera que pueda programar es inteligente

Premisa: John puede programar

Conclusión: Por lo tanto, John es inteligente

En un argumento, las premisas se utilizan como evidencia para respaldar las conclusiones. También se les llama **antecedentes**, y la conclusión es la **consecuencia**. La característica esencial de la lógica deductiva es que la conclusión verdadera *debe* obtenerse de premisas verdaderas. Se traza una línea para separar las premisas de la conclusión, como se muestra en el ejemplo precedente, de modo que no es necesario rotular explícitamente las premisas y la conclusión.

El argumento podría haberse escrito de forma más breve como:

Cualquiera que pueda programar es inteligente

John puede programar

∴ John es inteligente

donde los tres puntos, ∴, significan “por tanto”.

Ahora veamos más de cerca a la lógica silogística. La principal ventaja del estudio de silogismos es que se trata de una simple y bien entendida rama de la lógica que puede probarse por completo. Además, a menudo los silogismos son útiles porque se pueden expresar a partir de reglas SI...ENTONCES. Por ejemplo, el silogismo anterior se puede recordar como:

SI            cualquiera que pueda programar es inteligente y

              John puede programar

ENTONCES John es inteligente

En general, un silogismo es cualquier argumento deductivo válido que cuenta con dos premisas y una conclusión. El silogismo clásico es un tipo especial llamado **silogismo categórico**; las premisas y conclusiones se definen como afirmaciones categóricas de las siguientes cuatro formas, como se muestra en la tabla 3.2.

Forma	Esquema	Significado
A	Toda S es P	afirmativo universal
E	No S es P	negativo universal
I	Alguna S es P	afirmativo particular
O	Alguna S no es P	negativo particular

Tabla 3.2 Afirmaciones categóricas

Observe que, en la lógica, el término *esquema* especifica la forma lógica de la afirmación. Esto ilustra también otro uso de la palabra, que es diferente de su utilización en AI, analizado en el capítulo 2. En la lógica, la palabra *esquema* se usa para mostrar la forma esencial de un argumento. También podría especificar la forma lógica de un silogismo entero como en:

Toda M es P  
Toda S es M  
∴ Toda S es P

Al sujeto de la conclusión, S, se le llama **término menor**, mientras que al predicado de la conclusión, P, se le llama **término mayor**. A la premisa que tiene el término mayor se le llama **premisa mayor** y a la que tiene el término menor se le llama **premisa menor**. Por ejemplo

Premisa mayor: Toda M es P  
Premisa menor: Toda S es M  
Conclusión: Toda S es P

es un silogismo que se dice que se encuentra en **forma normal** con sus premisas mayor y menor identificadas. El **sujeto** es el objeto que se describe y el **predicado** describe algunas propiedades del sujeto. Por ejemplo, en la afirmación

Todas las microcomputadoras son computadoras  
el sujeto es “microcomputadoras” y el predicado es “computadoras”. En la afirmación:

Todas las microcomputadoras con 8 megabytes  
son computadoras con gran memoria

el sujeto es “microcomputadoras con 8 megabytes” y el predicado es “computadoras con gran memoria”.

Las formas de las afirmaciones categóricas se han identificado desde la antigüedad con las letras A, E, I y O. La A y la I indican afirmación y vienen de las dos primeras vocales de la palabra latina “affirmo”; mientras que E y O vienen de “nego”. Se dice que las formas A e I son **afirmativas en calidad** al afirmar que los sujetos se incluyen en la clase del predicado. La E y O son **negativas en calidad** porque excluyen los sujetos de la clase del predicado.

A los verbos *ser* o *estar* se les llama **cópula**, que significa “conectar” en latín. La cópula conecta las dos partes de la afirmación. En el silogismo categórico normal, es la forma del tiempo presente del verbo *ser* o *estar*. Así que otra versión es:

Todas las S son P

Al tercer término del silogismo, M, se le llama **término medio** y es común para ambas premisas. Éste es esencial porque un silogismo se define de modo tal que la conclusión no puede deducirse únicamente de una u otra de las premisas. De modo que el argumento:

Toda A es B  
Toda B es C  
∴ Toda A es B

no es un silogismo válido porque se desprende únicamente de la primera premisa.

La **cantidad** o **cuantificador** describe la porción de la clase incluida. Los cuantificadores *Todo* y *No* son **universales** porque se refieren a clases enteras; al cuantificador *Algunos* se le llama **particular** porque sólo alude a una parte de la clase.

El **modo** de un silogismo está definido por las tres letras que dan forma a la premisa mayor, la premisa menor y la conclusión, respectivamente. Por ejemplo, el silogismo

Toda M es P  
Toda S es M  
 ∴ Toda S es P

es un modo AAA.

Hay cuatro patrones posibles para ordenar los términos S, P y M, como se muestra en la tabla 3.3. A cada patrón se le llama **figura**, y el número de la figura especifica su tipo

	Figura 1	Figura 2	Figura 3	Figura 4
Premisa mayor	M P	P M	M P	P M
Premisa menor	S M	S M	M S	M S

Tabla 3.3 Patrones de afirmaciones categóricas

De modo que el ejemplo anterior se describe totalmente como un tipo de silogismo AAA-1. El solo hecho de que un argumento tenga forma silogística no significa que sea válido. Considerando la forma del silogismo AEE-1:

Toda M es P  
No S es M  
 ∴ No S es P

no es un silogismo válido, como puede verse en el ejemplo:

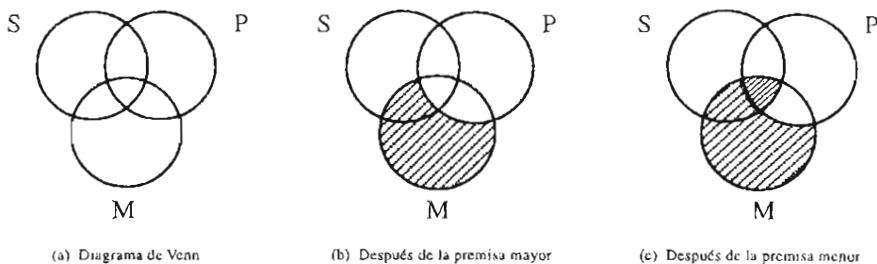
Todas las microcomputadoras son computadoras  
Ningún mainframe es una microcomputadora  
 ∴ Ningún mainframe es una computadora

En lugar de inventar ejemplos para probar la validez de argumentos silogísticos, puede utilizarse un **procedimiento de decisión**. Se trata de un método para probar la validez y es un método mecánico o algoritmo un poco general, por medio del cual puede automatizarse el proceso de determinación de la validez. Aunque hay procedimientos de decisión para la lógica silogística y la lógica de proposición, Church mostró en 1936 que no hay ninguno para la lógica de predicado. En cambio, las personas deben aplicar la creatividad para generar pruebas.

El procedimiento de decisión para las proposiciones consiste simplemente en construir una tabla de verdad y revisarla en busca de tautologías. El procedimiento de decisión para los silogismos puede darse utilizando diagramas de Venn con tres círculos superpuestos que representen S, P y M, como se muestra en la figura 3.15(a). Para la forma del silogismo AEE-1:

Toda M es P  
Ninguna S es M  
 ∴ Ninguna S es P

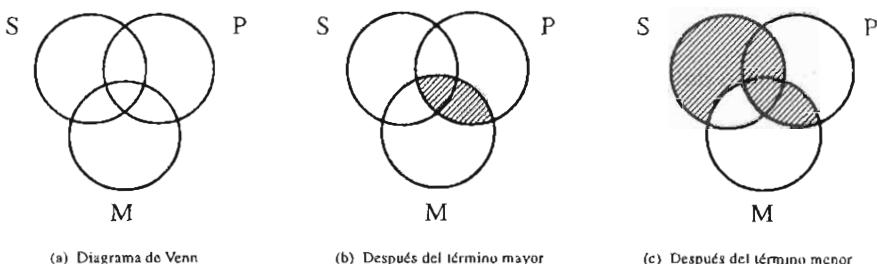
la premisa mayor se ilustra en la figura 3.15(b). La sección de M sombreada con líneas indica que no hay elementos en esa porción. En (c) se incluye la premisa menor, trazando líneas en su porción sin elementos. En (c) puede verse que la conclusión de AEE-1 es falsa porque hay algunas S en P.



**Figura 3.15 Procedimiento de decisión para el silogismo AEE-1**

En otro ejemplo, el siguiente EAE-1 es válido, como puede observarse en la figura 3.16(c):

Ninguna M es P  
Toda S es M  
 $\therefore$  Ninguna S es P



**Figura 3.16 Procedimiento de decisión para el silogismo EAE-1**

Los diagramas de Venn que incluyen “algunos” cuantificadores son un poco más difíciles de trazar. Las reglas generales para dibujar silogismos categóricos bajo la concepción de Boole de que tal vez no haya miembros en las afirmaciones A y E son:

1. Si una clase está vacía, se sombra.
2. Las afirmaciones universales, A y E, siempre se dibujan antes que las particulares.
3. Si una clase tiene por lo menos un miembro, márquela con un asterisco.
4. Si una afirmación no especifica en cuál de las dos clases adyacentes existe un objeto, coloque un asterisco en la línea entre las clases.
5. Si un área ha sido sombreada, no puede introducirse un asterisco en ella.

Por ejemplo:

Algunas computadoras son laptops  
Todas las laptops son portátiles  
 $\therefore$  Algunas portátiles son computadoras

que se puede poner en un tipo IAI-4 como

Algunas P son M  
Todas las M son S  
 $\therefore$  Algunas S son P

Siguiendo las reglas 2 y 1 de los diagramas de Venn, iniciamos con la afirmación universal para la premisa menor y la sombreamos, como lo muestra la figura 3.17(a). A continuación, se aplica la regla 3 a la premisa mayor en particular y se dibuja un \*, como se muestra en la figura 3.17(b). Como en el diagrama se muestra la conclusión "Algunas portátiles son computadoras", se deriva que el argumento IAI-4 es un silogismo válido.

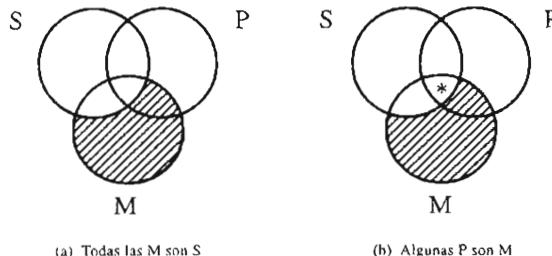


Figura 3.17 Silogismo del tipo IAI-4

### 3.6 REGLAS DE INFERENCIA

Aunque los diagramas de Venn son un procedimiento de decisión para silogismos, son inconvenientes para argumentos más complejos, porque se vuelven más difíciles de leer. Sin embargo, hay un problema más importante con los silogismos, debido a que sólo atienden una pequeña parte de las afirmaciones lógicas posibles. En particular, los silogismos categóricos sólo atienden afirmaciones categóricas de la forma A, E, I y O.

La lógica proposicional ofrece otros mecanismos para describir argumentos; en efecto, a menudo la utilizamos sin darnos cuenta. Por ejemplo, considerando el siguiente argumento de proposición:

Si hay energía eléctrica, la computadora trabajará  
Hay energía eléctrica  
 $\therefore$  La computadora trabajará

Este argumento puede expresarse de manera formal utilizando letras para representar las proposiciones, como sigue:

A = Hay energía eléctrica  
 B = La computadora trabajará

de modo que el argumento puede escribirse como

$$\begin{array}{c} A \rightarrow B \\ \hline A \\ \therefore B \end{array}$$

Argumentos como éste se dan a menudo; un esquema general para la representación de argumentos de este tipo es:

$$\begin{array}{c} p \rightarrow q \\ \hline p \\ \therefore q \end{array}$$

donde p y q son variables lógicas que pueden representar cualquier afirmación. El uso de variables lógicas en la lógica proposicional permite afirmaciones más complejas que las de las cuatro formas silogísticas A, E, I y O. Al esquema de inferencia de esta forma proposicional se le llama con diversos nombres: **razonamiento directo, modus ponens, ley de desprendimiento y suposición del antecedente**.

Observe que este ejemplo también puede expresarse en la forma silogística:

Todas las computadoras con energía eléctrica trabajarán  
Esta computadora tiene energía eléctrica  
Esta computadora trabajará

que demuestra que el *modus ponens* es realmente un caso especial de lógica silogística. El *modus ponens* es importante porque forma la base de los sistemas expertos basados en reglas. La proposición compuesta  $p \rightarrow q$  corresponde a la regla y la p corresponde al patrón que debe marcar el antecedente de la regla que habrá de satisfacerse. Sin embargo, como se analiza en el capítulo 2, el condicional  $p \rightarrow q$  no es equivalente exacto de una regla, porque el condicional es una definición lógica definida por una tabla de verdad y hay muchas definiciones posibles para él.

En este texto se utilizará la convención de la teoría lógica de utilizar mayúsculas como A, B, C ... para representar proposiciones constantes como "Hay energía eléctrica". Las minúsculas como p, q, r ..., representarán variables lógicas que pueden corresponder a diferentes proposiciones constantes. Observe que esta convención es opuesta a la de PROLOG, que utiliza mayúsculas para las variables.

Este esquema de *modus ponens* pudo haberse escrito con variables lógicas denominadas de otra manera como:

$$\begin{array}{c} r \rightarrow s \\ \hline r \\ \therefore s \end{array}$$

y el esquema aún significaría lo mismo.

Otra notación para el esquema es:

$$r, r \rightarrow s; \therefore q$$

donde la coma se utiliza para separar una premisa de otra y el punto y coma indica el final de las premisas. Aunque hasta ahora sólo hemos visto argumentos con dos premisas, una forma más general de un argumento es:

$$P_1, P_2, \dots, P_N; \therefore C$$

donde las mayúsculas  $P_i$  representan premisas como  $r$ ,  $r \rightarrow s$ , y  $C$  es la conclusión. Observe cómo se parece a la instrucción de cumplimiento de objetivos de PROLOG, analizado en el capítulo 2.

$$p :- p_1, p_2, \dots, p_N.$$

El objetivo,  $p$ , se satisface si se satisfacen todos los subobjetivos  $p_1, p_2, \dots, p_N$ . Un argumento análogo para reglas de producción puede escribirse en la forma general:

$$C_1 \wedge C_2 \wedge \dots \wedge C_N \rightarrow A$$

lo que significa que si se satisface cada condición,  $C_i$ , de una regla, entonces se realiza la acción,  $A$ , de la regla. Como se analizó previamente, una afirmación lógica de la forma anterior no es estrictamente equivalente a una regla, porque la definición lógica del condicional no es la misma que una regla de producción. Sin embargo, esta forma lógica es una ayuda intuitiva útil para razonar acerca de las reglas.

La notación de los operadores lógicos Y y O toma diferentes formas en PROLOG, en comparación con los usuales  $\wedge$  y  $\vee$ . La coma entre subobjetivos en PROLOG significa una conjunción,  $\wedge$ , y la disyunción,  $\vee$ , se indica con un punto y coma. Por ejemplo:

$$p :- p_1; p_2.$$

significa que  $p$  se satisface si  $p_1$  o  $p_2$  se satisfacen. Las conjunciones y las disyunciones pueden combinarse. Por ejemplo:

$$p :- p_1, p_2; p_3, p_4.$$

es lo mismo que las dos afirmaciones de PROLOG:

$$\begin{aligned} p &:- p_1, p_2. \\ p &:- p_3, p_4. \end{aligned}$$

En general, si las premisas y la conclusión son esquemas, el argumento:

$$P_1, P_2, \dots, P_N; \therefore C$$

es un argumento deductivo formalmente válido si y sólo si:

$$P_1 \wedge P_2 \wedge \dots \wedge P_N \rightarrow C$$

es una tautología. Como ejemplo:

$$(p \wedge q) \rightarrow p$$

es una tautología porque es verdadero para cualquier valor, V o F, de p y q. Esto se puede verificar construyendo la tabla de verdad.

El argumento de *modus ponens*:

$$\begin{array}{c} p \rightarrow q \\ \hline \therefore q \end{array}$$

es válido porque puede expresarse como tautología.

$$(p \rightarrow q) \wedge p \rightarrow q$$

Observe que se está suponiendo que la flecha tiene precedencia inferior que la conjunción y la disyunción. Esto ahorra la escritura de paréntesis adicionales como:

$$((p \rightarrow q) \wedge p) \rightarrow q$$

En la tabla 3.4 se muestra la tabla de verdad para el *modus ponens*. Es una tautología porque todos los valores del argumento, mostrados en la columna de la derecha, son verdaderos sin importar los valores de sus premisas. Observe que en la tercera, cuarta y quinta columnas, los valores de verdad se escriben bajo ciertos operadores, como  $\rightarrow$  y  $\wedge$ . Se les llama **conectores principales** porque conectan las dos partes principales de una proposición compuesta.

p	q	$p \rightarrow q$	$(p \rightarrow q) \wedge p$	$(p \rightarrow q) \wedge p \rightarrow q$
V	V	V	V	V
V	F	F	F	V
F	V	V	F	V
F	F	V	F	V

Tabla 3.4 Tabla de verdad para *modus ponens*

Aunque este método funciona para determinar argumentos válidos, requiere que se verifique cada fila de la tabla de verdad. El número de filas es  $2^N$ , donde N es el número de premisas, de modo que las filas se incrementarán rápidamente. Por ejemplo, para cinco premisas se requerirá verificar 32 filas, mientras que para 10 premisas se verificarían 1024 filas. Un método más corto para determinar un argumento válido, es considerar únicamente aquellas filas de la tabla de verdad en las que todas las premisas son verdaderas; la definición equivalente de un argumento válido establece que es válido si y sólo si la conclusión es verdadera para cada una de estas filas. Es decir, la conclusión está tautológicamente implícita en las premisas. Para el *modus ponens*, la premisa  $p \rightarrow q$  y la premisa p sólo son verdaderas en la primera fila, y también en la conclusión, por tanto, el *modus ponens* es un argumento válido. Si hubiera alguna otra fila en la que todas las premisas fueran verdaderas y la conclusión falsa, entonces el argumento no sería válido.

En la tabla 3.5 se muestra la manera más corta de expresar la tabla de verdad para el *modus ponens*, donde todas las filas se muestran explícitamente. En la práctica, sólo necesitan considerarse aquellas filas que tienen premisas verdaderas, como la primera.

La tabla de verdad para el *modus ponens* muestra que es válida porque la primera fila tiene premisas verdaderas y una conclusión verdadera, y no hay otras filas que tengan premisas verdaderas y una conclusión falsa.

Premisas			Conclusión	
p	q	$p \rightarrow q$	p	q
V	V	V	V	V
V	F	F	V	F
F	V	V	F	V
F	F	V	F	F

Tabla 3.5 Forma corta de una tabla de verdad para el *modus ponens*.

Los argumentos pueden ser engañosos. Para demostrar esto, consideremos primero el siguiente ejemplo válido de *modus ponens*:

Si no hay errores, entonces el programa se cumple  
No hay errores  
 $\therefore$  El programa se cumple

Comparando esto con el siguiente argumento que, de alguna manera, semeja al del *modus ponens*

Si no hay errores, entonces el programa se cumple  
El programa se cumple  
 $\therefore$  No hay errores

¿Se trata de un argumento válido? El esquema para argumentos de este tipo es:

$$\begin{array}{c} p \rightarrow q \\ q \\ \hline \therefore p \end{array}$$

y su tabla de verdad en forma corta se muestra en la tabla 3.6.

Premisas			Conclusión	
p	q	$p \rightarrow q$	q	p
V	V	V	V	V
V	F	F	F	V
F	V	V	V	F
F	F	V	F	F

Tabla 3.6 Forma corta de la tabla de verdad para  $p \rightarrow q, q; \therefore p$ 

Observe que este argumento no es válido. Aunque la primera fila muestra que la conclusión es verdadera si todas las premisas lo son, la tercera fila muestra que si la premisa es verdadera, la conclusión es falsa. Por tanto, este argumento falla para el criterio si y sólo si de un argumento válido. Aunque muchos programadores desearían que argumentos como éstos fueran verdaderos, la lógica (y la experiencia) prueban que es una **falacia**, o argumento no válido. A este argumento en particular se le llama *falacia del recíproco*.

Tomando otro ejemplo, el esquema de argumento:

$$\begin{array}{c} p \rightarrow q \\ \neg q \\ \hline \therefore \neg p \end{array}$$

es la razón por la que la tabla 3.7 muestra que la conclusión es verdadera sólo cuando las premisas son verdaderas.

Premisas			Conclusión	
<i>p</i>	<i>q</i>	<i>p → q</i>	<i>q</i>	$\neg p$
V	V	V	F	F
V	F	F	V	F
F	V	V	F	V
F	F	V	V	V

Tabla 3.7 Forma corta de la tabla de verdad para  $p \rightarrow q, \neg q; \therefore \neg p$

A este esquema en particular se le llama de diversas maneras: *razonamiento indirecto*, *modus tollens* y *ley de la contraposición*.

*Modus ponens* y *modus tollens* son **reglas de inferencia**, a veces llamadas *leyes de inferencia*. En la tabla 3.8 se muestran algunas de ellas.

La palabra *en modus* significan “modo” en latín, mientras que *ponere* significa “asertar” y *tollere* significa “negar”. Los nombres reales de las reglas del *modus* y sus significados literales se muestran en la tabla 3.9. *Modus ponens* y *modus tollens* son abreviaturas de los dos primeros tipos (Stebbing 50). Los números de la regla de inferencia corresponden a los de la tabla 3.8.

Las reglas de inferencia pueden aplicarse a argumentos con más de dos premisas. Por ejemplo, considerando el siguiente argumento:

Los precios bajos suben sólo si el yen sube  
 El yen sube sólo si el dólar baja y  
 si el dólar baja entonces el yen sube.  
 Como los precios bajos han subido,  
 el dólar debe haber bajado.

Definamos las proposiciones como sigue:

$$\begin{aligned} B &= \text{precios bajos suben} \\ Y &= \text{yen sube} \\ D &= \text{dólar baja} \end{aligned}$$

Si se recuerda de la sección 2.12 que uno de los significados del condicional es “*p, sólo si q*”. Una proposición como “El yen sube sólo si el dólar baja” tiene este significado y se representa como  $B \rightarrow Y$ . Entonces todo el argumento tiene la siguiente forma.

$$\begin{array}{c} B \rightarrow Y \\ (Y \rightarrow D) \wedge (D \rightarrow Y) \\ \hline \therefore D \end{array}$$

La segunda premisa tiene una forma interesante, que puede reducirse aún más si se utiliza una variante del condicional. El condicional  $p \rightarrow q$  tiene diversas variantes, que son la **recíproca**, la **inversa** y la **contrapositiva**. Se presentan en la lista con el condicional, para completar este aspecto, en la tabla 3.10.

Ley de inferencia	Esquema
1. Ley del desprendimiento	$\begin{array}{c} p \rightarrow q \\ \hline \therefore q \end{array}$
2. Ley de la contrapositiva	$\begin{array}{c} p \rightarrow q \\ \hline \therefore \neg q \rightarrow \neg p \end{array}$
3. Ley de <i>modus tollens</i>	$\begin{array}{c} p \rightarrow q \\ \neg q \\ \hline \therefore \neg p \end{array}$
4. Regla de cadena (Ley del silogismo)	$\begin{array}{c} p \rightarrow q \\ q \rightarrow r \\ \hline \therefore p \rightarrow r \end{array}$
5. Ley de la inferencia disyuntiva	$\begin{array}{c} p \vee q \\ \neg p \\ \hline \therefore q \end{array} \qquad \begin{array}{c} p \vee q \\ \neg q \\ \hline \therefore p \end{array}$
6. Ley de la doble negación	$\begin{array}{c} \neg(\neg p) \\ \hline \therefore p \end{array}$
7. Ley de De Morgan	$\begin{array}{c} \neg(p \wedge q) \\ \hline \therefore \neg p \vee \neg q \end{array} \qquad \begin{array}{c} \neg(p \vee q) \\ \hline \therefore \neg p \wedge \neg q \end{array}$
8. Ley de simplificación	$\begin{array}{c} p \wedge q \\ \hline \therefore p \end{array} \qquad \begin{array}{c} \neg(p \vee q) \\ \hline \therefore q \end{array}$
9. Ley de conjunción	$\begin{array}{c} p \\ q \\ \hline \therefore p \wedge q \end{array}$
10. Ley de la adición disyuntiva	$\begin{array}{c} p \\ \hline \therefore p \vee q \end{array}$
11. Ley del argumento conjuntivo	$\begin{array}{c} \neg(p \wedge q) \\ p \\ \hline \therefore \neg q \end{array} \qquad \begin{array}{c} \neg(p \wedge q) \\ q \\ \hline \therefore \neg p \end{array}$

Tabla 3.8 Algunas reglas de inferencia para la lógica proposicional

Número de regla de inferencia	Nombre	Significado
1	<i>modus ponendo ponens</i>	“modo por el que la...” afirmación, afirma
3	<i>modus tollendo tollens</i>	negación, niega
5	<i>modus tollendo ponens</i>	negación, afirma
11	<i>modus ponendo tollens</i>	afirmación, niega

Tabla 3.9 Los significados de modus

Condicional	$p \rightarrow q$
recíproco	$q \rightarrow p$
inverso	$\neg p \rightarrow \neg q$
contrapositivo	$\neg q \rightarrow \neg p$

Tabla 3.10 El condicional y sus variantes

Como siempre, se supone que el operador de negación tiene una mayor prioridad que los otros operadores lógicos, de modo que no se usan paréntesis con  $\neg p$  y  $\neg q$ .

Si el condicional  $p \rightarrow q$  y su recíproco  $q \rightarrow p$  son verdaderos, entonces  $p$  y  $q$  son equivalentes; es decir,  $p \rightarrow q \wedge q \rightarrow p$  es equivalente al bicondicional  $p \leftrightarrow q$  o a la equivalencia  $p \equiv q$ . En otras palabras,  $p$  y  $q$  siempre toman los mismos valores de verdad, si  $p$  es verdadero entonces  $q$  es verdadero y si  $p$  es falso entonces  $q$  es falso. El argumento se vuelve

$$\begin{array}{l} (1) \quad B \rightarrow Y \\ (2) \quad Y \equiv D \\ (3) \quad B \\ \hline \therefore D \end{array}$$

donde los números se utilizan ahora para identificar las premisas. Como  $Y$  y  $D$  son equivalentes a partir de (2), podemos sustituir  $D$  por  $Y$  en (1) para obtener:

$$(4) \quad B \rightarrow D$$

donde (4) es una inferencia hecha con base en (1) y (2). Las premisas (3) y (4) y la conclusión son:

$$\begin{array}{l} (4) \quad B \rightarrow D \\ (3) \quad B \\ \hline \therefore D \end{array}$$

que puede reconocerse como un esquema de *modus ponens*. Por tanto, el argumento es válido.

La sustitución de una variable que es equivalente a otra, es una regla de inferencia llamada *regla de sustitución*. Las reglas de *modus ponens* y de sustitución son básicas en la lógica deductiva.

Una comprobación de lógica formal al enumerar las premisas, la conclusión y las inferencias, suele escribirse como se muestra a continuación:

1. $B \rightarrow Y$	
2. $(Y \rightarrow D) \wedge (D \rightarrow Y)$	
3. $B$	/ $\wedge D$
4. $Y \equiv D$	2 Equivalencia
5. $C \rightarrow D$	1 Sustitución
6. $D$	3,5 Modus ponens

Las líneas 1, 2 y 3 son las premisas y la conclusión; 4, 5 y 6 son las inferencias obtenidas. La columna de la derecha presenta la regla de inferencia y los números de línea se utilizan para justificar la inferencia.

### 3.7 LIMITACIONES DE LA LÓGICA DE PROPOSICIÓN

Considerando nuestro familiar y clásico argumento:

Todos los hombres son mortales  
Sócrates es un hombre  
 Por tanto, Sócrates es mortal

Sabemos que es un argumento válido porque es un silogismo válido. ¿Se puede probar su validez utilizando la lógica proposicional? Para responder a esta pregunta, primero se escribe el argumento como un esquema:

$p =$  Todos los hombres son mortales  
 $q =$  Sócrates es un hombre  
 $r =$  Sócrates es mortal

de este modo, el esquema de argumento es:

$\begin{array}{c} p \\ q \\ \hline \therefore r \end{array}$

Observe que no hay conectores lógicos ni conclusiones en las premisas y, por tanto, cada premisa y cada conclusión tiene una variable lógica diferente. Además, la lógica proposicional no ha previsto cuantificadores, así, no hay manera de representar al cuantificador "todos" en la primera premisa. Por lo tanto, la única representación de este argumento en la lógica proposicional es el esquema anterior con tres variables independientes.

Para determinar si es un argumento válido, se considera la tabla de verdad con tres variables independientes para todas las combinaciones posibles de V y F que se muestran en la tabla 3.11. La segunda fila de esta tabla muestra el argumento que habrá de invalidarse porque las premisas son verdaderas pero la conclusión es falsa.

La invalidez de este argumento *no* debe interpretarse como si significara que la conclusión es incorrecta. Cualquier persona reconocería que se trata de un argumento correcto, la invalidez simplemente significa que *el argumento no puede probarse con base en la lógica proposicional*. Puede probarse que el argumento es válido si examinamos la estructura interna de las premisas; por ejemplo, tendríamos que atribuir algún significado a "todos" y reconocer "hombres" como plural de "hombre". Sin embargo, los silogismos y el cálculo proposicional no permiten examinar la estructura interna de las proposiciones. Esta limita-

ción se supera con la lógica de predicados; y el argumento es válido bajo esta lógica. En realidad, toda la lógica silogística es un subconjunto válido de la lógica de predicados de primer orden, y con ella puede probarse su validez.

<b>p</b>	<b>q</b>	$\therefore r$
V	V	V
V	V	F
V	F	V
V	F	F
F	V	V
F	V	F
F	F	V
F	F	F

Tabla 3.11 Tabla de verdad para el esquema  $p, q; \therefore r$

La única forma silogística válida de la proposición es:

Si Sócrates es un hombre, entonces Sócrates es mortal.  
Sócrates es un hombre.  
 Por tanto, Sócrates es mortal.

Digamos que:

$$\begin{aligned} p &= \text{Sócrates es un hombre} \\ q &= \text{Sócrates es mortal} \end{aligned}$$

El argumento se convierte en:

$$\begin{array}{c} p \rightarrow q \\ \hline \therefore q \end{array}$$

que es una forma silogística válida de *modus ponens*.

Consideremos el siguiente argumento clásico como otro ejemplo.

Todos los caballos son animales  
 Por tanto, la cabeza de un caballo es la cabeza de un animal

Se sabe que este argumento es correcto, y aún así no podemos probarlo bajo una lógica de proposición; aunque puede probarse bajo la lógica de predicado (véase el problema 3.12).

## 3.8 LÓGICA DE PREDICADO DE PRIMER ORDEN

La lógica silogística puede describirse por completo con la lógica de predicado. En la tabla 3.12 se muestran las cuatro afirmaciones categóricas y sus representaciones en esta clase de lógica de predicado.

Tipo	Esquema	Representación de predicado
A	Todo S es P	$(\forall x) (S(x) \rightarrow P(x))$
E	Ningún S es P	$\neg(\forall x) (S(x) \rightarrow \neg P(x))$
I	Algun S es P	$(\exists x) (S(x) \wedge P(x))$
O	Algun S no es P	$(\exists x) (S(x) \wedge \neg P(x))$

Tabla 3.12 Representación de los cuatro silogismos categóricos utilizando lógica de predicados

Además de las reglas de inferencia previamente analizadas, la lógica de predicados tiene reglas que tratan con cuantificadores.

La Regla de Asignación Universal de Casos establece, en esencia, que un individuo puede sustituirse por uno universal. Por ejemplo, si  $\phi$  es cualquier proposición o función de proposición:

$$\frac{(\forall x) \phi(x)}{\therefore \phi(a)}$$

es una inferencia válida, donde  $a$  es un caso; es decir, alude a un individuo específico, mientras que  $x$  es una variable que abarca todos los individuos. Por ejemplo, esto puede utilizarse para probar que Sócrates es un ser humano:

$$\frac{(\forall x) H(x)}{\therefore H(\text{Sócrates})}$$

donde  $H(x)$  es la función proposicional que indica que  $x$  es un ser humano. Lo anterior establece que para toda  $x$ , esa  $x$  es un ser humano y por inferencia Sócrates es un ser humano.

Otros ejemplos de la Regla de Asignación Universal de Casos son:

$$\frac{(\forall x) A(x)}{\therefore A(c)}$$

$$\frac{(\forall y) (B(y) \vee C(b))}{\therefore B(a) \vee C(b)}$$

$$\frac{(\forall x) [A(x) \wedge (\exists x) (B(x) \vee C(y))] }{\therefore A(b) \wedge (\exists x) (B(x) \vee C(y))}$$

En el primer ejemplo, el caso  $c$  se sustituye con  $x$ , pero en el segundo ejemplo, nótese que el caso  $a$  se ha sustituido por  $y$ , pero no por  $b$ , porque  $b$  no está incluida en el **ámbito** del cuantificador; es decir, un cuantificador como  $\forall x$  sólo se aplica a las variables  $x$ . A las variables como  $x$  y  $y$  utilizadas con cuantificadores se les llama **delimitadas** y a los otros se les llama **libres**. En el tercer ejemplo, el cuantificador  $x$  sólo tiene como ámbito  $A(x)$ . Es decir,  $\forall x$  no se aplica al cuantificador existencial  $\exists x$  y a su ámbito sobre  $B(x) \vee C(y)$ . La convención de cuantificadores anidados como éste, es que el ámbito termina cuando se utiliza un nuevo cuantificador, aunque utilice la misma variable. La prueba formal del silogismo:

Todos los hombres son mortales  
Sócrates es un hombre  
 $\therefore$  Sócrates es mortal

se muestra a continuación, donde H = hombre, M = mortal y s = Sócrates:

- |  |                                 |
|--|---------------------------------|
| 1. $(\forall x) (H(x) \rightarrow M(x))$ |                                 |
| 2. $H(s)$                                | $\therefore M(s)$               |
| 3. $H(s) \rightarrow M(s)$               | 1 Asignación universal de casos |
| 4. $M(s)$                                | 2,3 Modus ponens                |

## 3.9 SISTEMAS LÓGICOS

Un sistema lógico es una recopilación de objetos como reglas, axiomas, instrucciones, etcétera, organizada de manera congruente. El sistema lógico tiene varios objetivos, el primero consiste en especificar las formas de los argumentos. Como los argumentos lógicos no tienen significado en un sentido semántico, una forma válida es esencial si habrá de determinarse la validez del argumento. Por tanto, una función importante de un sistema lógico es determinar las **fórmulas bien formadas (fbf)** que se utilizan en los argumentos; sólo éstas pueden utilizarse en argumentos lógicos. Por ejemplo, en la lógica silogística:

Todo S es P

podría ser una fbf, pero:

Todo  
 Todo es S P  
 Es S todo

no son fbf's. Aunque los símbolos del alfabeto carecen de significado, la secuencia de símbolos que conforman la fbf sí lo tiene.

El segundo objetivo de un sistema lógico es indicar las reglas de inferencia que son válidas. El tercer objetivo de un sistema lógico es extenderse al descubrir nuevas reglas de inferencia y, por tanto, extender el rango de argumentos que pueden probarse. Al extender el rango de argumentos, pueden probarse nuevas fbf, llamadas **teoremas**, con un argumento lógico.

Cuando un sistema lógico está bien desarrollado, puede utilizarse para determinar la validez de los argumentos de manera análoga a los cálculos en sistemas como la aritmética, la geometría, el cálculo, la física y la ingeniería. Se han desarrollado sistemas lógicos como el Cálculo de frases o proposiciones, el Cálculo de afirmaciones, etc. Cada sistema depende de definiciones formales de sus **axiomas o postulados**, que son sus definiciones fundamentales. A partir de estos axiomas, la gente (y a veces los programas de computadora como el AM), tratan de determinar lo que puede probarse. Cualquier persona que haya estudiado geometría euclíadiana en la preparatoria está familiarizado con los axiomas y la derivación de teoremas geométricos. Al igual que los teoremas geométricos pueden derivarse de los axiomas geométricos, los teoremas lógicos pueden derivarse de axiomas lógicos.

Un axioma es sólo un hecho o **aseveración** (o aserto) que no puede probarse desde el interior del sistema. A veces aceptamos ciertos axiomas porque tienen "sentido" al apelar al sentido común o la observación. Otros axiomas, como "las líneas paralelas se unen en el infinito", no tienen un sentido intuitivo porque parecen contradecir el axioma de Euclides de que las líneas

paralelas nunca se unen. Sin embargo, este axioma acerca de las líneas paralelas que se unen en el infinito es tan razonable desde un punto de vista puramente lógico como el de Euclides y es la base de un tipo de geometría no euclidiana.

Un sistema formal requiere lo siguiente:

1. Un alfabeto de símbolos.
2. Un conjunto de cadenas finitas de estos símbolos, las fbf.
3. Axiomas, las definiciones del sistema.
4. Reglas de inferencia, que permiten una fbf, A, que habrá de deducirse como la conclusión de un conjunto finito, G, de otras fbf donde  $G = \{A_1, A_2 \dots A_n\}$ . Estas fbf deben ser axiomas u otros teoremas del sistema lógico. Por ejemplo, un sistema de lógica proposicional puede definirse utilizando sólo *modus ponens* para derivar nuevos teoremas.

Si el argumento:

$$A_1, A_2, \dots A_n; \therefore A$$

es válido, entonces se dice que A es un **teorema** del sistema de lógica formal y se escribe con el símbolo  $\vdash$ . Por ejemplo,  $\Gamma \vdash A$  significa que A es un teorema del conjunto de fbf,  $\Gamma$ . Un esquema más explícito de una prueba de que A es un teorema se muestra a continuación:

$$A_1, A_2, \dots A_n \vdash A$$

El símbolo  $\vdash$ , que indica que el siguiente fbf es un teorema, no es un símbolo del sistema. En cambio,  $\vdash$  es un **metasímbolo**, porque se utiliza para describir al propio sistema. Una analogía es un lenguaje de computadora como Pascal, pues aunque pueden especificarse programas utilizando la sintaxis de Pascal, no hay sintaxis en Pascal para indicar un programa válido.

Una regla de inferencia en un sistema formal especifica la manera exacta en que pueden obtenerse nuevas asertos, los teoremas, a partir de axiomas y teoremas previamente derivados. Un ejemplo de un teorema es nuestro silogismo sobre Sócrates, escrito en forma de lógica de predicados:

$$(\forall x) (H(x) \rightarrow M(x)), H(s) \vdash M(s)$$

donde H es la función de predicado para hombre y M es la función de predicado para mortal. Como M(s) puede probarse a partir de sus axiomas de la izquierda, es un teorema de estos axiomas. Sin embargo, nótese que M(Zeus) no sería un teorema porque Zeus, el dios griego, no es un hombre, y no hay una manera alterna de demostrar M(Zeus).

Si un teorema es una tautología, se deduce que  $\Gamma$  es un conjunto vacío porque la fbf siempre es verdadera y, por tanto, no depende de ningún otro axioma o teorema. Un teorema que es una tautología se escribe con el símbolo  $\models$ , como en  $\models A$ . Por ejemplo, si  $A \models p \vee \neg p$ , entonces  $\models p \vee \neg p$  establece que  $p \vee \neg p$  es un teorema, que es una tautología. Observe que, independientemente de cualquier valor asignado a p, V o F, el teorema  $p \vee \neg p$  siempre es verdadero. Una asignación de valores verdaderos es una **interpretación** de una fbf. Un **modelo** es una interpretación en la cual la fbf es verdadera. Por ejemplo, un modelo de  $p \rightarrow q$  es  $p = V$  y  $q = V$ . A una fbf se le llama **consistente** si hay una interpretación que la hace verdadera, e **inconsistente** si la fbf es falsa en *todas* las interpretaciones. Un ejemplo de una fbf inconsistente es  $p \wedge \neg p$ .

Una fbf es **válida** si es verdadera en todas las interpretaciones; de otra manera es **inválida**. Por ejemplo, la fbf  $p \vee \neg p$  es válida, mientras que  $p \rightarrow q$  es una fbf inválida porque no es verdadera para  $p = V$  y  $q = F$ . Una fbf está **probada** si puede demostrarse que es válida. Todas las fbf de proposiciones pueden comprobarse con el método de la tabla de verdad, porque sólo hay un número finito de interpretaciones para fbf y, así, se puede determinar el cálculo proposicional **decidible**. Sin embargo, el cálculo de predicados no es decidible porque no hay un método general de comprobación como las tablas de verdad para todas las fbf del cálculo de afirmaciones.

Un ejemplo de una fbf de cálculo de predicados válida que puede probarse es la de cualquier predicado B;

$$(\exists x) B(x) \rightarrow \neg (\forall x) \neg B(x)$$

que muestra cómo el cuantificador existencial puede reemplazarse con el cuantificador universal. Por tanto, esta fbf de cálculo de predicados es un teorema.

Hay una gran diferencia entre una expresión como  $\vdash A$  y  $\vdash B$ . La A es un teorema y puede probarse a partir de los axiomas, mediante las reglas de inferencia. La B es una fbf y tal vez no haya prueba conocida para demostrar su derivación. Mientras que la lógica proposicional es decidible, la lógica de predicado no lo es. Es decir, no hay un procedimiento ni algoritmo mecánico para comprobar un teorema de lógica proposicional con un número finito de pasos. En realidad, existe una prueba teórica de que no hay un procedimiento de decisión para la lógica de predicados. Sin embargo, no hay procedimientos de decisión para sus subconjuntos como los silogismos y la lógica de proposición. Debido a esto, a veces se alude a la lógica de predicado como **semidecidible**.

Un ejemplo muy simple de un sistema formal completo, define lo siguiente:

*Alfabeto:* el símbolo “1”

*Axioma:* La cadena “1” (que resulta igual que el símbolo 1)

*Regla de inferencia:* si cualquier cadena \$ es un teorema, entonces la cadena \$11 lo es. Esta regla puede escribirse como una regla de producción de Post:

$$\$ \rightarrow \$11.$$

Si  $\$ = 1$  entonces la regla da  $\$11 = 111$

Si  $\$ = 111$  entonces la regla da  $\$11 = 11111$  y en general  
1, 111, 11111, 1111111, ...

Estas cadenas son los teoremas de este sistema formal (Minsky 67).

Aunque algo como 11111 no se parece a los teoremas que acostumbramos ver, son teoremas lógicos perfectamente válidos. Estos teoremas particulares también tienen un significado semántico, porque son los números nenes expresados en un **sistema de números unitario** del símbolo 1. Al igual que el sistema binario sólo tiene los símbolos del alfabeto 0 y 1, el sistema de números unitario sólo tiene el símbolo 1. Los números de los sistemas unitario y decimal se expresan como sigue:

Unitario	Decimal
1	1
11	2
111	3
1111	4
11111	5

y así sucesivamente.

Observemos que debido a la regla de inferencia y al axioma, las cadenas 11, 1111 y demás, no pueden expresarse en nuestro sistema formal. Es decir, 11 y 1111 son cadenas de nuestro alfabeto formal, pero no son teoremas ni fbf porque no pueden probarse utilizando solamente la regla de inferencia y el axioma. El sistema formal sólo permite la derivación de los números nones, no la de los pares. Debe agregarse el axioma “11” para poder derivar los números pares.

Otra propiedad de un sistema formal es su **capacidad de estar completo**. Un conjunto de axiomas está **completo** si cada fbf puede probarse o **refutarse**, este término significa probar que alguna aseveración es falsa. En un sistema completo, cada fbf lógicamente válida es un teorema. Sin embargo, como no es posible determinar la lógica de predicados, obtener una prueba depende de nuestra suerte e inteligencia. Por supuesto, otra posibilidad es escribir un programa de computadora que tratará de derivar pruebas y dejar que actúe.

Una propiedad aún más deseable de un sistema lógico es que sea **sólido**. Por sistema sólido se entiende que cada teorema es una fbf lógicamente válida. En otras palabras, un sistema sólido no permitirá que se infiera una conclusión que no sea una consecuencia lógica de sus premisas. No se inferirán argumentos inválidos como válidos.

Hay diferentes **órdenes** de lógica. Un lenguaje de **primer orden** se define para que los cuantificadores operen sobre objetos que son variables como  $\forall x$ . Un lenguaje de **segundo orden** tendría características adicionales como dos tipos de variables y cuantificadores. Además de variables y cuantificadores de orden, la lógica de segundo orden puede tener cuantificadores que abarcan símbolos de función y de predicado. Un ejemplo de lógica de segundo orden es el **axioma de igualdad**, que establece que dos objetos son iguales si todos sus predicados son iguales. Si P es cualquier predicado de un argumento, entonces:

$$x = y \equiv (\forall P) [P(x) \leftrightarrow P(y)]$$

es una afirmación del axioma de igualdad que usa un cuantificador de segundo orden,  $\forall P$ , que abarca todos los predicados.

### 3.10 RESOLUCIÓN

La muy eficaz regla de **resolución** de inferencia, introducida por Robinson en 1965, con frecuencia se implanta en programas de AI para prueba de teoremas (Robinson 65). En realidad, la resolución es la principal regla de inferencia de PROLOG. En lugar de muchas reglas de inferencia de aplicación limitada, como *modus ponens*, *modus tollens*, combinación, encadenamiento, etcétera, PROLOG utiliza una regla de inferencia general de resolución. Esta aplicación de la resolución hace que los probadores automáticos de teoremas como PROLOG resulten herramientas prácticas para resolver problemas, por que en lugar de tener que probar diferentes reglas de inferencia esperando que una tenga éxito, puede aplicarse la regla de resolución simple, método que puede reducir mucho el espacio de búsqueda.

Como una manera de introducir la resolución, consideremos primero el silogismo sobre Sócrates expresado en PROLOG como se muestra a continuación, donde los comentarios se muestran con un signo de porcentaje:

```
mortal(X) :- hombre(X).      % Todos los hombres son mortales
hombre(sócrates).            % Sócrates es un hombre
:- mortal(sócrates).         % Consulta: ¿Sócrates es mortal?
sí                           % PROLOG responde sí
```

PROLOG utiliza una notación **libre de cuantificador**. Obsérvese que el cuantificador universal,  $\forall$ , está implícito en la afirmación de que todos los hombres son mortales.

PROLOG se basa en lógica de predicados de primer orden. Sin embargo, también tiene varias extensiones para facilitar las aplicaciones de programación. Estas funciones de programación especiales violan la lógica de predicados pura y se les denomina **funciones extralógicas**: entrada/salida, corte (que altera el espacio de búsqueda) y asegurar/retractar (para modificar valores de verdad sin justificación lógica).

Antes de que pueda aplicarse la resolución, la fbf debe estar en una forma **normal** o estándar. Los tres principales tipos de formas normales son la **forma normal conjuntiva**, la forma de cláusula y su subconjunto cláusula de Horn. La idea básica de la forma normal es expresar la fbf de una forma estándar que sólo utilice  $\wedge$ ,  $\vee$  y tal vez  $\neg$ . El método de resolución se aplica entonces a las fbf de forma normal en la que todos los conectores y cuantificadores se han eliminado. Esta conversión es necesaria porque la resolución es una operación sobre pares de disyunciones, que produce nuevas disyunciones, lo que simplifica la fbf.

A continuación se ilustra una fbf de forma normal conjuntiva, que se define como la conjunción de disyunciones, que son **literales**.

$$(P_1 \vee P_2 \vee \dots) \wedge (Q_1 \vee Q_2 \vee \dots) \wedge \dots (Z_1 \vee Z_2 \vee \dots)$$

Términos como  $P_i$  deben ser literales, lo que significa que no contienen conectores lógicos como el condicional o bicondicional, o cuantificadores. Una literal es una fórmula atómica o una fórmula atómica negada. Por ejemplo, la siguiente fbf:

$$(A \vee B) \wedge (\neg B \vee C)$$

está en forma normal conjuntiva. Los términos entre paréntesis son cláusulas:

$$A \vee B \text{ y } \neg B \vee C$$

Como se mostrará más adelante, cualquier fbf de lógica de predicados, que incluye a la lógica proposicional como un caso especial, puede escribirse en forma de cláusulas. La **forma de cláusula completa** puede expresar cualquier fórmula de la lógica de afirmaciones, pero tal vez no sea tan natural o legible para una persona (Kowalski 79). La sintaxis de PROLOG es el conjunto de la cláusula de Horn, que facilita en gran medida la comprobación de los teoremas mecánicos con PROLOG y hace su implantación más eficiente que la notación de lógica de predicados normal o la forma de cláusula completa. Como se mencionó en el capítulo 1, PROLOG sólo permite un encabezado. Por lo general, una expresión en forma de cláusula completa se escribe de una forma especial llamada forma de cláusula de Kowalski:

$$A_1, A_2, \dots, A_N \rightarrow B_1, B_2, \dots, B_M$$

la cual se interpreta como si se dijera que si todos los subobjetivos  $A_1, A_2, \dots, A_N$  son verdaderos, entonces uno o más de  $B_1, B_2, \dots, B_M$  también son verdaderos. Obsérvese que a veces la dirección de la flecha está invertida en esta notación. Esta cláusula, escrita en notación de predicados normal, es:

$$A_1 \wedge A_2 \dots A_N \rightarrow B_1 \vee B_2 \dots B_M$$

Esto puede expresarse en **forma disyuntiva** como la disyunción de literales utilizando la equivalencia:

$$p \rightarrow q \equiv \neg p \vee q$$

así:

$$\begin{aligned} A_1 \wedge A_2 \dots A_N &\rightarrow B_1 \vee B_2 \dots B_M \\ &\equiv \neg(A_1 \wedge A_2 \dots A_N) \vee (B_1 \vee B_2 \dots B_M) \\ &\equiv \neg A_1 \vee \neg A_2 \dots \neg A_N \vee B_1 \vee B_2 \dots B_M \end{aligned}$$

donde la ley de De Morgan:

$$\neg(p \wedge q) \equiv \neg p \vee \neg q$$

se usa para simplificar la última expresión.

Como se comentó en el capítulo 1, PROLOG utiliza un tipo restringido de forma de cláusula, la cláusula de Horn, en la que sólo se permite un encabezado.

$$A_1, A_2, \dots, A_N \rightarrow B$$

que se escribe en sintaxis de PROLOG como:

$$B :- A_1, A_2, \dots, A_N$$

El problema que surge al tratar de probar directamente un teorema, es la dificultad de deducirlo utilizando sólo las reglas de inferencia y los axiomas del sistema. Esto puede requerir mucho tiempo para deducir un teorema o tal vez no seríamos lo suficientemente listos para hacerlo. Para probar que un teorema es verdadero, se utiliza el método clásico de **reducción al absurdo**, o método de contradicción; en éste, se trata de probar que la fbf negada es un teorema, si resulta una contradicción, entonces la fbf original no negada es un teorema.

El objetivo básico de la resolución es inferir una nueva cláusula, el **resolutivo**, de otras dos cláusulas, llamadas **cláusulas padre**; el resolutivo tendrá menos términos que los padres. Al seguir el proceso de resolución, terminará por obtenerse una contradicción o el proceso se terminará porque no se ha hecho ningún progreso. En el siguiente argumento se muestra un ejemplo simple de resolución.

$$\begin{array}{l} A \vee B \\ \underline{A \vee \neg B} \\ \therefore A \end{array}$$

Una manera de ver cómo se llega a la conclusión, es escribiendo las premisas como:

$$(A \vee B) \wedge (A \vee \neg B)$$

Uno de los axiomas de distribución es:

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$

Aplicando esto a las premisas se tiene:

$$(A \vee B) \wedge (A \vee \neg B) \equiv A \vee (B \wedge \neg B) \equiv A$$

donde el último paso seguido desde  $(B \wedge \neg B)$  siempre es falso. Esto se desprende de la Ley del medio excluido, que establece que algo no puede ser verdadero y falso a la vez. En la lógica de confusión, analizada en el capítulo 5, veremos que esta ley no se sostiene. Otra

Tabla 3.13

### 3.11 SIS

manera de escribir esto utiliza el término **nulo**, que significa vacío, nada o falso. Por ejemplo, un apuntador nulo en Pascal apunta a nada, y la Ley del medio excluido establece que  $(B \wedge \neg B) \equiv$  nulo.

El ejemplo de resolución muestra la manera en que las cláusulas padre ( $A \vee B$ ) y ( $A \vee \neg B$ ) pueden simplificarse en el resolutivo A. En la tabla 3.13 se resumen algunas cláusulas padre básicas y sus resolutivos en notación de cláusula, donde las comas que separan a las cláusulas significa  $\wedge$ .

Cláusulas padre	Resolutivo	Significado
$p \rightarrow q, p$ o $\neg p \vee q, p$	q	<i>Modus ponens</i>
$p \rightarrow q, q \rightarrow r$ o $\neg p \vee q, \neg q \vee r$	$p \rightarrow r$ o $\neg p \vee r$	Encadenamiento o silogismo hipotético
$\neg p \vee q, p \vee q$	q	Combinación
$\neg p \vee \neg q, p \vee q$	$\neg p \vee p$ o $\neg q \vee q$	VERDADERO (una tautología)
<u><math>\neg p, p</math></u>	nulo	FALSO (una contradicción)

Tabla 3.13 Cláusulas y resolutivos

### 3.11 SISTEMAS DE RESOLUCIÓN Y DEDUCCIÓN

Dadas las fbf  $A_1, A_2, \dots, A_N$  y una conclusión lógica o teorema C, sabemos que

$$A_1 \wedge A_2 \dots A_N \vdash C$$

es equivalente a establecer que

$$(1) \quad A_1 \wedge A_2 \dots A_N \rightarrow C \quad \equiv \neg(A_1 \wedge A_2 \dots A_N) \vee C \\ \equiv \neg A_1 \vee \neg A_2 \dots \neg A_N \vee C$$

es válida. Suponga que tomamos la negación como se muestra en seguida:

$$\neg [A_1 \wedge A_2 \dots A_N \rightarrow C]$$

Ahora

$$p \rightarrow q \equiv \neg p \vee q$$

y así lo anterior se vuelve:

$$\neg[A_1 \wedge A_2 \dots A_N \rightarrow C] \equiv \neg[\neg(A_1 \wedge A_2 \dots A_N) \vee C]$$

A partir de la ley de De Morgan,

$$\neg(p \vee q) \equiv \neg p \wedge \neg q$$

y así lo anterior se vuelve:

$$(2) \quad \neg[A_1 \wedge A_2 \dots A_N \rightarrow C] \equiv [\neg(\neg(A_1 \wedge A_2 \dots A_N)) \wedge \neg C] \\ \equiv A_1 \wedge A_2 \dots A_N \wedge \neg C$$

Ahora si (1) es válida, entonces su negación (2) debe ser inválida. En otras palabras, si (1) es una tautología, entonces (2) debe ser una contradicción. Las fórmulas (1) y (2) representan dos maneras equivalentes de comprobar que una fórmula C es un teorema. La fórmula (1) puede utilizarse para probar un teorema mediante verificación para ver si es verdadera en todos los casos; al mismo tiempo, la fórmula (2) puede utilizarse para probar un teorema al mostrar que (2) conduce a una contradicción.

Como se mencionó en la sección anterior, la comprobación de un teorema mostrando que su negación lleva a una contradicción, se prueba mediante la *reducción al absurdo*. La parte principal de este tipo de prueba es la **refutación**; refutar algo significa probar que es falso. La resolución es una regla de inferencia sólida que es también una **refutación completa**, porque la cláusula vacía siempre será el resultado eventual si hay una contradicción en el conjunto de cláusulas. En esencia, esto significa que si hay una contradicción la **refutación de resolución** terminará en un número finito de pasos. Aunque la refutación de resolución no puede indicarnos cómo *producir* un teorema, sin duda nos indicará si una fbf es un teorema.

Como un ejemplo simple de prueba por refutación de resolución, tomemos el argumento:

$$\begin{array}{c} A \rightarrow B \\ B \rightarrow C \\ C \rightarrow D \\ \hline \therefore A \rightarrow D \end{array}$$

Para probar que la conclusión  $A \rightarrow D$  es un teorema por refutación de resolución, primero se convierte a forma disyuntiva utilizando la equivalencia:

$$p \rightarrow q \equiv \neg p \vee q$$

Así;

$$A \rightarrow D \equiv \neg A \vee D$$

y su negación es:

$$\neg(\neg A \vee D) \equiv A \wedge \neg D$$

La conjunción de las formas disyuntivas de las premisas y la conclusión negada da la forma conjuntiva normal adecuada para refutar la resolución:

$$(\neg A \vee B) \wedge (\neg B \vee C) \wedge (\neg C \vee D) \wedge A \wedge \neg D$$

El método de resolución puede aplicarse ahora a las premisas y la conclusión. En la figura 3.18 se muestra un método para representar la refutación de una resolución con la forma de un diagrama de **árbol de refutación de una resolución**, donde se resuelven las cláusulas del mismo nivel. La raíz, que es el resolutivo final, es nula, como puede verificarse en la última fila de la tabla 3.13 para  $\neg p$ ,  $p$ , y así la conclusión original  $A \rightarrow D$  es un teorema.

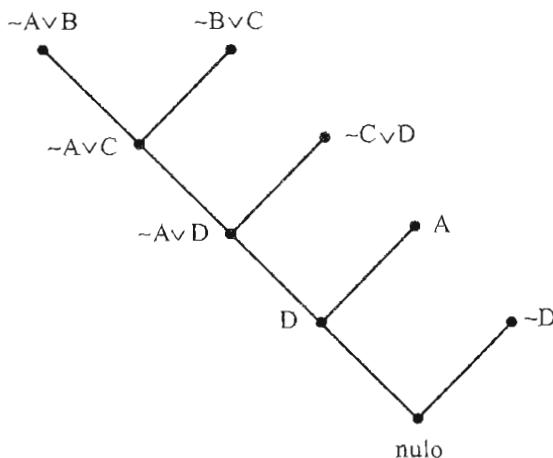


Figura 3.18 Árbol de refutación de resolución

### 3.12 RAZONAMIENTO SUPERFICIAL Y CAUSAL

Los sistemas de resolución y de reglas de producción son dos paradigmas populares para la comprobación de teoremas. Aunque la mayoría considera a los teoremas en el sentido matemático, se ha visto que un teorema en realidad es la conclusión de un argumento lógico válido. Ahora, consideremos un sistema experto que utiliza una cadena de inferencia; en general, una cadena más larga representa conocimiento más causal o profundo, mientras que el razonamiento superficial suele utilizar una sola regla o unas cuantas inferencias. Además de la longitud de la cadena de inferencia, la calidad del conocimiento en las reglas también es un factor importante para distinguir el razonamiento profundo y superficial. A veces se usa otra definición de conocimiento superficial, llamada **conocimiento de experiencia**, que es el conocimiento basado en la experiencia.

La conclusión de una cadena de inferencia es un teorema porque lo prueba la cadena de inferencia, como se demostró en el ejemplo anterior:

$$A \rightarrow B, B \rightarrow C, C \rightarrow D \vdash A \rightarrow D$$

De hecho, los sistemas expertos que usan una cadena de inferencia para establecer una conclusión están utilizando teoremas; este resultado es muy importante porque de otra manera no sería posible utilizarlos para la inferencia causal. Por supuesto, los sistemas expertos estarían restringidos a inferencias superficiales de reglas simples sin encadenamiento.

A continuación se verán algunas reglas para comparar mejor el razonamiento superficial y profundo. Para empezar, considere la siguiente regla, en la que el número entre paréntesis es sólo para propósitos de identificación:

(1)     SI un automóvil tiene  
              una buena batería  
              buenas bujías  
              gasolina  
              buenas llantas  
      ENTONCES el automóvil puede avanzar

Ésta es una regla perfectamente correcta que podría utilizarse en un sistema experto.

Una de las funciones importantes de un sistema experto es su capacidad de explicación, como se analiza en el capítulo 1. Los sistemas expertos basados en reglas facilitan que el sistema explique su razonamiento. En este caso, si el usuario pregunta cómo puede avanzar el automóvil, el sistema experto podría responder con una lista de sus elementos condicionales:

una buena batería  
 buenas bujías  
 gasolina  
 buenas llantas

Se trata de un tipo de mecanismo de explicación elemental, porque el sistema presenta una lista sólo de los elementos condicionales de la regla. Es posible diseñar medios de explicación más sofisticados para presentar una lista de reglas previas que se han disparado y que dieron como resultado el disparo de la regla actual. Otros medios de explicación pueden permitir que el usuario haga preguntas del tipo "qué pasaría si" para explorar rutas alternativas de razonamiento.

Esta regla también es un ejemplo de **razonamiento superficial**, es decir, en este hay poca o nula comprensión de la causa y el efecto, porque existe poca o nula cadena de inferencia. En esencia, la regla anterior es heurística, porque todo el conocimiento está contenido en una regla. Ésta se activa cuando sus elementos condicionales se satisfacen y no porque el sistema experto comprenda la función de los elementos condicionales. En el razonamiento superficial, hay poca o nula **cadena causal** de la causa y el efecto de una regla a otra. En el caso más simple, la causa y el efecto están contenidos en una regla que no guarda relación con ninguna otra. Si se considera a las reglas desde el punto de vista de los fragmentos de conocimiento analizados en el capítulo 1, el razonamiento superficial no realiza las conexiones entre los fragmentos, de modo que es como una reacción refleja simple.

La ventaja del razonamiento superficial en comparación con el razonamiento causal, es la facilidad de programación; esto significa que el tiempo de desarrollo es más corto y el programa es más pequeño y rápido, y su desarrollo es más barato.

Los marcos y las redes semánticas son dos modelos útiles para el **razonamiento profundo** o causal. El término *profundo* se usa a menudo como sinónimo de razonamiento causal para indicar un conocimiento profundo del tema. Sin embargo, una comprensión profunda requiere que además de comprender la cadena causal por la que ocurre el proceso, también se comprenda el proceso en un sentido abstracto.

Un sistema experto clásico que utiliza marcos y que tiene una comprensión causal es el sistema Steamer, construido por la Marina de Estados Unidos para enseñar la operación de naves con plantas de propulsión (Hollan 84). Un sistema experto profundo es el Tutor LISP, que está diseñado para actuar como un tutor inteligente para enseñar LIPS a los estudiantes (Reiser 85).

Podemos agregar razonamiento causal simple a nuestra regla por medio de la definición de reglas adicionales como:

- (2) SI la batería es buena  
ENTONCES hay electricidad
- (3) SI hay electricidad  
y las bujías son buenas  
ENTONCES las bujías producirán una chispa
- (4) SI las bujías producen una chispa  
y hay gasolina  
ENTONCES el motor encenderá
- (5) SI el motor enciende  
y hay buenas llantas  
ENTONCES el automóvil avanzará

Observe que con el razonamiento causal, el mecanismo de explicación puede indicar lo que hace cada componente del automóvil, porque cada elemento está especificado en una regla. Este tipo de sistema causal también facilita la escritura de un sistema de diagnóstico para determinar cuál efecto tendrá un mal componente. El razonamiento causal puede utilizarse para un refinamiento arbitrario de la operación de un sistema limitado por la velocidad de ejecución, el tamaño de la memoria y el aumento en el costo de desarrollo.

El razonamiento causal puede utilizarse para construir un **modelo** del sistema real que se comporte en todos los aspectos como el elemento real. Este tipo de modelo puede utilizarse para simulación, con el fin de explorar el razonamiento hipotético del tipo de consultas "qué pasaría si". Sin embargo, los modelos causales no siempre son necesarios ni deseables. Por ejemplo, el sistema MUD sirve como consultor para ingenieros de líquidos de perforación (Kahn 85). El líquido de perforación, llamado *mud* (fango) debido a su parecido con el fango, es una ayuda importante en la perforación por muchas razones, como el enfriamiento y la lubricación del taladro. MUD diagnostica problemas con el fango y sugiere tratamientos.

Un sistema causal no sería muy útil porque, en condiciones normales, el ingeniero de perforación no podría observar la cadena causal de los acontecimientos que ocurren más allá de la superficie. En cambio, el ingeniero sólo puede observar los síntomas en la superficie y no los eventos intermedios no observables de importancia potencial para el diagnóstico.

La situación es muy diferente en medicina, donde los médicos tienen un amplio rango de pruebas de diagnóstico que pueden utilizarse para verificar los eventos intermedios. Por ejemplo, si una persona se queja de sentirse enferma, el médico puede comprobar si hay fiebre; de ser así, puede tratarse de una infección y realizará un análisis de sangre, si éste arroja una infección por tétanos, el médico puede verificar si la persona ha sufrido cortadas con un objeto oxidado en fecha reciente. Por el contrario, si el líquido de perforación se vuelve salado, el ingeniero sospecharía que el taladro ha perforado un manto de sal. Sin embargo, no hay una manera simple de verificar esto, porque no es posible entrar en la perforación, y una prueba sísmica es cara y no siempre resulta confiable. En virtud de que los ingenieros de perforación por lo general no pueden probar las hipótesis intermedias como los médicos, no resuelven los problemas de diagnóstico de la misma manera que éstos, y MUD refleja este método.

Otra razón para no utilizar el razonamiento causal en MUD es que existe un número muy limitado de diagnósticos y síntomas. La mayor parte de las pruebas relevantes que utiliza MUD son rutinarias y se introducen de antemano, por lo que hay pocas ventajas en una sesión de consulta interactiva con un ingeniero para explorar rutas de diagnóstico alternas, si el sistema ya conoce todas las pruebas y rutas de diagnóstico relevantes. Si hubiera muchas

rutas de diagnóstico posibles a seguir con una hipótesis verificable intermedia, habría ventajas para el conocimiento causal porque el ingeniero podría trabajar con el sistema para reducir la búsqueda a rutas factibles.

Debido al mayor número de requisitos para el razonamiento causal, sería necesario combinar ciertas reglas en un razonamiento superficial. El método de resolución con refutación puede utilizarse para probar que una sola regla es una conclusión verdadera de varias reglas. La regla simple es el teorema que se probará mediante resolución.

Por ejemplo, suponiendo que queremos probar que la regla (1) es una conclusión lógica de las reglas (2) a (5). Utilizando las siguientes definiciones proposicionales, las reglas pueden expresarse como sigue:

$B =$ la batería es buena	$A =$ el automóvil avanzará
$E =$ hay electricidad	$C =$ las bujías producen una chispa
$G =$ hay gasolina	$P =$ el motor prenderá
$S =$ las bujías son buenas	$L =$ hay buenas llantas

- (1)  $B \wedge S \wedge G \wedge L \rightarrow A$
- (2)  $B \rightarrow E$
- (3)  $E \wedge S \rightarrow C$
- (4)  $C \wedge G \rightarrow P$
- (5)  $P \wedge L \rightarrow A$

El primer paso para aplicar la refutación de resolución es negar la conclusión o regla de objetivo:

$$(1') \neg(B \wedge S \wedge G \wedge L \rightarrow A) = \neg[\neg(B \wedge S \wedge G \wedge L) \vee A] \\ = \neg[\neg B \vee \neg S \vee \neg G \vee \neg L \vee A]$$

Ahora, cada una de las otras reglas se expresa en forma disyuntiva utilizando equivalencias como:

$$p \rightarrow q \equiv \neg p \vee q \text{ y } \neg(p \wedge q) \equiv \neg p \vee \neg q$$

para obtener las siguientes nuevas versiones de (2) a (5):

- (2')  $\neg B \vee E$
- (3')  $\neg(E \wedge S) \quad C = \neg E \vee \neg S \vee C$
- (4')  $\neg(C \wedge G) \quad P = \neg C \vee \neg G \vee P$
- (5')  $\neg(P \wedge L) \quad A = \neg P \vee \neg L \vee A$

Como se describe en la sección anterior, una manera conveniente de representar los resolutivos sucesivos de (1') a (5') es con un árbol de refutación de resolución, como se muestra en la figura 3.19. Empezando en la parte superior del árbol, las cláusulas se representan como nodos, que se resuelven para producir el siguiente resolutivo. Por ejemplo:

$$\neg B \vee E \text{ y } \neg E \vee \neg S \vee C$$

se resuelven para inferir:

$$\neg B \vee \neg S \vee C$$

que se resuelve luego con:

$$\neg C \vee \neg G \vee P$$

para inferir:

$$\neg B \vee \neg S \vee \neg G \vee P$$

y así sucesivamente. Para simplificar el dibujo, los últimos resolutivos están implícitos, en vez de dibujar cada uno de ellos.

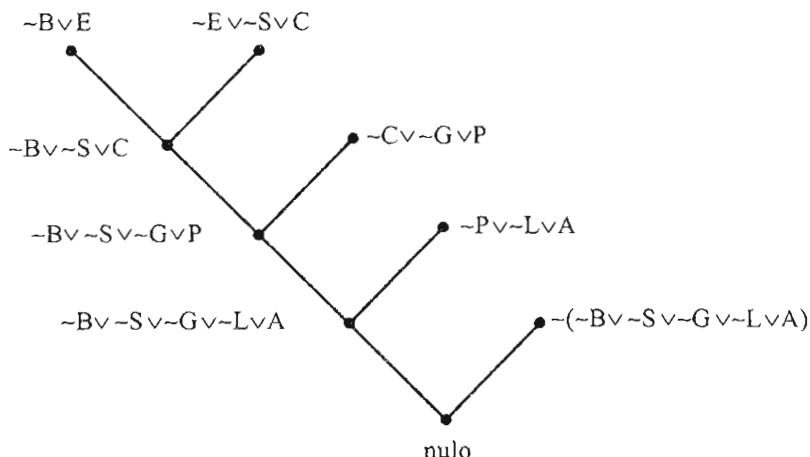


Figura 3.19 Árbol de refutación de la resolución para el ejemplo del automóvil

Como la raíz del árbol es nula, esta resolución es una contradicción. Por refutación, la conclusión original:

$$B \wedge S \wedge G \wedge L \rightarrow A$$

es un teorema porque su negación lleva a una contradicción. Por tanto, la regla (1) se desprende lógicamente de la regla (2) a la (5).

### 3.13 RESOLUCIÓN Y LÓGICA DE PREDICADO DE PRIMER ORDEN

El método de resolución también se utiliza con la lógica de predicados de primer orden. En realidad, es el principal mecanismo de inferencia de PROLOG. Sin embargo, antes de que pueda aplicarse la resolución, debe ponerse una ffb en forma de cláusula. Por ejemplo:

Algunos programadores odian las fallas  
Ningún programador odia el éxito  
 $\therefore$  Ninguna falla es un éxito

Definiendo los siguientes predicados:

P(x) = x es un programador  
F(x) = x es una falla  
E(x) = x es un éxito  
O(x, y) = x odia a y

Las premisas y la conclusión negada se escriben como:

- (1)  $(\exists x) [P(x) \wedge (\forall y) (F(y) \rightarrow O(x, y))]$
- (2)  $(\forall x) [P(x) \rightarrow (\forall y) (E(y) \rightarrow \neg O(x, y))]$
- (3)  $\neg(\forall y) (F(y) \rightarrow \neg E(y))$

donde la conclusión se ha negado como preparación para la resolución.

### Conversión a la forma de cláusula

Los siguientes nueve pasos son un algoritmo para convertir las fbf de predicado de primer orden a la forma de cláusula. Este procedimiento se ilustra utilizando la fbf (1) de la página anterior.

1. Eliminar condicionales,  $\rightarrow$ , utilizando la equivalencia

$$p \rightarrow q \equiv \neg p \vee q$$

de modo que la fbf(1) se convierte en:

$$(\exists x) [P(x) \wedge (\forall y) (\neg F(y) \rightarrow O(x, y))]$$

2. Cuando sea posible, eliminar las negaciones o reducir el ámbito de la negación a un átomo. Utilizar las equivalencias mostradas en el apéndice A como:

$$\begin{aligned} \neg\neg p &\equiv q \\ \neg(p \wedge q) &\equiv \neg p \vee \neg q \\ \neg(\exists x) P(x) &\equiv (\forall x) \neg P(x) \\ \neg(\forall x) P(x) &\equiv (\exists x) \neg P(x) \end{aligned}$$

3. Estandarizar las variables dentro de una fbf, para que las variables de límite o modelos de cada cuantificador tengan nombres únicos. Nótese que los nombres de variable de un cuantificador son modelos. Es decir:

$$(\forall x) P(x) \equiv (\forall y) P(y) \equiv (\forall z) P(z)$$

y así, la forma estandarizada de:

$$(\exists x) \neg P(x) \vee (\forall x) P(x)$$

es:

$$(\exists x) \neg P(x) \vee (\forall y) P(y)$$

4. Eliminar cuantificadores existenciales,  $\exists$ , utilizando **funciones de Skolem**, denominadas en honor del lógico noruego Thoralf Skolem. Considerando la fbf:

$$(\exists x) L(x)$$

donde  $L(x)$  se define como el predicado, que es verdadera si  $x < 0$ . Esta fbf puede reemplazarse con:

$L(a)$

donde  $a$  es una constante como  $-1$ , que hace que  $L(a)$  sea verdadera. A la  $a$  se le llama constante de Skolem, que es un caso especial de la función de Skolem. Para el caso en el que haya un cuantificador universal antes de uno existencial;

$$(\forall x) (\exists y) L(x, y)$$

donde  $L(x, y)$  es verdadera si el entero  $x$  es menor que el entero  $y$ . Esta fbf significa que para cada entero  $x$ , hay un entero  $y$  mayor que  $x$ . Observe que la fórmula no indica cómo calcular  $y$ , dado un valor para  $x$ . Suponiendo que existe una función  $f(x)$  que produce una  $y$  mayor que  $x$  la fbf anterior se le aplica la función de Skolem como:

$$(\forall x) L(x, f(x))$$

La función de Skolem de una variable existencial dentro del ámbito de un cuantificador universal, es una función de todos los cuantificadores de la izquierda. Por ejemplo:

$$(\exists u) (\forall v) (\forall w) (\exists x) (\forall y) (\exists z) P(u, v, w, x, y, z)$$

se le aplica la función de Skolem como:

$$(\forall v) (\forall w) (\forall y) P(a, v, w, f(v, w), y, g(v, w, y))$$

donde  $a$  es alguna constante y la segunda función de Skolem,  $g$ , debe ser diferente a la primera función,  $f$ . Nuestra fbf del ejemplo se convierte en:

$$P(a) \wedge (\forall y) (\neg F(y) \vee H(a, y))$$

5. Convertir la fbf a una **forma prenex**, que es una secuencia de cuantificadores,  $Q$ , seguida de una **matriz**,  $M$ . En general, los cuantificadores pueden ser  $\forall$  o  $\exists$ . Sin embargo, en este caso, en el paso 4 ya se han eliminado todos los cuantificadores existenciales y la  $Q$  sólo puede ser  $\forall$ . Además, como cada  $\forall$  tiene su propia variable modelo, todas las  $\forall$  pueden moverse a la izquierda de la fbf y el ámbito de cada  $\forall$  puede ser toda la fbf.

Nuestro ejemplo se convierte en:

$$(\forall y) [P(a) \wedge (\neg F(y) \vee H(a, y))]$$

donde la matriz es el término entre corchetes.

6. Convertir la matriz a la forma conjuntiva normal, que es una conjunción de cláusulas. Cada cláusula es una disyunción. El ejemplo ya es una forma normal conjuntiva donde una cláusula es  $P(a)$  y la otra es  $(\neg F(y) \vee H(a, y))$ . Si es necesario, puede utilizarse la siguiente regla distributiva para poner la matriz en la forma normal conjuntiva:

$$p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$$

7. Desechar los cuantificadores universales innecesarios porque en esta etapa todas las variables de una fbf deben estar delimitadas. La fbf ahora es la matriz. Nuestra fbf del ejemplo se convierte en:

$$P(a) \wedge (\neg F(y) \vee O(a,y))$$

8. Eliminar los signos  $\wedge$ , escribiendo la fbf como un conjunto de cláusulas. El ejemplo ahora es:

$$\{ P(a), \neg F(y) \vee O(a,y) \}$$

que por lo general se escribe sin las llaves, como las cláusulas:

$$\begin{aligned} P(a) \\ \neg F(y) \vee O(a,y) \end{aligned}$$

9. Cambiar el nombre de las variables en las cláusulas, si es necesario, para que el mismo nombre de la variable sea el único utilizado en una cláusula. Por ejemplo, si tuviéramos las cláusulas:

$$\begin{aligned} P(x) \wedge Q(x) \vee L(x,y) \\ \neg P(x) \vee Q(y) \\ \neg Q(z) \vee L(z,y) \end{aligned}$$

pueden renombrarse como:

$$\begin{aligned} P(x1) \wedge Q(x1) \vee L(x1,y1) \\ \neg P(x2) \vee Q(y2) \\ \neg Q(z) \vee L(z,y3) \end{aligned}$$

Si se realiza el procedimiento para convertir la segunda premisa a la forma de cláusula y la conclusión negada de nuestro ejemplo, al final se obtendrán las cláusulas:

- (1a)  $P(a)$
- (1b)  $\neg F(y) \vee O(a,y)$
- (2a)  $\neg P(x) \vee \neg E(y) \vee \neg O(x,y)$
- (3a)  $F(b)$
- (3b)  $E(b)$

donde los números aluden a las premisas originales y a la conclusión negada. Por tanto, las premisas (1) y (3) se convierten en dos cláusulas con sufijos (a) y (b), mientras que la premisa (2) se convierte en la cláusula (2a).

## Unificación y reglas

Una vez que se han convertido las fbf a la forma de cláusula, suele ser necesario encontrar **casos de sustitución** apropiados para las variables. Es decir, cláusulas como:

$$\begin{aligned} \neg F(y) \vee O(a,y) \\ F(b) \end{aligned}$$

no pueden resolverse con base en el predicado  $F$ , hasta que coincidan los argumentos de  $F$ . Al proceso de encontrar sustituciones para las variables, de modo que los argumentos coincidan, se le llama **unificación**.

La unificación es una función que distingue a los sistemas expertos de los simples árboles de decisión. Sin unificación, los elementos condicionales de las reglas sólo se podrían igualar

con las constantes, lo que significa que tendría que escribirse una regla específica para cada hecho posible. Por ejemplo, suponiendo que alguien quiere hacer sonar una alarma si un sensor indica humo. Si hay N sensores, se necesitaría una regla para cada sensor, como se muestra a continuación:

```
SI sensor 1 indica humo ENTONCES suena alarma 1
SI sensor 2 indica humo ENTONCES suena alarma 2
...
SI sensor N indica humo ENTONCES suena alarma N
```

Sin embargo, con la unificación, puede utilizarse una variable llamada ?N para el identificador del sensor, de modo que puede escribirse la siguiente regla:

```
SI sensor ?N indica humo
ENTONCES suena alarma ?N
```

Cuando dos literales complementarias están unificadas, pueden eliminarse por resolución. En las dos cláusulas anteriores, la sustitución de y por b resultaría:

```
~F(b) ∨ O(a,b)
F(b)
```

El predicado F ahora se ha unificado y puede resolverse en:

```
O(a,b)
```

Una sustitución se define como el reemplazo simultáneo de variables por términos que los diferencian de las variables, estos pueden ser constantes, variables o funciones. La sustitución de términos por variables se indica con el conjunto:

```
{ t1/v1, t2/v2 ... tn/vn }
```

Si θ es un conjunto como éste y A es un argumento, entonces Aθ se define como el caso de sustitución de A. Por ejemplo, si:

```
θ = { a/x, f(y)/y, x/z }
A = P(x) ∨ Q(y) ∨ R(z)
```

entonces:

```
Aθ = P(a) ∨ Q(f(y)) ∨ R(z)
```

Obsérvese que la sustitución es simultánea, de modo que obtenemos R(z) y no R(a).

Suponiendo que hay dos cláusulas, C<sub>1</sub> y C<sub>2</sub>, definidas como:

```
C1 = ~P(x)
C2 = P(f(x))
```

Una posible unificación de P es:

```
C1' = C1 { f(x)/x } = ~P(f(x))
```

Otra posible unificación para P son las dos sustituciones utilizando la constante a:

$$\begin{aligned} C_1'' &= C_1 \{ f(a)/x \} = \neg P(f(a)) \\ C_2' &= C_2 \{ a/x \} = P(f(a)) \end{aligned}$$

Observe que  $P(f(x))$  es más general que  $P(f(a))$ , porque hay un número infinito de casos de  $P(f(x))$  que pueden producirse al sustituir una constante por x. A una cláusula como  $C_1$  se le llama la **cláusula más general**.

En general, a una sustitución  $\theta$  se le llama **unificador** para un conjunto de cláusulas  $\{A_1, A_2, \dots, A_n\}$  si y sólo si  $A_1\theta = A_2\theta = \dots = A_n\theta$ . A un conjunto que tiene un identificador se le llama **unificable**. El **unificador más general (umg)** es aquel del que todos los demás unificadores son casos. Esto puede expresarse de manera más formal estableciendo que  $\theta$  es el unificador más general si y sólo si para cada unificador,  $\alpha$ , de un conjunto, hay una sustitución  $\beta$  tal que:

$$\alpha = \theta\beta$$

donde  $\theta\beta$  es la sustitución compuesta creada al aplicar primero  $\theta$  y luego  $\beta$ . El **algoritmo de unificación** es un método para encontrar el unificador más general, para un conjunto unificable de argumentos finito (Chang 73; Nilsson 80).

Para el ejemplo, con cláusulas (1a) a (3b), los resultados de la sustitución y la unificación se muestran en el árbol de refutación de la resolución de la figura 3.20. Como la raíz es nula, la conclusión negada es falsa y, por tanto, la conclusión "ninguna falla es un éxito" es válida.

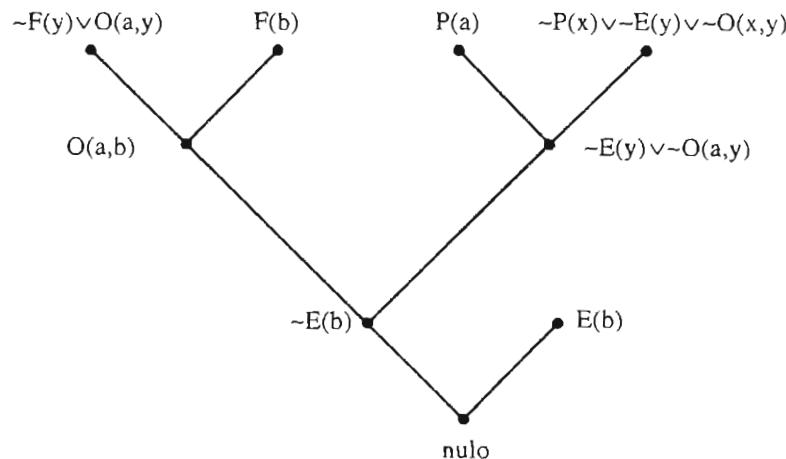


Figura 3.20 Árbol de refutación de la resolución para probar que ninguna falla es un éxito

Los ejemplos utilizados hasta ahora son simples y su resolución es directa, aunque tediosa para los seres humanos. Sin embargo, en muchas otras situaciones el proceso de resolución puede llegar a un punto muerto, de modo que es necesario el rastreo hacia atrás para probar cláusulas alternativas para resolución. Aunque la resolución es muy eficaz y es la base de PROLOG, puede resultar ineficiente para algunos problemas. Uno de los problemas de la resolución es que

no tiene una estrategia eficiente de búsqueda integrada y, por tanto, el programador debe proporcionar heurística, como el corte de PROLOG, para una búsqueda eficiente.

Se han investigado varias versiones modificadas de resolución, como la **preferencia de unidad**, la **resolución de entrada**, la **resolución lineal** y el **conjunto de soporte** (Amble 87). La principal ventaja de la resolución reside en que es una técnica eficaz y sencilla que resulta adecuada para muchos casos. Esto hace más fácil la construcción de sistemas mecánicos como PROLOG que un sistema que pretende implantar muchas reglas de inferencia diferentes. Otra ventaja de la resolución es que cuando tiene éxito, proporciona automáticamente una prueba al mostrar la secuencia de pasos que lleva a nulo.

### 3.14 ENCADENAMIENTO HACIA ADELANTE Y HACIA ATRÁS

A un grupo de varias inferencias que conectan un problema con su solución se le llama **cadena**. A una cadena que se explora o recorre desde un problema hasta su solución se le llama cadena hacia adelante. Otra manera de describir el encadenamiento hacia adelante es el razonamiento que va de los hechos a las conclusiones que se desprenden de ellos. Una cadena que se recorre de la hipótesis a los hechos que la sustentan, es una cadena hacia atrás. Otra manera de describir una cadena hacia atrás es desde el punto de vista de un objetivo que puede lograrse al satisfacer los subobjetivos. Como se muestra con esto, la terminología utilizada para describir el encadenamiento hacia adelante y hacia atrás depende del problema que se está analizando.

El encadenamiento puede expresarse con facilidad a partir de la inferencia. Por ejemplo, suponiendo que se tienen reglas del tipo *modus ponens*:

$$\frac{p \rightarrow q \\ p}{\therefore q}$$

que forma una cadena de inferencia, como la siguiente:

$$\begin{array}{l} \text{elefante}(x) \rightarrow \text{mamífero}(x) \\ \text{mamífero}(x) \rightarrow \text{animal}(x) \end{array}$$

Estas reglas pueden usarse en una cadena causal de inferencia hacia adelante que deduzca que Clyde es un animal, dado que Clyde es un elefante. La cadena de inferencia se ilustra en la figura 3.21. Observe que el mismo diagrama también ilustra el encadenamiento hacia atrás.

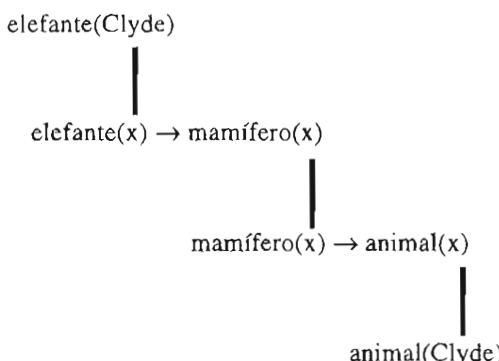
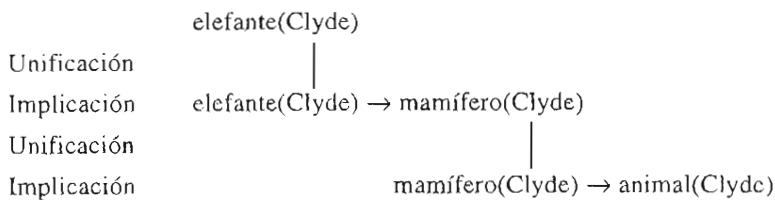


Figura 3.21 Cadena causal hacia adelante

En la figura 3.21, la cadena causal se representa con la secuencia de barras, que conectan el consecuente de una regla con el antecedente de la siguiente. Una barra también indica la unificación de las variables con los hechos. Por ejemplo, la variable  $x$  en el predicado elefante( $x$ ) debe unificarse primero con el hecho elefante(Clyde), antes de que pueda aplicarse la regla del elefante. La cadena causal es en realidad una secuencia de implicaciones y unificaciones, como se muestra en la figura 3.22.



**Figura 3.22 Cadena causal explícita**

El encadenamiento hacia atrás es el proceso inverso. Suponiendo que queremos probar la hipótesis animal(Clyde), el problema central del encadenamiento hacia atrás es encontrar una cadena que vincule la evidencia con la hipótesis. Al hecho elefante(Clyde), se le llama **evidencia** en el encadenamiento hacia atrás para indicar que se usa para sustentar la hipótesis, de la misma manera en que la evidencia en un juzgado se utiliza para probar la culpabilidad del acusado.

Como un ejemplo simple de encadenamiento hacia adelante y hacia atrás, suponga que está conduciendo un automóvil y de pronto ve una patrulla con las luces encendidas. Por encadenamiento hacia adelante puede inferir que la policía quiere detenerlo a usted o a alguien más; es decir, los hechos iniciales sustentan dos conclusiones posibles. Si el carro de policía se detiene justo atrás del suyo o si le hace señas, una inferencia adicional es que el oficial quiere detenerlo a usted y no a alguien más. Adoptando esto como hipótesis de trabajo, usted puede aplicar el encadenamiento hacia atrás para razonar el por qué.

Algunas posibles hipótesis intermedias son que conduce ebrio, con exceso de velocidad, con algún malfuncionamiento en el equipo y que conduce un vehículo robado. Ahora debe examinarse la evidencia para sustentar estas hipótesis intermedias. ¿Fue la botella de cerveza que arrojó por la ventana, ir a 160 en una zona con velocidad máxima de 50 kilómetros por hora, las luces traseras rotas o que las placas identifican el automóvil robado que está conduciendo? En este caso, cada pieza de evidencia sustenta una hipótesis intermedia y así todas son verdaderas. Cualquiera de estas hipótesis intermedias, o todas ellas, son razones posibles para probar la hipótesis de trabajo de que la policía quiere detenerlo.

Es útil visualizar el encadenamiento hacia adelante y hacia atrás como una ruta a través del espacio de problema en la que los estados intermedios corresponden a hipótesis intermedias bajo encadenamiento hacia atrás o conclusiones intermedias bajo encadenamiento hacia adelante. En la tabla 3.14 se resumen algunas de las características comunes del encadenamiento hacia adelante y hacia atrás. Observe que las características de esta tabla sólo pretenden ser una guía. Sin duda es posible diagnosticar en un sistema de encadenamiento hacia adelante y la planeación en un encadenamiento hacia atrás. En particular, la explicación se facilita en el encadenamiento hacia atrás debido a que el sistema puede explicar con facilidad y exactitud qué objetivo trata de cumplirse. En el encadenamiento hacia adelante, la explicación no se facilita tanto, porque los subobjetivos no se conocen de manera explícita sino hasta que se descubren.

En la figura 3.23 se ilustra el concepto básico del encadenamiento hacia adelante en un sistema basado en reglas. Las reglas se disparan por los hechos que satisfacen su antecedente o lados izquierdos (LI). Por ejemplo, para activar la regla  $R_1$ , deben satisfacerse los hechos  $B$  y  $C$ ; sin embargo, sólo el hecho  $C$  está presente, de modo que  $R_1$  no se activa. La regla  $R_2$  se

activa con los hechos C y D, que están presentes, de modo que R<sub>2</sub> produce el hecho intermedio H. Otras reglas que se satisfacen son R<sub>1</sub>, R<sub>6</sub>, R<sub>7</sub>, R<sub>8</sub> y R<sub>9</sub>. La ejecución de las reglas R<sub>8</sub> y R<sub>9</sub> produce las conclusiones del proceso de encadenamiento hacia delante. Estas conclusiones pueden ser otros factores, salida, etcétera.

Encadenamiento hacia adelante	Encadenamiento hacia atrás
Planeación, supervisión, control	Diagnóstico
Presente a futuro	Presente a pasado
Antecedente a consecuencia	Consecuencia a antecedente
Controlado por datos, razonamiento de abajo hacia arriba	Controlado por objetivos, razonamiento de arriba hacia abajo
Trabaja hacia adelante para encontrar cuáles soluciones se desprenden de los hechos	Trabaja hacia atrás para encontrar los hechos que sustenten la hipótesis
Búsqueda primero a lo ancho facilitada	Búsqueda primero a fondo facilitada
Los antecedentes determinan la búsqueda	Las consecuencias determinan la búsqueda
Explicación no facilitada	Explicación facilitada

Tabla 3.14 Algunas características del encadenamiento hacia adelante y hacia atrás

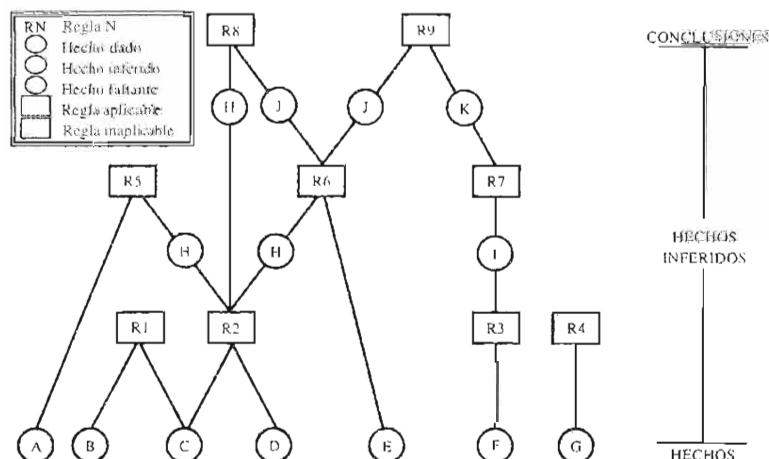


Figura 3.23 Encadenamiento hacia adelante

Al encadenamiento hacia adelante se le llama **razonamiento de abajo hacia arriba**, porque se razona a partir de la evidencia o los hechos de nivel más bajo a las conclusiones de nivel superior que se basan en los hechos. El razonamiento de abajo hacia arriba de un sistema experto es análogo a la programación convencional de abajo hacia arriba que se analiza en el capítulo 1. Los hechos son las unidades elementales del paradigma basado en el conocimiento, porque no pueden descomponerse en ninguna unidad más pequeña con significado. Por ejemplo, el hecho "pato" tiene significados definidos como un sustantivo o como un verbo. Sin embargo, si se divide aún más, el resultado son las letras p, a, t y o, que no tienen significado especial. En programas convencionales, las unidades básicas de significado son los datos.

Por costumbre, los constructores de nivel superior que están compuestos por los de nivel inferior se ponen en la parte superior. De modo que al razonamiento que parte de constructores de nivel superior, como las hipótesis, hacia abajo, hasta los hechos de nivel inferior que pueden sustentarla, se le llama **razonamiento de arriba hacia abajo** o encadenamiento hacia atrás. En la figura 3.24 se ilustra el concepto de encadenamiento hacia atrás. Para probar o refutar la hipótesis H, debe probarse por lo menos una de las hipótesis intermedias, H<sub>1</sub>, H<sub>2</sub> o

$H_3$ . Nótese que este diagrama se dibuja como un árbol Y-O para indicar que en algunos casos, como  $H_2$ , deben estar presentes todas las hipótesis de nivel inferior para sustentarla. En otros casos, como las hipótesis de nivel superior,  $H$ , sólo es necesaria una hipótesis de nivel inferior. En el encadenamiento hacia atrás, por lo general el sistema producirá evidencia del usuario para ayudar a probar o refutar la hipótesis. Esto contrasta con un sistema de encadenamiento hacia delante, en el que todos los hechos relevantes suelen conocerse por anticipado.

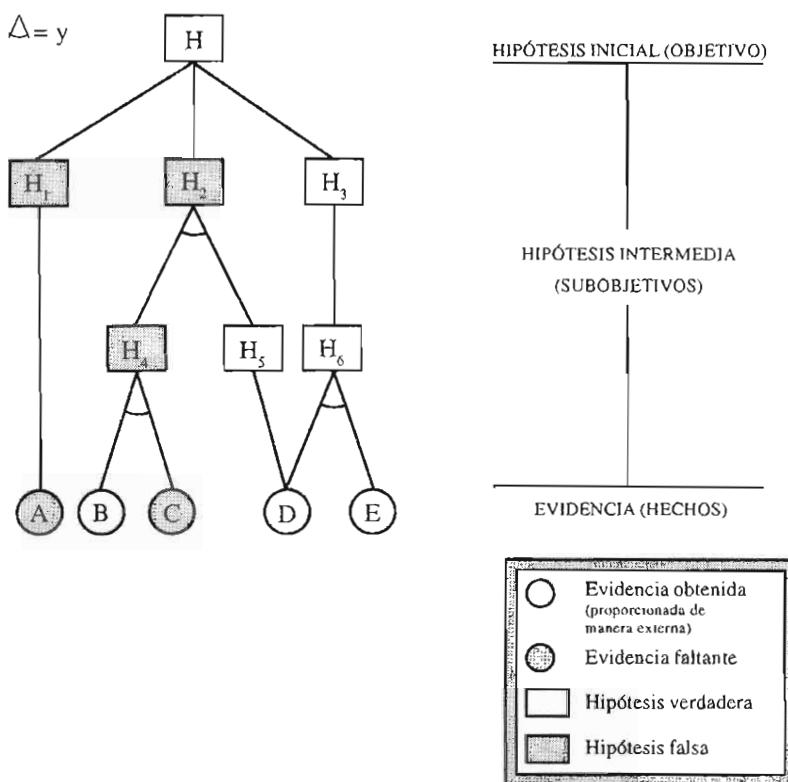


Figura 3.24 Encadenamiento hacia atrás

Si se vuelve a observar la figura 3.4 (página 100), se verá que la red de decisión está bien organizada para el encadenamiento hacia adelante. Las hipótesis de nivel superior son los tipos diferentes de frambuesas como la frambuesa con flores, la negra, la morada y la roja. La evidencia para sustentar estas hipótesis desciende al nivel inferior. Es posible escribir con facilidad las reglas para identificar una frambuesa. Por ejemplo:

SI las hojas son simples ENTONCES es frambuesa con flores

Para obtener evidencia es importante hacer las preguntas correctas; aquéllas que mejoran la eficiencia para determinar la respuesta correcta. Un requisito obvio para esto es que el sistema experto sólo debe hacer preguntas que se relacionen con la hipótesis que se está tratando de probar; aunque el sistema puede hacer cientos o miles de preguntas, debe pagarse un costo en tiempo y dinero para obtener la evidencia que permita responder las preguntas. Además, la acumulación de ciertos tipos de evidencia, como los resultados de pruebas mèdicas

cas, puede ser incómoda y tal vez peligrosa para el paciente (en realidad, es difícil pensar en un examen médico placentero).

En condiciones ideales, el sistema experto también debe permitir al usuario presentar voluntariamente la evidencia, aunque el sistema no la haya pedido. Al permitir que el usuario presente evidencia se acelera el proceso de encadenamiento hacia atrás y hace que el sistema sea más conveniente. La evidencia presentada de manera voluntaria puede llevar al sistema a omitir algunos vínculos en la cadena causal o seguir un método totalmente nuevo. La desventaja es que se requiere una programación más compleja del sistema experto, porque éste tal vez no siga una cadena vínculo por vínculo.

En la figura 3.25 se muestran buenas aplicaciones para el encadenamiento hacia adelante y hacia atrás. Por simplicidad, estos diagramas se dibujan como árboles en lugar de hacerlo como una red general. Una buena aplicación para el encadenamiento hacia adelante ocurre si el árbol es ancho y no muy profundo; esto se debe a que el encadenamiento hacia adelante facilita una búsqueda primero a lo ancho. Es decir, este encadenamiento es bueno si la búsqueda de conclusiones avanza nivel por nivel. Por el contrario, el encadenamiento hacia atrás facilita una búsqueda primero a fondo. Un buen árbol para este tipo de búsqueda es estrecho y profundo.

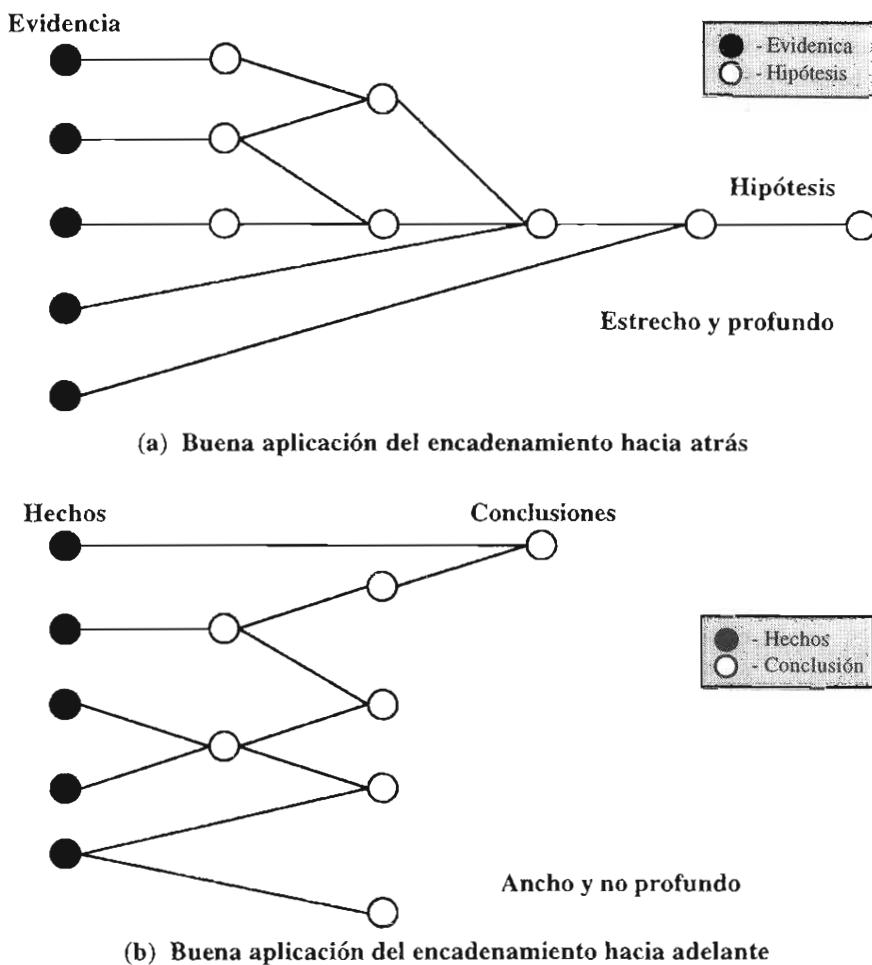


Figura 3.25 Encadenamiento hacia atrás y hacia adelante

Obsérvese que la estructura de las reglas determina la búsqueda de una solución. Es decir, la activación de una regla depende de los patrones que según su diseño, debe satisfacer. Los patrones del lado izquierdo determinan si una regla puede activarse mediante hechos. Las acciones del lado derecho determinan los hechos que se afirman y eliminan, de modo que afectan otras reglas. Existe una situación análoga en el encadenamiento hacia atrás, pero se utilizan hipótesis en lugar de reglas. Desde luego, una hipótesis intermedia puede ser tan sólo una regla que se equipara con su consecuencia en lugar de su antecedente.

Consideremos las siguientes reglas del tipo SI...ENTONCES:

```
SI A ENTONCES B
SI B ENTONCES C
SI C ENTONCES D
```

Si se da el hecho A y el mecanismo de inferencia está diseñado para comparar los hechos con los antecedentes, entonces los hechos intermedios B y C serán acertados y se alcanzará la conclusión D. Este proceso corresponde al encadenamiento hacia adelante.

En contraste, si el hecho (en realidad, la hipótesis) D es acertado y el mecanismo de inferencia compara los hechos con las consecuencias, el resultado corresponde al encadenamiento hacia atrás. En los sistemas diseñados para este tipo de encadenamiento, como PROLOG, el mecanismo incluye varias funciones, como el rastreo automático hacia atrás, para facilitar el encadenamiento.

El encadenamiento hacia atrás puede lograrse en un sistema de encadenamiento hacia adelante, y viceversa, si se rediseñan las reglas. Por ejemplo, las reglas anteriores para el encadenamiento hacia delante pueden rescribirse como:

```
SI D ENTONCES C
SI C ENTONCES B
SI B ENTONCES A
```

Ahora C y B se consideran subobjetivos o hipótesis intermedias que deben cumplirse para satisfacer la hipótesis D. La evidencia A es el hecho que indica el final de la generación de subobjetivos. Si hay un hecho A, entonces se sustenta a D y se considera verdadera bajo esta cadena de inferencia hacia atrás. Si no hay A, entonces no se sustenta a la hipótesis D y se considera falsa.

Una dificultad con este método es la eficiencia. Un sistema de encadenamiento hacia atrás facilita la búsqueda primero a fondo, mientras que un sistema de encadenamiento hacia delante facilita la búsqueda primero a lo ancho. Aunque es posible escribir una aplicación de encadenamiento hacia atrás en un sistema de encadenamiento hacia delante, y viceversa, el sistema no será tan eficiente en su búsqueda de una solución. La segunda dificultad es conceptual, el conocimiento obtenido del especialista tendrá que modificarse para cumplir las demandas del mecanismo de inferencia. Por ejemplo, un mecanismo de inferencia de encadenamiento hacia adelante compara el antecedente de las reglas y uno de encadenamiento hacia atrás compara la consecuencia. Es decir, si el conocimiento del especialista es, de manera natural, un encadenamiento hacia atrás, tendrá que reestructurarse por completo para aplicarlo en un modo de encadenamiento hacia delante, y viceversa.

### 3.15 OTROS MÉTODOS DE INFERENCIA

A veces, se usan otros tipos de inferencia con los sistemas expertos. Aunque estos métodos no son de propósito tan general, como la deducción, son muy útiles.

## Analogía

Además de la deducción y la inducción, otro método de inferencia eficaz es la **analogía**. La idea básica del razonamiento por analogía es tratar de relacionar situaciones previas como guías para las nuevas. La mayor parte de las criaturas vivas son muy buenas para aplicar el razonamiento por analogía en sus vidas, lo que es esencial debido al enorme número de situaciones nuevas que se encuentran en el mundo real. En lugar de tratar cada situación nueva como única, a menudo es útil tratar de relacionarlas con aquellas que ya son familiares. El razonamiento analógico se relaciona con la inducción, pero mientras que la inducción hace inferencias de lo específico a lo general de la misma situación, la analogía trata de hacer inferencias a partir de situaciones que no son iguales. La analogía no puede hacer pruebas formales como la deducción. En cambio, la analogía es una herramienta de razonamiento heurístico que a veces funciona.

Un ejemplo de razonamiento por analogía es el diagnóstico médico. Cuando usted acude a un doctor debido a un problema de salud, el doctor obtendrá información de usted y anotará los síntomas; si sus síntomas son idénticos o muy similares a los de otras personas con el problema X, el doctor puede inferir por analogía que usted tiene el problema X. Por supuesto, también puede usarse el encadenamiento hacia atrás.

Obsérvese que este diagnóstico no es una deducción, porque usted es único. El simple hecho de que alguien con el mismo problema muestre ciertos síntomas, no significa que usted mostrará esos síntomas; en cambio, el doctor supone que sus síntomas lo hacen análogo a una persona con los mismos síntomas y problema conocido. Este diagnóstico inicial es una hipótesis que los exámenes médicos pueden probar o refutar. Es importante tener una hipótesis de trabajo inicial, porque ésta reduce los miles de problemas potenciales. Sería muy caro y consumiría tiempo empezar a realizar cada examen posible sin una hipótesis inicial.

Como ejemplo de la utilidad del razonamiento por analogía, supóngase que dos personas participan en un juego llamado el Juego del 15 (Fischler 87). Eligen números entre el 1 y el 9 por turnos, con la restricción de que no puede usarse el mismo número dos veces, y gana la primera persona cuyos números sumen 15. Aunque a primera vista parece un juego que requiere algún razonamiento, una analogía puede facilitar el juego.

Veanos el tablero de gato con valores asignados a cada celda, como se muestra abajo:

6	1	8
7	5	3
2	9	4

Se trata de un **cuadro mágico** porque la suma de los valores en las filas, columnas y diagonales es una constante, 15. Este tablero con los valores del cuadro mágico puede considerarse una analogía del Juego del 15. Ahora resulta muy fácil jugarlo si lo considera un gato y luego traduce la estrategia ganadora al Juego del 15.

A este cuadro se le llama el **cuadro estándar de orden 3**. El término orden alude al número de filas o columnas de un cuadro, por tanto, sólo hay un cuadro de orden 3. Es posible crear otros cuadros mágicos si se gira o invierte el cuadro estándar. Otra manera de construir cuadros mágicos es agregando la misma constante a cada celda. Conocer esta información nos permite deducir la estrategia ganadora para el Juego del 18, donde los números deben seleccionarse del conjunto:

$$\{2, 3, 4, 5, 6, 7, 8, 9, 10\}$$

o el juego del 21, utilizando el conjunto:

$$\{3, 4, 5, 6, 7, 8, 9, 10, 11\}$$

Ahora podemos utilizar la inducción para inferir la estrategia ganadora para el juego del  $15 + 3N$ , donde  $N$  es cualquier número natural  $1, 2, 3, \dots$  al considerar al juego como análogo al tablero de gato, que es análogo a un cuadro mágico de valores:

$$\{1+N, 2+N, 3+N, 4+N, 5+N, 6+N, 7+N, 8+N, 9+N\}$$

Utilizando la analogía del cuadro de orden 3 para juegos de tres movimientos, por inducción podemos inferir que los cuadros de orden superior pueden usarse para juegos que implican más de tres movimientos. Por ejemplo, el cuadro de orden 4:

16	3	2	13
5	10	11	8
9	6	7	12
4	15	14	1

le permite jugar la estrategia ganadora del Juego del 34 de cuatro movimientos al considerarlo desde el punto de vista del gato. En contraste con el cuadro estándar de orden 3, hay 880 cuadros estándar de orden 4, lo que le permite muchos juegos más (Smith 84).

El razonamiento por analogía es una parte importante del razonamiento de sentido común, que es muy difícil para las computadoras (y los niños). Otras aplicaciones de la analogía han servido para el aprendizaje (Carbonell 82).

## Ensayo y error

Otro método de inferencia es la clásica estrategia de la AI de **ensayo y error**, a veces llamada generación y prueba, que requiere la generación de una solución probable y luego probarla para ver si la solución propuesta cumple todos los requisitos. Si la solución es satisfactoria, entonces se abandona, de otro modo genera una nueva solución, prueba de nuevo y así sucesivamente. Este método se utilizó en el primer sistema experto, DENDRAL, concebido en 1965, para ayudar a identificar estructuras moleculares orgánicas (Buchanan 78). Un espectrómetro de masas proporciona datos de una muestra desconocida y los introduce en DENDRAL, que genera todas las estructuras moleculares potenciales que podría producir el espectrograma desconocido. Luego DENDRAL prueba las moléculas más probables estimulando sus espectrogramas de masa y comparándolos con el original desconocido. Otro programa que utiliza generar y probar es el programa AM (Artificial Mathematician: matemático artificial), que infiere nuevos conceptos matemáticos (Lenat 82).

Con el fin de reducir el enorme número de soluciones posibles, ensayo y error suele utilizarse con un programa de planeación que limita las soluciones probables para generación. A esta variación se le llama **plan-ensayo y error** y se usa por eficiencia en muchos sistemas. Por ejemplo, el sistema experto de diagnóstico médico MYCIN también tiene la capacidad de planear un tratamiento terapéutico con medicamentos, después de que se ha diagnosticado la enfermedad de un paciente (Chancey 85). En esencia, un **plan** consiste en encontrar cadenas de reglas o inferencias que conectan un problema con una solución u objetivo, con la eviden-

cía para sustentarla. La planeación se hace con mayor eficiencia al buscar de manera simultánea hacia delante a partir de los hechos y hacia atrás a partir del objetivo.

El planeador MYCIN crea primero una lista de los medicamentos terapéuticos a los que el paciente es sensible. Para reducir interacciones indeseables entre los medicamentos, es mejor limitar el número de éstos que recibe un paciente, aunque se piense que sufre de varias infecciones diferentes. El generador toma la lista con prioridades del planeador y genera sublistas de uno o dos medicamentos, si es posible. Luego se prueban la eficacia de estas sublistas contra las infecciones, las alergias del paciente y otras consideraciones, antes de tomar la decisión de administrarlas.

Generar y probar también puede considerarse el paradigma de inferencia básico de reglas. Si se satisfacen los elementos condicionales de una regla, genera algunas acciones como nuevos hechos. El mecanismo de inferencia prueba estos hechos contra los elementos condicionales de las reglas en la base de conocimiento. Las reglas que se satisfacen se ponen en la agenda y la regla de mayor prioridad genera sus acciones, que entonces se prueban, etcétera. Por tanto, generar y probar produce una cadena de inferencia que puede llevar a la solución correcta.

## Abducción

La inferencia por **abducción** es otro método que se utiliza con frecuencia en la solución de un problema de diagnóstico. El esquema de abducción se parece al de *modus ponens*, pero en realidad es muy diferente, como se muestra en la tabla 3.15.

<b>Abducción</b>	<b>Modus ponens</b>
$p \rightarrow q$	$p \rightarrow q$
<u>q</u>	<u>p</u>
$\therefore p$	$\therefore q$

Tabla 3.15 Comparación de abducción y *modus ponens*

*Abducción* es otro nombre para un argumento falaz que analizamos en la sección 3.6, como la Falacia del recíproco. Aunque la abducción no es un argumento deductivo válido, es un método útil de inferencia y se ha usado en los sistemas expertos. Al igual que la analogía, que tampoco es un argumento deductivo válido, la abducción puede ser útil como una regla heurística de inferencia. Es decir, cuando no tenemos un método deductivo de inferencia, la abducción puede resultar útil, pero no está garantizado que funcione. Ni la analogía, ni generar y probar, ni la abducción son deductivos ni está garantizado que funcionen todo el tiempo. A partir de premisas verdaderas, estos métodos no pueden probar conclusiones verdaderas. Sin embargo, estas técnicas son útiles en la reducción del espacio de búsqueda al generar hipótesis razonables que luego pueden utilizarse con la deducción.

A veces se hace referencia a la abducción como un razonamiento de hechos observados para una mejor explicación (Reggia 85). Consideremos lo siguiente como ejemplo de abducción:

SI  $x$  es un elefante ENTONCES  $x$  es un animal  
 SI  $x$  es un animal ENTONCES  $x$  es un mamífero

Si sabemos que Clyde es un mamífero, ¿Podemos concluir que Clyde es un elefante?

La respuesta a esta pregunta depende de si estamos hablando de la realidad o de nuestro sistema experto. En la realidad no podríamos obtener esta conclusión con algún grado de certidumbre. Clyde podría ser un perro, un gato, una res o cualquier otro tipo de animal que sea un mamífero pero no un elefante. En realidad, al considerar cuántas especies de animales existen, sin conocer ninguna información adicional sobre Clyde, la probabilidad de que sea un elefante es muy baja.

No obstante, en un sistema experto que sólo incluye las reglas anteriores, diríamos por abducción con un 100 por ciento de certidumbre que si Clyde es un mamífero, entonces es un elefante. Esta inferencia se desprende de una **suposición de mundo cerrado** en la que suponemos que no existe nada más fuera del mundo cerrado de nuestro sistema experto, cualquier cosa que no pueda probarse en él se considera como falsa. Bajo la suposición del mundo cerrado, se conocen todas las posibilidades. Como el sistema experto sólo contiene estas dos reglas y sólo un elefante puede ser un mamífero, entonces si Clyde es un mamífero, debe ser un elefante.

Suponiendo que añadimos una tercera regla, como se muestra en seguida:

SI  $x$  es un perro ENTONCES  $x$  es un animal

Aun podemos operar nuestro sistema experto bajo una suposición de mundo cerrado; sin embargo, ya no podemos concluir con un 100 por ciento de certeza que Clyde es un elefante, todo lo que podemos asegurar es que Clyde es un elefante o un perro. Para decidir entre los dos, se necesita más información. Por ejemplo, si hay otra regla;

SI  $x$  es un perro ENTONCES  $x$  ladra

y hay evidencias de que Clyde ladra, las reglas pueden revisarse como se muestra a continuación:

- (1) SI  $x$  es un animal ENTONCES  $x$  es un mamífero
- (2) SI  $x$  ladra ENTONCES  $x$  es un animal
- (3) SI  $x$  es un perro ENTONCES  $x$  ladra
- (4) SI  $x$  es un elefante ENTONCES  $x$  es un animal

Ahora puede hacerse una cadena de inferencia abductiva hacia atrás, empleando las reglas (1), (2) y (3) para demostrar que Clyde debe ser un perro.

Una cadena de abducción hacia atrás no es lo mismo que el significado acostumbrado de encadenamiento hacia atrás. El término *encadenamiento hacia atrás* significa que estamos tratando de probar una hipótesis al buscar evidencia para sustentarla; y se usaría para tratar de probar que Clyde es un mamífero. Desde luego, para nuestro pequeño sistema no hay otras posibilidades. Sin embargo, podrían añadirse otras clasificaciones para reptiles, aves, etcétera.

Si se sabe que Clyde es un mamífero, la abducción podría utilizarse para determinar si Clyde es un elefante o un perro. El encadenamiento hacia delante se usaría si se sabe que Clyde es un elefante y queremos saber si es un mamífero. Como puede verse, la elección del método de inferencia depende de lo que habrá de determinarse. Como el encadenamiento hacia delante es deductivo, sólo se garantiza que sus conclusiones son válidas. En la tabla 3.16 se resume el propósito de cada una de las tres técnicas de inferencia.

Varios sistemas de AI y expertos han usado la abducción basada en marcos para solución de problemas de diagnóstico (Basili 85; Miller 82; Pople 82; Reggia 83). En estos sistemas, la base de conocimiento contiene **asociaciones causales** entre trastornos y síntomas. Se hacen inferencias utilizando ensayo y error de hipótesis para los trastornos.

**Tabla 3**  
atrás y

Inferencia	Inicio	Propósito
Encadenamiento hacia delante	Hechos	Conclusiones que deben seguir
Encadenamiento hacia atrás	Conclusión incierta	Hechos para sustentar la conclusión
Abducción	Conclusión verdadera	Hechos que deben seguir

Tabla 3.16 Resumen del propósito de encadenamiento hacia delante, encadenamiento hacia atrás y abducción

### Razonamiento no monotónico

En condiciones normales, la adición de nuevos axiomas a un sistema lógico significa que pueden probarse más teoremas porque hay más axiomas a partir de los cuales pueden derivarse nuevos teoremas. A esta propiedad de aumentar los teoremas con el aumento de axiomas se le conoce como **monotonía** y a los sistemas como la lógica deductiva se les llama **sistemas monotónicos**.

Sin embargo, puede surgir un problema si un axioma recién introducido contradice parcial o completamente un axioma previo. En este caso, los teoremas que se han probado tal vez ya no sean válidos. Por tanto, en un **sistema no monotónico**, los teoremas no necesariamente aumentan a medida que lo hace el número de axiomas.

El concepto de no monotonía tiene una aplicación importante en los sistemas expertos. A medida que se generan nuevos hechos (lo que es análogo a la comprobación de los teoremas), un sistema experto monotónico seguiría elaborando hechos. Puede ocurrir un problema importante si uno o más hechos se vuelven falsos porque un sistema monotónico no pudo tratar con cambios en la veracidad de axiomas y teoremas. Por ejemplo, suponiendo que hay un hecho que afirma la hora, en cuanto pase un segundo, el hecho anterior ya no es válido. Un sistema monotónico no sería capaz de tratar con la nueva situación. Otro ejemplo, sería que el sistema de identificación de una aeronave afirma el hecho de que un blanco era hostil, y más adelante, nueva evidencia prueba que el blanco es amigable. En un sistema monotónico, la identificación original no podría cambiar. Un sistema no monotónico permite la retracción de los hechos.

Como otra aplicación, supongamos que se quiere escribir un medio de explicación para un sistema experto que permitiría al usuario regresar a inferencias previas y explorar rutas de inferencia alternas del tipo “*¿Qué pasaría si?*”. Todas las inferencias hechas después de la inferencia deseada previa deben retractarse del sistema. Además de los hechos, las reglas también deben haberse extirpado del sistema, reintegrándose a la base del conocimiento para no monotonía. Otras complicaciones surgen en sistemas como OPS5, en los que las reglas pueden crearse automáticamente en el lado derecho de reglas durante la ejecución. Para la no monotonía, cualquier regla inferida hecha después de la inferencia deseada anterior tendría que haberse eliminado del sistema. El registro de todas las inferencias hechas puede consumir una gran cantidad de la memoria y reducir en gran medida la velocidad del sistema.

Para proporcionar una no monotonía, es necesario adjuntar una justificación o dependencia a cada hecho y regla, que explique la razón para creer en ella. Si se toma una decisión no monotónica, entonces el mecanismo de inferencia puede examinar la justificación de cada hecho y regla, para ver si todavía es creíble, y también para una posible restauración de las reglas extirpadas y los hechos retractados en los que se cree otra vez.

El problema de justificar hechos se señaló por primera vez en el **problema de cuadro**, (McCarthy 69). Éste es un término descriptivo que recibe su nombre del problema de identificar lo que ha cambiado o no ha cambiado en el cuadro de una película. Las películas se fotografían como una sucesión de fotos fijas, llamadas cuadros. Cuando se reproducen a 24 cuadros por segundo o más rápido, el ojo humano no puede distinguir los cuadros individuales y así se da la ilusión de movimiento. Un problema de cuadro para la AI es reconocer los

cambios en un ambiente a través del tiempo. Por ejemplo, considerando el problema del mono y los plátanos (figura 3.7), supongamos que el mono debe pararse en la caja roja para alcanzar los plátanos, de modo que la acción es “empuja la caja roja hasta abajo de los plátanos”. Ahora el problema de cuadro es ¿Cómo reconocemos que la caja todavía es roja después de la acción? Al empujar la caja no debe cambiar el ambiente. No obstante, otras acciones como “pinta la caja de azul” *cambiarán* el ambiente. En algunas herramientas de los sistemas expertos, un ambiente se conoce como **mundo** y consta de un conjunto de hechos relacionados (Filman 88). Un sistema experto puede llevar el registro de varios mundos para hacer razonamiento hipotético de manera simultánea. Al problema de mantener la corrección o el valor de verdad de un sistema se le llama **conservación de verdad** (Doyle 79). Ésta, o una variación llamada **conservación de la verdad basada en suposiciones**, es esencial para mantener la pureza de cada mundo al retractarse de hechos injustificados (de Kleer 86).

Consideremos el clásico ejemplo del pájaro Piolín como un ejemplo simple de razonamiento no monotónico. En ausencia de cualquier otra información, supondríamos que como Piolín es un pájaro, puede volar. Éste es un ejemplo de **razonamiento predeterminado**, algo muy parecido a las opciones predeterminadas usadas en las ranuras de los marcos. El razonamiento predeterminado puede considerarse como una regla que hace inferencias acerca de reglas, una **metarregla** que establece

SI se sabe que X no es cierto, y  
No hay evidencia de contradicción en X  
ENTONCES tentativamente concluye Y

Las metarreglas se analizan de manera más extensa en la siguiente sección.

En nuestro caso la metarregla tiene la forma específica:

X es la regla "Todos los pájaros pueden volar" y  
El hecho es que "Piolín es un pájaro"  
Y es la inferencia "Piolín puede volar"

Desde la perspectiva de las reglas de producción, esto puede expresarse como una regla en nuestra base de conocimiento, que indica:

SI X es un pájaro ENTONCES X puede volar

y el hecho que existe en la memoria de trabajo es:

Piolín es un pájaro

La unificación del hecho con el antecedente de la regla, produce la inferencia de que Piolín puede volar.

Ahora viene el problema. Supongamos que se añade un hecho adicional a la memoria de trabajo que dice que Piolín es un pingüino. Sabemos que los pingüinos no pueden volar, de modo que la inferencia de que Piolín puede volar es incorrecta. Por supuesto, debe haber una regla en el sistema que también establezca este conocimiento o el hecho se ignorará.

Para que un sistema siga siendo correcto, debe eliminarse la inferencia incorrecta. Sin embargo, tal vez no baste con esto si otras inferencias se basaron en la inferencia incorrecta. Es decir, tal vez otras reglas usaron la inferencia incorrecta como evidencia para dibujar inferencias adicionales, etcétera. Este es un problema de conservación de la verdad. La inferencia de que Piolín puede volar fue una **inferencia plausible**, basada en el razonamiento

predeterminado. (El término *plausible* significa “no imposible” y se analizará más a fondo en el capítulo 4.)

Una manera de permitir la inferencia no monotónica es al definir un operador de frase M, que puede definirse informalmente como “es consistente” (McDermont 82). Por ejemplo,

$$(\forall x) [Pájaro(x) \wedge M(Puede\_volar(x)) \rightarrow Puede\_volar(x)]$$

puede definirse como “Para toda x, si x es un pájaro y es consistente que un pájaro puede volar, entonces x puede volar”. Una manera más informal de establecer esto es “Casi todos los pájaros pueden volar”. El término “es consistente” significa que no hay contradicción con otro conocimiento. Sin embargo, se ha criticado esta interpretación como si en realidad estableciera que sólo los pájaros que no pueden volar son aquellos que se ha inferido que no pueden volar (Moore 85). Se trata de un ejemplo de **razonamiento autoepistemológico**, que significa literalmente razonamiento sobre el conocimiento propio. Ambos tipos de razonamiento, predeterminado y autoepistemológico se usan en el **razonamiento de sentido común**, que los seres humanos suelen aplicar muy bien pero que resulta muy difícil para las computadoras.

El razonamiento autoepistemológico es un razonamiento acerca del propio conocimiento que es distinto del conocimiento en general. Por lo general, una persona puede hacer esto muy bien porque conoce los límites de su conocimiento. Por ejemplo, supongamos que un extraño llega frente a usted y le dice que fue su cónyuge, de inmediato sabría (a menos que padeciera amnesia) que esto no es cierto porque no lo conoce. La metarregla general del razonamiento autoepistemológico es:

SI no tengo conocimiento de X  
ENTONCES X es falso

Observe cómo el razonamiento autoepistemológico depende de la suposición de mundo cerrado. Se supone que cualquier hecho que no se conoce es falso. En este razonamiento, el mundo cerrado es su conocimiento propio.

Tanto el razonamiento autoepistemológico como el predeterminado son no monotónicos. Sin embargo, las razones son diferentes; el razonamiento predeterminado es no monotónico porque es **derrotable**. Este término significa que cualquier inferencia es tentativa y tal vez tenga que retirarse cuando se dispone de nueva información. No obstante, el razonamiento autoepistemológico puro no es derrotable debido a la suposición del mundo cerrado que declara que usted ya conoce todo el conocimiento verdadero. Por ejemplo, como la gente casada sabe quién es su cónyuge (a menos que quieran olvidarlo), una persona casada no aceptará que un extraño fue su cónyuge, aunque se lo diga. Como la mayoría reconoce que no tiene recuerdos perfectos, no se adhieren al razonamiento autoepistemológico puro. Por supuesto, las computadoras no tienen este problema.

El razonamiento autoepistemológico es no monotónico debido al significado de la afirmación autoepistemológica **sensible al contexto**. El término *sensible al contexto* significa que el significado cambia con el contexto. Como ejemplo simple de sensibilidad al contexto, puede considerarse la manera en que se pronuncia la palabra “sí” en las dos frases:

Claro que sí quiero ese aumento  
Creo que si te voy a pagar

La pronunciación de “sí” es sensible al contexto.

Ahora consideremos un sistema que consta de los dos siguientes axiomas:

$$(\forall x) [Pájaro(x) \wedge M(Puede\_volar(x)) \rightarrow Puede\_volar(x)] \\ Pájaro(Piolín)$$

En este sistema lógico, *Puede\_volar(Piolín)* es un teorema derivado de la unificación de Piolín con la variable *x* y la implicación.

Ahora supongamos que se agrega un nuevo axioma que establece que Piolín no puede volar y, por tanto, contradice el teorema previamente derivado.

$\neg Puede\_volar(Piolín)$

El operador *M* debe cambiar su operación sobre su argumento sobre todo porque ahora *M(Puede\_volar(Piolín))* no es consistente con el nuevo axioma. En este nuevo contexto de tres axiomas, el operador *M* no daría un resultado VERDADERO para *Puede\_volar(Piolín)* porque entra en conflicto con el nuevo axioma. El valor devuelto por el operador *M* debe ser FALSO en el nuevo contexto, y así la conjunción es FALSA. Por ello, no hay implicación para producir el teorema *Puede\_volar(Piolín)* y no hay conflicto.

Una manera de implantar esto por medio de reglas es como sigue:

SI *x* es un pájaro Y *x* es típico  
 ENTONCES *x* puede volar  
 SI *x* es un pájaro Y *x* es atípico  
 ENTONCES *x* no puede volar

Piolín es un pájaro  
 Piolín es atípico

Observe que este sistema no invalida la conclusión *Puede\_volar(Piolín)*, sino que más bien evita que se dispare la regla incorrecta. Éste es un método mucho más eficiente de conservación de la verdad que si tenemos una regla y el axioma especial  $\neg Puede\_volar(Piolín)$ . Ahora tenemos un sistema más general que puede manejar fácilmente a otros pájaros no voladores sin que se tengan que agregar continuamente nuevas inferencias, lo que se supone que hace el sistema.

### 3.16 METACONOCIMIENTO

El programa Meta-DENDRAL usa la inducción para inferir nuevas reglas de estructura química. Meta-DENDRAL es un intento por sobreponerse al cuello de botella del conocimiento de tratar de extraer reglas estructurales moleculares de los especialistas humanos. Meta-DENDRAL ha tenido mucho éxito en redescubrir reglas conocidas e inferir nuevas.

Por ejemplo, la siguiente metaregla se toma del programa de adquisición de conocimiento TEIRESIAS para MYCIN, el sistema experto para diagnosticar infecciones sanguíneas y meningitis (Feigenbaum '79).

METARREGLA 2

SI

El paciente es un anfitrión comprometido, y

Hay reglas que mencionan pseudomonas en su premisa, y

Hay reglas que mencionan klebsiellas en su premisa

ENTONCES

Hay evidencia que sugiere (.4) que lo primero  
debe hacerse antes de lo último

(El número .4 en la acción de la regla es un grado de certidumbre y se analizará en un capítulo posterior.)

TEIRESIAS adquiere interactivamente el conocimiento de un perito. Si MYCIN hace un diagnóstico incorrecto, entonces TEIRESIAS llevará al especialista de regreso por la cadena del razonamiento incorrecto hasta que establezca el punto donde inicia. Mientras recorre hacia atrás la cadena de razonamiento, TEIRESIAS también interactúa con el perito para modificar reglas incorrectas o adquirir nuevas reglas.

El conocimiento sobre nuevas reglas no se introduce inmediatamente en MYCIN. En cambio TEIRESIAS verifica si la nueva regla es compatible con reglas similares. Por ejemplo, si la nueva regla describe la forma en que una infección entra en el cuerpo y otras reglas aceptadas tienen un elemento condicional que establece la puerta de entrada entonces la nueva regla deberá tener el mismo elemento. Si la nueva regla no establece la puerta de entrada, entonces TEIRESIAS consultará al usuario acerca de esta discrepancia. TEIRESIAS tiene un patrón de **modelo de regla**, de reglas similares, conocidas y trata de adecuar la nueva regla a él. En otras palabras, el modelo de regla es el conocimiento que TEIRESIAS tiene acerca de su conocimiento. Una situación análoga para una persona ocurriría si va con un distribuidor de automóviles para comprar un automóvil nuevo y el vendedor trata de venderle un automóvil con tres ruedas.

El metaconocimiento de TEIRESIAS es de dos tipos: la METARREGLA 2, descrita anteriormente, es una estrategia de control que indica la manera en que se aplican las reglas. Por otra parte, el tipo de modelo de regla de metaconocimiento determina si la nueva regla tiene la forma apropiada para introducirse en la base del conocimiento. En un sistema experto basado en reglas, al hecho de determinar si la nueva regla está en la forma correcta se le llama **verificación** de la regla. Al hecho de determinar si una cadena de inferencias correctas lleva a la respuesta correcta se le llama **validación**. Ambas son tan interdependientes que suele usarse el acrónimo **VyV** para referirse a ellas. Una definición más coloquial de los términos, tomada de la ingeniería de software es (Boehm 84):

Verificación: "¿Estoy elaborando el producto correctamente?"  
Validación: "¿Estoy elaborando el producto correcto?"

V y V se analizará con más detalle en un capítulo posterior.

## 3.17 RESUMEN

En este capítulo se expusieron los métodos de inferencia para los sistemas expertos de uso frecuente. La inferencia es particularmente importante en ellos porque es la técnica con que resuelven los problemas. Se analizó la aplicación de árboles, gráficas y la representación de conocimiento. Se ilustraron también las ventajas de estas estructuras para desarrollar inferencias.

Se analizó la lógica proposicional y de predicados de primer orden. Las tablas de verdad y las reglas de inferencia se describieron como medios para probar teoremas y afirmaciones. Se mencionaron las características de los sistemas lógicos: estar completos, ser sólidos y permitir decisiones.

El método de resolución para comprobar teoremas se analizó para la lógica proposicional y la de predicados de primer orden. Los nueve pasos relacionados con la conversión de una

fórmula bien adecuada a la forma de cláusula se ilustró con un ejemplo. La aplicación de la función de Skolem, la forma normal prenex y la unificación se analizaron en el contexto de la conversión de las fbf a forma de cláusula.

Se analizó además otro importante método de inferencia, la analogía. Aunque no se usa ampliamente en sistemas expertos debido a la dificultad para implantarla, la gente la utiliza con frecuencia y debe considerarse en el diseño de sistemas expertos. También se expuso la de ensayo y error con un ejemplo de su uso en MYCIN. Se describió la aplicación de metaconocimiento en TEIRESIAS y su relación con la verificación y validación de sistemas expertos.

## PROBLEMAS

- 3.1 Escriba un programa de árbol de decisión que sea de autoaprendizaje. Enséñele el conocimiento de los animales de la figura 3.3.
- 3.2 Escriba un programa que traducirá automáticamente el conocimiento almacenado en un árbol de decisión binario en reglas del tipo SI ... ENTONCES. Pruebelo con el árbol de decisión animal del problema 3.1.
- 3.3 Dibuje una red semántica que contenga el conocimiento sobre frambuesas de la figura 3.4.
- 3.4 Dibuje el diagrama de estado que muestre una solución al problema clásico del granjero, el zorro, la cabra y la calabaza. En este problema, un granjero, un zorro, una cabra y una calabaza se encuentran en el banco de un río. Debe utilizarse un bote para transportarlos a la otra orilla. Sin embargo, sólo caben dos en el bote, al mismo tiempo (y sólo el granjero puede remar). Si se deja al zorro con la cabra y el granjero no está presente, el zorro se comerá a la cabra. Si se deja a la cabra sola con la calabaza, la cabra se la comerá.
- 3.5 Dibuje un diagrama de estado para un problema de viaje bien estructurado que tenga
  - (a) tres métodos de pago: efectivo, cheque y tarjeta
  - (b) los intereses del viajero: sol, nieve
  - (c) cuatro destinos posibles, dependiendo de los intereses y el presupuesto del viajero
  - (d) tres tipos de transporte.

Escriba reglas SI ... ENTONCES para aconsejar al viajero a dónde ir, dependiendo del presupuesto y los intereses. Elija destinos reales y encuentre los costos para ir allí desde el lugar en que se encuentra.

- 3.6 Determine si los siguientes son argumentos válidos o no:
  - (a)  $p \rightarrow q, \neg p \rightarrow r, r; \therefore p$
  - (b)  $\neg p \vee q, p \rightarrow (r \wedge s), s \rightarrow q; \therefore q \vee r$
  - (c)  $p \rightarrow (q \rightarrow r), q; \therefore p \rightarrow r$
- 3.7 Utilizando el procedimiento de decisión con diagrama de Venn, determine si los siguientes son silogismos válidos o no:
  - (a) AEE-4
  - (b) AOO-1
  - (c) OAO-3
  - (d) AAI-1
  - (e) OAI-2