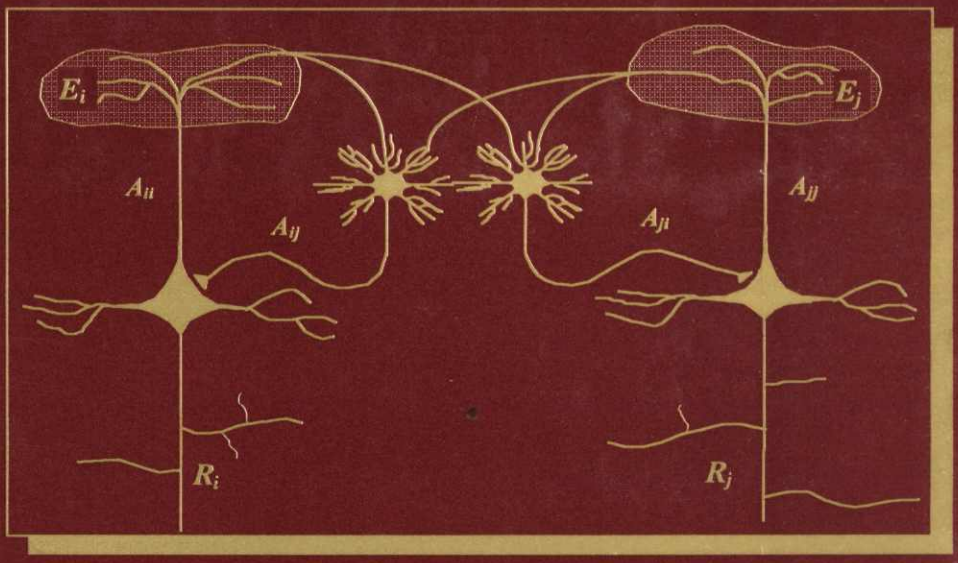
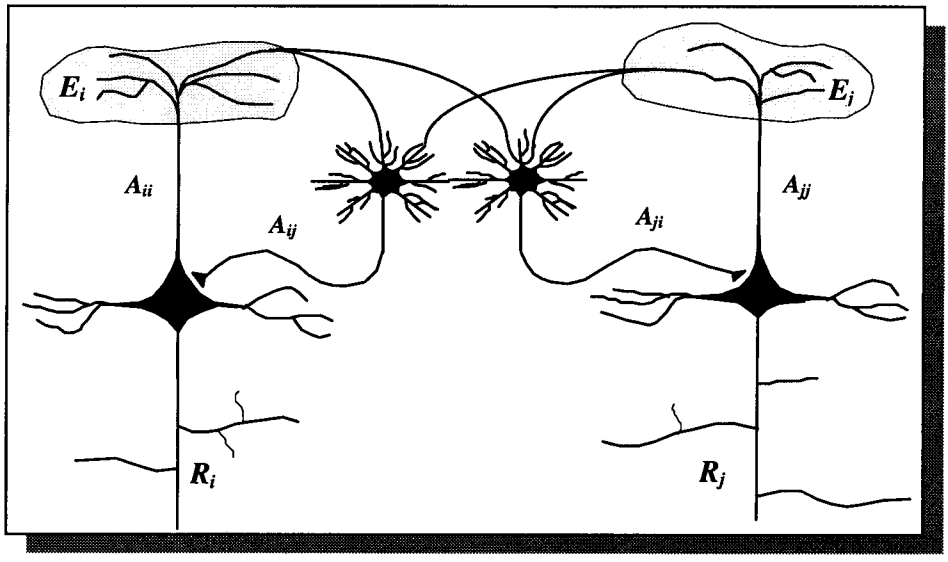


ASPECTOS BÁSICOS DE LA INTELIGENCIA ARTIFICIAL

J. Mira
A. E. Delgado
J. G. Boticario
F. J. Díez

UNED

SANZ Y TORRES



ASPECTOS BÁSICOS DE LA INTELIGENCIA ARTIFICIAL

J. Mira
A. E. Delgado
J. G. Boticario
F. J. Díez

UNED

SANZ Y TORRES

ASPECTOS BÁSICOS DE LA INTELIGENCIA ARTIFICIAL

Pinos Alta, 49 - 28029 Madrid
Teléfs.: 314 52 51 - 314 55 99
Fax: 323 15 59

ISBN: 84-88667-13-2
Depósito legal: M. 5.879-1995

Impreso por: Impresos y Revistas, S. A. (IMPRESA)
Herreros, 42. Políg. Ind. Los Ángeles
GETAFE (Madrid)

ÍNDICE

Presentación

Este libro está pensado como texto básico para dos grupos distintos de alumnos de la Escuela Universitaria de Informática de la UNED: los que cursan la asignatura de ***Introducción a la Inteligencia Artificial***, de segundo curso de Informática de Sistemas y los alumnos que estudien alguna de las asignaturas optativas de tercer curso en la línea temática de *Inteligencia Artificial* (*IA*). Es decir, alguna de las asignaturas de Sistemas Basados en Conocimiento (I y II), Razonamiento y Aprendizaje, Programación Orientada a la *IA* y/o Percepción y Control Basados en Conocimiento.

Para cada uno de estos grupos hay una selección de temas y una guía de lectura diferentes, de acuerdo con los planes de estudio y los contenidos de la guía de curso. Los alumnos de Introducción a la Inteligencia Artificial de segundo curso, encontrarán su programa en los temas 3 a 8, ambos inclusive, dedicados al estudio de las técnicas de búsqueda y a los formalismos de representación del conocimiento (lógica, reglas, redes, marcos y guiones). También deberán considerarse los temas 1, 2 y 9 a un nivel más superficial. Los dos primero temas estudian la perspectiva histórico-conceptual de la *IA* y los aspectos metodológicos. El tema 9 está dedicado al estudio de los Sistemas Expertos y encontrarán su aplicación más adecuada como temas básicos en las asignaturas dedicadas en tercero al estudio de los sistemas basados en conocimiento.

Los temas 10 y 11 están dedicados al estudio del aprendizaje y a la computación neuronal. Son temas de contenido más avanzado que servirán de referencia en las asignaturas *Razonamiento y Aprendizaje* y *Sistemas Basados en Conocimiento II* (Redes Neuronales).

Junto a las funciones de libro de texto de la UNED que hemos comentado los contenidos del libro han sido organizados de forma tal que también puede ser interesante su lectura para alumnos de otras universidades y para cualquier profesional interesado en el campo de la Inteligencia Artificial.

Para estos posibles lectores de ámbito general y para nuestros alumnos de la UNED, queremos resaltar el carácter distintivo de nuestro propósito a la hora de seleccionar los contenidos y los puntos de vista que aquí se manifiestan. Este carácter viene determinado por el énfasis en la metodología y por la visión integradora entre las perspectivas Simbólica y Conexionista de la *IA*.

Los autores.

1 PERSPECTIVA HISTÓRICA Y CONCEPTUAL

J. Mira y A.E. Delgado

En este primer capítulo intentamos acercarnos al concepto de ***Inteligencia Artificial*** *(**IA**) usando criterios en extenso y en intenso y estableciendo la distinción entre las perspectivas de "Ciencia de lo Natural" (análisis) y "Ciencia de lo Artificial" (síntesis). La distinción entre la perspectiva conexionista, y la simbólica se establecerá en el siguiente capítulo. La IA, que nació conexionista se convirtió en simbólica en la década de los sesenta y ha vuelto a considerar a las redes neuronales como un complemento necesario a partir de la década de los ochenta.*

Usamos el comentario de las distintas etapas históricas (neurocibernética, computación previa a la IA, heurística y micromundos, énfasis en el conocimiento, aprendizaje y conexionismo) para ofrecer una visión amplia del campo que justifique el estado actual. La historia que se cuenta no es completa ni neutra. Dicho de otra forma, hacemos énfasis en aquellos hechos que nos han parecido más significativos por proximidad conceptual. Por otro lado, aprovechamos la evolución historia para introducir y comentar superficialmente el contenido del resto de los capítulos dedicados a la representación computacional del conocimiento y al estudio de los sistemas expertos, el aprendizaje simbólico y las redes neuronales. Finalmente, al comentar las distintas aportaciones históricas, introducimos el test de Turing como método experimental de acumular inferencia inductiva sobre el grado de conocimiento codificado en un programa de IA.

La redacción del capítulo está pensada para que admita al menos dos tipos de lectura. Una inicial, introductoria, en la que el lector no debe preocuparse si no conoce con precisión el significado de todos los términos (sistema experto, conexionismo, heurística, red semántica, etc...). Al estudio de estos términos se

dedicarán capítulos enteros y es después del estudio de los capítulos correspondientes a estos temas cuando tiene sentido volver a una segunda lectura de este capítulo inicial para dar coherencia al nuevo conocimiento adquirido e integrarlo en la idea global de IA que aquí se pretende transmitir.

1.1 CONCEPTO DE INTELIGENCIA ARTIFICIAL

El propósito de la Inteligencia Artificial (*IA*) es hacer computacional el conocimiento humano no analítico por procedimientos simbólicos, conexionistas o híbridos. Para el conocimiento analítico existen otras ramas de la computación que estudian los métodos y las técnicas adecuadas para su representación formal y posterior desarrollo de los programas de ordenador correspondientes.

Para conseguir una visión razonablemente amplia del contenido de la *IA* usaremos criterios ***extensionales*** (es decir, proporcionando una relación lo más completa posible de los temas que estudia la *IA*), junto a otros criterios ***intensionales***, que establecen las reglas de clasificación, de forma que al enfrentarnos con un problema computacional (o con un método de solución) específico podamos concluir si pertenece o no al campo de la *IA* en función, por ejemplo, de cuán completo, analítico o complejo es el conocimiento necesario para la solución del problema. Una mezcla adecuada de argumentos en extenso e intenso nos ayudará a especificar qué entendemos por *IA*.

En *IA* existen dos perspectivas básicas, al igual que en prácticamente todas las ramas del conocimiento humano:

*a) IA como ciencia de lo natural (**análisis**).*

*b) IA como ciencia de lo artificial (ingeniería de **síntesis**).*

A su vez en ambas ramas cooperan dos paradigmas, que constituyen dos formas de analizar un proceso y, esencialmente, dos metodologías de síntesis de una solución:

*c) Computación **simbólica**, de grano grueso y programable.*

*d) Computación **conexionista**, de grano pequeño y autoprogramable por aprendizaje. Aquí, parte del conocimiento está en la propia estructura de la red.*

Comentaremos primero los puntos a) y b) y dejaremos para más adelante la comparación entre las perspectivas *simbólica* y *conexionista*, aunque ya adelantamos que toda computación es conexionista a nivel de procesadores. El simbolismo nace en el dominio del observador, al asociar tablas semánticas a las *estructuras de datos*, a los *procesos* que las manejan y al *control* de esos procesos.

La *IA* como ciencia de lo *natural* es una ciencia de *análisis* en su doble aspecto correlacional y teórico. Su objeto formal no es la materia ni la energía, sino el conocimiento que, al igual que la información, es pura forma. Por correlacional se entiende el registro de la ocurrencia simultánea o sucesiva de diferentes procesos inferenciales a los que se asocian magnitudes simbólicas. El procedimiento teórico busca una "explicación" de esa correlación en términos de un conjunto de leyes generales de un nivel superior, que permiten predecir lo que ocurriría en otros casos no observados. Es decir, busca un modelo del conocimiento humano organizado en general en varios niveles (estático, dinámico y estratégico) y susceptible de ser usado en predicción, como hacen usualmente otras ciencias.

La fenomenología de la *IA* es el conjunto de hechos asociados a los procesos cognoscitivos y a los principios organizacionales y estructurales que dan lugar al comportamiento humano, al que usualmente etiquetamos como *inteligente*. Por eso, su objeto formal coincide en parte con el de la neurología y la ciencia cognoscitiva y su método pretende aproximarse al de la física, integrando teoría y experimento. Para la formalización de ese conocimiento la *IA* usa todas las herramientas que tiene a su alcance (la lógica, las matemáticas, la algorítmica y la heurística) junto con otras nuevas que han nacido como consecuencia de sus necesidades específicas, como el modelado del conocimiento, las técnicas de evaluación de prototipos o los mecanismos de explicación y aprendizaje simbólico. Se reconoce también que es probable que no dispongamos todavía de todas las herramientas formales adecuadas para la representación computacional de los aspectos más genuinos del comportamiento del sistema nervioso, por ejemplo la comprensión y producción del lenguaje natural. Finalmente, el laboratorio de la *IA* es la simulación. Los experimentos (las tareas) se programan y el resultado de esa programación se *evalúa*. Finalmente, como consecuencia de esa evaluación, se reformula el problema o se rediseñan los mecanismos de inferencia y se sacan conclusiones para nuevos prototipos.

Lo distintivo de la *IA* como ciencia, en relación con la física, es que ahora la información y el conocimiento se han convertido en el nuevo objeto formal de la ciencia, como alternativa a la materia y la energía. Al ser el conocimiento el objeto formal de la *IA* estará sujeto a la *observación taxonómica*, *análisis*, *modelado*, *explicación* y *transformación*. Lo que se busca a largo plazo es una teoría del conocimiento computable con capacidad predictiva análoga a la de una ley física. Es decir, impersonal, comprobable experimentalmente y transferible. Evidentemente la tarea no es sencilla.

La *IA como ciencia de lo artificial* es una ciencia de ***síntesis*** que aspira a convertirse en una ingeniería en sentido estricto, con la metodología y la eficacia propias de las otras ingenierías que manejan la materia y la energía. Ahora se parte de un conjunto de especificaciones funcionales y se busca la síntesis de un sistema (programa más máquina) que las satisfaga.

Cuando la ingeniería tiene que ver con la materia o la energía, el propósito (la necesidad a satisfacer) será diseñar un puente, un automóvil, un amplificador electrónico o una calculadora digital. Se empieza entonces con un conjunto de especificaciones funcionales del problema y un estudio de las posibilidades de síntesis a partir de las técnicas disponibles. Después se propone un procedimiento algorítmico para alcanzar la realización física y, finalmente, se evalúa la solución. Este es el procedimiento usual en ingeniería. A partir de un conjunto de especificaciones, en general insuficientes, y usando las técnicas, métodos y materiales disponibles, la ingeniería propone un procedimiento constructivo que nos lleva a un sistema artificial que satisface las necesidades iniciales.

En *IA* trabajamos con información y conocimiento, y ambos son pura forma, totalmente independiente del sistema físico que las soporta. Por consiguiente la "necesidad" o el propósito del diseño siempre tiene que ver con la identificación, modelado, representación y uso de ese conocimiento en inferencia. El equivalente a la necesidad de partida en la ingeniería convencional es aquí una tarea de percepción, decisión, planificación o control. El resultado que buscamos es un *programa* de ordenador sobre una máquina específica desarrollado a partir de un modelo del conocimiento que supuestamente usaba el operador humano que realizaba esa tarea.

Las tareas que aborda la *IA* de síntesis son tareas de alto nivel, correspondientes a lo que en los expertos humanos llamamos, en general, procesos cognoscitivos y pueden clasificarse en tres grandes grupos ordenados en grado de dificultad creciente. Esta dificultad se evalúa en función de la

diversidad en las entidades y las relaciones que los componen y en el grado de semántica necesario para su descripción de forma completa e inequívoca:

1. *Dominios formales.*
2. *Dominios técnicos.*
3. *Funciones básicas y genuinas del comportamiento humano.*

Las tareas en dominios formales toman la forma genérica de "*solucionadores de problemas*" mediante *búsquedas* en un espacio de estados de conocimiento y pueden ser juegos (por ejemplo ajedrez) o problemas *lógico-matemáticos* (por ejemplo deducción de teoremas, geometría, integración simbólica, etc...). Son en general tareas en las que no hay imprecisión en el conocimiento, hay pocos elementos y su comportamiento se puede describir de forma completa e inequívoca. Se trata de "micromundos formales", que corresponden a simplificaciones muy fuertes del mundo real. Los resultados que se obtienen aquí son difíciles de extrapolar pero los problemas que aparecen, las técnicas de solución y las limitaciones que emergen constituyen un aprendizaje muy valioso para abordar posteriormente problemas del mundo real. Este primer apartado (dominios formales) marcó la etapa inicial de la *IA* y nos dejó, entre otras cosas, los procedimientos de búsqueda (ciega o heurística) como tarea genérica de validez prácticamente universal. Las técnicas básicas de búsqueda, los espacios de representación, la inferencia, y el uso del conocimiento del dominio, junto con algunos ejemplos sobre integración simbólica se estudiarán en los capítulos 3 y 4.

Las tareas que usan conocimiento científico-técnico en dominios estrechos tienen que ver, por ejemplo, con el *diagnóstico médico*, la *detección de fallos*, la *planificación* de trayectorias en robots, etc... En la mayor parte de estos casos, la tarea a sintetizar admite una representación dentro de una jerarquía de tareas genéricas de análisis (identificación, monitorización, clasificación, diagnóstico o predicción), de modificación (reparación, control, supervisión, aprendizaje) o de síntesis (diseño jerárquico o incremental, configuración, planificación o modelado) que son válidas en muchas aplicaciones con sólo modificar la parte del conocimiento que hace referencia a entidades específicas del dominio de la aplicación. Lo característico de estas tareas científico-técnicas es el carácter limitado del conocimiento que manejan (dominios "estrechos") y la posibilidad

de formalizar ese conocimiento con las técnicas disponibles (representación mediante reglas o marcos e inferencia por concatenación de reglas o activación de ciertos campos de los marcos). Esta formalización es posible porque hay poca variabilidad en los conceptos y existen ciertas leyes que si bien no son tan fuertes como las de la física (por ejemplo), son suficientes para codificar los razonamientos del experto humano que hacen referencia a esa tarea.

Este segundo apartado (tareas técnicas en dominios estrechos), ha crecido espectacularmente en los últimos años y ha dado lugar a la *Ingeniería del Conocimiento* que, en general, se desconecta de las raíces biológicas-cognoscitivas de la *IA* y busca procedimientos de síntesis de sistemas con las siguientes facetas:

1. Se parte de la descripción de la tarea a nivel de conocimiento, en el sentido de Allen Newell, del que hablaremos en el capítulo 2. Para ello es necesario realizar un proceso de obtención de ese conocimiento a partir del experto humano que lo posee. Veremos en el capítulo 9 que esta obtención equivale de hecho a una reconstrucción en forma de modelo.

2. Se busca una representación de ese conocimiento separándolo de los mecanismos de aplicación del mismo (inferencia) de forma que pueda acumularse por procedimientos incrementales. Este propósito sólo se consigue hasta cierto punto y, además, no sería del todo deseable una separación total. En los capítulos 5, 6, 7 y 8 estudiaremos las formas de representación más usuales (lógica, reglas, redes, marcos y guiones) y veremos cómo cada forma de representación lleva implícitas al resto de las características (modelado, uso, capacidad de explicación, etc...). Así, cada representación se proyecta de hecho en un procedimiento de programación de una tarea genérica.

3. Se seleccionan las técnicas adecuadas para su implementación y se desarrolla un primer prototipo con la ayuda de ciertas herramientas para la adquisición del conocimiento y de entornos de programación. Veremos estos temas en el capítulo 9.

   En la perspectiva conexionista de la *IA*, la arquitectura genérica de la red neuronal es la de un clasificador multicapa, con capas especializadas en las que se ha inyectado todo el conocimiento disponible para inicializar la red. El resto se adquiere por aprendizaje supervisado o no supervisado.

4. Se hace énfasis en el carácter de *ingeniería* buscando procedimientos sistemáticos de implementación, evaluación y refinamiento de esos

prototipos. Estos procedimientos deben ser explícitos, reproducibles y parcialmente independientes del dominio. No es tampoco necesario que estos procedimientos estén inspirados en lo que se conoce sobre la solución humana al problema.

5. Se usan lenguajes y entornos de programación que facilitan el desarrollo rápido y eficiente de aplicaciones.

Nos queda comentar finalmente el conjunto de tareas que están asociadas al comportamiento humano "inespecífico". Es decir, lo que hacemos a todas horas sin darnos cuenta: ver, oír, caminar, pensar, hablar, comprender el lenguaje, etc. Su importancia hace que le dediquemos un apartado, al que llamaremos de "comportamiento artificial", porque busca la síntesis de las funciones que la neurofisiología y la psicología cognoscitiva usan para describir el comportamiento humano.

Ya hemos comentado que los resultados de la *IA* en este dominio son muy limitados. Existen programas de visión artificial, de control motor de un robot o de razonamiento que son eficientes como *IA de síntesis* (es decir, resuelven problemas técnicos importantes) pero no podemos afirmar que se aproximen a la solución humana (*IA de análisis*).

1.2 IDEA INTUITIVA DEL COMPORTAMIENTO ARTIFICIAL

El concepto de ***inteligencia*** que nosotros utilizamos está sacado de la psicología cognoscitiva y pertenece a una familia más amplia de ***construcciones teóricas*** (comprensión, propósito, intención, memoria, aprendizaje, etc...) inventadas por el hombre para ayudarnos en la descripción del comportamiento observable de ***sistemas complejos*** en interacción con su ***medio***.

Hablar de *IA* en este sentido supone querer comprender y duplicar las funciones que caracterizan los aspectos más genuinos del comportamiento humano ordinario. Por ejemplo, lo que hace un médico todos los días cuando va a su consulta, independientemente de su tarea científico-técnica específica, consistente en analizar síntomas y proponer un diagnóstico y una terapia. La representación computacional de este conocimiento, que *no es específico* de su actividad como experto en un dominio técnico, pero que es imprescindible para que esta actividad pueda realizarse, constituye la frontera más genuina entre la *IA* y la ciencia cognoscitiva. Algunas de sus características más distintivas son:

a) *Su aparente simplicidad en el ser vivo.* Así, si abrimos los ojos "vemos" de forma inmediata un conjunto de perceptos y lo mismo pasa al integrar los cinco sentidos en perceptos más complejos. Análogamente, nos hablan y "comprendemos" los conceptos e ideas que nos quieren transmitir. A la vez, razonamos y contestamos, etc... Y todo esto lo hacemos sin esfuerzo aparente.

b) *La enorme complejidad* de estos procesos cognoscitivos "elementales" cuando queremos sintetizarlos. No existen soluciones satisfactorias al problema de la comprensión de imágenes o del lenguaje natural. Las soluciones más avanzadas de la *IA* en estos campos están todavía lejos de las soluciones biológicas. Basta pensar en el razonamiento de sentido común o en el uso del lenguaje natural para detectar la distancia entre las soluciones naturales y las artificiales.

c) *El uso masivo de conocimiento* y la sospecha de que los procedimientos usuales de representación (lógica, redes, reglas, marcos) y los mecanismos de inferencia por encadenamiento de reglas (por ejemplo), son totalmente insuficientes para modelar estas tareas cognoscitivas. Para su modelado hace falta un lenguaje de representación con la capacidad y robustez del lenguaje natural.

d) *El estilo peculiar de computación* que aparentemente usa el ser vivo. Toda la computación artificial es extensional y se realiza sobre información estructurada. Por el contrario, hay cierta evidencia sobre el carácter intensional (por propiedades) e intencional (por propósitos) de la computación biológica, donde la inferencia es inmediata (refleja) y la información se reestructura constantemente por procedimientos muy robustos tanto a nivel sintáctico como semántico.

e) El *reconocimiento*, de acuerdo con Maturana y Varela [1990], de que todo conocer depende de la *estructura que conoce*. Está en su organización y *lo cognoscitivo* es propio de *lo vivo*. Por eso es difícil que el conocer computacional coincida con el conocer vivo, porque sus unidades materiales (silicio, tejido nervioso) y sus arquitecturas especifican fenomenologías distintas. Toda la organización de lo vivo es inherente a su historia evolutiva y a la materia viva que la soporta. De forma análoga, toda computación artificial termina en la electrónica digital y en uniones PN sobre cristales de silicio que, simplemente, conducen o no conducen.

f) *La hipótesis fuerte de la IA* es que también es posible hacer computacional este conocimiento propio de lo vivo. Es decir, los procesos cognoscitivos

pueden ser reducidos al nivel simbólico y a partir de aquí ya hay programas traductores que llegan hasta el nivel físico. No todos los profesionales del campo están de acuerdo con esta afirmación, pero trabajar en esa dirección es un objetivo común y apasionante.

La figura 1.1 muestra un esquema del conjunto de tareas cognoscitivas usadas por la neurofisiología y la psicología para describir el comportamiento interactivo de un ser vivo con su medio. Corresponde esencialmente a lo que Newell y Simon [1972] llamaron "diagrama funcional de un agente inteligente".

El agente interacciona con su medio a través de un conjunto de *sensores* (visión, tacto, audición,...) que representan físicamente las configuraciones espacio-temporales de ese medio. Posteriormente, se realiza un procesamiento multisensorial de más alta semántica, con referencia a contenidos de memoria al que llamamos *percepción*. El objetivo de ambos procesos es identificar al medio de acuerdo con un *modelo* de representación interna que permite *comprender* el significado de imágenes y palabras. No queremos pasar por alto el conjunto de conceptos que hemos mencionado (percepción, identificación, modelo del medio, representación interna, comprensión, significado de imágenes y significado de palabras). El lenguaje natural nos ha facilitado esta mención, pero cuando se reflexiona sobre cualquiera de estos conceptos y se pretende reproducirlos en un programa, nos encontramos con todas las dificultades que hemos comentado anteriormente. Sigamos sin embargo con la descripción del "agente inteligente" en el otro extremo de interacción con el medio.

El agente realiza ahora tareas *motoras* que inciden en el medio controlando manipuladores, produciendo palabras, coordinando acciones elementales para navegar evitando obstáculos etc. Estas tareas suelen ser también de semántica baja (efectores) o media (secuenciación). Conviene aclarar que el grado de semántica de una señal es proporcional a la capacidad operacional del símbolo que transporta o, dicho de otra forma, a la cantidad de conocimiento adicional que tendría que usar un observador externo para comprender su significado. En las fronteras con el mundo físico (fotorreceptores de la retina y motoneuronas activando haces musculares a la salida), la semántica es baja. En cambio, cuando se penetra en el sistema nervioso y las señales ya han experimentado varias codificaciones sucesivas, la semántica es alta.

Entre estas dos familias de tareas (perceptuales y motoras) existe un conjunto intermedio de *tareas de decisión* que trabajan entre espacios de representación, sin conexión directa con el medio. Incluyen los procesos cognoscitivos asociados al pensamiento y al lenguaje (formación de conceptos,

comprensión y producción del lenguaje, razonamiento inductivo, deductivo y abductivo, memoria asociativa, aprendizaje, planificación estratégica, procesos creativos, etc...). Para la realización de estas tareas el agente posee un modelo del medio y un conjunto de propósitos en ese medio y para alcanzar sus metas usa lo que Newell llama ***"principio de racionalidad"***.

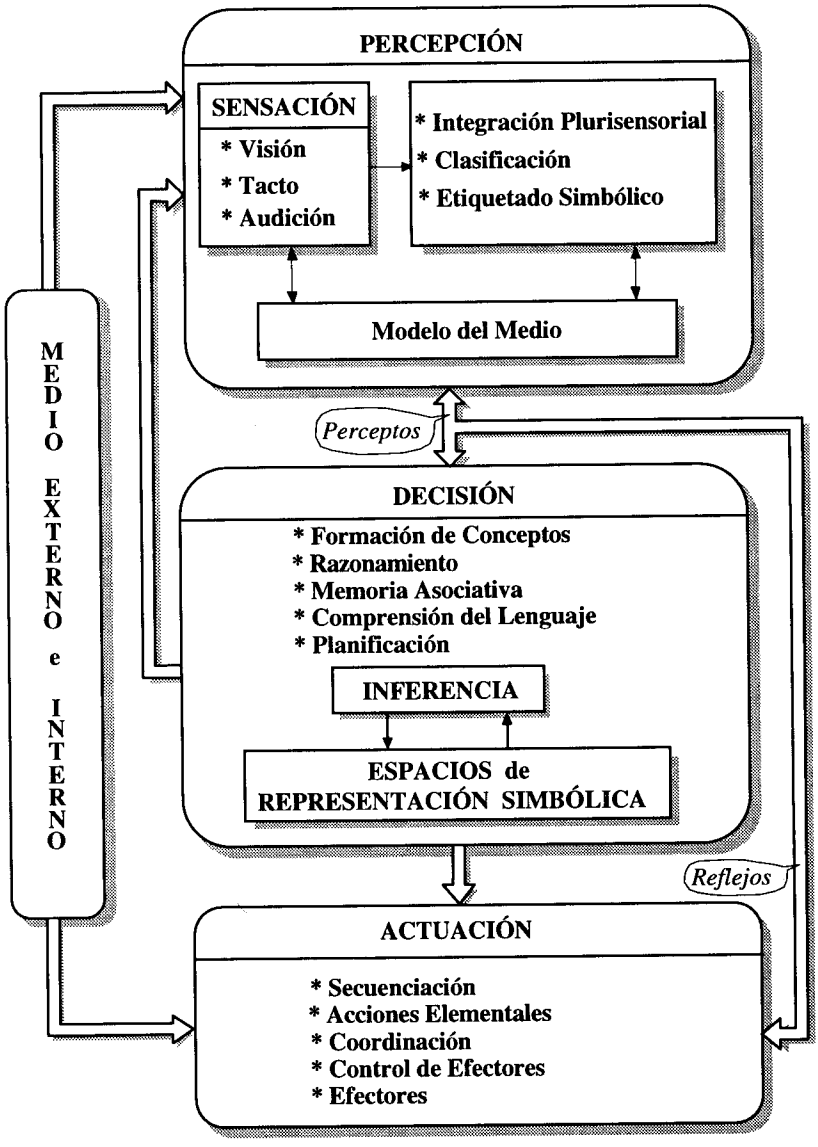


Fig. 1.1. Resumen de tareas que describen la interacción con el medio de un "agente inteligente".

El problema es que el principio de racionalidad no es operacional. Es decir, nos dice *"**qué hacer**"* pero no *"**cómo hacerlo**"*. Cada vez que encontremos un procedimiento efectivo para identificar, capturar y representar de forma computable el conocimiento necesario para la síntesis no trivial de estas tareas de *percepción*, *decisión* o *planificación* motora, diremos que tenemos un sistema de *IA*: visión artificial, comprensión del lenguaje natural, decisión (*SE*) o robótica perceptual, etc. Nosotros sólo estudiaremos los aspectos básicos de representación y uso del conocimiento en funciones de decisión para tareas técnicas en dominios limitados (sistemas expertos). También estudiaremos el aprendizaje simbólico y conexionista, pero no hay capítulos específicos dedicados a la visión artificial, o a la robótica. Su estudio queda fuera de los alcances de este libro. Sí que queremos decir que no existen problemas específicos y técnicas de solución especiales para cada una de estas tareas. El día que seamos capaces de resolver el problema de la percepción (por ejemplo), tendremos resuelto los problemas del aprendizaje, la memoria y el razonamiento. Y lo mismo es cierto si el primer problema que resolvemos es el del aprendizaje. En todos los casos estamos buscando formas de modelar conocimiento que puedan reducirse al nivel simbólico.

El problema de la *IA* aplicada, como en cualquier otra ingeniería que busca sistemas físicamente realizables, es pasar del nivel de especificaciones al nivel de componentes. Obsérvese que, finalmente, toda *computación* (sea simbólica, conexionista, o numérica convencional) termina en el nivel físico de la Electrónica Digital, como configuraciones determinísticas de estados lógicos –niveles de 0 ó 5 voltios– en inversores o biestables). La clave de la *IA* es conseguir programas traductores intermedios que conecten las primitivas de bajo nivel con las de un lenguaje de representación cada vez más próximo al lenguaje natural.

¿Donde empieza entonces la IA?. Decíamos al comienzo que para conseguir una definición de *IA* usaríamos criterios *extensivos* e *intensivos*. Ya hemos visto los primeros. Su resumen sería la lista de tareas que hemos enumerado dentro de cada una de las tres familias de dominios (formales, científico-técnicos y humanos inespecíficos). Veamos ahora algunos ***criterios intensionales*** para distinguir las fronteras de la *IA* con las otras ramas de la computación. De acuerdo con Rich y Knight [1991], hay que distinguir entre problemas propios de la *IA* y técnicas de *IA* para resolver problemas.

Tenemos un problema de *IA* siempre que:

C.1. No exista una solución analítica o algorítmica conocida.

C.2. Cuando existiendo esa solución, la explosión combinatoria la haga ineficiente.

C.3. Cuando el conocimiento necesario es masivo, incompleto, complejo y difícil de representar.

C.4. Cuando es necesario el aprendizaje y la inyección de conocimiento del dominio.

C.5. Siempre que abordemos tareas cognoscitivas que usen conocimiento de sentido común.

De forma complementaria, diremos que tenemos una técnica de solución de problemas propia de la *IA*, cuando:

C.6. Utiliza una estructura de tareas genéricas que permite capturar los aspectos generales del problema y de sus procedimientos de solución de forma que las situaciones individuales se tratan por los métodos asociados a las clases a las que pertenecen.

C.7. Usa heurísticas que intentan capturar el conocimiento accesible (en general incompleto e impreciso) del dominio.

C.8. Separa el conocimiento de su uso en inferencia y hace énfasis en el primero.

C.9. Permite manejar el razonamiento impreciso y temporal.

C.10 Incluye algún tipo de aprendizaje: Simbólico o Conexionista. Sin aprendizaje no hay IA.

Vamos a ilustrar con el ejemplo de la visión el paso de la computación analítico-algorítmica a la *IA*. La idea de partida es que este paso se produce cuando es necesario dar un salto cualitativo en el conocimiento que es necesario inyectar desde el exterior de un sistema para comprender el significado del proceso. Este salto se da en visión al pasar del ***nivel bajo*** (transductores,

preproceso, extracción de características locales bordes, y segmentación) al ***nivel alto*** en el que se define el espacio de objetos y se realiza el reconocimiento de patrones y la interpretación del *significado* de una escena, tal como se ilustra en la figura 1.2.

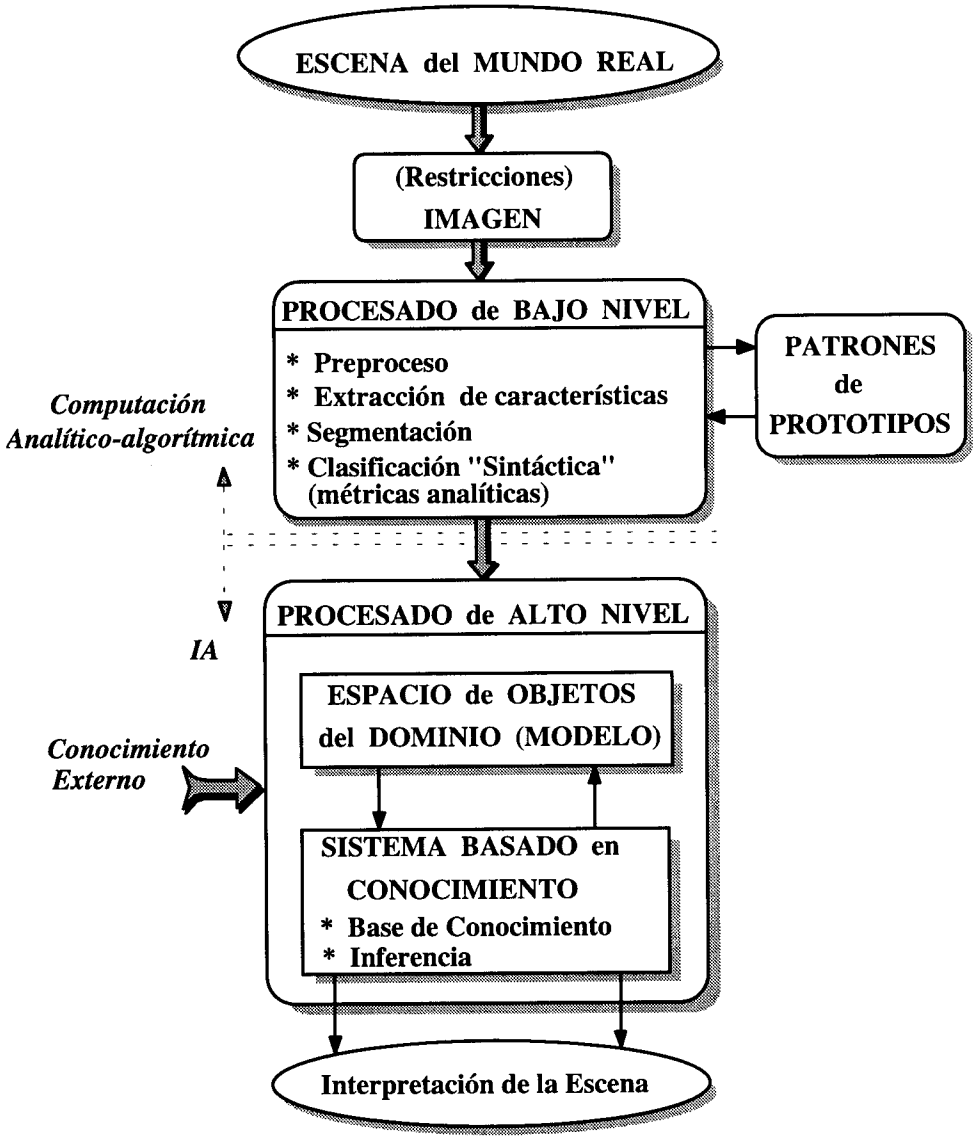


Fig. 1.2. Representación esquemática del proceso de visión artificial, distinguiendo el procesado analítico de bajo nivel del procesado de alto nivel, basado en conocimiento y propio de la *IA*.

En el procesado de bajo nivel prácticamente toda la información está en la imagen. El transductor sigue una ley física, en general de tipo logarítmico. Por ejemplo, pasa de voz o de imagen a una señal eléctrica continua y variable con el tiempo $v(x,y;t)$, usando un amplificador o una cámara de vídeo. A su vez, esta señal analógica se digitaliza mediante un conversor analógico-digital que cambia drásticamente su forma pero mantiene la información. El segundo paso es el *preproceso* que extrae características locales o integrales de naturaleza analítica (derivadas espacio-temporales, transformadas de Fourier, etc...) que no exigen conocimiento complementario para ser entendidos por un observador. Este preproceso produce una descripción complementaria de la señal inicial que facilita el posterior proceso de clasificación. Aunque hemos dicho que aquí no es necesaria la inyección de conocimiento a nivel explícito, sí que existe de forma implícita a través de la selección del conjunto de características que consideramos más adecuadas para el reconocimiento de la voz o de una determinada familia de imágenes.

La etapa final del procesado de bajo nivel es el reconocimiento de formas basado en la definición analítica de distancia entre el valor que toman las propiedades usadas para describir la imagen y los valores correspondientes a esas variables en un conjunto de patrones. La salida de este nivel es la etiqueta del patrón al que más se aproxima la descripción. Aquí ya hay un primer paso de semántica al definir las clases de equivalencia y las métricas usadas para discriminar, aunque el salto cualitativo está en el segundo nivel.

En el procesado de alto nivel (percepción), nos hace falta recurrir a la inyección de conocimiento externo específico del dominio para dar significado a las estructuras de datos y a los procesos porque su sentido sólo queda claro para quien posee ese conocimiento del dominio y en ningún caso es evidente a partir de las entidades del nivel simbólico o del nivel físico (en el caso del conexionismo). El punto frontera es la definición del espacio de objetos en el modelo del medio. Dos patrones con la misma etiqueta a nivel sintáctico pueden representar entidades de la escena totalmente diferentes a nivel semántico. Este segundo nivel, de comprensión de imágenes, es responsabilidad de la *IA*.

Comprender una imagen es asociarle la escena del mundo real que representa de acuerdo con un modelo del medio. Pensemos por ejemplo en el proceso que realiza un médico para interpretar una radiografía y emitir un informe diagnóstico. Es claro que en la imagen se encuentran elementos identificables a nivel analítico (contrastes, determinadas bandas del histograma, etc...). Sin embargo, el diagnóstico final (la comprensión de la imagen) es una

consecuencia del *modelo* de los "objetos del dominio" que el médico posee sobre las configuraciones normales o patológicas de la anatomía del órgano cuya imagen se está procesando y este conocimiento no está en la imagen. Hemos visto *dónde empieza* la *IA*. Veremos ahora ***dónde acaba***. Es decir, conocemos algo sobre los límites de la *IA* por debajo, en su frontera con la analítica y la computación numérica. Veamos ahora su frontera con lo genuinamente humano. Hay profesionales del campo que aseguran que no existe tal frontera y que sólo es cuestión de tiempo y recursos el encontrar las estructuras y los procesos de información que dupliquen o superen al cerebro humano. De hecho, si consiguiéramos un programa capaz de hacer un *uso activo* del conocimiento que existe en las bases de datos actuales, superaríamos muchas funciones del cerebro.

Sin embargo, el obstáculo básico [Schwartz, 1987], es la necesidad de la programación en detalle, en vez de la capacidad biológica de madurar y aprender a organizar la información presentada de forma poco ordenada en estructuras internas autoorganizativas y robustas, capaces de ser aplicadas a una amplia variedad de circunstancias. La perspectiva conexionista aborda parcialmente este obstáculo pero sus resultados son todavía muy modestos. De nuevo la dificultad está en hacer computacional el aprendizaje como único camino conocido para evitar la necesidad de programación detallada.

Los límites de la *IA* pueden encontrarse, al menos, en los siguientes puntos:

a) Desconocimiento del operador humano.

b) Falta de teoría (principios organizacionales y estructuras).

c) Diferencias fundamentales entre el nivel físico de la computación (cristales semiconductores) y el propio de los seres vivos (tejido biológico).

Difícilmente podemos afirmar que la mente es reducible a computación si realmente no conocemos cómo funcionan las neuronas biológicas, ni cómo está organizado el cerebro, ni cómo realizamos nuestros procesos cognoscitivos. Ni siquiera podemos afirmar que la *metáfora computacional* sea la más adecuada para comprender el funcionamiento del sistema nervioso. La metáfora computacional supone que todos los procesos pueden modelarse mediante dos espacios de representación (el de entradas y el de salida), y un conjunto de reglas

de transformación que proyectan configuraciones del espacio de entradas en las configuraciones correspondientes del espacio de salidas.

Por otro lado, la *IA* de síntesis no tiene por qué depender de la comprensión de lo vivo. Recuérdese que no volamos moviendo las alas, sino por conservación de la cantidad de movimiento. Es decir, una forma alternativa y eficiente de extender los límites de la *IA* es desarrollarla como ciencia y tecnología de lo artificial, sin referencia directa con la biología. Entonces deben resolverse, al menos, las siguientes cuestiones:

1. Modelado y representación del conocimiento a nivel estratégico, de tareas genéricas de tipos de inferencia y de entidades y relaciones propias del dominio usando nuevos lenguajes de representación más próximos al lenguaje natural. La robustez, flexibilidad y capacidad representacional del lenguaje natural está todavía muy lejos de la de los lenguajes formales.
2. Búsqueda de nuevos principios de autoorganización capaces de generar estructuras simbólicas robustas y de amplio uso a partir de entradas de información más desorganizadas y fragmentarias. Es decir, búsqueda de soluciones alternativas a la programación completa de todas las aplicaciones.
3. Desarrollo de nuevos lenguajes de programación que incluyan las funcionalidades de los actuales (Lisp, Prolog, Reglas,...) pero que permitan a la vez cierto nivel de autoprogramación y ayuda a la edición de conocimiento.
4. Énfasis en las teorías computacionales del aprendizaje tanto simbólico como conexionista o híbrido. Así, parte de las tareas de adquisición del conocimiento podrían automatizarse.

1.3 PERSPECTIVA HISTÓRICA DE LA *IA*

El sueño de mecanizar los procesos del pensamiento (lo que ahora llamamos "hacer computacional el conocimiento humano" o sintetizar sus procesos cognoscitivos en sistemas de *IA*) es muy antiguo y procede –como casi todo– de los griegos. Aunque se suele reconocer que el nacimiento de la *IA* debe asociarse a la conferencia que se organizó en 1956 en el Dartmouth College, su nacimiento real pasa por Platón, Descartes, Boole, Leibnitz y Hobbes, encuentra una etapa neurocibernética a partir de 1943 con los trabajos pioneros de W.S. McCulloch, W. Pitts, J. von Neumann, N. Wiener, J. McCarthy, A. Newell,

M.L. Minsky, S.C. Kleene, M. D. Davis, A.M. Uttley y C. Shannon y entra después en una etapa de dominios formales (micromundos) y heurística. Mención especial merece el trabajo de Alan Turing, tanto por sus aportaciones al modelo de computación como por su intento de formalizar la *IA* pasando de la pregunta inicial acerca de si las máquinas pueden "pensar" a otra más científica y práctica sobre la evaluación de las funcionalidades de un programa (lo que ahora conocemos como test de Turing).

Hacia 1975 se da un cambio brusco en la orientación de los trabajos en *IA* haciendo énfasis en la importancia del conocimiento frente a los mecanismos generales de inferencia. Así, se abandona la búsqueda de "solucionadores generales de problemas" y se focaliza el trabajo en la búsqueda de tareas técnicas concretas en dominios estrechos, dando lugar al campo de los ***sistemas basados en conocimiento*** (los ***SBC***) y, en particular, a los ***sistemas expertos*** (***SE***). Los problemas de esta época son la adquisición y representación de ese conocimiento y el desarrollo de aplicaciones en gran variedad de campos, entre los que destaca el diagnóstico en medicina.

Finalmente, podemos considerar que en torno a 1980 comienza la etapa en la que nos encontramos ahora, caracterizada esencialmente por un fuerte renacimiento de la computación neuronal, primero como alternativa a la *IA* simbólica y después como complemento en los sistemas híbridos. Junto a este renacimiento se reconoce un cierto estancamiento en los desarrollos teóricos, la consolidación de las técnicas elaboradas previamente y la importancia creciente del aprendizaje en ambas perspectivas, simbólica y conexionista.

Vamos a comentar esta evolución histórica distinguiendo las siguientes etapas:

- *Etapa Neurocibernética.*
- *Computación: de Platón a Turing.*
- *Heurística y Micromundos.*
- *Énfasis en el Conocimiento.*
- *Aprendizaje y Renacimiento del Conexionismo.*

1.3.1 Neurocibernética

La inteligencia artificial comenzó siendo computación neuronal cuando en 1943 Warren S. McCulloch y Walter Pitts introducen el primer modelo formal al que en la actualidad llamaríamos circuito secuencial mínimo, formado por una función lógica seguida de un retardo y en el que la programación se sustituye por el aprendizaje. Así una red de neuronas formales de McCulloch-Pitts es equivalente a una máquina de Turing de cinta finita. Si la red es programable entonces es equivalente a una máquina universal. Comienzan así, en paralelo, ambos modelos de computación.

Las ideas básicas de esta época aparecen bajo el nombre de ***neurocibernética*** y se basan en considerar que los seres vivos y las máquinas pueden ser comprendidos usando los mismos principios organizacionales y las mismas herramientas formales (la lógica y las matemáticas). Se piensa también que la aproximación a ambos debe realizarse a nivel de *procesadores* (bajo nivel), de forma que para comprender como computa el cerebro, hay que estudiar las neuronas biológicas, modelarlas formalmente, construir redes y ver como emerge el comportamiento a partir de procesos locales de autoorganización, memoria asociativa y aprendizaje. Los tres trabajos de 1943, que podemos considerar fundacionales, fueron:

- *"Conducta, Propósito y Teleología"*
  (Rosemblueth, Wiener, Bigelow)
- *"Un Cálculo Lógico de las Ideas Inmanentes en la Actividad Nerviosa"*
  (W.S. McCulloch, W. Pitts)
- *La Naturaleza de la Explicación"*
  (K. Craik)

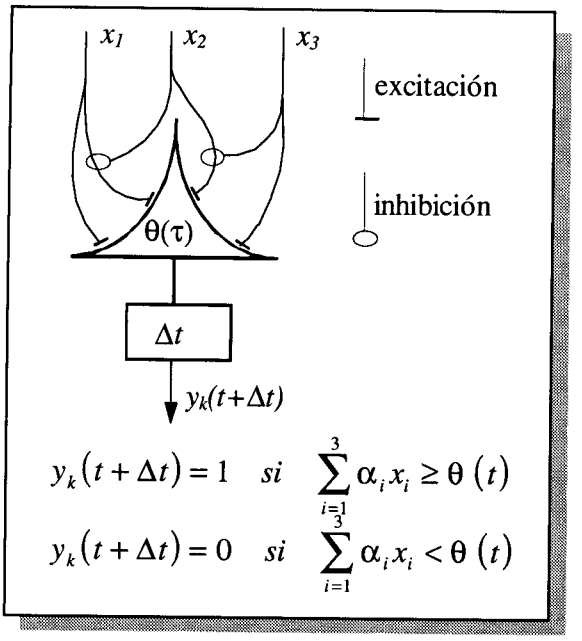
En el primero se introducen tres conceptos importantes en *IA*: la *realimentación* como principio organizacional, la *computación* por propósitos y la idea de *información* como pura forma, separable de la señal física que la transporta. La propuesta básica es que la conducta de un sistema en su medio (hombre o robot) debe interpretarse en términos de unos objetivos (propósitos) y del intento de alcanzarlos usando la realimentación negativa que compara el estado actual con el deseado y corrige la actuación.

El trabajo de Wiener, ampliado en 1948 en el libro "Cybernetics" y complementado con el trabajo de Shannon sobre la teoría matemática de la comunicación [1949] establece claramente que lo importante en control y en comunicación es la *información* –el mensaje– que es pura forma sin dimensión física. Lo mismo haría Alan Turing con la computación al distinguirla de la máquina física y lo mismo ocurre con el *conocimiento*, que también es pura forma. El significado de la información y el conocimiento sólo existe en el dominio del observador. Al pasar al nivel simbólico todo se reduce a estructuras de datos y procesos. Volveremos sobre este tema en el capítulo segundo, al hablar del "nivel de conocimiento", introducido por Allen Newell [1981].

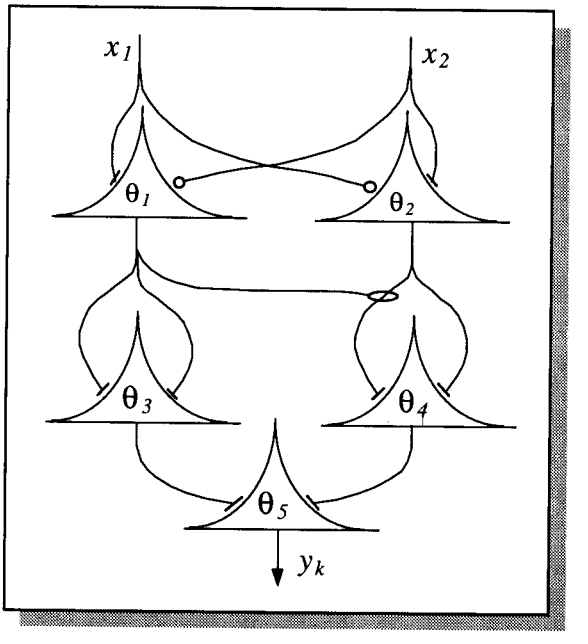
El trabajo de W.S. McCulloch y W. Pitts sobre redes de neuronas formales inicia la ***"Teoría Neuronal del Conocimiento"***, buscando las redes de procesadores que sean capaces de reconocer caracteres, recordar, inferir de forma distribuida, cooperar, aprender por refuerzo o autoorganizarse. En términos actuales diríamos que esta primera etapa de la *IA* busca la solución de los problemas a nivel físico, donde estructura y función coinciden. Da comienzo a la teoría modular de autómatas y usa la lógica (determinística y probabilística) como forma de representar el conocimiento que queda distribuido en las conexiones de la red. La figura 1.3 muestra un esquema del modelo inicial de neurona formal y red neuronal.

El segundo aspecto que merece destacarse de esta época de la *IA* es la aparición del problema de la ***intencionalidad***. Hablar de computación conlleva de forma implícita hablar de computación en extenso, pero los aspectos más genuinos del pensamiento humano están asociados al cálculo intensional (por conprehensión) y al uso de propósitos para guiar la acción. Las lógicas temporales y el estudio de la causalidad y las creencias en *IA* han activado de nuevo los trabajos iniciales sobre representaciones computables de operadores de significado e intención.

El tercer trabajo que estamos comentando es la obra de Kenneth Craik (***"La Naturaleza de la Explicación"***) en la que interpreta la actividad del sistema nervioso en términos de un conjunto de procesos encaminados a construir una representación interna del medio (modelo) y usarla para predecir. Aprender aquí es acumular conocimiento actualizando el modelo del medio. *Craik* contribuyó a la moderna *IA* con dos aportaciones clave: *razonamiento abductivo* y *espacios de representación.*



(a)



(b)

Fig. 1.3. Neuronas formales de la primera época. a) Función umbral sobre la suma de entradas excitadoras e inhibidoras. b) Red de capas. La función de la red queda especificada por los umbrales de disparo (θ_1, ..., θ_5) y la conectividad.

La inferencia en *IA* está asociada al uso individual o combinado de tres tipos de razonamiento: deductivo, inductivo o abductivo. En la ***deducción lógica*** se parte de un conjunto de fórmulas que se consideran de validez general (axiomas) y sobre ellas se aplican un conjunto de reglas o procedimientos de demostración que nos permiten obtener nuevas fórmulas válidas. Pasamos de lo general a lo particular y podemos garantizar la certeza del resultado obtenido. El problema de este tipo de inferencia es que necesitamos establecer con precisión el conjunto de creencias (axiomas) que definen de forma completa el conocimiento del dominio.

En la ***inferencia inductiva*** pasamos de lo particular a lo general, generalizando la información extraible de casos particulares. Para ello usamos "pistas" (heurísticas) con el conocimiento del dominio, pero nunca podemos garantizar la complitud y certeza de la inferencia. Siempre hay incertidumbre sobre la validez de las suposiciones.

Finalmente, en el ***razonamiento abductivo*** se parte de una conclusión conocida y se busca un hecho que la explique. Trata situaciones (tales como el diagnóstico) en las que se conoce una relación causa-efecto y un suceso (efecto) y tenemos que lanzar hipótesis sobre su causa más probable.

Craik, junto con Pierce, son claros antecedentes del trabajo actual en el campo [Peng y Reggia, 1990]. En 1943 Craik, en su libro "La Naturaleza de la Explicación", introduce el primer intento de definición operacional de conceptos tales como causalidad, significado, implicación y consistencia, en la misma línea que siguió más tarde Turing al hablar de inteligencia. Las preguntas importantes no son qué es causalidad o inteligencia, sino cómo podemos modelarlas (reconstruirlas) construyendo programas que las dupliquen. En palabras de Craik, "nuestra pregunta no es qué es implicación o causalidad, sino cuál es la estructura y los procesos necesarios para que un sistema mecánico sea capaz de imitarlos correctamente y predecir sucesos del mundo externo a la vez que crear nuevas entidades".

El segundo punto de la obra de Craik es la propuesta de un mecanismo de razonamiento por analogía en el modelo del medio donde la implicación formal es el equivalente a la causalidad en el mundo físico. Distinguía Craik tres procesos:

1. Traslación de los procesos externos a símbolos en un espacio de representación.

2. Obtención de otros símbolos mediante inferencia en el modelo del medio que paraleliza la causalidad externa.

3. Retraslación de esos símbolos transformados al dominio de sus referentes externos (predicción).

Este proceso de inferencia en el modelo del medio produce los resultados simbólicos equivalentes a los que hubiera producido la causalidad física que hemos modelado.

La idea de un modelo del medio ha permanecido desde esta época fundacional hasta la moderna robótica y las nuevas concepciones del proceso de adquisición del conocimiento de los expertos humanos mediante un proceso de modelado. La característica fundamental de lo que ahora llamamos un SBC es su poder para modelar sucesos basados en dos tipos de símbolos: números y palabras. La propuesta implícita en la obra de Craik es que la codificación de estos símbolos no es arbitraria, sino que mantiene su identidad desde la entrada sensorial hasta la salida motora. Las relaciones internas entre los símbolos mantienen la consistencia e interdependencia de las entidades externas a las que representan.

Hemos comentado la influencia de los trabajos fundacionales de 1943 en la *IA* actual. Desde 1943 hasta 1980, aproximadamente, aparecen trabajos en neurocibernética, que se solapan con la *IA* desde 1956, en los temas que resumimos en la tabla de la figura 1.4.

Esta etapa conexionista fue ensombrecida por la *IA* simbólica desde 1956 hasta 1980 en la que vuelve a renacer pero es interesante señalar que la línea de investigación en *IA* a bajo nivel (procesadores) siguió durante estos años y que muchos de los nombres de investigadores relevantes aparecen en ambos campos. Tal es el caso de J. von Neumann, S.C. Kleene, O.G. Selfridge, J. McCarthy, A. Newell, C. Shannon, N. Shapiro, M. Minsky y S. Papert, J.T. Culbertson, M.A. Davis, A.M: Uttley, W.R. Ashby y K. de Leeuw, entre otros. El contenido del libro "Automata Studies", editado por Shannon y McCarthy el 1956 en la Universidad de Princeton es referencia obligada para entender el desarrollo de la *IA* en esta época [Shannon & McCarthy, 1956].

La distinción entre procesadores y procesos y el establecimiento de distintos niveles de computación, desde el físico de la computación neuronal hasta el simbólico de la *IA* convencional será estudiada en el capítulo segundo dedicado a los aspectos metodológicos.

- *Redes de Neuronas Formales.*
- *Algoritmos de Análisis y Síntesis.*
- *Aprendizaje por ajuste de Parámetros.*
- *Tolerancia a Fallos.*
- *Modelos de Memoria Asociativa.*
- *Reconocimiento de Caracteres.*
- *Aprendizaje Asociativo y por Refuerzo.*
- *Desarrollos en Teoría Modular de Autómatas.*
- *Codificación y Teoría de la Información.*
- *Principios Organizacionales:*
  - *Autoprogramación.*
  - *Autorreproducción.*
- *Programas Programadores.*
- *Traducción Automática.*
- *Comprensión del Leguaje Natural.*

Fig. 1.4. Tabla resumen de los temas abordados por la *IA* en su primera etapa neurocibernética.

1.3.2 Computación: de Platón a Turing

En la década de los cincuenta se da el paso de la computación numérica a la simbólica y se desarrollan programas para jugar al ajedrez o demostrar teoremas, a la vez que Turing propone su conocido test. Sin embargo, independientemente de la naturaleza numérica o simbólica de un cálculo, la computación encuentra sus raíces en Grecia. Dreyfus sugiere que la *IA* comenzó alrededor del año 450 a.C. cuando, de acuerdo con Platón, Sócrates pregunta a Euthyphro por un conjunto de ***reglas de decisión*** definidas de forma tan precisa

que en cada momento pudiéramos calcular la respuesta del sistema aplicando esas reglas a la entrada.

John Haugeland hace énfasis en la propuesta de Hobbes (1588-1679) quien afirmaba que el pensamiento (raciocinio) consistía en ejecutar operaciones simbólicas siguiendo de forma metódica un conjunto de ***reglas de decisión***.

Descartes (1596-1650) descubre el carácter invariante de la matemática para representar conocimiento. Las matemáticas conciernen con símbolos y sus relaciones abstractas y podemos considerar los contenidos del pensamiento como representaciones simbólicas. Es interesante recordar que Descartes ya estableció claramente la distinción entre la computación "de diccionario" y el genuino proceso de reconocimiento de significados que más tarde ha sido usado por Searle (entre otros) para criticar la validez del test de Turing.

Descartes intenta formalizar el razonamiento usando tres tipos de procesos: ***enumeración***, ***deducción*** e ***intuición*** (intuitus), siendo "intuitus" el más difícil de mecanizar porque hace referencia a la aprehensión directa mediante la cual captamos el significado de un concepto. Por otro lado, algunas de las primitivas propuestas por Descartes (causalidad, duración, magnitud, igualdad, analogía y duda) siguen teniendo vigencia en el intento actual de construir una teoría de la inteligencia.

Aunque Leibniz (1646-1716) es más conocido por su contribución al "hardware" añadiéndole la capacidad de multiplicar y dividir a la máquina de Pascal, su contribución a la *IA* fue la búsqueda de un lenguaje simbólico con contrapartida numérica. Leibniz introduce el concepto de programa y de programación: "artificio elegante mediante el cual ciertas relaciones se pueden representar y fijar numéricamente de forma que su determinación posterior se reduce a ***cálculo numérico***".

Aunque Leibniz precedió a Boole en la búsqueda de un lenguaje simbólico, fue George Boole (1815-1868) quien introdujo el modelo lógico haciendo énfasis en la separación de símbolos y números y en la posibilidad de tratar a los primeros como objetos de cálculo. Su obra, "Una Investigación de las Leyes del Pensamiento en las que están basadas las Teorías Matemáticas de la Lógica y las Probabilidades", introdujo en la computación el modelo lógico, del que en cierto modo todavía no hemos podido escapar. De hecho, si incluimos la obra de Shannon sobre la realización física de los operadores lógicos, toda la *IA* desde el nivel físico de la electrónica digital hasta el nivel simbólico, está impregnada de la lógica formal. La polémica acerca de si es suficiente (o no) el modelo lógico

para la *IA* todavía continúa. Volveremos sobre este punto al comparar los distintos formalismos de representación del conocimiento en los capítulos 5, 6, 7 y 8. El capítulo quinto está dedicado por completo a la lógica como mecanismo de representación y uso del conocimiento. La deducción automática sigue siendo un apartado básico de la *IA*. De hecho el programa "Logic Theorist" [Newell, Shaw y Simon, 1956] es el primer trabajo publicado sobre razonamiento automático, dentro de la *IA*.

Un aspecto importante de los desarrollos teóricos en *IA* parte de intentar mantener las ventajas del modelo lógico y disminuir sus inconvenientes asociados al carácter binario e invariante de sus valores de verdad. La lógica probabilística (en particular en su vertiente bayesiana), la lógica borrosa de Zadeh y las extensiones del razonamiento no monótono y temporal son muestras de estos intentos. Su estudio en profundidad queda fuera del alcance de este libro que se preocupa sólo por los aspectos básicos de la *IA*. Sin embargo se harán algunas referencias a estas extensiones en los temas correspondientes.

Las contribuciones de von Neumann a la computación numérica y a la *IA*, tanto en su perspectiva simbólica como en las redes neuronales, fueron extraordinarias, a pesar de su corta vida. Podemos esquematizarlas en varios puntos:

- *Arquitectura de Computadores.*
- *Teoría Modular de Autómatas.*
- *Redes Neuronales y Teoría del Cerebro.*

Independientemente de sus trabajos previos en física y economía, el trabajo de 1945 dentro del proyecto EDVAC posee dos aportaciones básicas a la computación. Por un lado separa el diseño lógico, al que ahora llamamos arquitectura de computadores, de las distintas realizaciones físicas de la máquina. Por otro lado define la arquitectura básica que todavía persiste. Es curioso señalar que el diseño lógico propuesto por von Neumann usa neuronas formales del tipo de McCulloch-Pitts, de forma que demuestra la equivalencia entre máquina de Turing y redes neuronales sobre una arquitectura programable y universal.

Su contribución a la teoría de autómatas y a la *IA* conexionista no fue sólo a nivel formal, sino que planteó cuestiones fundamentales que siguen siendo claves [Symposium de Hixon, en 1949]:

- *Reformulando la Máquina de Turing en términos de Autómatas Celulares.*
- *Autoprogramación* (autómatas que diseñan otros autómatas).
- *Autorreproducción* y *Evolución* (contructores universales que se reproducen).
- *Tolerancia a Fallos* y *Estabilidad Lógica* ante cambios de función local.

Finalmente, dentro de este apartado de la perspectiva histórico-conceptual de la *IA*, llegamos a Alan Turing y sus dos contribuciones básicas a la teoría de la computación y a la *IA*,

- *Un Modelo Computacional Universal*
  (al que ahora llamamos máquina de Turing) [1936].
- *Un Procedimiento Experimental de Medir la IA de un Programa*
  (al que ahora llamamos test de Turing) [1950].

La máquina de Turing es una máquina matemática. Al igual que von Neumann separó la máquina física de la arquitectura, Turing sube a otro nivel y separa la arquitectura del modelo lógico. Todo proceso que se pueda describir de forma clara, completa, precisa e inequívoca en lenguaje natural puede representarse mediante un autómata. A partir de esta representación se puede encontrar una secuencia de instrucciones que al operar sobre símbolos codificados genera un programa ejecutable en cualquier computador. Así, computacionalmente todas las arquitecturas (y por consiguiente todas sus realizaciones físicas) son equivalentes en el sentido de Turing. Las diferencias tienen que ver con factores de complejidad, tamaño, tiempo, eficacia o recursos necesarios. Comentamos ahora el ***test de Turing.***

La cuestión recurrente desde Descartes acerca de "si las máquinas podrán algún día pensar" fue reformulada en 1950 por Alan Turing en términos más precisos usando el juego de la imitación. La formulación posterior [Moor, 1987] en términos de la acumulación de evidencia inductiva lo ha convertido en una de las pruebas básicas para evaluar la calidad de un programa de *IA* o el nivel de conocimiento en un *SE*. Veremos más adelante que esta forma de valoración no es suficiente ya que no es lo mismo imitar el pensamiento que pensar. Sin embargo, su valor e influencia a lo largo de toda la historia de la *IA* es innegable.

El juego de la imitación fue planteado inicialmente para ser jugado por tres personas: un hombre (*A*), una mujer (*B*) y un interrogador (*C*) que está en una habitación aparte y puede preguntar a ambos. Estos le contestan sobre un soporte (por ejemplo una pantalla) que no permite la distinción del origen de las respuestas por la sola forma del mensaje. Se supone que los tres agentes (*A*, *B* y *C*) conocen las reglas del juego y los respectivos propósitos. El propósito de *A* (hombre) es confundir a *C* (interrogador) imitando a *B* (mujer). El propósito de *B* es informar a *C* acerca de su identidad.

Turing modificó este juego en la forma que ahora conocemos como Test de Turing (figura 1.5), sustituyendo a uno de los actores (el *B* por ejemplo) por un programa de ordenador y dejando en el *A* al operador humano. Ahora, el interrogador (*C*) realiza un conjunto de preguntas $\{P_i\}$ a ambos agentes (programa y experto humano) y compara las respuestas del operador humano, $\{R_i^A\}$, con las que obtiene del programa de *IA*, $\{R_i^B\}$. Si de la simple comparación de estas respuestas el observador *C* no es capaz de distinguir que fuente de información corresponde al operador humano (*A*) y cual al programa de *IA*, diremos que para esa tarea y en ese dominio al programa de *IA* se le debe suponer el mismo nivel de inteligencia que al operador humano al que imita.

Aunque el test necesita ser más específico acotando el tipo de tareas y dominios, especificando la sintaxis y la semántica del lenguaje en el que se deben realizar las preguntas y las respuestas y el tiempo que dura el interrogatorio, etc..., es evidente que su efecto fue excelente ya que cambió una pregunta muy ambigua y general (¿pueden pensar las máquinas?) por un procedimiento efectivo para valorar experimentalmente las funcionalidades de un programa de *IA*, comparándolas con las del operador humano. Una modificación de esta prueba consiste en sustituir al interrogador por un procedimiento de "final de juego" y hacer competir al programa (*B*) con el operador humano (*A*), por ejemplo en el juego del ajedrez. No hay que olvidar nunca que junto a (*B*) está todo el conocimiento inyectado por el programador.

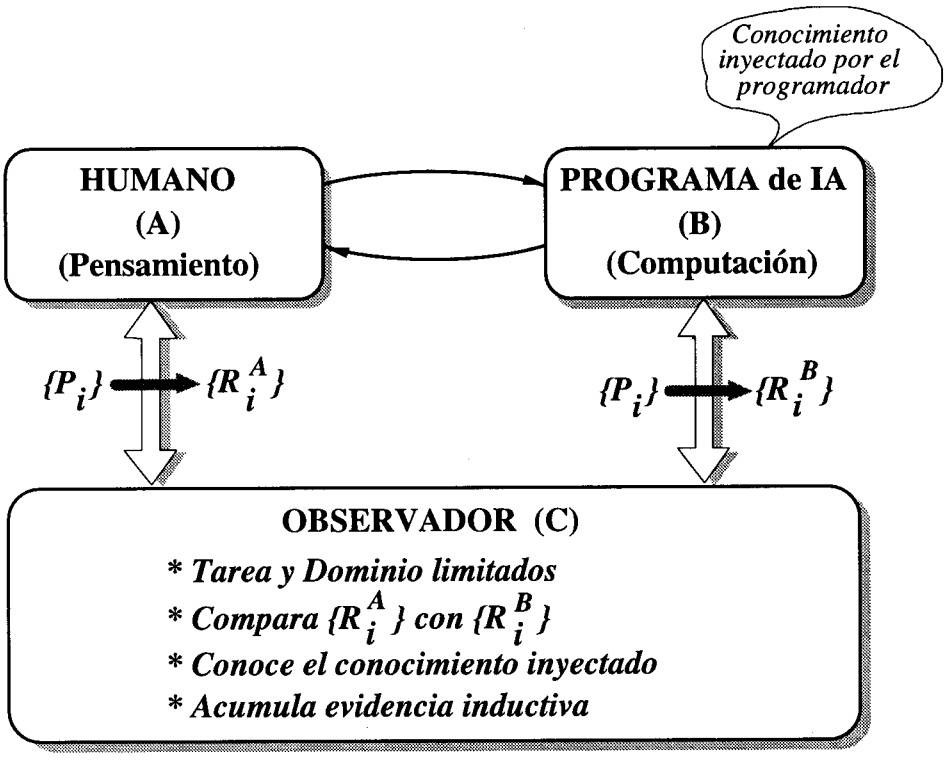


Fig. 1.5. Interpretación del test de Turing como procedimiento experimental para acumular evidencia sobre la eficacia de un programa en la realización de una tarea.

Usaremos el esquema propuesto por Moor para analizar las críticas al test de Turing. La crítica más fuerte es su carácter conductista. Es decir, el test evalúa el "qué" pero no el "cómo". Una cotorra sofisticada podría producir contestaciones análogas en algunos casos. Imitar la conducta abierta de quien piensa sobre una tarea no es lo mismo que conocer el mecanismo que organiza las estructuras simbólicas a partir de otras menos organizadas para producir la misma conducta mediante los mismos procedimientos. Por otra parte, desde el punto de vista de la ingeniería tampoco sería necesario. Basta que el sistema sea eficiente y no importa como lo consiga.

La contestación adecuada a esta crítica consiste en considerar al test de Turing no tanto una medida de la inteligencia de un programa como un procedimiento experimental para recoger y acumular evidencia inductiva acerca de la eficacia de ese programa para resolver un problema. Evidentemente, como

todo razonamiento inductivo basado en ejemplos, el resultado del test siempre estará marcado por un cierto grado de incertidumbre. Es decir, sus deducciones nunca podrán ser consideradas absolutamente ciertas. Si *A* gana a *B* y/o hace mejores jugadas, diremos que posee más inteligencia para esa tarea. Cuantas más veces le gane, más evidencia acumularemos para afirmar que *A* es más inteligente que *B*. Fuera de la computación, cuando evaluamos la calidad y la eficacia de un operador humano (médico, ingeniero,...) o cuando se plantean situaciones de competición, también utilizamos este procedimiento experimental.

Esta interpretación inductiva elimina también, aunque sólo parcialmente, la crítica de Searle sobre la diferencia entre el conocimiento superficial (de diccionario) y el profundo. En este segundo caso se supone que el agente debe comprender el significado del mensaje a nivel semántico generando en otra lengua otro mensaje con el mismo significado, sin referencia necesaria a la sintaxis del mensaje de partida. Ambos traductores (superficial-sintáctico, profundo-semántico) podrían pasar los primeros niveles del test, con lo que serían considerados equivalentes a nivel superficial. Obviamente al complicar el tipo de preguntas haciéndolas más sensibles al contexto y al conocimiento de sentido común, el primer traductor dejaría de superar el test.

El segundo tipo de crítica hace referencia a que el test no es muy severo. Imitar no es muy complicado y la eficiencia en tareas formales y técnicas para dominios estrechos y modelables no justifica el carácter cognoscitivo (humano) de la computación en *IA*. Toda la información está completamente estructurada, todo el proceso está programado y hay poco lugar para el *aprendizaje*.

La capacidad de aprendizaje de un programa es, claramente, la siguiente medida importante para evaluar el nivel de *IA* que se le puede adscribir a ese programa. Esta es la tendencia dominante en la actualidad. El conocimiento estático, estructurado y programado no puede considerarse equiparable al conocimiento humano, activo, estructurante y con aprendizaje. Sin aprendizaje no hay auténtica *IA*. El carácter más genuino del conocimiento humano es su capacidad de aprendizaje.

1.3.3 Búsqueda Heurística y Dominios Formales

Tras los antecedentes neurocibernéticos y computacionales que ya hemos comentado, suele considerarse 1956 como el año en que nació la *IA*, pues el término fue acuñado por John McCarthy al convocar la Conferencia de Dartmouth, basada en "la conjetura de que todo aspecto del aprendizaje o

cualquier otra característica de la inteligencia puede ser simulado mediante una máquina". En esta conferencia estaban cuatro de los investigadores más influyentes del campo: H. Simon, A. Newell, M. Minsky y el propio McCarthy.

El primer trabajo, resultado de la búsqueda de esquemas universales de deducción, fue el programa *"Logic Theorist"* que demostró las posibilidades reales de la *IA*, al menos para el campo de los dominios formales (probando teoremas de los *"Principia Mathematica" de Whitehead y Russell*), y dando origen a toda la rama del razonamiento automático que persiste en la *IA*, reforzado por el *principio de resolución* de Robinson [1965] y sus refinamientos posteriores, incluyendo la creación del lenguaje Prolog.

Comienza también en esta época la preocupación por "lenguajes para procesar información" [IPL de Newell y Shaw, 1957], más orientados al manejo de símbolos que al cálculo numérico, iniciándose el camino hacia el Lisp, desarrollado por John McCarthy en el MIT [1958].

Otro trabajo representativo del pensamiento de esta época y debido también a Newell, Shaw y Simon es el programa GPS [Solucionador General de Problemas, 1959] complementado en 1960 [Newell, et al., 1960] con un intento de dotarlo de capacidad de aprendizaje y autoorganización. El GPS es un programa que incorpora medios heurísticos para resolver problemas en entornos formales (demostrar teoremas, identidades algebraicas y trigonométricas, integración y derivación). El procedimiento de solución usa la metodología de análisis "medios-fines", seleccionando una secuencia de operadores que haga disminuir la diferencia entre el estado inicial y el estado meta hasta anularla.

Todos los trabajos de esta primera época (1956-196x), se centraron en problemas propios de dominios formales ("micromundos"), que eran idealizaciones del mundo real: demostración de teoremas en cálculo proposicional, estrategias heurísticas en juegos, planificación de acciones y trayectorias en el mundo de los bloques y problemas de juegos (misioneros y caníbales, ocho reinas, "tic-tac-toe", etc...). En todos los casos la *IA* se entiende como búsqueda en un espacio de estados y se supone que no es necesario mucho conocimiento específico sobre el dominio, haciéndose énfasis en las estrategias de control usadas para guiar la búsqueda.

A pesar de que hoy en día la IA camina por otras vías hay sin embargo, al menos tres intentos serios de recuperar las propuestas iniciales del GPS y de los procedimientos de búsqueda en espacios de conocimiento. El propósito es

separar los aspectos estructurales comunes al procedimiento de solución de una clase general de problemas, del conocimiento específico de una tarea particular:

- *Las arquitecturas tipo SOAR.*
- *Las metodologías basadas en una estructura de tareas genéricas (Chandrasekaran y KADS).*
- *Los nuevos paradigmas multiestrategia en aprendizaje simbólico [Michalski & Tecuci, 1994].*

A mediados de los sesenta aparecen cambios graduales en la orientación de la *IA*, reconociendo las limitaciones de los procedimientos generales, acercándose más a los problemas del mundo real y dando cada vez más importancia al conocimiento específico del dominio y por consiguiente a los problemas asociados a su representación y posterior uso en inferencia, donde se separan conocimiento y uso del mismo. Hablaremos de esto en el siguiente apartado.

En 1968 Marvín Minsky edita el libro "Procesado de Información Semántica" [Minsky, 1968], con trabajos de Raphael (programa SIR), Bobrow (programa STUDENT), Quillian (redes semánticas), Evans (programa ANALOGY), Black (sistema deductivo de contestar preguntas) y J. McCarthy (programas de sentido común). En este libro se resume un conjunto de tesis doctorales dirigidas por el propio Minsky en el MIT y es representativo del trabajo en *IA* en esta época histórica. En todos los casos se trata de programas en general relacionados con la comprensión del lenguaje natural. Lo mismo ocurre con el trabajo de Winograd sobre la representación procedimental del conocimiento. La controversia entre la conveniencia de representaciones procedimentales o declarativas todavía es actual. El sistema SHRDLU [Winograd, 1972] es representativo del límite de una metodología que aborda el lenguaje natural limitando el dominio para conseguir un modelo completo de las estructuras y los procesos de ese dominio. Dreyfus critica esta época recordando que, en prácticamente todos los casos, los autores poseían una visión demasiado optimista sobre las posibilidades de sus programas, tal como ha resultado evidente veinte años más tarde. La llamada falacia del "primer paso" es muy significativa. De prácticamente todos los programas se decía que eran un primer paso hacia la comprensión del lenguaje natural, pero el segundo paso nunca llegó por ese camino, sino por la vía de los sistemas basados en conocimiento.

Mención especial merece el trabajo de McCarthy sobre ***programas con sentido común***. Todavía permanece como una de las fronteras de la *IA* el problema asociado al razonamiento de sentido común capaz de inferir de forma inmediata el amplio conjunto de consecuencias lógicas de cualquiera de los hechos que conoce. En palabras de McCarthy [1968], "podemos decir que un programa posee sentido común si deduce de forma automática y por sí mismo una clase suficientemente amplia de consecuencias inmediatas de cualquier cosa que se le dice y que él ya conoce" (pag. 403). El objetivo de McCarthy, de conseguir programas que aprendieran de la experiencia de forma tan efectiva como hacemos los humanos, aun no se ha conseguido.

1.3.4 Énfasis en el Conocimiento (197x-198x)

El paso desde la primera etapa de la *IA* en la que se dedicaron todos los esfuerzos a la búsqueda heurística en dominios formales, hasta la etapa *intermedia* (197x-198x), dominada por el predominio de los sistemas basados en conocimiento (***SBC***) y, en particular los sistemas expertos (***SE***), se debió, como ya hemos comentado, al reconocimiento del hecho que para resolver problemas del mundo real, lo importante es saber modelar el conocimiento específico de ese dominio. Los mecanismos de inferencia pueden ser razonablemente sencillos, si el modelo de conocimiento estático es suficientemente completo. Existe además una opinión bastante general sobre las limitaciones de la lógica como lenguaje único de representación. Nosotros no entraremos en este debate. Nos limitaremos a describir sus posibilidades expresivas e inferenciales en el capítulo cinco. De todas formas la complejidad de los problemas que aborda la IA ha demostrado una década más tarde que es aconsejable no sólo combinar todas las técnicas de representación sino también integrar el simbolismo con las redes neuronales.

La segunda idea que caracteriza esta etapa intermedia de la *IA* es que no basta con el énfasis en el conocimiento sino que además, la *IA* de síntesis (ingeniería) debe focalizarse en tareas científico-técnicas en dominios estrechos donde podemos abarcar el conocimiento de forma razonablemente completa y donde no sea dominante la influencia del conocimiento de sentido común. Hablaremos así de monitorización inteligente, clasificación basada en conocimiento, consejeros de terapia para distintas especialidades médicas, sistemas de supervisión y detección de alarmas y otras muchas funciones de este tipo. De todas ellas tratará el capítulo noveno y también encontraremos ejemplos distribuidos en los capítulos dedicados a la representación del conocimiento. Lo

importante aquí es señalar el nacimiento de los *SBC* y los *SE* en una época dominada por dos preocupaciones:

(a) *Énfasis en la representación computacional del conocimiento para tareas del mundo real. El conocimiento específico del dominio es "poder".*

(b) *Selección de tareas técnicas en dominios estrechos (SE) donde se separa el conocimiento de sus mecanismos de aplicación (inferencia).*

1.3.4.1 Representación del Conocimiento

El problema de la representación del conocimiento en sus aspectos básicos se estudia en los temas 5 a 8, pero sigue abierto tanto en ***IA teórica*** (¿qué constituye conocimiento?, ¿qué relación de dependencia existe entre el conocer y la estructura que conoce?) como en ***IA de síntesis***. (¿qué representación es más eficiente, expresiva y completa?, ¿cuál permite un uso posterior más eficiente cuando ese conocimiento se usa al razonar?, ¿cómo se aborda el problema de la semántica?, ¿cómo formalizamos el razonamiento aproximado, no monótono, temporal o cualitativo?). La recopilación de trabajos que editaron Brachman y Levesque en 1985 es una buena referencia para captar el estado de la cuestión en esta época que estamos comentando. La conclusión de esta etapa es una propuesta de representación modular e híbrida que incluye aspectos de los cuatro procedimientos básicos: ***lógica***. ***reglas***, ***redes asociativas*** y ***marcos*** (objetos estructurados).

En esta época histórica se consideraba al conocimiento como "algo" que había que transportar desde la cabeza del experto humano a la memoria del computador. Por eso es usual hablar de "captura" o adquisición del conocimiento. La visión actual considera el problema de la representación formal del conocimiento como un proceso de modelado o de reconstrucción, a través del diálogo y la introspección, de lo que se supone que sabe el experto humano, estructurado en varios niveles (estratégico, de tareas, inferencial y específico del dominio).

La ***lógica*** (cap. 5) como lenguaje de representación del conocimiento en *IA* se ha usado desde el comienzo y continúa el debate sobre su suficiencia para representar y resolver problemas del mundo real. En la representación lógica todo el conocimiento tiene que codificarse mediante sentencias en las que se

afirma que *"x tiene la propiedad y"* y que tal afirmación es verdadera o falsa. Inferir aquí es combinar sentencias de este tipo de acuerdo con las leyes de la lógica de proposiciones y el *cálculo de predicados*.

Las principales ventajas de la lógica son su semántica, su expresividad y su eficacia en procesos deductivos. Sus inconvenientes proceden de la falta de estructuración, del problema de la completitud y de la existencia de problemas en el mundo real en los que hay que obtener conclusiones "por defecto", en los que no podemos garantizar que los resultados son ciertos pero la "ausencia de información en sentido contrario" nos permite avanzar en muchos razonamientos de sentido común. Esta ruptura de la monotonía (suposición usual en lógica) es la que ha dado origen a la búsqueda de nuevos formalismos para el razonamiento "no monótono" que junto con el aproximado (bayesiano o borroso), el temporal y el cualitativo, han dado lugar al estado actual de las investigaciones en esta rama de la *IA*. Estudiaremos la representación lógica en el capítulo quinto.

Para intentar resolver el problema de la ineficiencia propia de las representaciones lógicas surgen las representaciones basadas en *reglas* ("si *condición* entonces *acción*"), donde se sacrifica la expresividad a cambio de la eficiencia (cap. 6). Los sistemas basados en reglas introducidos por Newell en la *IA* aportan un nuevo paradigma de programación. La idea de partida es que todo el conocimiento relevante para realizar una tarea (de diagnóstico, por ejemplo) puede representarse mediante un conjunto de condicionales. Aplicar ese conocimiento supone un ciclo iterativo (*selección*, *ejecución*) que comienza identificando una regla y aplicándola. El resultado de aplicar la regla produce nuevos hechos y activa otras reglas de forma que el razonamiento está asociado a ese encadenamiento de reglas. En el sistema PROSPECTOR se comprobó la utilidad de este tipo de representación. Un trabajo reciente de Davis, Buchanan y Shortliffe [1993] pone de manifiesto las contribuciones de este tipo de representación que no siempre fueron comprendidas. Nosotros estudiaremos la representación y uso del conocimiento mediante reglas en el capítulo sexto.

Las ***redes semánticas*** (cap. 7) fueron introducidas por Ross Quillian en 1966 como modelo de memoria asociativa adecuado para la representación computacional del conocimiento necesario para la comprensión y traducción del lenguaje natural. Para Quillian la cuestión central era encontrar una representación formal del significado de las palabras que permitiera un uso posterior con eficiencia análoga a la que mostramos los humanos con nuestra memoria asociativa y relacional, en la que unos sucesos evocan a otros por asociación. Seltz, en 1922, en su teoría de la anticipación esquemática propuso

una organización relacional del conocimiento en términos de redes de conceptos (*nodos*) enlazadas por relaciones (*arcos*) y con una organización jerárquica que permitía la herencia de propiedades y que era activada por la información aferente. Ross Quillian [1968] retomó la propuesta de Seltz construyendo un programa que usa esta representación. La base de conocimientos es ahora una gran red de nodos (significados de palabras) interconectados por diferentes tipos de *arcos* asociativos. Basadas en la misma idea aparecen en *IA* otras propuestas de representación por redes entre las que cabe destacar:

- *Descripciones estructuradas de Patrick Winston [1975].*
- *Sistema SCHOLAR de Jaime Carbonell [1970].*
- *Grafos de Dependencia Conceptual de Schank [1975] y Schank y Abelson [1977] para representar el lenguaje natural.*
- *Redes proposicionales de Sowa [1976,1979].*
- *Sistema CASNET para diagnóstico médico [Weis, Kulikowski, Amarel y Safir, 1978].*

En este libro hemos preferido hablar de redes asociativas (cap. 7) como nombre más adecuado para englobar las distintas formas de representación que relacionan conceptos mediante nodos y arcos. Las redes semánticas serían entonces un caso particular que hace referencia a su origen, como forma de representar el significado (semántica) de las palabras en lenguaje natural. Otros ejemplos de redes asociativas serían, por ejemplo, las *redes causales* (entre las que se encuentran las redes bayesianas, las redes de diagnóstico tipo CASNET y las redes cualitativas) y las *redes neuronales*. Las tres características que distinguen los distintos tipos de redes son:

- *Tipo de conocimiento representado.*
- *Método de inferencia.*
- *Forma de abordar el problema del aprendizaje.*

Finalmente, hablaremos de los marcos, introducidos por Minsky en 1974, a partir de los modelos de memoria de Bartlett. Esta forma de representación del conocimiento supone que nuestro sistema nervioso organiza su experiencia

perceptiva y su memoria en términos de un conjunto de esquemas que describen de forma estructurada nuestro conocimiento referente a un conjunto de situaciones reales de forma que cuando los activamos en percepción o cuando los evocamos de la memoria para usarlos en razonamiento, estamos aceptando una gran cantidad de conocimiento implícito con el que rellenamos todos los campos o terminales ("slots") que quedan sin especificar.

Una característica importante de los marcos es su ordenación jerárquica, en la que un nodo superior indica una clase más amplia. Los marcos-nodo que están por debajo no sólo heredan las propiedades de sus superiores sino que pueden heredar también valores y métodos asociados a ellas. La herencia de valores constituye un mecanismo de razonamiento por defecto característico de los sistemas basados en marcos y de la programación orientada a objetos.

Los guiones y los planes son análogos a los marcos, sólo que ahora se hace referencia a estructuras que describen secuencias temporales de sucesos. La propuesta de Schank y Minsky es que el ser humano organiza su conocimiento en términos de millones de marcos, guiones y planes que usa constantemente en todas las tareas que comentamos en el segundo apartado de este capítulo: percepción, razonamiento, planificación motora, diagnóstico, etc...

Terminamos este apartado sobre la introducción histórica de los distintos métodos de representación, haciendo referencia a las representaciones híbridas y modulares que son usuales en la actualidad. La representación mejor es aquella que usa todos los métodos disponibles dependiendo de la naturaleza de la tarea. Así, los marcos pueden incluir reglas en sus terminales, las reglas pueden convertirse en objetos estructurados y ambos (reglas y marcos) pueden organizarse en forma de red. El primer paso histórico en esta línea de integración de distintos métodos de representación se debe a Aikins quien introdujo en 1979 la idea de prototipo en su sistema CENTAUR. En la misma línea se encuentra el sistema PIP de Pauker y colaboradores [1976]. El sistema CENTAUR interpreta datos referentes a la valoración de la función pulmonar a partir de una representación mixta (marcos y reglas) del conocimiento médico sobre la fisiopatología pulmonar. Las hipótesis diagnósticas se asocian a prototipos con reglas en algunos de sus terminales y con interdependencias entre ellos. La inferencia consiste en confirmar (identificar) cuál de los prototipos se acopla mejor al conjunto de datos de cada enfermo concreto.

En un trabajo reciente sobre el futuro de la representación del conocimiento Ronald Brachman [1990] resumen los avances desde 1980 hasta nuestros días en los siguientes puntos:

- *Mayor rigor en los desarrollos teóricos.*
- *Énfasis en las representaciones declarativas frente a las procedimentales.*
- *Intento de formalización e integración de los distintos tipos de razonamiento (inductivo, abductivo, temporal, basado en casos, no-monótono y cualitativo). Se camina hacia una teoría unificada del razonamiento.*
- *Renacimiento del conexionismo y surgimiento de las redes bayesianas.*
- *Manejo de grandes bases de conocimiento, lo que exige el desarrollo de métodos automáticos de adquisición y actualización (aprendizaje).*

1.3.4.2 Sistemas Basados en Conocimiento (SBC) y Sistemas Expertos (SE)

Cuando en un sistema se hace uso intensivo del conocimiento del dominio y se separa de los mecanismos que controlan su uso en inferencia, decimos que tenemos un *SBC*. Dentro de los *SBC* hay un grupo de sistemas en los que el conocimiento procede de un experto humano especialista en una tarea concreta (diagnóstico, supervisión, terapia, detección de fallos, etc...) en un dominio técnico. Decimos entonces que tenemos un *SE*. En la actualidad ambos conceptos se usan de forma indistinta aunque, pasada la explosión de los *SE*, se tiende a volver al concepto más comprehensivo de *SBC*, independientemente de cual sea la fuente del conocimiento.

El desarrollo de los métodos básicos de representación del conocimiento ha caminado paralelo al desarrollo de los sistemas expertos. Cada forma de representación está asociada históricamente al primer sistema que la usó, mostrando así sus posibilidades a nivel práctico. Por ejemplo, la posibilidad de programar la solución de un problema usando reglas, se conoce desde Post [1943] y Newell y Simon lo introducen en la *IA* en 1972 pero hizo falta que se desarrollara el sistema MYCIN para que la programación por reglas se consolidara como un método general de desarrollar sistemas expertos. Lo mismo ocurrió con la relación entre las redes semánticas y el sistema CASNET o entre los marcos y el sistema CENTAUR.

Las características fundamentales de un *SE* son:

- *Dominio reducido.*
- *Competencia en su campo.*
- *Separación conocimiento / inferencia.*
- *Capacidad de explicación.*
- *Flexibilidad en el diálogo.*
- *Tratamiento de la incertidumbre.*

La estructura básica de un *SE* de la primera época es la de la figura 1.6. Los módulos básicos son la *base de hechos ciertos*, la *base de conocimientos*, el *mecanismo de inferencia* que usa los elementos de la base de conocimientos, guiado por los hechos ciertos, para producir nuevos hechos que constituyen las respuestas del *SE*, y dos módulos adicionales. El primero es una base de datos convencional, distinta de la memoria de trabajo del sistema, y el segundo es un módulo de explicación y diálogo que ya existía en los prototipos de esta época inicial, aunque el concepto de explicación que incorporaban era bastante limitado (una traza de las reglas usadas). Aunque no aparece de forma explícita en el dibujo de la figura 1.6, este esquema corresponde a un *sistema basado en reglas*, de los que estudiaremos en el capítulo sexto. Cuando la representación del conocimiento se realiza usando marcos o redes semánticas, la estructura del *SE* no se corresponde de forma tan clara con estos módulos. El capítulo noveno está dedicado al estudio de los *SE*, tras haber visto las distintas técnicas de representación del conocimiento y los mecanismos de inferencia asociados. Una conclusión importante de esta etapa de la *IA* es que los sistemas basados en reglas constituyen un procedimiento efectivo de representación y uso del conocimiento esencialmente distinto de la programación convencional. Aquí la computación se realiza mediante un ciclo iterativo de identificar primero la regla adecuada a un hecho cierto y aplicarla. Ver después si, como consecuencia de la aplicación de esa regla, se ha resuelto el problema y en caso contrario, buscar una nueva regla en cuyo campo de condición se adapta al campo de acción de la anterior. Como consecuencia del disparo de una regla se altera el contenido de la memoria dinámica del sistema.

Esta forma de programar una aplicación supone, al menos, dos cambios esenciales:

1. Todo el conocimiento debe representarse de forma modular y declarativa.
2. En el modelado del conocimiento hay que tener un cuidado especial con el contenido que asignamos a los campos de condición y acción de cada una de las reglas porque ahí, de forma implícita, estamos incluyendo todo el conocimiento estratégico que guiará después el encadenamiento de las reglas.

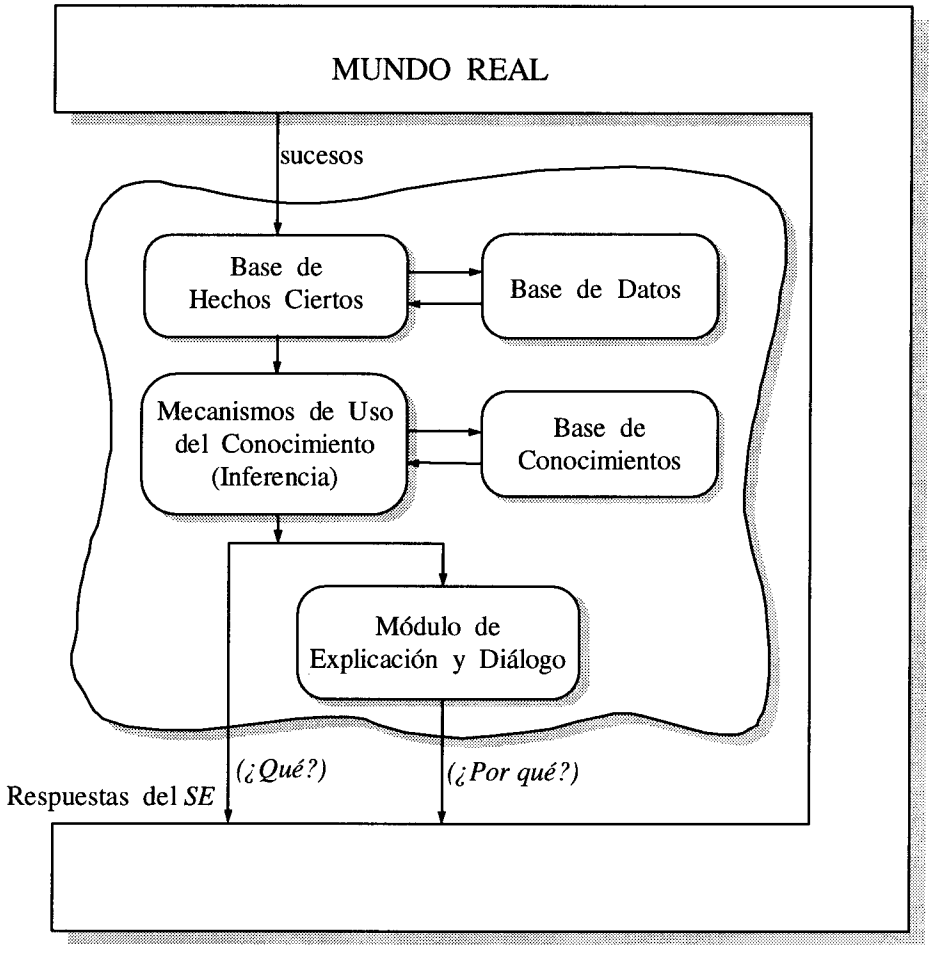


Fig. 1.6. Estructura básica de un *SE* de la primera época. La base de conocimiento está separada del mecanismo de aplicación de ese conocimiento.

Hacia la mitad de los años sesenta comienza en Stanford el desarrollo del sistema DENDRAL [Buchanan y Feigenbaum, 1978] que puede considerarse

como el primer *SE*. El propósito inicial era encontrar el conjunto de posibles estructuras moleculares a partir del análisis de los datos que ofrecía la espectroscopía de una determinada molécula. Sus resultados mostraron algunas de las características básicas en todos los *SE*:

1. Selección de un dominio limitado del conocimiento científico-técnico donde el programa alcanza niveles de competencia análogos a los del experto humano.
2. Evidencia de que el conocimiento esencial no es de carácter general sino específico del dominio.
3. Separación entre el conocimiento y el mecanismo de aplicación de ese conocimiento (inferencia) con la posibilidad de ampliar o modificar el conocimiento que posee el sistema, sin tener que modificar los mecanismos de inferencia.
4. Validez de las reglas como forma de representación del conocimiento (químico en este caso), sin necesidad de modelar el proceso de pensamiento del experto humano.

Paralelamente a DENDRAL se desarrolló MYCIN, sin duda alguna el sistema más influyente en toda la historia de la *IA*. Se desarrolló también en la Universidad de Stanford como sistema de consulta para el diagnóstico y tratamiento de enfermedades infecciosas [Davis et al., 1977; Buchanan & Sortliffe, 1984; Davis, Buchanan & Shortliffe, 1983] e incluía con más claridad que DENDRAL lo que también serían más tarde algunas de las características esenciales en todos los *SE*:

5. Validez del razonamiento por encadenamiento de reglas.
6. Tratamiento del problema de la incertidumbre (habitual en el conocimiento humano) mediante mecanismos sencillos y eficientes que combinan distintos factores de certeza.
7. Capacidad de *explicación* del razonamiento seguido para alcanzar la meta (diagnóstico o terapia) que propone.
8. Mención de conceptos que se consolidaron más tarde, tales como las *metarreglas* y la "adquisición de conocimiento" como tarea genérica en *IA*.

Las metarreglas (es decir, reglas sobre como utilizar otras reglas) se introdujeron en una extensión de MYCIN, TEIRESIAS, con tres fines: construir

la base de conocimientos mediante un diálogo con el experto humano, guiar el razonamiento mediante objetivos explícitos y mejorar la capacidad de explicación a partir de dichos objetivos [Davis, 1980; Davis & Buchanan, 1984].

La separación del conocimiento de su uso en inferencia permitió utilizar ambas componentes de forma independiente. Por una parte, la base de conocimiento fue usada en GUIDON para construir un sistema tutor que instruyera a los estudiantes sobre enfermedades infecciosas [Clancey, 1979; 1984]. Por otra parte, al "vaciar" el sistema de reglas sobre enfermedades infecciosas quedó EMYCIN (el nombre significa Essential MYCIN), que incluye las estrategias de razonamiento y las facilidades de explicación, y fue utilizado para desarrollar nuevos *SE* dando origen a los actuales entornos de desarrollo tipo NEXPERT, GoldWorks ó G2, por ejemplo [van Melle et al., 1984; van Melle, 1989]. Dos de los sistemas más importantes surgidos a partir de MYCIN fueron CENTAUR, en el que Aikins combinó el uso de marcos y reglas en lo que llamó prototipos [Aikins, 1980] y ONCOCIN, destinado a la gestión de protocolos de oncología [Tu et al., 1989].

El sistema CASNET fue desarrollado en la Universidad de Rutgers por Weiss y Kulikowski [1982] entre principios y mediados de los años 70 para el diagnóstico de glaucoma. A pesar de ser uno de los primeros sistemas expertos, incluye características ausentes en la mayor parte de los sistemas posteriores. El conocimiento se representa mediante una red causal, Causal Asociational Network. Los nodos se agrupan en tres niveles: observaciones, estados patofisiológicos y enfermedades, más un cuarto grupo correspondiente a planes terapéuticos. Los arcos representan relaciones causales, temporales y jerárquicas (de clasificación). Cada relación causal tiene asociado un factor de confianza que indica –informalmente– la probabilidad de que se produzca el efecto. El diagnóstico consiste en hallar caminos que unan estados fisiológicos entre sí y con las observaciones disponibles, tratando de explicar los hallazgos y la evolución de la enfermedad.

La evaluación del sistema produjo resultados muy satisfactorios y, del mismo modo que MYCIN dio lugar a EMYCIN, CASNET dio lugar a EXPERT, una herramienta con la que se construyeron sistemas expertos para reumatología y endocrinología.

La experiencia de MYCIN inspiró la construcción de PROSPECTOR [Duda et al., 1979] en el Stanford Research Institute. Se trata de un sistema destinado a la exploración geológica, en el cual el conocimiento viene

representado por una red de reglas y la inferencia se realiza mediante técnicas de inspiración bayesiana.

A pesar de las inconsistencias que presenta desde el punto de vista matemático, PROSPECTOR constituyó el primer éxito comercial de la *IA*, pues este sistema experto ayudó a detectar un depósito de molibdeno valorado en 100 millones de dólares. El otro gran éxito comercial fue el sistema experto R1, también llamado XCON, que aún se utiliza en Digital Equipment Corporation para configurar las estaciones de trabajo VAX [McDermott, 1982].

El sistema INTERNIST es un programa de consulta en medicina interna desarrollado por Pople y Myers [1977] en la Universidad de Pittsburg. Acepta como entradas un conjunto de hechos tales como síntomas, datos de laboratorio, signos físicos o historias clínicas y responde con un diagnóstico, haciendo competir distintas hipótesis alternativas. Representa el conocimiento mediante árboles en los que se evalúan las relaciones entre entidades mediante probabilidades condicionales.

El sistema PIP (Present Illness Program) se desarrolló durante los años 70 en el M.I.T. Su interés histórico se debe a que es el primer sistema experto basado en marcos. En cuanto al control del razonamiento, hay tres estados en que puede encontrarse un marco. Inicialmente, todos ellos están ***inactivos***. Cuando se introduce en el sistema un determinado hallazgo, los marcos correspondientes a las enfermedades que podrían explicarlo son ***activados*** y los marcos relacionados con ellos (por ejemplo, mediante los campos "puede-complicarse-por" o "diagnóstico-diferencial"), pasan a estar ***semiactivos***. Con ello, se pretende simular el comportamiento del médico que investiga uniendo las razones a favor o en contra de una determinada enfermedad (un marco activo), aunque teniendo en cuenta otras enfermedades relacionadas con ella (marcos semiactivos), hasta llegar finalmente a una conclusión que englobe toda la información disponible.

El sistema CENTAUR [Aikins, 1980] se realizó para interpretar datos referentes a la valoración de la función pulmonar a partir de una representación mixta (marcos más reglas) del conocimiento médico acerca de la fisiopatología pulmonar. Las hipótesis diagnósticas se asocian a ***prototipos*** con reglas de producción en algunos de sus campos. Estos prototipos de patologías están enlazados por una red que representa sus interdependencias de forma análoga a cómo se hacía en el tercer nivel de CASNET, sólo que aquí todos los prototipos

se estructuran en términos del mismo conjunto de campos que describen el modelo general de patología.

La meta del sistema CENTAUR es confirmar cuál (o cuáles) de los prototipos se acopla mejor al conjunto de datos del enfermo. Los prototipos activados en una primera fase constituyen la "*lista ordenada de hipótesis*". En el proceso de rellenado de los campos, cuando las reglas de que se dispone no pueden calcular un valor, éste se solicita al experto humano. Finalmente, cuando todos los terminales tienen valor, se desarrolla un *proceso de decisión* para estimar hasta qué punto los valores obtenidos coinciden con los esperados para ese prototipo. Se repite el proceso para todos los prototipos activados hasta que todos los datos reales han sido clasificados en uno o varios de los prototipos.

Cada prototipo (asma OAD, por ejemplo) contiene terminales *generales* (información, punteros de enlace con otros prototipos, descripciones breves), *de control* (si se confirma ... acción, si se rechaza ... acción) y *de componentes* (valores plausibles para las variables fisiológicas, valores a seleccionar por defecto, reglas). Las reglas de producción se usan para inferir información dentro de un prototipo. La parte de "*condiciones*" de la regla incluye un valor o combinación de valores de algunas de las componentes (capacidad pulmonar, por ejemplo). La parte de "*acción*" se refiere al valor diagnóstico inferido.

SI (1) **A_1:** La variable X_1 toma un valor Y_1 menor que Z_1

Y

B_1: La variable X_2 toma un valor Y_2 mayor que Z_2

O

(2) **A_2:** La variable $X_1 = Y_1 < 15$

Y

B_2: La variable $X_2 = Y_2 > 80$

ENTONCES: (1) Hay evidencia de que el grado de asma OAD es severo

Y

(2) Que uno de los hallazgos es : H_7

Esta estructura usada inicialmente por CENTAUR es bastante general. En el sistema PIP, que ya hemos comentado por ser el primer sistema basado en marcos, cada prototipo pertenece a la clase: enfermedad, estado clínico o estado fisiológico y se encuentran en términos de hallazgos, valoración, complicaciones, prototipos antecedentes y consecuentes y esquema de diagnóstico diferencial.

Así, la red se genera a partir de los propios enlaces entre prototipos. Una vez que se activa un prototipo quedan semiactivados indirectamente todos aquellos prototipos que aparecen como *referentes* en el activo ("*padres*", "*hijos*" y "*análogos*"). Cada prototipo puede estar: *dormido*, *semiactivo*, *activo* o *aceptado*. Inicialmente todos están en estado durmiente, pero cualquiera de ellos puede pasar al estado activo como consecuencia de la ejecución de procesos de activación asociados a la información aferente. El PIP es característico de la forma de razonar en medicina que ya hemos comentado anteriormente.

El sistema ONCOCIN, desarrollado en la Universidad de Stanford [Hickam et al., 1985; Tu et al., 1989], es el primer consejero de terapia en oncología y sigue siendo referencia obligada de un tipo general de sistemas de planificación temporal que constan de una secuencia de pasos generalizados con decisiones estratégicas al final de cada paso sobre una estructura "esqueletal" previamente definida. No existe necesidad de crear un plan que nos lleve a alcanzar un objetivo. El plan está creado de antemano, como consecuencia de haber modelado el conocimiento estratégico del medio sobre los protocolos de quimioterapia. La función del *SE* es decidir cómo se ejecuta a lo largo del tiempo, seleccionando intervalos entre decisiones estratégicas y tratamientos alternativos en los puntos de decisión, con el cálculo de las dosis correspondientes de cada droga. Para ello se recoge información en tiempo real (durante la ejecución del plan) sobre efectos laterales de la quimioterapia y evolución general del entorno.

Desde esta etapa inicial se ha producido una explosión del número de *SE* desarrollados. Desafortunadamente, no ha pasado lo mismo en cuanto sus contribuciones teóricas. En el texto de Waterman (1986) aparece una clasificación por tareas y dominios de los distintos *SE* desarrollados en la década. Van desde la agricultura (PLANT, POMME) hasta la medicina, donde han sido dominantes (MYCIN, NEOMYCIN, CASNET, CENTAUR, EMERGE, INTERNIST, CADUCEUS, NEUROLOGIST, ONCOCIN, PUFF,...), pasando por la química (CONGEN, CRYSTALIS, C-13, DENDRAL, GA1, META-DENDRAL, MOLGEN, ...), las propias ciencias de la computación (ACE, COMPASS, CRITTER, DFT, EURISCO), la ingeniería (DELTA, REACTOR,...), la geología (HYDRO, PROSPECTOR,...), la abogacía (AUDITOR,...), la meteorología (WILLARD,...) la educación (SCHOLAR, WHY, SOPHIE, GUIDON,...), la banca, la física y las matemáticas. Prácticamente todas las áreas del conocimiento aplicado han sido objeto de descripción computacional mediante sistemas basados en conocimiento.

Sólo por haber sido desarrollados en nuestro grupo queremos citar los sistemas CAEMF [Marín, 1986; Marín & Mira, 1987], SUTIL [Barro, 1987], MEDTOOL [Otero & Mira, 1988; Otero, 1991], TAO [Mira et al., 1987; 1991], TAO-MEDTOOL [Barreiro, 1991] y DIAVAL [Díez, 1994], los primeros desarrollados en la Universidad de Santiago de Compostela y el último en la UNED. El sistema CAEMF está dedicado al diagnóstico y seguimiento anteparto del estado materno-fetal. SUTIL aborda el problema de la monitorización inteligente en una unidad de cuidados coronarios y resuelve algunos problemas importantes relacionados con los sistemas expertos en tiempo real. MEDTOOL es una herramienta para el desarrollo de sistemas expertos que incluye un procedimiento propio para la representación del conocimiento mediante "magnitudes generalizadas", una especie de micromarcos. TAO es un consejero de terapia en oncología que incorpora el conocimiento estratégico necesario para la inclusión de enfermos en protocolos de quimioterapia y para el seguimiento del efecto del protocolo. Finalmente, DIAVAL es un *SE* para ecocardiografía basado en redes bayesianas.

Terminamos este apartado comentando las tendencias actuales en el campo de los *SE*:

- *Desarrollos de SE con una metodología razonablemente establecida, usando entornos comerciales y aceptando los métodos usuales de representación e inferencia.*
- *Desarrollos teóricos en temas frontera relacionados con la extensión de los métodos de representación y razonamiento.*
- *Énfasis en el aprendizaje y renacimiento del conexionismo.*

El primer punto es la clara distinción entre las perspectivas aplicada y teórica. En la primera perspectiva, se selecciona una aplicación, se adquiere el conocimiento, se usa un entorno de desarrollo para editar el conocimiento y se obtiene un prototipo que utiliza los procedimientos de inferencia propios de la herramienta. Este primer prototipo se evalúa en su entorno real y pasa después a un proceso de refinamiento y aprendizaje (en algunos casos). El capítulo 9 tal como hemos comentado anteriormente, lo dedicaremos a desarrollar esta perspectiva.

La perspectiva teórica no es muy optimista, a pesar que se han producido avances en algunos temas, en general relacionados con la extensión de los tipos de conocimiento y con los mecanismos de razonamiento. La extensión del conocimiento busca incluir el tiempo, la incertidumbre y el sentido común. La extensión del razonamiento busca en general extensiones de la lógica que no pierdan sus ventajas esenciales. En particular se busca la formalización del razonamiento no monótono y temporal y el tratamiento de la incertidumbre y la imprecisión. El tema del aprendizaje y el conexionismo lo comentaremos en el siguiente apartado.

1.3.5 Aprendizaje y Renacimiento del Conexionismo

Tal como comentábamos en el apartado sobre neurocibernética, la *IA* empezó siendo conexionista. Es decir, los primeros trabajos estaban relacionados con la síntesis de redes de neuronas artificiales que fueran capaces de reconocer caracteres, almacenar información o "razonar". A su vez, el aprendizaje se entendía también a nivel de procesadores. Aprender era modificar el valor de los parámetros de un sistema (Perceptron, Adalines, Polinomios de Gabor,...) para mejorar su comportamiento. Esta mejora se medía mediante una función del error resultante de comparar la respuesta deseada con la obtenida y los parámetros se modificaban en la dirección que hacía mínimo ese error. Entre esta etapa inicial y los alrededores de 1980 la *IA* ha sido simbólica y ha caminado en las líneas que hemos comentado anteriormente. Finalmente, en la última década hay un fuerte renacimiento de las redes neuronales como alternativa o complemento a la *IA* simbólica y la opinión generalizada de que sin aprendizaje será difícil avanzar en *IA* y en todo programa que tenga que gestionar información y conocimiento masivos procedentes de un medio cambiante.

1.3.5.1 Aprendizaje

Desde la etapa conexionista inicial hasta la llegada de los sistemas híbridos (cap. 10 y 11) (con parte neuronal y parte simbólica) y las arquitecturas que integran múltiples estrategias (Michalski y Tecuci, 1994), el aprendizaje en *IA* ha estado dominado por la perspectiva simbólica. Su fuente de inspiración ha sido el aprendizaje cognoscitivo que estudia la formación de conceptos, la solución de problemas, el razonamiento por analogía, los procesos inductivos, en los que se generaliza y traslada lo aprendido en casos particulares y el aprendizaje creativo, por discernimiento ("insight").

Ante la calidad del aprendizaje humano y el hecho evidente de que un programa sólo hace aquellas cosas para las que ha sido "programado", la *IA* siempre ha sido criticada diciendo que un ordenador nunca podrá aprender. Este argumento es incorrecto puesto que lo único que necesitamos hacer es decirle cómo tiene que aprender, es decir programarlo para que sea capaz de interpretar los datos de entrada de forma tal que los cambios en sus estructuras de datos y en sus procesos le permitan mejorar su comportamiento con la experiencia. Evidentemente, es probable que los aspectos más genuinos del aprendizaje humano nunca puedan duplicarse computacionalmente, pero esta posible limitación no es relevante, con tal que sigamos avanzando por el camino correcto. Una buena teoría del aprendizaje deberá incluir al aprendizaje humano como caso particular.

La idea más general de aprendizaje es la acumulación de conocimiento. Por consiguiente, un programa aprende cuando es capaz de acumular conocimiento sobre una tarea. Para que esto sea posible tenemos que representar a nivel simbólico, en términos de cambios en estructuras de datos y algoritmos, esos procesos de acumulación de conocimiento. Difícilmente se puede aprender algo que no podemos representar por lo que una buena teoría del aprendizaje comienza con la búsqueda de representaciones plásticas (transformables) que admitan cambios adaptativos como consecuencia del entrenamiento. La medida del grado de aprendizaje de un programa es entonces el incremento en eficiencia o generalidad que se obtiene como consecuencia de ese entrenamiento.

Planteado así el problema es demasiado general de forma que bajo el nombre de aprendizaje se engloban procesos muy diversos que podemos clasificar en términos de la tarea genérica a la que se refiere y del mecanismo de razonamiento en el que se basa. Hay tres familias de tareas: ***perceptivas***, *de* ***organización central*** (decisión) y *de* ***planificación motora***.

El aprendizaje en ***percepción*** incluye aspectos tales como el reconocimiento de caracteres y la formación de conceptos, que a su vez incluye la selección de características para describir los objetos de entrada, la creación de un lenguaje de descripción de conceptos (conjunto de hipótesis entre las que decidir) y la representación simbólica de todo el conocimiento previo que se usa para guiar y acotar el espacio de búsqueda. La *IA* reconoce desde hace tiempo que hace falta mucho conocimiento inicial para que un programa pueda empezar a aprender.

El aprendizaje en ***planificación*** incluye los aspectos complementarios de la percepción. Ahora en vez de clasificar, se parte de conceptos centrales en un

modelo del medio (planes) y se programa un generador de acciones elementales. Salvo la influencia de los factores temporales, el problema es *simétrico* (en el caso humano) respecto a un hipotético plano central en corteza cerebral. La percepción es la imagen especular de la planificación y la acción. De hecho, en todas las aplicaciones de la *IA* en visión artificial y robótica perceptual, planificación y percepción están integradas.

El aprendizaje de *organización central* incluye aspectos tales como la adquisición automática de nuevo conocimiento declarativo y su integración en una organización interna, la puesta al día de una base de conocimiento y los cambios en los mecanismos de inferencia.

Si consideramos ahora el aprendizaje como el cambio en los *mecanismos de razonamiento*, hay tres paradigmas básicos: inductivo, deductivo y abductivo (que en un sentido amplio incluye también el razonamiento por analogía). En el aprendizaje inductivo [Mitchell, 1982; Michalski, Carbonell y Mitchell, 1984], los cambios en las estructuras de datos y en los algoritmos van encaminados a la generalización del conocimiento extraible de los ejemplos (positivos y negativos) usados en el entrenamiento. Ese proceso de generalización está guiado por criterios heurísticos tanto más válidos cuanto más limitado y modelable es el dominio. Cuando se dispone de mucho conocimiento sobre el dominio no es necesario usar grandes conjuntos de entrenamiento. Basta con algunos casos para poder generalizar. Decimos entonces que tenemos una estrategia de aprendizaje basado en casos (CBL, Case-Based Learning). El aprendizaje deductivo está asociado a situaciones en las que se dispone de un conocimiento general bastante completo y una reglas de inferencia para obtener casos bajo la ley y explicar el proceso deductivo. Los problemas están asociados a la dificultad de encontrar teorías completas, por lo que normalmente hay que complementarlo con otras estrategias inductivas o abductivas.

El aprendizaje por analogía arranca de Craik [1943] y está basado en la búsqueda de correspondencias entre entidades y relaciones pertenecientes a dos dominios (dominio fuente y dominio diana). Estas correspondencias se usan para trasladar los razonamientos del dominio fuente a las situaciones análogas del dominio diana. Dado que las inferencias analógicas tienen carácter inductivo, estas dos estrategias de aprendizaje poseen puntos comunes. La analogía, al igual que la inducción, se basa en realizar una suposición que responde a una generalización de naturaleza incierta. En el aprendizaje por analogía lo que se elige es el caso conocido más próximo al nuevo que queremos resolver.

El aprendizaje abductivo busca un procedimiento para proponer y seleccionar las hipótesis que mejor expliquen las conclusiones conocidas. La elección entonces, también de naturaleza inductiva, está asociada a las hipótesis y tiene el carácter de una explicación.

El estado actual del aprendizaje computacional está marcado por dos tendencias complementarias:

(1) Desarrollo de sistemas con multiestrategia, basados en la comprensión de las funcionalidades y las condiciones de aplicación de las distintas estrategias individuales (deducción, inducción, analogía y red neuronal) y en la cooperación de estas estrategias en función del tipo de problema, del conocimiento del dominio y del objetivo del aprendizaje.

(2) Desarrollos teóricos y metodológicos distinguiendo entre las funcionalidades en el dominio propio (primitivas del lenguaje de representación) y en el dominio del observador (programador) y proponiendo un nivel de procesos comunes (tales como identificación, indexación, comparación, modificación e incorporación) en términos de los cuales sean formalmente equivalentes las distintas estrategias.

Nosotros estudiaremos los aspectos básicos del aprendizaje simbólico en el tema 10. Otros aspectos más avanzados quedan fuera del alcance de este libro.

1.3.5.2 Conexionismo

El crecimiento de la computación simbólica y el reconocimiento de las limitaciones de las redes de neuronas formales tipo perceptrón puestas de manifiesto por Minsky y Papert en su trabajo sobre configuraciones que no son separables (reconocibles) mediante funciones de discriminación lineales y en el libro sobre perceptrones [Minsky & Papert, 1969] frenaron el conexionismo al final de los años 60. Hay una etapa intermedia (1970-1980) en la que se sigue trabajando en conexionismo desarrollándose las redes probabilísticas [Moreno & McCulloch, 1968; Moreno et al., 1972], las memorias asociativas [Anderson & Rosenfeld, 1989], el reconocimiento de caracteres [Fukushima, 1980], el aprendizaje por refuerzo [Grossberg, 1972; Mira, 1971], la visión artificial a nivel de procesadores y a nivel de procesos [Marr y Poggio, 1976; Marr, 1982], la decisión cooperativa [Kilmer y McCulloch, 1968] y los modelos biofísicos e inferenciales que reclaman una extensión de la computación neuronal hacia el simbolismo de las formulaciones analíticas.

El salto cualitativo en la perspectiva conexionista de la *IA* aparece al comienzo de los años 80 con las propuestas de Rumelhart y colaboradores [1986] y Barto [1983]. Todavía permanece el concepto inicial de red neuronal como cálculo realizado por una arquitectura modular con gran número de elementos con alto grado de conectividad y realizando una función analógica no lineal. La diferencia esencial está en la sustitución de la función umbral por una función derivable que permita la retropropagación de la función de error que guía el aprendizaje supervisado por el método del gradiente. Este argumento técnico junto con un cierto sentimiento de estancamiento en el campo de los sistemas expertos y el carácter pendular de la historia han potenciado el renacimiento del conexionismo.

En la *IA* conexionista se está intentando ahora resolver los mismos problemas que han preocupado a la *IA* simbólica. Es decir, ¿cómo se representa el conocimiento en una red neuronal?, ¿cómo se usa en inferencia?, ¿cómo aprende la red?, ¿cómo se usa el conocimiento inicial del problema para definir la arquitectura, la conectividad, la función local y los algoritmos de aprendizaje?, ¿cómo se puede establecer un puente simbólico-conexionista?.

El contenido en extenso de la *IA* conexionista aborda los siguientes temas:

1. *Conexiones con la neurociencia*, buscando inspiración en las redes de neuronas biológicas, su organización, plasticidad y esquemas de conectividad. La opinión general en este apartado es que las neuronas biológicas realizan computaciones mucho más complejas que las usadas actualmente en las neuronas artificiales.
2. *Modelos de computación distribuida y autoprogramable*, incluyendo los no lineales, lógico-secuenciales, borrosos, probabilísticos, bayesianos e inferenciales.
3. *Búsqueda de arquitecturas* multicapa e incrementales y genéticas que faciliten la cooperación y la autoorganización de forma análoga a las propuestas de la *IA* distribuida.
4. *Estudio del aprendizaje* supervisado y no supervisado y su conexión con el aprendizaje simbólico y las formulaciones híbridas que toman lo mejor de ambos paradigmas y pasan de uno a otro en función de las características de la tarea.
5. *Desarrollo de neurosimuladores*, que son entornos de programación, evaluación y desarrollo de redes neuronales a nivel software, antes de

pasar a la implementación. Se buscan entornos orientados a aplicaciones y/o algoritmos y de uso general.

6. *Implementación* de redes como arquitecturas paralelas de propósito especial a nivel de preprocesadores, coprocesadores o neuro-circuitos completos.

7. Un gran capítulo de *aplicaciones*, en *percepción* (segmentación, extracción de características, reconocimiento de caracteres, texturas, análisis del movimiento y los niveles semánticos superiores que completan las tareas necesarias para sintetizar un sistema de visión artificial, en particular y de percepción multimodal en general) y *control* (identificación, planificación motora, control de efectores en robótica) y problemas genéricos de *clasificación* en tiempo real.

La conclusión de este apartado es que ambas perspectivas (simbólica y conexionista) han reconocido sus limitaciones y han decidido cooperar buscando formulaciones híbridas de tareas genéricas y soluciones cooperativas que usan ambos métodos tomando de cada uno lo más adecuado para la solución del problema. Aun así, la dimensión del problema global que aborda la *IA* nos obligará a ser modestos durante algunas generaciones. Las redes neuronales las estudiaremos en el último capítulo de este libro.

2 ASPECTOS METODOLÓGICOS EN IA

J. Mira y A.E. Delgado

En el primer capítulo presentamos una revisión histórico-conceptual del contenido de la IA como ciencia y como tecnología y desde la doble perspectiva extenso-intenso. A partir del capítulo tercero vamos a entrar en el desarrollo de contenidos específicos. En este segundo capítulo abordamos algunos aspectos metodológicos a los que usualmente no se les dedica mucha atención pero que nosotros consideramos esenciales para el avance de la IA como ciencia y como tecnología.

Es usualmente reconocido que la IA carece de metodología, comparada no sólo con ciencias establecidas, como la Física, sino incluso en comparación con otras ramas de la computación en las que se especifica un objetivo y los procedimientos para alcanzarlos mediante un programa. Usualmente es además posible evaluar este programa, incluyendo sus limitaciones en cuanto a complejidad, eficiencia y coste en recursos físicos, temporales o humanos.

Desafortunadamente, en muchos campos de IA aplicada se siguen métodos más imprecisos y rudimentarios. Muchas publicaciones describen un programa que, según sus autores, duplica una función humana supuestamente inteligente, sin incluir otros contenidos que los propios del nivel simbólico, de forma que en general es difícil determinar qué calcula de verdad el programa, qué conocimiento ha sido representado y cuál se inyecta al interpretar el programa. Tampoco existen en general referencias a otras técnicas alternativas y a las eficiencias relativas [Schwartz, 1987]. Si esto es cierto en general, lo es más en los sistemas de aprendizaje, tal como veremos en el tema correspondiente.

Para intentar contribuir a la solución de este problema, y antes de entrar en materia específica, vamos a introducir en este capítulo la ***taxonomía*** *de*

niveles de computación *debida a David Marr (teoría, algoritmo, implementación) y el nivel de conocimiento, propuesto por Allen Newell en 1981, con lo que se completa una primera visión metodológica de la IA a través de la distinción de distintos niveles de computación. Se hace cierto énfasis en los problemas asociados al primer salto de nivel, a la reducción del nivel de conocimiento al nivel simbólico y al paso inverso de interpretación.*

El siguiente paso metodológico procede de la Física y de la Biología y consiste en introducir la figura del ***observador*** *y la distinción –a cualquier nivel– de dos dominios (el propio del nivel y el del observador). En particular, comentamos el hecho evidente, aunque no siempre explícito, de la* ***inyección de conocimiento*** *siempre presente en la reducción e interpretación de los procesos del nivel simbólico en el dominio del observador.*

El tercer aspecto metodológico que se aborda en este capítulo es la distinción y posterior integración de las ***perspectivas simbólica*** *y* ***conexionista*** *(redes neuronales) en la IA, en particular en el caso de las redes basadas en conocimiento.*

Aunque se harán algunas referencias al tema de las estructuras de tareas genéricas y al procedimiento de desarrollo de sistemas basados en conocimiento, el grueso de los comentarios sobre este tema se aplaza hasta el capítulo 9, dedicado al estudio de los Sistemas Expertos. Entonces ya habremos estudiado los distintos métodos de representación y los mecanismos de inferencia correspondientes y, por consiguiente, dispondremos de bases más sólidas para comentar los aspectos metodológicos correspondientes.

Al igual que en el capítulo anterior los contenidos de éste también aconsejan una doble lectura. La primera para captar los aspectos básicos y la segunda para integrar contenidos y comprobar en dominios concretos cuál es el significado de los distintos niveles de computación, el conocimiento inyectado, la teoría de los dos dominios y la complementariedad o equivalencia entre las perspectivas simbólica y conexionista.

2.1 NIVELES DE COMPUTACIÓN

Cuando nos enfrentamos a la complejidad de un sistema, tanto en tareas de ***análisis*** (en las que intentamos comprender su funcionamiento) como en tareas de ***síntesis***, en las que queremos obtener el sistema a partir de un conjunto de

especificaciones funcionales, es útil realizar las descripciones usando una ***jerarquía de niveles***, que nos permiten segmentar esa complejidad.

Así, en física es usual distinguir entre el nivel mecano-cuántico y relativista y el nivel clásico, que a su vez puede mirarse con una óptica ***microscópica*** o ***macroscópica***. Pensemos por ejemplo en el comportamiento de un gas encerrado en un recipiente de volumen V y a una temperatura T. Para describir el comportamiento de este sistema de N partículas a nivel microscópico necesitamos $6 \cdot N$ números (3 coordenadas de posición y 3 componentes de velocidad por cada partícula), junto con las leyes de la dinámica. Si ahora nos fijamos en la descripción del sistema a nivel macroscópico, las magnitudes características del nivel son la presión (P), el volumen (V) y la temperatura (T) y la dinámica viene descrita por la relación que las enlaza causalmente, $(PV = nRT)$.

En ***biología*** también es usual distinguir distintos niveles de organización e integración jerárquica, pasando del nivel ***protoplasmático*** (subcelular, con procesos fisico-químicos) al ***celular*** (bioquímico y eléctrico) al ***orgánico*** y al de comportamiento ***global*** (animal y humano).

Cada nivel de descripción está caracterizado por una fenomenología, un conjunto de entidades y relaciones y unos principios organizativos y estructurales propios (figura 2.1). Un nivel también se caracteriza por la existencia de un conjunto de restricciones que limitan su capacidad [Pylyshyn, 1988]. Cada nivel tiene su medio, que es el conjunto de señales que entiende, y con el que interactúa usando un lenguaje común, propio del nivel. Cuando la descripción de un nivel la realiza un observador externo, siempre existe una semántica propia (relación signo-significado) y un conjunto de cuestiones que son las pertinentes a ese nivel.

Es una fuente frecuente de error en *IA* el mezclar entidades pertenecientes a niveles distintos e intentar explicar datos y procesos de un nivel de alta semántica, como el nivel de conocimiento, a partir de estructuras de datos y operadores lógicos propios de niveles inferiores, tales como la electrónica digital o la teoría de autómatas.

Todos los niveles están parcialmente cerrados a organización y poseen un conjunto de herramientas formales propias (lógica, teoría de autómatas, algoritmos, lenguajes naturales). Este cierre organizativo y estructural permite la estabilidad del nivel y su inclusión en una jerarquía en la que cada nivel enlaza con el inferior mediante un proceso de ***reducción*** de acuerdo con unas leyes

preestablecidas e inequívocas (programas traductores, compiladores e intérpretes) y con el superior mediante procesos de ***emergencia*** que para ser comprendidos necesitan de la ***inyección*** de ***conocimiento*** por parte de un observador externo que añade semántica a las entidades del nivel inferior. Del conocimiento detallado de lo que ocurre en todos los inversores y biestables de un computador, no se puede deducir el algoritmo ni las estructuras de datos del programa que se está ejecutando en ese momento. Análogamente, del conocimiento del algoritmo no se puede deducir el contenido a nivel de conocimiento de la computación que se ha programado. Tampoco el paso inverso es unívoco. Hay varios algoritmos posibles para una misma computación y hay múltiples implementaciones posibles para un mismo algoritmo (serie, paralelo, distinto lenguaje y entorno y, finalmente, distinta máquina física).

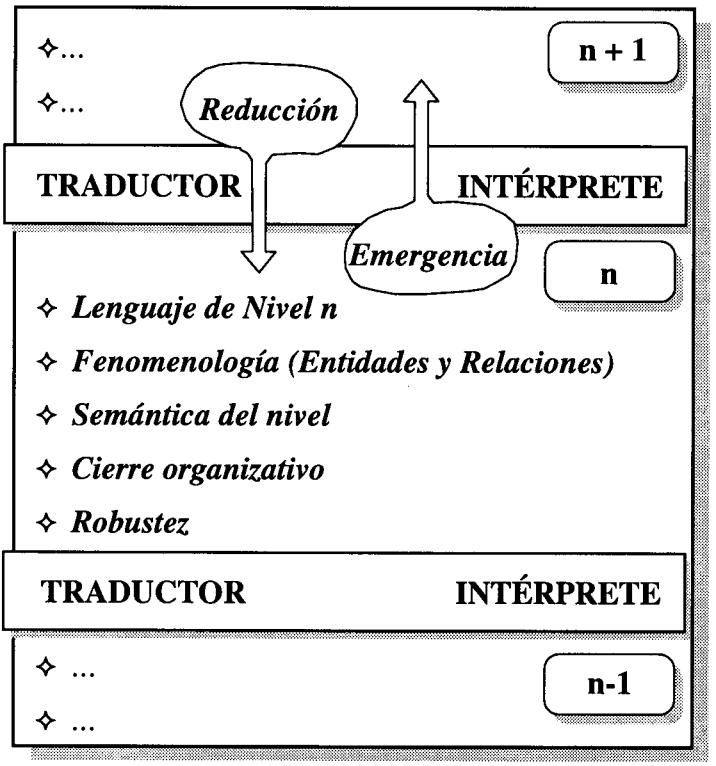


Fig. 2.1. Jerarquía de niveles de descripción.

Las características específicas de los distintos procesos de emergencia y reducción dependen del dominio, de la tarea y de la perspectiva (análisis o

síntesis) en donde estamos distinguiendo niveles de organización. Entre estas características es básica la robustez frente a perturbaciones sintácticas y semánticas. Los niveles inferiores son mucho menos robustos que el nivel de conocimiento.

En ***computación***, la teoría de niveles fue introducida por David Marr [1982] y Allen Newell [1981], aunque existían antecedentes claros en [Chomsky, 1968] quien introdujo los conceptos de "***competencia***" y "***ejecución***" para distinguir los lenguajes naturales de los sistemas arbitrarios y formales de manipulación de símbolos. Tener dominio de una lengua significa ser capaz de comprender lo que alguien nos dice y responder con otras señales que muestren que transportan una interpretación semántica e intencionada del primer mensaje. Las sentencias de un lenguaje poseen un significado intrínseco determinado por ciertas reglas. Quien posee estas reglas decimos que ha desarrollado una ***competencia lingüística*** específica. Sin embargo, el uso de ese lenguaje (la ***ejecución***) no es un simple reflejo de esas reglas, sino que comprende otros muchos factores (conocimiento adicional de la persona que habla, contexto, creencias, etc...) que son determinantes para comprender y generar el discurso. La competencia lingüística es separable de la ejecución que está a otro nivel. El paralelismo de la obra de Chomsky con la de Newell se consigue al asociar el concepto de competencia lingüística de Chomsky con el nivel de conocimiento de Newell.

Por otro lado, y tal como señala el propio Newell, las ideas latentes en la descripción de la computación por niveles están presentes en la práctica ordinaria de todos los profesionales del campo. La clave está en hacer explícitas estas ideas y en tenerlas en cuenta a la hora de analizar o sintetizar un programa de *IA*.

Todo nivel de computación admite una representación general en términos de un ***espacio de entradas***, un ***espacio de salidas*** y un conjunto de ***reglas de transformación*** que enlazan las representaciones en ambos espacios, tal como se ilustra en la figura 2.2. Un punto importante es reconocer que estos espacios son de ***representación***, con una estructura signo-significado que sólo adquiere valor en el dominio del observador, como veremos más adelante y que es característica de la gramática de ese nivel. De acuerdo con Chomsky, podemos decir que la gramática del lenguaje $L^{(n)}$, propia del nivel n genera y reconoce un conjunto de acoplamientos (X, S), donde X es la representación física o formal de las señales que acepta el nivel y S es la interpretación semántica asignada a X por las reglas del lenguaje característico de ese nivel. Estas gramáticas para los

lenguajes formales propios de toda la computación están ***predefinidas***, son absolutamente rígidas y se estudian en teoría de la computación bajo el nombre de lenguajes regulares, independientes del contexto y estructurados por frases junto con sus autómatas equivalentes. El problema que tiene la *IA* es que el lenguaje de representación que necesita para modelar el conocimiento humano está muy cerca del lenguaje natural y para este nivel todavía no conocemos una gramática universal suficientemente robusta frente a perturbaciones sintácticas y semánticas.

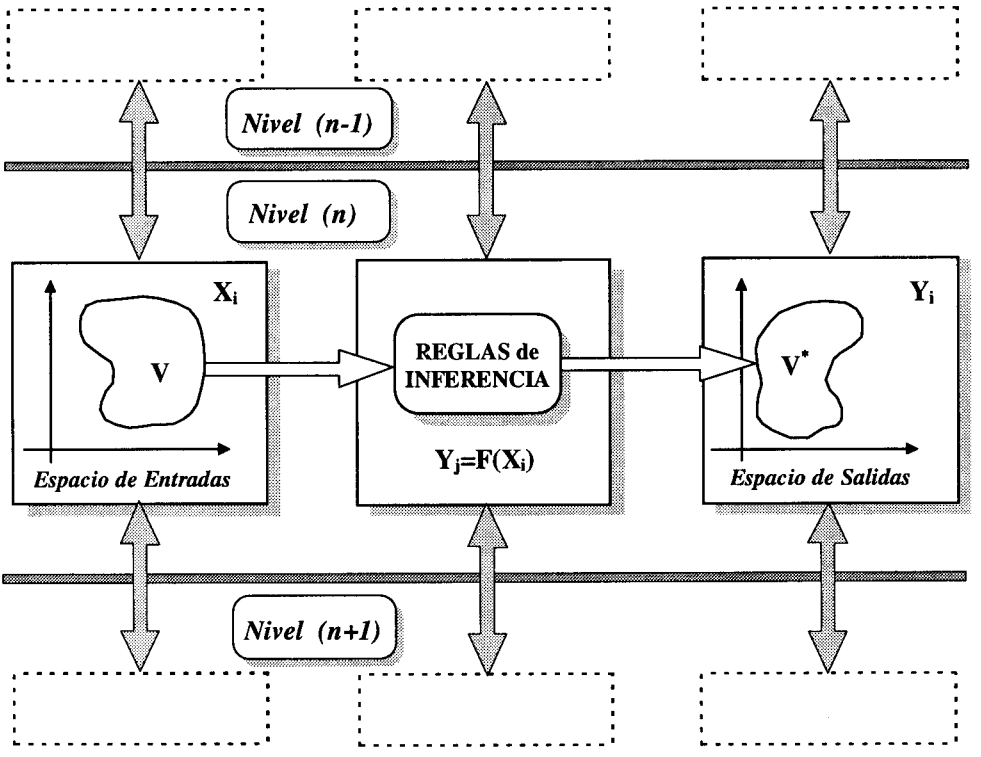


Fig. 2.2. Modelo de computación en un nivel como transformaciones que enlazan dos espacios de representación usando reglas y las primitivas de ese nivel.

El espacio de entradas es, en principio, un espacio multidimensional de aquellas características del proceso que se consideran relevantes. Así, por ejemplo, si estamos interesados en la percepción visual, el espacio de entradas viene determinado por la naturaleza física de los estímulos y por sus propiedades

relevantes (iluminación, brillo, color, contraste) y el tiempo. Cuando se consideran sistemas que no están enfrentados directamente a un mundo físico exterior, como puede ser un problema de integración simbólica o de búsqueda heurística en un espacio de estados o los potenciales eléctricos en corteza cerebral, la definición de las coordenadas del espacio de entradas tiene que ver con nuestro conocimiento del problema y nuestras concepciones teórico-prácticas acerca de las estructuras del nivel. En sistemas de aprendizaje el espacio de entradas puede ser un espacio de características o de hipótesis. Lo importante aquí es recordar que en todos los casos a las distintas variables del espacio de entradas, X_i, se les asocia una tabla de significados, S_i, de forma que la representación completa son los pares (X_i, S_i).

El espacio de salidas es de nuevo un espacio de representación, también multidimensional en general, en el que se seleccionan los resultados de la computación en función del tiempo, las entradas y las reglas de transformación propias del proceso en ese nivel. De nuevo, cuando el nivel es bajo, las señales poseen baja semántica y el espacio de salidas puede ser físico, como en el caso de un manipulador en robótica. En general, sin embargo, este espacio será simbólico o lingüístico, con unas variables formales, Y_j, a las que de nuevo hay que asociar una tabla de semántica, S_j, para obtener la representación completa de los pares (Y_j, S_j).

Finalmente, la computación de un nivel se completa con la descripción de las transformaciones que producen nuevos valores en el espacio de salidas a partir de la secuencia de valores previos en ambos espacios. Aquí hay, sin embargo, una marcada diferencia entre los distintos niveles. A nivel físico existe una ley analítica que enlaza X_i con Y_j, independientemente de la semántica (S_i, S_j), de forma que podemos escribir: $Y_i = F(X_i, Y_j)$. Para la obtención de estos resultados formales no es necesaria ninguna referencia a los significados de las variables. Es un puro proceso de manipulación de símbolos. La asignación de significados a estos resultados (S_j) y su correlación con los de entrada (S_i), genera las interpretaciones de la computación a ese nivel. Estas interpretaciones sólo existen en el dominio del observador. Cuando el nivel es simbólico, también se pueden establecer correspondencias formales. El problema está en el nivel de conocimiento, donde tales correspondencias no se conocen (de lo contrario no sería un problema de *IA*). Nada decimos aquí de la dificultad asociada al problema de construir una representación de los espacios de entrada y salida adecuados para cada problema. Sin embargo su importancia es decisiva, tal como se ha visto en visión artificial y en aprendizaje simbólico. Una buena representación es una condición necesaria para la solución del problema. De

hecho, el esfuerzo asociado a toda computación se reparte entre una componente de representación y otra de transformación. En un caso extremo, cuando la representación es óptima la inferencia necesaria es trivial. El compromiso de la *IA* entre "solucionadores generales de problemas" y "espacios de representación del conocimiento" es paradigmático de lo que aquí estamos diciendo.

Veamos ahora la introducción de los tres niveles de computación, tal como los propuso Davis Marr en 1982. Marr buscaba una teoría computacional de la percepción visual pero sus propuestas son trasladables a todas las tareas propias de la *IA,* cambiando percepción por planificación, decisión o aprendizaje, por ejemplo. El punto de partida es reconocer que cualquier explicación de la percepción visual que se base sólo en el funcionamiento de las redes neuronales desde retina a corteza será absolutamente insuficiente. Lo que necesitamos tener es una "clara comprensión de lo que se debe calcular, cómo es preciso hacerlo, los supuestos físicos en los que se basa el método y algún tipo de análisis sobre los algoritmos que son necesarios para llevar a cabo ese cálculo"[Marr, 1982].

No basta entonces el conocimiento de la computación a nivel de procesadores físicos (electrónica digital y arquitectura de ordenadores) ni el nivel previo de algoritmos y estructuras de datos, "debe existir un nivel adicional de comprensión en el que el carácter de las tareas de procesamiento de información llevadas a cabo durante la percepción se analice y comprenda de modo independiente a los mecanismos y estructuras particulares que los implementan en nuestros cerebros" [Marr, 1982]. Si cambiamos cerebro por ordenador tenemos clara la propuesta de Marr. Para analizar o sintetizar una tarea computacional es conveniente usar tres niveles, tal como se muestra en la figura 2.3.

En el primer nivel tenemos los fundamentos teóricos de la computación, el planteamiento del problema en lenguaje natural y un posible esquema de solución en términos del conocimiento del dominio. Veremos más tarde como enlaza esta "teoría de cálculo" con el nivel de conocimiento de Newell [1981] y con la idea de tarea genérica de Chandrasekaran [Chandrasekaran, 1986; Chandrasekaran, et al., 1992] propuesta como un medio para modelar el conocimiento que queremos inyectar en un sistema. Cuando Marr introdujo este primer nivel usó como ejemplo el problema de la suma (caja registradora en un supermercado). Desafortunadamente, no todos los problemas de la *IA* son tan sencillos y por consiguiente no siempre es posible disponer de una "teoría del

cálculo". En muchas ocasiones debemos conformarnos con un conjunto impreciso e incompleto de especificaciones funcionales.

El segundo nivel de análisis o síntesis de un proceso es la elección de un lenguaje de ***representación*** para los espacios de entrada y salida y de un ***algoritmo*** que haga efectivas las transformaciones que enlazan esas representaciones. Este segundo nivel de D. Marr coincide con el nivel simbólico de A. Newell. Es decir, una vez que hemos descrito todo lo que conocemos de un proceso a nivel de conocimiento, hay que reducir esa descripción al nivel simbólico, en términos de un conjunto de estructuras de datos y procesos de transformación de esas estructuras.

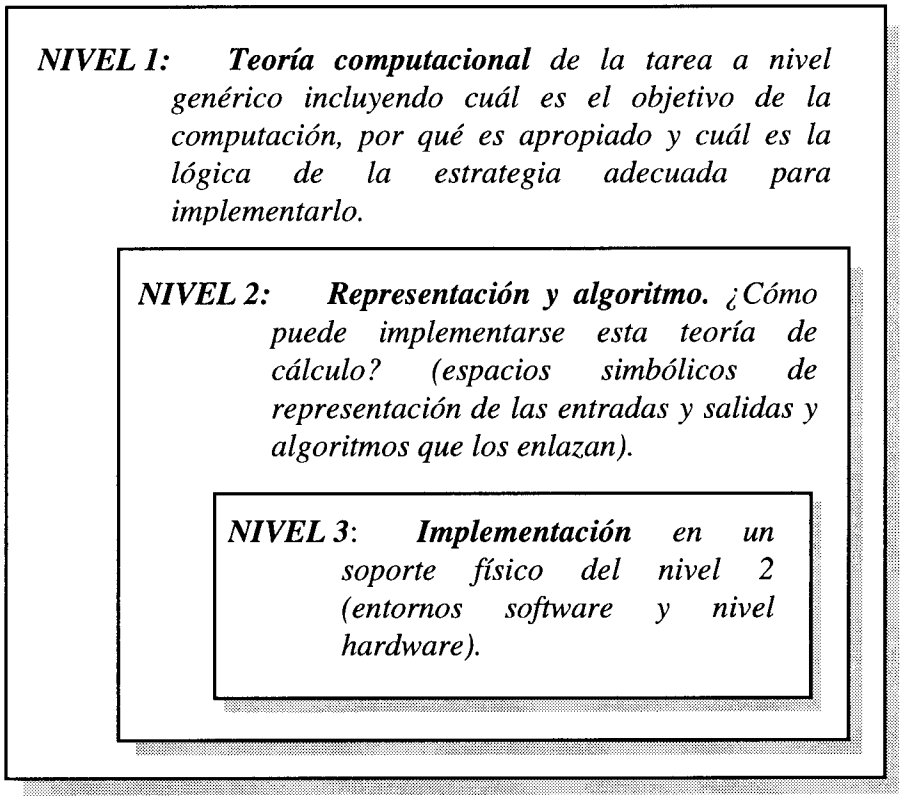


Fig. 2.3. Niveles computacionales introducidos por Marr.

El tercer nivel tiene que ver con todo el proceso de implementación que nos lleva del algoritmo a los procesadores físicos. Incluye la selección de un

lenguaje de programación y la construcción del programa. A partir de aquí ya existe un ***traductor*** que, tras el análisis léxico y sintáctico, genera el código objeto directamente ejecutable por los procesadores físicos. Obsérvese que en computación terminan nuestras preocupaciones siempre que alcancemos una descripción a partir de la cual un programa traductor nos conecta con el nivel físico. Esto ocurre entre el segundo y el tercer nivel, pero no entre el primero y el segundo. Es decir, no existe un procedimiento general y efectivo para traducir el nivel de conocimiento al nivel simbólico para el tipo de tareas que aborda la *IA*.

Para una tarea concreta, tanto en biología como en computación, los tres niveles de descripción están relacionados, pero no de forma unívoca y causal. Cuando se baja de nivel siempre se pierde información, tal como es fácil de comprobar si intentamos después recorrer el camino inverso (de los procesadores al nivel simbólico y de aquí al de conocimiento o "teoría del cálculo"). La razón es que no hay una representación única en el nivel (*n-1*). No está unívocamente determinada la máquina que implementa un algoritmo ni el algoritmo que resuelve un problema. La figura 2.4 ilustra estos cambios de nivel.

La hipótesis fuerte de la *IA* es que a pesar de estas pérdidas de semántica al bajar desde el nivel de conocimiento al nivel de procesadores físicos donde la semántica es intrínseca (lo que ocurre en electrónica digital está determinado por la arquitectura y las funciones locales de sus operadores básicos –puertas y retardos–) todavía es posible hacer computacional la inteligencia humana.

La única semántica bien establecida para el nivel físico (electrónica digital) es la de Frege-Tarski que se preocupa de la verdad o falsedad de una afirmación. Para cada regla gramatical existe una regla semántica asociada que define el significado de la configuración en términos del significado de sus partes componentes. El disparo de un circuito inversor o el cambio de estado de un biestable representan la verdad o falsedad de una afirmación (la función lógica que produce su disparo). Sin embargo, hay muchas facetas del conocimiento humano que son difíciles de reducir a la verdad o falsedad de un conjunto de proposiciones. Sentidos, referencias, creencias, contradicciones e incertidumbres son componentes usuales del razonamiento humano representadas por el lenguaje natural a nivel de conocimiento. Al intentar codificarlas siempre se produce una *pérdida* de semántica que se inyecta ("recupera") posteriormente al interpretar (de nuevo en lenguaje natural) las supuestas funcionalidades de un programa (primer salto de la figura 2.4). En el segundo salto, al pasar de algoritmo (nivel simbólico en general) al nivel físico, parte del conocimiento se

oculta (pero no se pierde) tras las tablas de semántica de las primitivas del lenguaje.

Finalmente, en el nivel físico la semántica es de nuevo de Tarski. Estructura y función coinciden y cuando un circuito se dispara nos está diciendo que es verdad la proposición que activa sus entradas.

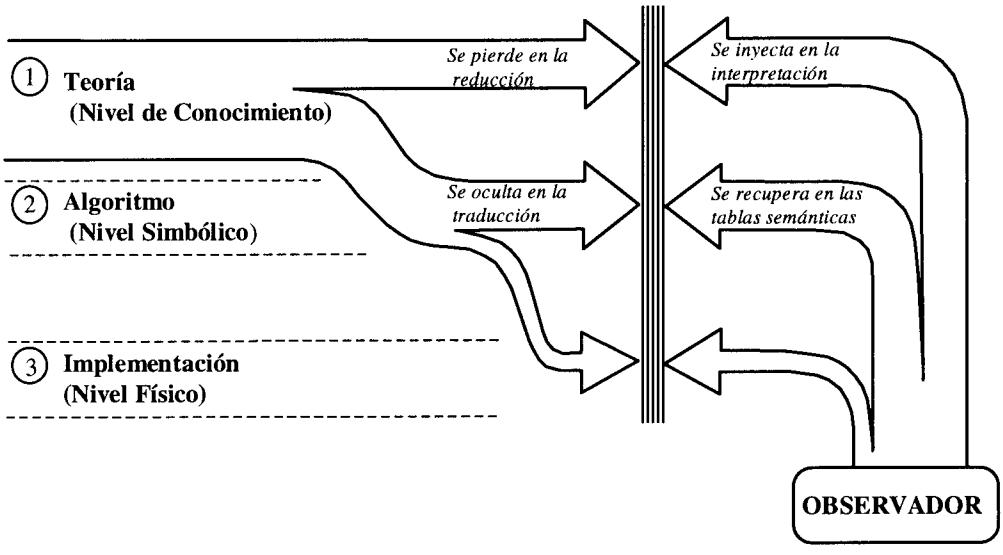


Fig. 2.4. Pérdidas de conocimiento en el proceso de reducción de un nivel al siguiente inferior y necesidad de inyectar ese conocimiento en el proceso inverso de análisis e interpretación del significado de un cálculo en los tres niveles.

2.2 EL NIVEL DE CONOCIMIENTO DE ALLEN NEWELL

Aunque ya hemos comentado que el nivel de conocimiento coincide parcialmente con el nivel de "teoría de la computación" de D. Marr, la coincidencia no es total porque existen muchos procesos que son descriptibles a nivel de conocimiento para los que todavía no disponemos de una teoría. Además, estos procesos son precisamente los más genuinos de la *IA*.

Newell introdujo en 1981 el nivel de conocimiento como un nuevo nivel de descripción por encima del nivel simbólico, que caracteriza el

comportamiento de un sistema en términos de sus metas, su conocimiento, sus creencias, sus procedimientos de inferencia y un principio general de "racionalidad" que garantiza que los agentes usarán ese conocimiento y esos mecanismos de razonamiento para alcanzar sus metas. La clave del nivel es su carácter abstracto, genérico e independiente tanto del dominio como de los lenguajes usados para representar el conocimiento (marcos, reglas, lógica o redes) y para usarlo (inducción, deducción o abducción). Busca entonces arquitecturas generales y reusables para especificar tareas, en línea con el concepto de ***competencia*** introducido por Chomsky, de la que hemos hablado en páginas anteriores y las metodologías tipo KADS [Tansley & Hayball, 1991], de las que hablaremos en la sección 2.4.2 y en el capítulo noveno, dedicado a los *SE*.

Pylyshyn ha asociado el nivel de conocimiento al cálculo intencional, que tiene que ver con los procesos que podemos explicar recurriendo al contenido semántico de las representaciones, sin bajar a su forma (sintaxis) y recientemente Clancey [1993] ha señalado el carácter relativista de las descripciones en el nivel de conocimiento. Son las descripciones de ***un agente*** en ***su medio*** y depende de sus metas y de los procedimientos usados para alcanzarlas. Estas descripciones nos permitirán predecir el comportamiento del sistema ante situaciones semánticamente análogas, aunque las estructuras de datos que las soporten sean completamente diferentes.

Una computación intencional (a nivel de conocimiento) es ***invariante*** ante cambios en la representación a nivel simbólico, de la misma forma que una computación a nivel simbólico es invariante ante cambios en el nivel de implementación.

En 1986 Dietterich aplica las ideas de Newell al problema de la adquisición de conocimiento en los sistemas de aprendizaje, y concluye que muchos sistemas a los que sus autores llaman "de aprendizaje" aparecen totalmente estáticos a nivel de conocimiento. Es decir, son capaces de mejorar sus prestaciones como consecuencia de cambios en el nivel simbólico pero sin modificar su conocimiento. Otra conclusión importante del trabajo de Dietterich es que el comportamiento de los programas de aprendizaje no puede describirse ni predecirse a nivel de conocimiento. Estos programas realizan saltos inductivos no justificados. Esta clasificación de los sistemas de aprendizaje usando el nivel de conocimiento quedará algo más clara cuando introduzcamos en el apartado siguiente la figura del ***observador*** y la teoría de los ***dos dominios***. Intentaremos

completar esta distinción en el capítulo 10, dedicado al estudio del aprendizaje simbólico.

Volvamos ahora al trabajo original de Newell [1981], en el que se plantea tres objetivos: (1) razonar acerca de la naturaleza del conocimiento, (2) proponer la existencia de un nivel específico del conocimiento cuyas entidades básicas son las creencias, objetivos, planes e intenciones y el principio de racionalidad, que conecta causalmente las intenciones con las acciones y (3) la descripción de este nivel y sus conexiones con los niveles inferiores (simbólico y de implementación).

En el primer punto, Newell distingue entre el conocimiento y su representación. El esfuerzo de Newell en el primer punto se centra en distinguir el conocimiento de sus posibles representaciones, y dotar al primero de entidad propia, de forma análoga a como Chomsky distinguía ***entre competencia lingüística*** y ***ejecución*** el el lenguaje natural. ¿Cuál es, para Newell, la naturaleza del conocimiento?. ¿Qué tiene un sistema cuando decimos que posee un conocimiento determinado?.

> *Si un sistema tiene –y usa– una estructura de datos o un procedimiento y el programador acepta que con una tabla de semántica adecuada estas entidades del nivel simbólico representan otras entidades específicas del dominio (objeto, concepto, relación causal, etc.), decimos que esa representación posee la versión reducida del conocimiento inicial.*

Aceptado que el conocimiento es una entidad previa y totalmente distinguible de su representación, Newell propone un nuevo nivel –nivel de conocimiento–, situado sobre el nivel simbólico y caracterizado por el conocimiento como medio y el principio de racionalidad como ley general de comportamiento.

Para poner este nivel en relación con los otros (simbólico y de implementación) se introducen cinco aspectos para caracterizar los niveles: sistema, medio, componentes, leyes de composición y leyes de comportamiento (figura 2.5). El ***sistema*** en el nivel físico es el ordenador, en el simbólico es la visión que del mismo tiene un programador que conoce el lenguaje y el sistema

operativo y en el nivel de conocimiento, el "agente" inteligente (el experto humano que posee el conocimiento).

El ***medio*** es la materia o la forma que va a ser procesada en la representación propia de cada nivel. Así, a nivel físico sólo tenemos estados de corte o saturación en los transistores de salida de los inversores con los que se construyen las puertas lógicas (niveles lógicos 0 y 1). A nivel simbólico la materia son los símbolos y sus expresiones relacionales, de acuerdo con la gramática del nivel que nos va a garantizar una traducción inequívoca al nivel físico. En cambio, a nivel de conocimiento la materia es el propio conocimiento que tenemos que ***reconstruir*** usando un lenguaje de representación (marcos, reglas, redes) que facilite su reducción al nivel simbólico.

	SISTEMA	MEDIO	COMPONENTES	OPERADORES	LEYES de COMPORTA-MIENTO
NIVEL de CONOCIMIENTO	Agente Inteligente	Conocimiento	· Metas · Acciones · Intenciones	Los que usa el lenguaje natural	Principio de Racionalidad
NIVEL SIMBÓLICO	Visión del Programador de Ordenador	Símbolos y Expresiones	Operadores y Memorias (Primitivas del Lenguaje)	· Designación · Asociación	Interpretación fija
NIVEL DE IMPLEMENTACIÓN	E. Digital + Arquitectura de Computadores	Vectores Booleanos y Estados Lógicos de puertas y biestables	· Registros · ALU's · Decodificadores de Instrucciones	Álgebra de Boole	Teoría de Autómatas

Fig. 2.5. Aspectos característicos propuestos por Newell para especificar los distintos niveles de computación.

Las ***componentes*** constituyen los procesos primitivos y básicos, a partir de los cuales se pueden construir todos los demás usando de forma repetida ciertas leyes de composición. En el nivel físico son los registros que almacenan palabras de n-bits en su doble caracterización, como dato y como instrucción y las operaciones elementales de la unidad aritmético-lógica que operan sobre el contenido de dos registros y descargan el resultado en un tercero. A nivel simbólico las componentes son las primitivas del lenguaje que usa el

programador, sus operadores y las estructuras de control. Finalmente, en el nivel de conocimiento las componentes son metas, acciones, propósitos, intenciones, creencias, deseos y significados. Este nivel es puramente ***semántico***. En él se razona sin bajar a la representación, y su lenguaje propio (extraordinariamente robusto) es el lenguaje natural, del que no hay compiladores conocidos. El conocimiento no es finito.

Las ***leyes de composición*** del nivel físico son las propias de la lógica combinatoria (álgebra de Boole) y a nivel simbólico se convierten en procesos de designar y asociar. A nivel de conocimiento estas leyes de composición no son conocidas. Se manifiestan a través del lenguaje natural y tienen que ver con la forma en la que nuestros deseos y actitudes se convierten en acciones. Gran parte de los trabajos sobre causalidad e intencionalidad [Rives, 1994] buscan la formalización de estas leyes de composición, aunque existe la sospecha de que todo cálculo formalizable lo es en extenso.

Las ***leyes de comportamiento*** del nivel físico son las propias de la lógica secuencial (Teoría de Autómatas). A nivel simbólico son las correspondientes a la interpretación secuencial de los programas. Finalmente, a nivel de conocimiento las leyes son las del ***principio de racionalidad***, o principio de causalidad semántica:

- *"Si un agente conoce que una de sus acciones conducirá a sus metas, entonces seleccionará esa acción"*
- *"Conocimiento es cualquier cosa que se pueda adscribir a un agente de forma que su conducta se puede calcular usando el principio de racionalidad"*

El enlace de este principio con el nivel simbólico se establece a través de un razonamiento por analogía:

Cuando decimos que el "programa ***x*** *conoce* ***k****" queremos decir que contiene alguna estructura de datos (****k*****) que representa a* ***k*** *y que esta estructura participa en la selección de la acción* ***A***** de la misma forma que demanda el principio de racionalidad para seleccionar la acción* ***A*** *en el nivel de conocimiento.*

Junto al aspecto positivo de que un sólo principio general pueda explicar y guiar la conducta de un agente a nivel semántico o intencional, existen fuertes problemas, entre ellos los asociados a la ***omnisciencia lógica*** (los agentes conocen todo, de forma que ante un conjunto de hechos y reglas infieren todas las consecuencias deducibles de forma instantánea) y a la falta de procedimiento efectivo para reducir el principio al nivel simbólico. Es decir, el paso de lo que tiene que hacerse (especificaciones funcionales) al cómo hacerlo.

Los niveles no son del todo independientes sino que están relacionados en términos de lo que es necesario añadir a uno de ellos para obtener el siguiente. Así, el ***medio*** de un nivel más un conjunto de estructuras estáticas adicionales definen el ***medio*** del nivel superior. Los vectores booleanos del nivel físico y su organización en registros más las estructuras para enlazar campos definen el medio del nivel simbólico. Desafortunadamente, eso falla en el enlace entre el nivel simbólico y el nivel de conocimiento, porque estamos tratando con unas entidades de partida (metas, propósitos, creencias y significados) que se describen usando lenguaje natural pero que no son fáciles de reconstruir en un lenguaje de representación formal.

Desde la perspectiva de *IA* aplicada, el problema ahora es encontrar un procedimiento (que en general será incompleto e impreciso) para reducir el nivel de conocimiento al nivel simbólico. Es decir, para pasar de descripciones en lenguaje natural de la tarea que realiza un agente a reescrituras (siempre recortadas por las limitaciones del nivel simbólico) en un lenguaje de representación del conocimiento directamente accesible al nivel simbólico, donde el principio de racionalidad, las metas, las creencias y las acciones se proyectan en estructuras de datos y algoritmos. Así, por ejemplo, para desarrollar un *SE* en una especialidad médica (sea oncología), empezamos hablando con el médico sobre su función diagnóstica y de terapia, intentamos descomponerla, buscamos una estructura de tareas genéricas que la represente (diagnóstico o planificación temporal) y, finalmente, seleccionamos los mecanismos de inferencia y los objetos estructurados que mejor representan el conocimiento dinámico y estático del dominio (dosis de drogas, cálculo de intervalos, etc...). Finalmente, todo se reduce a las primitivas de un lenguaje formal y a su representación lógica en el nivel físico.

Para facilitar este proceso de reducción se están buscando entidades de un ***nivel intermedio,*** entre las que se encuentran la teoría de agentes cooperativos (descomposición, segmentación y especialización), la estructura de tareas genéricas y la metodología KADS, entre otras propuestas. Cuando estudiemos el

capítulo dedicado al aprendizaje simbólico veremos también otras propuestas de niveles intermedio entre el nivel de conocimiento y el simbólico.

2.3 EL AGENTE OBSERVADOR Y LOS DOS DOMINIOS DE DESCRIPCIÓN

Para comprender el significado de la computación en los tres niveles descritos anteriormente (conocimiento, simbólico y físico) y valorar en su justo término muchos de los resultados de la *IA* en general, y de los sistemas de aprendizaje en particular, es conveniente usar la distinción de dos dominios de descripción: el ***dominio propio*** del nivel (***DP***) y el ***dominio del observador*** externo (***DO***) que interpreta la computación a ese nivel.

La introducción de la figura del observador y la distinción entre una fenomenología y su descripción, procede de la física y ha sido reintroducida y elaborada en el campo de la biología por Maturana [1975] y Varela [1979] y en la *IA* y la computación neuronal por Mira y Delgado [1987, 1988 y 1991]. Recientemente ha sido aplicada al estudio del aprendizaje por Boticario [1994].

Al reconocer la existencia de un observador externo a la computación en la prescripción y en la descripción de las funcionalidades de un programa de *IA* estamos introduciendo la idea de distintos sistemas de referencia en los que se representan las magnitudes y sus significados (figura 2.6). Cuando observamos una computación a nivel físico o simbólico, las descripciones de lo observado deben realizarse en dos sistemas de referencia. Uno es el propio del nivel físico (donde las variables son vectores lógicos y los operadores proceden del álgebra de Boole) o el nivel simbólico (donde las variables son las estructuras de datos y los procesos que las enlazan, ambos representados en las primitivas del lenguaje formal usado). A este dominio, que engloba a los niveles físico y simbólico, le llamamos dominio ***propio*** (*DP*) o autocontenido. El otro dominio es el del ***observador*** (*DO*) que usa el lenguaje natural para describir y dotar de significado a los procesos del *DP*.

Recordemos que cada nivel está caracterizado por una fenomenología y por las propiedades intrínsecas de las entidades que lo constituyen. Así, en las descripciones en su propio dominio (tanto dentro del nivel físico como del simbólico) todo lo que ocurre es ***causal***, las relaciones son de ***necesidad***. Lo que ocurre es "lo que tiene que ocurrir" porque estructura y función coinciden y las conexiones entre las magnitudes observables siguen sus leyes propias. En el *DP*

del nivel físico los procesos son inseparables de los procesadores que los realizan. Los inversores invierten, los sumadores suman, un biestable D retarda la señal un intervalo que es función del reloj y de su estructura interna, las ALU's realizan las operaciones aritmético-lógicas de acuerdo con su tabla de verdad, etc. En todos los casos, la semántica es intrínseca.

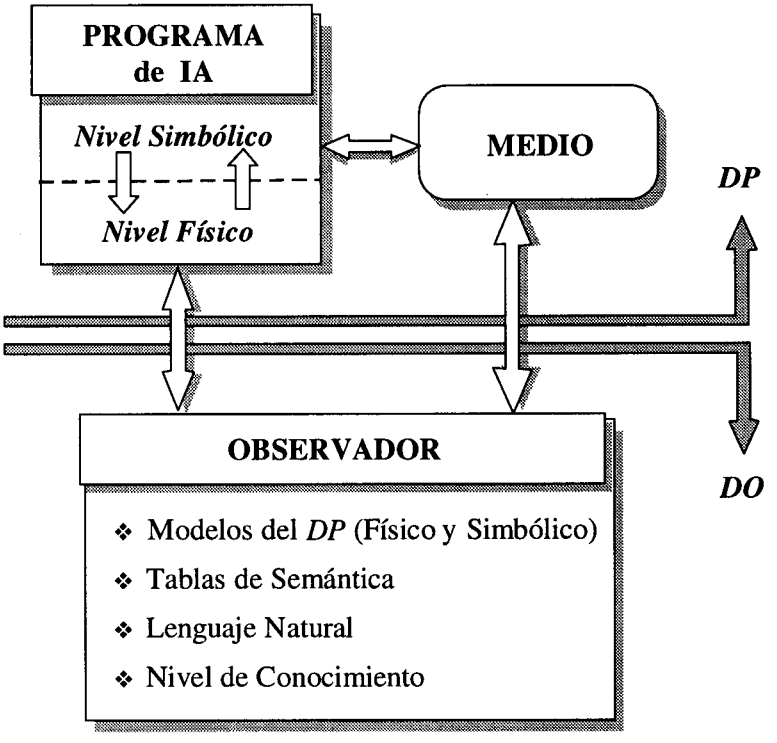


Fig. 2.6. Introducción del observador y distinción entre los dos dominios de descripción.

De forma análoga, en las descripciones propias del nivel simbólico también encontramos conexiones causales. Así, las relaciones entre las primitivas de un lenguaje de alto nivel están especificadas por su compilador, y siguen las leyes de su gramática.

Por otro lado, el observador siempre actúa a nivel de conocimiento. Su lenguaje es el lenguaje natural y su función de diseño e interpretación de la computación en los niveles físico y simbólico está caracterizada por su modelo del conocimiento, por las limitaciones de los otros dos niveles y por la ***inyección***

de conocimiento necesaria para recuperar en la interpretación todo lo que se perdió en el proceso de reducción de niveles. En el *DO* siempre hay más conocimiento del que se puede deducir de la sola consideración del contenido de los otros dos niveles.

David Marr explicaba de forma clara la necesidad de inyectar conocimiento para comprender una computación en los tres niveles: "tratar de entender la percepción estudiando sólo a las neuronas es como tratar de comprender el vuelo de los pájaros estudiando sólo sus plumas: simplemente, no es posible. Para comprender el vuelo de los pájaros tenemos que comprender la aerodinámica, sólo entonces nos aparece con pleno sentido la estructura de las plumas y las diferentes formas de las alas de los pájaros" [Marr, 1982].

Cuando la computación la realiza un ser vivo, la dificultad está en conocer las estructuras y funciones locales del dominio propio (problema de análisis), junto con la historia evolutiva que nos permite intuir el resto del conocimiento necesario para comprender la conducta observada (percepción, aprendizaje, etc.).

Cuando la computación la realiza un ordenador a través de un programa escrito en un lenguaje de alto nivel, el problema está en mezclar entidades y relaciones del *DP* con otras del *DO* con las que no tienen ninguna relación causal. Dicho de otro modo, es evidente que podemos hablar de onversores, registros y decodificadores (nivel físico), junto a las primitivas de un lenguaje (nivel lógico de implementación directamente reducible al nivel físico), usando el lenguaje natural (lo estamos haciendo ahora), pero hay que ser muy cuidadoso para no mezclar estas entidades que operan causalmente en su *DP* (electrónica y lenguaje de programación) pero no a nivel de conocimieto (*DO*). El problema importante en *IA* aparece cuando se mezclan las tablas de semántica de ambos dominios y se pretende dar significados de entidades propias del nivel de conocimiento en *DO* a las entidades de los otros dos niveles (*DP*).

F. Varela (1979) distingue claramente entre las explicaciones en el *DO* y las explicaciones ***operacionales*** (causales en el *DP*). En ambos casos las entidades y los procesos del nivel están descritos en dos lenguajes consistentes por separado y susceptibles de referencias cruzadas. La diferencia está en que en las descripciones operacionales (*DP*), los términos usados para referirnos a los procesos no pueden salirse ni en sintaxis ni en semántica del dominio en el que operan (no podemos sacarlos). En cambio, en las descripciones en el nivel de conocimiento (*DO*), los términos usados pertenecen al lenguaje natural y hacen referencia al conocimiento del dominio pero sus referentes no están obligados a

seguir las leyes físicas o formales del *DP*. Los enlaces en *DO* no operan en *DP*, salvo que nos limitemos a los modelos que el observador posee del nivel físico y del simbólico. Es decir, salvo que sólo hablemos de electrónica, lógica combinatoria o teoría de autómatas. El resto de los significados, incluido el propio concepto de ***conocimiento***, se queda en el *DO*.

El punto metodológico importante es que en ***IA*** podemos y debemos usar descripciones en ambos dominios con tal que no generemos confusión al mezclar entidades de distinta semántica usando la enorme capacidad integradora del lenguaje natural. Un analizador sintáctico no tendría problemas al aceptar la frase "Ayer fui a cenar y bailar con GoldWorks". El error aparece al reconocer que estamos hablando de un paquete de "software".

Muchas de las críticas recibidas por la *IA* proceden de la falta de distinción entre las entidades del *DP* y las del *DO* en la reducción del nivel de conocimiento al nivel simbólico y en la posterior interpretación de las supuestas funcionalidades de un programa. La razón del error (voluntario o no) es que en las descripciones con lenguaje natural, las entidades propias del *DP* ylas del *DO* se mezclan y las palabras que las representan coinciden (comprensión ⇨ "comprensión"). Sólo cuando se plantea la reducción del nivel de conocimiento al nivel simbólico y se hacen explícitas las tablas de semántica aparece clara la distinción. Así tiene que ser, porque en el *DP* no hay propósitos, ni metas, ni agentes inteligentes, ni aprendizaje, ni conocimiento. Sólo estructuras de datos y algoritmos con leyes causales propias. Y más abajo, sólo estados lógicos en circuitos electrónicos con nuevas leyes propias absolutamente causales e inmutables. El error más frecuente en *IA* es partir de una fenomenología compleja, asignar variables y operadores de los niveles bajos e interpretar los resultados de nuevo en el nivel superior sin hacer mención explícita a las subidas y bajadas de semántica y al conocimiento externo que ha sido necesario inyectar en estos saltos de nivel. La ley, sin embargo, es clara: hay que explicitar el conocimiento y la semántica asociados a todas las entidades que no constituyen elementos causales intrínsecos a los niveles físico o simbólico. Inversamente, para interpretar en el *DO* una computación hay que añadir a las entidades del *DP* todo lo que es dispensable de la fenomenología del nivel, lo que no nos haría falta si nos quedamos sólo con la descripción formal. De acuerdo de nuevo con F. Varela, lo característico de las descripciones en lenguaje natural en el *DO* es que nos permiten ignorar los enlaces causales del nivel físico.

¿La entidad X juega un papel causal en el nivel simbólico (o físico)?

Sí ⇨ Entonces X pertenece al DP

No ⇨ Entonces X pertenece al DO

La figura 2.7 resume las relaciones entre el *DP* y el *DO*. En el *DO* el observador siempre opera a nivel de conocimiento y sus descripciones las realiza en lenguaje natural. En el *DP* se encuentran los niveles simbólico y físico y las relaciones entre estos dos niveles no ofrecen dificultad, de forma que cuando hablamos de los dos dominios siempre nos referimos a las relaciones entre el nivel de conocimiento y el nivel simbólico, salvo en la *IA* conexionista en la que se busca la reducción del nivel de conocimiento directamente sobre el nivel físico. De esta perspectiva hablaremos en el apartado final de este capítulo y en el tema correspondiente a la computación neuronal.

Volviendo de nuevo a la figura 2.7, vemos que las conexiones ***DO ⇨ DP*** están relacionadas con procesos de síntesis en los que se pasa de la formulación en lenguaje natural a la descripción correspondiente en el lenguaje formal del nivel simbólico. Inversamente, las conexiones ***DP ⇨ DO*** están asociadas con los problemas de análisis de las funcionalidades de un sistema de *IA* (observación taxonómica e interpretación semántica).

En el *DP* se pueden distinguir tres capas. La más profunda está relacionada con la estructura de tareas genéricas (clasificación, diagnóstico, planificación temporal, etc.) y con la arquitectura global del sistema. Su contenido debe valer para muchas aplicaciones e incluye ***el conocimiento estratégico*** independiente del dominio.

En la capa intermedia se encuentran las "herramientas y los métodos", es decir el conjunto de formalismos de representación (lógica, marcos, reglas, redes y cualquier otro formalismo híbrido que incluya una combinación de las técnicas básicas, por ejemplo mezclando marcos con reglas) y mecanismos de inferencia (inducción, deducción, abducción y las combinaciones resultantes de matizar estos procedimientos básicos con las características que aportan los lenguajes de representación: resolución, herencia, etc). Para una misma tarea genérica, podemos usar distintas combinaciones de inferencia sin hacer todavía uso directo del conocimiento del dominio, aunque con influencias obvias pues distintas tareas en dominios diferentes siempre aconsejan formas concretas de representación e inferencia.

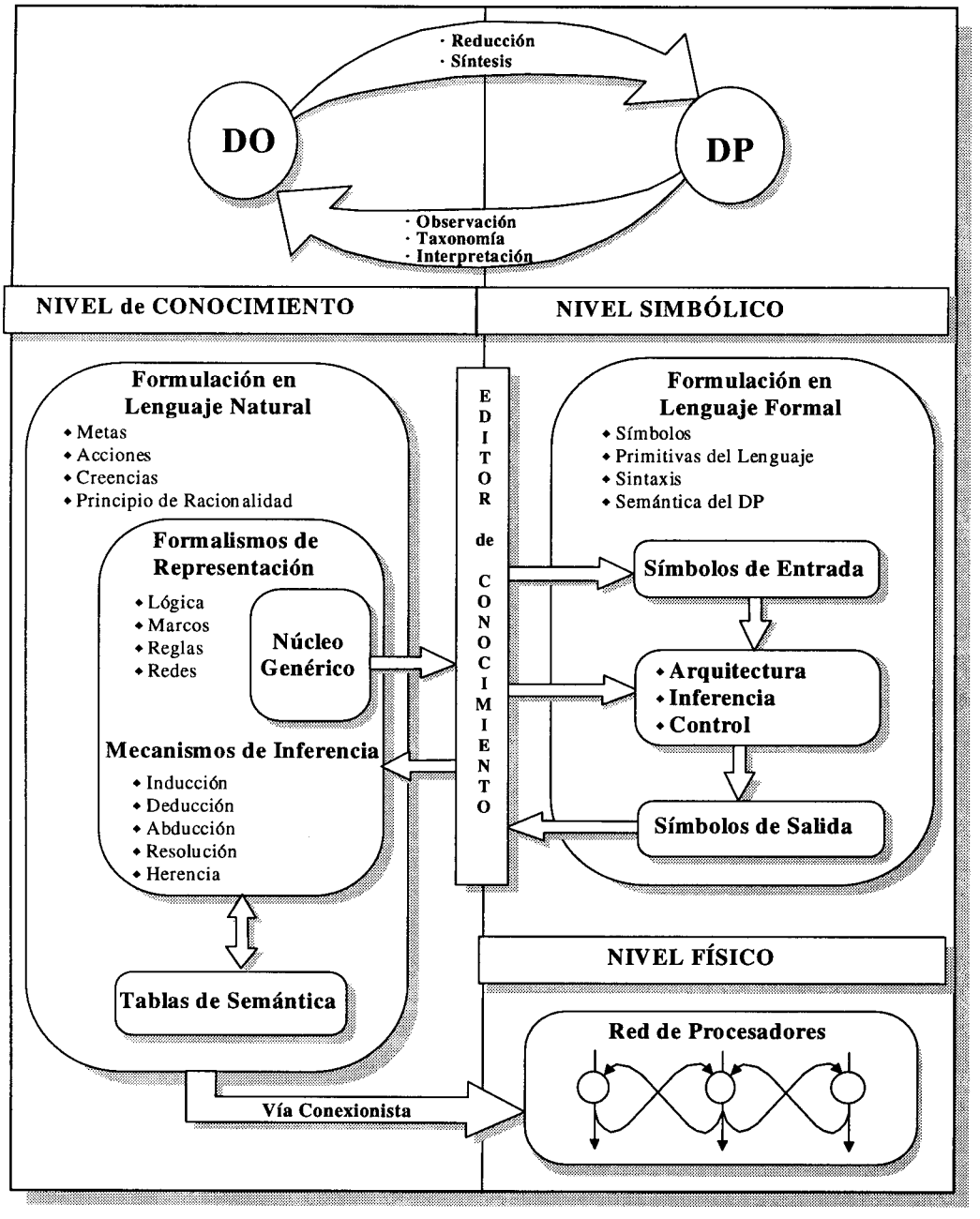


Fig. 2.7. Resumen de las relaciones entre el *DP* y el *DO*. El nivel de conocimiento siempre opera en el *DO*. Los niveles físico y simbólico pertenecen al *DP*.

La tercera capa, la más externa en el *DO*, incluye la arquitectura global de la tarea, los aspectos organizativos del modelo, los mecanismos de adquisición (modelado) del conocimiento del dominio, la segmentación en distintos agentes cooperantes y la asignación de tareas a esos agentes y, finalmente, la construcción de las tablas de semántica que se volverán a usar en la interpretación del nivel simbólico. En esta capa externa del *DO* se evalúan (y modifican) las herramientas seleccionadas en la segunda capa para rellenar los huecos del núcleo genérico y se seleccionan las estrategias de control.

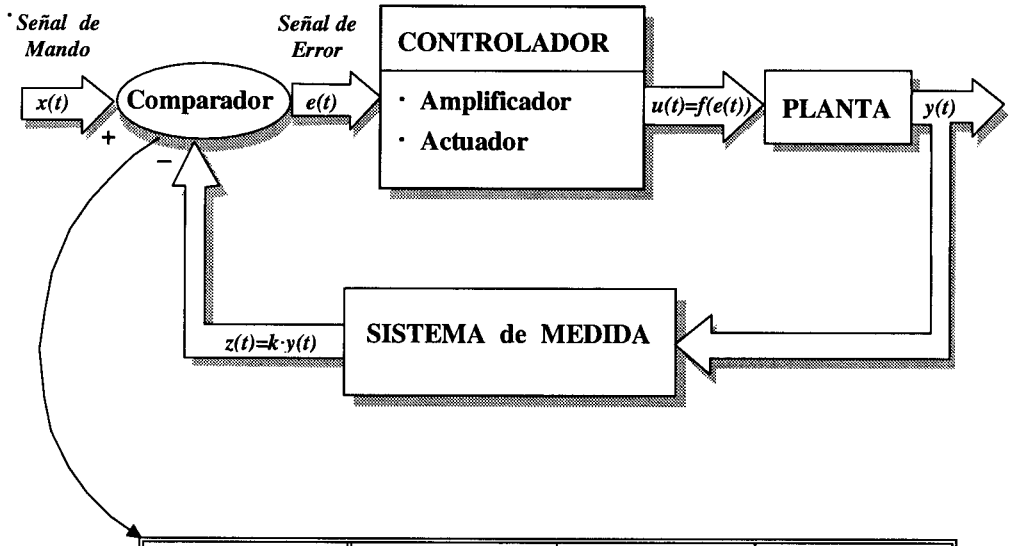
2.4 ESTRUCTURA DE TAREAS GENÉRICAS PARA MODELAR CONOCIMIENTO EN EL *DO*

Para facilitar el proceso de adquisición de conocimiento y su posterior reducción al nivel simbólico (inferencia y representación) se han intentado desarrollar métodos abstractos de modelar conocimiento en términos de un conjunto de ***tareas genéricas*** (***TG***) y ***métodos*** (***M***) para desarrollarlas. Tareas y métodos se usan como conceptos intermediarios para pasar desde los agentes y las metas de Newell hasta el nivel de implementación.

La idea básica es que el conocimiento puede ser modelado de acuerdo con un plan estratégico que enlaza tareas para conseguir una meta y que esas tareas son genéricas, es decir valen para un amplio grupo de problemas, independientemente del dominio específico y de las formas de representación.

Al ser estas *TG* de validez general nos permiten modelar el conocimiento de forma análoga a como lo hace la teoría de sistemas continuos. Así, se reconoce habitualmente que un sistema de control puede modelarse de acuerdo con el *diagrama de bloques* de la figura 2.8. La meta (macrotarea) es mantener constante el valor de una magnitud, $y(t)$, o hacer que esa magnitud siga la evolución temporal de una señal de mando, $x(t)$. Para conseguirlo se parte de un análisis del problema a nivel estratégico y se identifican las funciones que se consideran necesarias (comparación, amplificación, actuación y medida), y su estructura de conexión: lazo de realimentación que compara el valor de la magnitud de salida $y(t)$, con el valor de consigna, $x(t)$, y actúa para minimizar la diferencia que se está midiendo constantemente para informar al comparador. Obsérvese que este análisis y el esquema resultante son totalmente independientes de si el control es analógico o digital y de cuál sea la función de transferencia de la planta. En cada caso, los ***métodos***, las formas de ***representar***

el conocimiento y los mecanismos de ***inferencia*** variarán, pero la organización de la tarea permanece. Su esquema conceptual es el mismo.



TG	Método	Representación del Conocimiento	Inferencia
Comparación	• Resta aritmética e = x - k·y	• Variables Analógicas	• Analítica
	• Algoritmo Digital	• Vectores Booleanos • Lógica	• Deductiva (Reglas)
	• Matrices de Semejanza (Establece y refina)	• Marcos • Redes • Reglas	• Búsqueda Heurística

Fig. 2.8. Diagrama de bloques de un sistema de control usado para ilustrar la idea de estructura de *TG* como modelo de conocimiento.

Vemos en la tabla incluida en la figura 2.8 que el método puede ser la resta aritmética, un algoritmo digital de resta o un método de medida de semejanzas entre patrones que establece la analogía a un nivel alto y después refina. Análogamente la representación del conocimiento es la adecuada a los tres métodos: variables analógicas, vectores booleanos o marcos y redes. Finalmente, la inferencia es la correspondiente a estas representaciones: analítica en el caso analógico, lógico-deductiva en el digital y búsqueda heurística en el caso de las matrices de semejanza. Lo relevante a nivel de estructura de tareas genéricas es que en todos los casos estamos realizando la misma tarea (***comparación***)

Esta idea de usar modelos conceptuales análogos a los que usa la teoría de sistemas para modelar conocimiento no analítico fue introducida en la *IA* por Clancey [1985, 1993], junto con Chandrasekaran [1986, 1992], Breuker y Wielinga [1989] y Tansley y Hayball [1993], entre otros.

Así, las tareas genéricas son bloques funcionales que participan en la solución de una familia amplia de problemas. En general están asociadas a métodos que son procedimientos efectivos de cálculo adecuados para realizar esa tarea o para reducirla a otras más elementales hasta alcanzar un nivel de *TG* primitivas que ya pueden ejecutarse por el uso directo del conocimiento del dominio.

En un sistema de control las *TG* son: ***comparación***, ***amplificación***, ***actuación*** y ***medida*** y la estructura de *TG* es la realimentación negativa. En *IA* veremos muy pronto que las tareas genéricas son la clasificación jerárquica y heurística, el diagnóstico, la monitorización, el diseño, etc, y los métodos son heurísticos (generar y probar, seleccionar y traducir, ...). Finalmente, la contrapartida del nivel organizativo (lazo de realimentación), la representa en *IA* el conocimiento estratégico que nos guía en la selección y conexión de *TG*.

El analizar o sintetizar un sistema basado en conocimiento mediante una estructura secuencial o concurrente de *TG* es una metodología estructurada que nos permite inyectar en cada aplicación todo el conocimiento que es invariante (todos los sistemas de control usan la realimentación y las funciones de comparación y medida). Así, una comprensión clara de las relaciones entre las *TG* permite estructurar tanto el proceso de adquisición del conocimiento como su uso posterior en un diseño modular.

En el ***análisis*** modelamos un segmento del razonamiento humano especificando lo que debe hacer el sistema. En ***síntesis***, proponemos una solución estructurada diciendo cómo debe resolverse el problema usando un

conjunto muy limitado de módulos genéricos (*TG*) cada uno de los cuales se concentra en un aspecto particular del proceso de solución. En ambos casos el trabajo en *TG* busca modelar el conocimiento mediante un lenguaje de bloques funcionales de alto nivel que facilite el enlace del nivel de conocimiento con el nivel simbólico. Es más fácil asociar formas de representación e inferencia a las *TG* primitivas y a sus métodos que a los agentes de Newell

2.4.1 La Clasificación como Tarea Genérica

Se diga o no explícitamente, muchos de los programas de *IA*, tanto simbólica como conexionista, están basados en la estructura genérica de un ***clasificador*** que asocia configuraciones de entrada, $\{X_i\}$, a configuraciones de salida, $\{Y_j\}$, tal como se muestra en la figura 2.9.a. Las entradas son datos, vectores de características o hipótesis iniciales. Las salidas son configuraciones de categorías o clases que, dependiendo del dominio, representan decisiones diagnósticas, posibles fallos de un sistema, o conceptos seleccionados. Ambos espacios deben estar especificados en extenso y de forma completa antes que el clasificador empiece a inferir porque la clasificación lo que hace es ordenar las categorías de salida y/o seleccionar una de ellas. Los métodos usados dependen del tipo de clasificación. Así, en la clasificación jerárquica el método suele ser "establecer y refinar". En la clasificación heurística, el método es la comparación a nivel abstracto, tal como se muestra en la figura 2.9.b, y en la clasificación conexionista, el método es el ajuste de parámetros mediante la minimización de una función de coste, tal como el error cuadrático medio entre la salida real y la deseada para una familia de pares (entrada, salida) usados en el entrenamiento de la red. Comentaremos aquí los últimos (heurístico y conexionista) y después introduciremos la metodología KADS que usa una librería de tareas genéricas en tres grandes apartados: ***análisis***, ***modificación*** y ***síntesis***. Al nivel metodológico al que nos estamos moviendo en este capítulo no nos interesa tanto la extensión y profundidad de los contenidos que comentamos como el hacer notar su importancia en el proceso de modelar el conocimiento para diseñar SE.

En 1985 Clancey introdujo la ***clasificación heurística*** como tarea genérica subyacente a muchos *SE* y sugirió su descomposición en tres subtareas: abstracción, equiparación heurística y refinamiento [Clancey, 1985].

La abstracción de los datos permite separar el valor numérico de una variable (temperatura = 39 grados) del módulo de conocimiento asociado de forma implícita. Por ejemplo, que ese valor de temperatura es alto, tiene valor

sintomático, y pertenece a una jerarquía y por consiguiente puedo usar la clase a la que pertenece para razonar (buscar asociaciones) a un nivel superior.

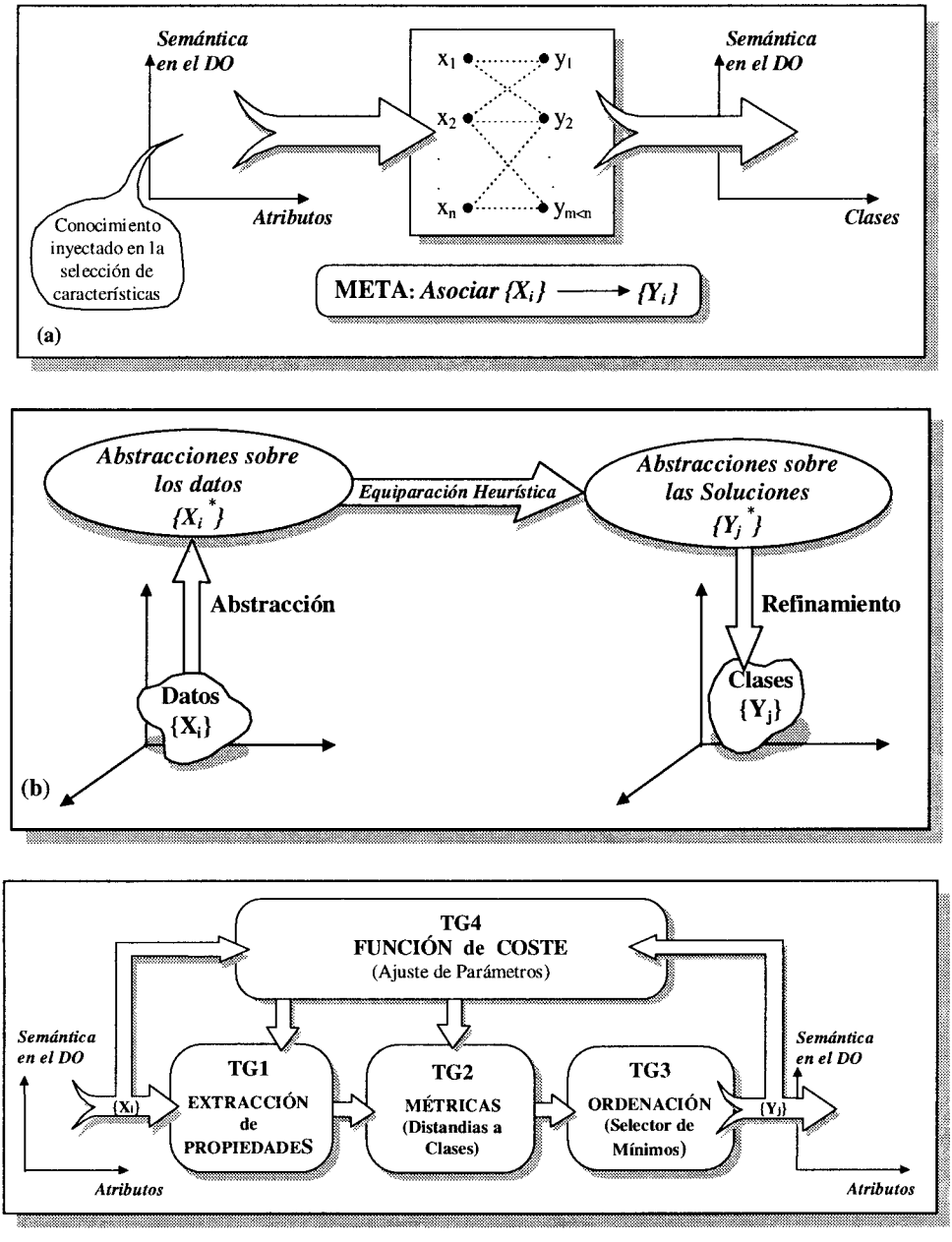


Fig. 2.9. Tarea genérica de clasificación: (a) Nivel de conocimiento. (b) Propuesta de Clancey. (c) Propuesta conexionista con tres capas (subtareas).

La equiparación heurística permite de nuevo buscar asociaciones entre "tipos generales de síntomas" y "tipos de patologías". Así alta temperatura indica fiebre y está asociada a infección, por lo que podemos acotar el espacio de búsqueda al área de las enfermedades, que cursan asociadas a una infección. Finalmente, el refinamiento está asociado al resto del conocimiento necesario para especificar el diagnóstico.

La clasificación conexionista (figura 2.9.c) usa en general el conocimiento del problema para definir una estructura con cuatro tareas genéricas, distribuidas por capas: extracción de propiedades, métricas, selección de máximos y refuerzo. Sea cual fuere la naturaleza de los datos a clasificar (señales de voz, imágenes, palabras,...), la clasificación conexionista comienza con la construcción de un espacio de características con una parte fija y otra autoprogramable por aprendizaje. Estas características pueden ser analíticas (filtros), algorítmicas (mínimo, máximo, histograma,...) o puramente simbólicas (condicionales).

La segunda *TG* incorpora las métricas asociadas a las clases de equivalencia. Su función es calcular la proximidad de cada punto del espacio de características a cada una de las clases. La tercera *TG* es la selección de la clase a la que la distancia es mínima o el cálculo de la función de pertenencia de ese punto del espacio de medida a las distintas clases (diagnósticos), en las formulaciones borrosas. Finalmente, la última *TG* del clasificador es el aprendizaje. Para cualquier especificación de las otras *TG*, siempre hará falta un procedimiento de evaluar la calidad o eficacia del clasificador y usar esa evaluación (el error o su valor cuadrático medio) para ajustar los parámetros de la red neuronal. De estas descripciones en términos de *TG* obtenemos "pistas" sobre los distintos tipos de conocimiento que vamos a necesitar, independientemente de cómo lo representemos después en la etapa de implementación.

2.4.2 La Metodología KADS

La metodología KADS para el desarrollo de sistemas basados en conocimiento se apoya en el nivel de *TG* descrito anteriormente [Breuker &Wielinga, 1989; Tansley & Hayball, 1993] y en la distinción de cuatro capas (estrategia, tareas, inferencias y conocimiento estático del dominio) para modelar el conocimiento. El ciclo temporal que se sigue en el desarrollo de un sistema consta de fases alternativas de análisis y síntesis, tal como se muestra en la tabla de la figura 2.10.

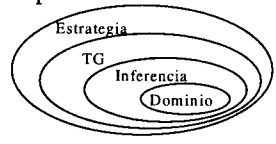
FASE	ETAPA	ACTIVIDAD	RESULTADO
ANÁLISIS	✧ Proceso global	❖ Analisis y Descomposición	◆ Documentación ◆ Descomposición del proceso ◆ Modelo de cooperación ◆ Especificación de la interfaz
	✧ Conocimiento del Experto	❖ Analiza el Conocimiento estático	◆ Estructuras del dominio
		❖ Selección del modelo de *TG* más adecuado	◆ Modelo de *TG*
		❖ Construcción del modelo	◆ 4 Capas: 
	✧ Restricciones	❖ Análisis de restricciones	◆ Modelo de restricciones
SÍNTESIS	✧ Global	❖ Subsistemas	◆ Arquitectura ◆ Definición de subsistemas ◆ Interface
	✧ *SBC*	❖ Definición de la Estructura	◆ Estructuras
		❖ Módulos Funcionales	◆ *TG*
		❖ Métodos	◆ Métodos específicos ◆ Repres. del conocimiento ◆ Inferencia
		❖ Elementos	◆ Especificación de los módulos y sus interfaces

Fig. 2.10. Descomposición de las etapas de análisis y diseño de SBC. Cada fase consta de distintas etapas y en cada etapa se distinguen diferentes actividades que dan como resultados los contenidos de la última columna..

Dentro de cada fase se pueden distinguir distintas actividades que producen resultados que actúan como entradas a las actividades siguientes. Las

actividades cumplen la misión análoga a la de los bloques funcionales en el diseño de sistemas continuos. Su interconexión especifica la composición del proceso global a partir de otros más elementales.

El proceso comienza con el análisis de la aplicación a nivel global, la adquisición del conocimiento del experto y el estudio de las restricciones. El análisis global permite una descomposición del problema al menos en tres módulos: El módulo basado en conocimiento, la interfaz y el resto de la aplicación (integración con bases de datos, toma de señales del mundo real, etc.).

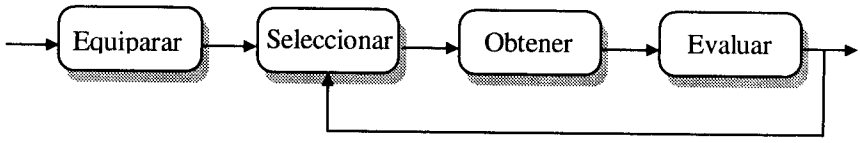
El análisis del conocimiento del experto busca construir un modelo de lo que se supone que sabe ese experto en relación con su especialidad. Para facilitar esta adquisición usamos un modelo "esqueletal" (vacío) con cuatro capas superpuestas. La interior incluye todo el conocimiento estático sobre las entidades y relaciones del dominio y puede usar marcos, reglas, redes semánticas y cualquier otro método de representar conocimiento estático.

La siguiente capa modela la inferencia. Está relacionada con las técnicas usadas en la repesentación del conocimiento estático pero no con su contenido específico. Es decir, hay distintos tipos de inferencia que pueden usarse para un mismo conocimiento estático

La tercera capa contiene la estructura de *TG* que hemos selecciomado o construido para la aplicación, con los tipos de inferencia que ahora ya son concretos. En la figura 2.8. hemos ilustrado una de estas estructuras, correspondiente a una tarea global de control. La monitorización, el diagnóstico, la clasificación y la planificación son otras tareas para las que también poseemos estructuras vacías que nos facilitan la adquisición del conocimiento.

Finalmente, la capa más externa se usa para modelar los aspectos estratégicos del conocimiento. Por ejemplo, cuando hay que modificar o cambiar una estructura. Cuando a las capas de estrategias y *TG* se les añade la inferencia y la representación del conjunto de hechos, conceptos y relaciones que especifican el conocimiento del dominio, decimos que tenemos un modelo de conocimiento. Aquí termina el proceso de análisis.

Si se ha adquirido el conocimiento de esta forma, la estructura del proceso de diseño del sistema queda ahora razonablemente clara. Supongamos que la estructura de tareas que nos sugiere el análisis es la siguiente:



El problema ahora es comparar los patrones de cada una de estas *TG* con los contenidos de nuestra biblioteca de *TG* para ver si encontramos módulos funcionales adecuados o, en caso contrario, obtener las modificaciones necesarias en los módulos existentes o sintetizar otros nuevos. El resultado de esta actividad es la estructura de *TG* más adecuada al problema.

La figura 2.11 muestra un resumen de una posible biblioteca de *TG* organizada en tres secciones: tareas de ANÁLISIS de sistemas, tareas de MODIFICACIÓN y tareas de SÍNTESIS. A su vez, estas tareas pueden subdividirse en otras más elementales tales como: comparar, seleccionar, confirmar, descomponer, verificar, especializar, abstraer, etc. Veremos en el tema dedicado al estudio del aprendizaje simbólico que es posible encontrar un conjunto de tareas primitivas para los paradigmas inductivo y deductivo. En el capítulo noveno, ampliaremos este apartado al hablar de la adquisición del conocimiento.

Tal como hemos comentado anteriormente, el objetivo de este apartado es de naturaleza cualitativa y está dentro de un capítulo cuyo propósito global es hacer énfasis en los temas metodológicos. Sólo queremos poner de manifiesto la necesidad de usar una metodología para pasar del nivel de conocimiento al nivel simbólico, al igual que hemos hecho antes énfasis en la necesidad de distinguir niveles de computación y dominios de descripción.

La propuesta de una estructura de *TG* para modelar conocimiento tiene el atractivo de toda teoría modular de sistemas (analógicos o digitales) y plantea la existencia de un conjunto de primitivas computacionales (tareas genéricas no reductibles) en términos de las cuales se podría modelar y sintetizar el conocimiento. Además, estos módulos serían reusables. Toda la electrónica analógica puede sintetizarse a partir de un único módulo funcional (el amplificador operacional) que cumple le función de "tarea genérica no reductible". Lo mismo ocurre con el operador NAND en la electrónica digital. Hay que decir sin embargo y en honor a la verdad que la metodología KADS que estamos comentando está todavía muy lejos de la precisión y eficiencia de las técnicas modulares habituales en teoría de sistemas analógicos o digitales.

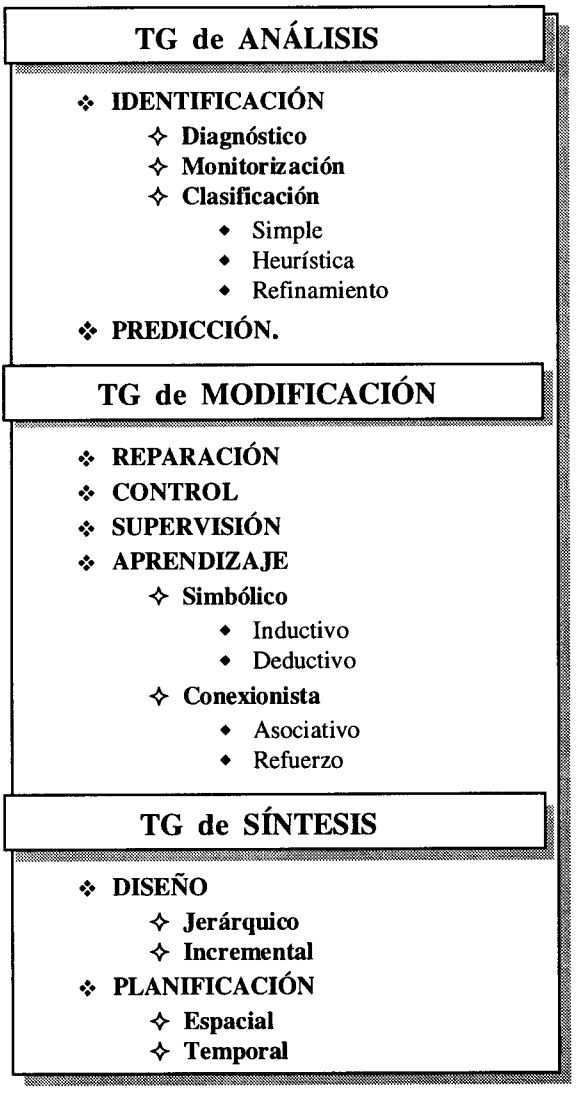


Fig. 2.11. Biblioteca de *TG* adaptada de [Tansley y Hayball, 1993].

2.5 *IA* SIMBÓLICA VERSUS *IA* CONEXIONISTA

El último apartado de este capítulo sobre aspectos metodológicos lo vamos a dedicar al comentario de las analogías y diferencias entre las dos perspectivas de la *IA* (simbólica y conexionista), intentando poner de manifiesto su complementariedad y su significado en la teoría de los dos dominios (*DP* y

DO). Una visión más completa de este tema se presenta al final del capítulo 11, tras haber estudiado las redes neuronales.

Tal como comentamos al estudiar la perspectiva histórico-conceptual de la *IA*, esta nació conexionista, fue después dominantemente simbólica y tras el renacimiento de la computación neuronal en los entornos de 1980, hemos llegado a una etapa de cooperación entre ambos alternativas, dependiendo de la naturaleza del problema, como dos técnicas complementarias y en algunos casos simbióticas.

Tras el estudio de los distintos niveles de computación (conocimiento, simbólico e implementación) y la distinción entre procesos y procesadores físicos que los soportan, junto con la introducción de la figura del observador que maneja descripciones en los dos dominios, es relativamente sencillo comprender la distinción entre computación simbólica y computación conexionista.

De hecho, toda computación termina siendo conexionista en el dominio propio del nivel físico en el sentido que se le ha dado aquí a estos términos. El simbolismo nace en el dominio del observador durante el proceso de reducción del nivel de conocimiento al nivel simbólico y en el proceso inverso de interpretación. Cuando la reducción se hace a través del nivel simbólico, terminando en las primitivas de un lenguaje, la *IA* es simbólica. La imprescindible traducción final al nivel físico la realizan un conjunto de programas que no son visibles al usuario. En cambio, cuando la reducción se hace directamente del nivel de conocimiento a la red de procesadores (ver la figura 2.6), la *IA* es conexionista.

El que los procesadores usuales en el nivel físico sean digitales en la *IA* simbólica (puertas NAND y biestables) y analógicos no lineales en la *IA* conexionista (suma ponderada seguida de una función de decisión tipo sigmoide) no es relevante, pues son equivalentes. Se puede realizar cálculo digital con lógica de umbral y cálculo analógico con procesadores digitales (la suma y el producto son fáciles de sintetizar en electrónica digital). La distinción a nivel de procesadores está en la necesidad de programación del caso simbólico, que en el caso conexionista se sustituye por el entrenamiento de la red neuronal por procedimientos correlacionales (asociación de estímulos) o por mecanismos supervisados de aprendizaje por refuerzo o minimización de una función del error que comete la red ante ciertas configuraciones de entrada para las que se conoce la salida deseada.

Se entiende por computación neuronal toda computación modular, distribuida, de "grano pequeño" y autoprogramable. Su arquitectura, en general, está organizada por capas sin o con realimentación (redes recurrentes) y la función local de cada uno de los procesadores suele ser la suma ponderada de sus entradas, seguida de una función de decisión no lineal (escalón abrupto o sigmoide). Las aproximaciones más interesantes son aquellas en las que se fuerza a que los procesadores sean ***autónomos,*** es decir, el aprendizaje, el control del modo de funcionamiento y el cálculo local deben ocurrir dentro del procesador y no en un ordenador convencional y externo del que la red termina siendo un **coprocesador** o un **periférico**. En el capítulo 11 hablaremos con detalle de estos temas.

Si recordamos ahora el resumen de las relaciones entre *DP* y *DO* que mostramos en la figura 2.7 y el ejemplo de la tarea genérica de clasificación de la figura 2.9, podemos razonar sobre las diferencias entre la *IA* simbólica y la *IA* conexionista (las redes neuronales).

Así, en la figura 2.7, vemos que una primera distinción está en el salto directo que las redes neuronales dan desde la formulación del problema a nivel de conocimiento hasta el nivel físico, (red de procesadores analógicos no lineales seguidos de una función no lineal tipo sigmoide para limitar las salidas). Esto supone un análisis más pormenorizado de la aplicación y la limitación a determinados tipos de problemas que permiten este paso. Hemos hablado en varias ocasiones al referirnos a las redes neuronales como computación distribuida de "grano pequeño", queriendo decir que la función local no puede ser excesivamente compleja. De otra forma sería muy difícil de autoprogramar y, además, estaríamos hablando de una red de computadores y no de una red de neuronas artificiales. Sin embargo, esta baja capacidad computacional de cada procesador hace que la reducción del nivel de conocimiento sea más complicada. No disponemos de nada equivalente a los niveles intermedios de representación e inferencia, ni el apoyo final de la capa de software (estructura de datos, algoritmos, ensambladores, etc.).

Como alternativa a este paso directo, la *IA* simbólica utiliza un conjunto de entornos y lenguajes de alto nivel que facilitan enormemente la reducción al nivel simbólico de un modelo de conocimiento. El problema suele estar en la fase de análisis, responsable de la completitud del modelo.

Otra distinción importante entre *IA* simbólica y conexionista es la riqueza de tareas genéricas y tipos de inferencia que ofrece la primera, frente a las limitaciones de una arquitectura por capas que siempre realiza una función de

clasificación (figura 2.9.c). Al pasar de la perspectiva simbólica a la conexionista el proceso de clasificación se segmenta en varias capas y se distribuye en paralelo sobre las neuronas de cada una de las capas de forma que parte de la computación queda impresa en la topología de la red y en las funciones locales y el resto se adquiere mediante un proceso de ajuste de parámetros.

En el capítulo 11 (figura 11.27) se muestra el paso del nivel de conocimiento a la red neuronal a partir de una estructura mínima de *TG*. Para que este paso sea posible el problema debe cumplir las exigencias usuales del paralelismo (ser segmentable y modular) y las propias del aprendizaje neuronal (medio cambiante y poco conocido y posibilidad de obtener un conjunto de entrenamiento). No todos los problemas tienen solución neuronal ni en todos los casos es la alternativa más conveniente. En los casos en que la solución neuronal es la aconsejable, se comienza definiendo la estructura de *TG* en el nivel de conocimiento, tal como hemos hecho para el caso de la tarea de clasificación.

Tenemos que reconocer, sin embargo, que no es tan fuerte la limitación de la función de clasificación como el modelo que habitualmente se usa para la computación local. Al no disponer de memoria en el sentido convencional del término, toda la función debe almacenarse en la estructura de la red, en el valor de las conexiones ("pesos") entre las distintas unidades.

Veremos en el capítulo final del libro que una forma de aproximar las redes neuronales a la computación simbólica es aumentar la capacidad de computación local, superando las limitaciones de un modelo analítico no lineal mediante el uso de un condicional o de un micro-marco con un campo de inferencia. En estas condiciones es posible hablar de sistemas expertos conexionistas y de redes neuronales basadas en conocimiento.

3 FUNDAMENTOS Y TÉCNICAS BÁSICAS DE BÚSQUEDA

J. G. Boticario

En los capítulos previos se ha presentado una visión de las características que definen la IA como ciencia y como tecnología. También se han presentado diferentes niveles de clasificación del tipo de problemas abordados. En un primer nivel se han distinguido problemas de percepción, razonamiento, aprendizaje, planificación y decisión. Tareas como clasificación, representación y búsqueda, pertenecen a un segundo nivel, ya que cada una de estas pueden aparecer, como tareas concretas en la resolución de las del primer nivel. A partir de ahora —en éste y en los capítulos siguientes— nos vamos a centrar en describir en el contexto adecuado las principales técnicas desarrolladas en este campo. Debido al gran número de problemas que pueden resolverse como problemas de búsqueda, apoyándonos en la evolución histórica de las investigaciones realizadas en el campo de la IA —en el que precisamente las técnicas de búsqueda fueron las que primero merecieron un mayor interés y desarrollo— y teniendo en cuenta la concreción de dichas estrategias, vamos a presentar en este capítulo aquellas que realizan un recorrido exhaustivo del espacio de búsqueda, dejando para el capítulo próximo las que utilizan el conocimiento del dominio para realizar un tratamiento informado —basado en conocimiento heurístico—, que permite reducir en gran medida la complejidad del proceso.

3.1 PLANTEAMIENTO DEL PROBLEMA

En la formulación de cualquier problema, inicialmente se parte de la declaración en lenguaje natural del mismo; que responde a su concepción en el *nivel del conocimiento*, término acuñado por Newell [1981] para establecer un nivel abstracto de interpretación de los sistemas en IA, en el que esencialmente se afirma que el sistema actúa siguiendo el "principio de racionalidad" (capítulo 2), es decir, *si un sistema tiene el conocimiento de que una de sus acciones conduce a una de sus metas, entonces se realiza dicha acción.* A partir de dicha descripción "en abstracto" se establecen planteamientos alternativos del problema en un nivel más concreto en el que aparecen elementos directamente representables por entidades que el sistema puede manipular directamente. En este capítulo nos vamos a centrar en el estudio de una de las técnicas más generales del segundo nivel: las *tareas de búsqueda.*

Los problemas computables requieren una gran cantidad de conocimiento que permite, partiendo de la especificación de los datos de entrada, realizar un procedimiento que conduce a sus soluciones, a situaciones de error, o al agotamiento de los recursos de cálculo disponibles. La IA estudia especialmente métodos que permiten resolver problemas en los que no existe el conocimiento sistemático para plantear una solución analítica. Estos métodos, considerados *débiles* por realizar procedimientos de búsqueda *no informada* (no aprovechan la información relevante del dominio), suelen ser poco eficientes, pero tienen la ventaja de poderse aplicar en una gran diversidad de clases de problemas. Para llevarlos a cabo efectiva y eficientemente, sobre dichos métodos se utiliza el *conocimiento heurístico* disponible. Vamos a presentar en éste y en el próximo capítulo los diferentes métodos de búsqueda como una serie de mecanismos generales para la solución de problemas en IA.

Ante cualquier problema se presentan dos situaciones posibles. Tener ya el conocimiento sobre lo que hay que hacer o tener que indagar qué podría hacerse para resolverlo. Nos centraremos en este último caso. Evidentemente, es posible resolver cualquier problema bien conociéndolo todo o bien buscando hasta encontrar lo que se necesita saber. Los problemas complejos requieren una formulación combinada de ambos enfoques. Las técnicas de representación que permiten describir los aspectos conocidos del problema se denominan *declarativas*. Las que se centran en especificar el proceso que hay que realizar para encontrar las soluciones buscadas se denominan *procedimentales*.

La elección de un marco específico de representación para el dominio en el cual se plantea el problema, determina la posibilidad de representar el conocimiento relevante. Por ejemplo, un sistema que quisiera distinguir los "números primos" de los que no lo son, debería tener la capacidad de comprobar la divisibilidad de dos números cualesquiera. Para ello se podría definir un predicado (capítulo 5) de dos términos: *divisible*(x, y), que tuviera asociado un *procedimiento de cálculo* que devolviera un valor booleano cierto o falso en función de la divisibilidad de x por y. Dicho procedimiento reflejaría el algoritmo necesario para realizar tal comprobación. Este modelo de representación puede calificarse como *procedimental.* Alternativamente, se podría *declarar* en una regla (capítulo 6) el conocimiento implicado en dicha comprobación, omitiendo los pasos específicos que deberían realizarse en el proceso de cálculo asociado.

R1: Si x sólo es divisible por x y por 1 ENTONCES x es un número primo

El conocimiento declarado en la regla *R1* sólo es utilizable si existe un *intérprete* capaz de utilizar dicho conocimiento, es decir, un procedimiento efectivo capaz de recibir como dato dicha regla y de devolver el valor booleano correspondiente al número dado.

Ningún sistema es completamente declarativo o exclusivamente procedimental. No tiene sentido práctico un conocimiento declarativo sin procedimientos que lo manipulen. Las técnicas de representación declarativas permiten especificar la información sin especificar las formas de uso posibles. Se declara "el qué". Cada hecho se almacena una sola vez. La gran ventaja de estas técnicas es su claridad y su predisposición a un planteamiento modular. Se pueden añadir fácilmente nuevos hechos a los ya existentes. Siempre puede introducirse una nueva regla en un conjunto de reglas que describan el conocimiento disponible.

Las técnicas procedimentales, en cambio, se centran en "el cómo". Permiten especificar más fácilmente el conocimiento del dominio que no se ajusta a esquemas declarativos. Dicho conocimiento sirve para orientar el proceso de búsqueda hacia los aspectos más relevantes. Se utiliza para guiar las líneas de razonamiento. La ventaja de estas técnicas reside en su eficiencia, ya que son más fáciles de mantener, debido a que se puede acceder a los caminos seguidos en el algoritmo aplicado. Un programa concreto escrito en un lenguaje simbólico como el LISP refleja una representación procedimental del problema.

En principio las técnicas declarativas y procedimentales son intercambiables (siempre y cuando exista el procedimiento de interpretación adecuado para las primeras). En general se puede afirmar que el carácter procedimental conlleva un tratamiento algorítmico y el declarativo uno heurístico. Las técnicas de búsqueda, analizadas como mecanismos genéricos de solución de problemas en IA, tienen un carácter fundamentalmente algorítmico. El conocimiento heurístico aplicado en aquellas se refleja en la utilización de las funciones de evaluación heurística para dirigir de una forma informada —dependiente del dominio de aplicación— el proceso de búsqueda (capítulo 4).

El planteamiento general de un problema, tal y como nosotros lo vamos a considerar, consiste en encontrar o construir una (puede haber varias) solución de dicho problema.

En este contexto un problema requiere:

1. Un "agente" con una serie de objetivos que se desean alcanzar. El "agente" es el sistema (programa) concreto que se está ejecutando.

2. Un conjunto de acciones que permiten obtener los *objetivos* o *metas*.

3. Un procedimiento de elección entre diferentes formas de llegar a ellas. Cada una de estas soluciones constituye una *secuencia de acciones* determinada. Suele denominarse *solucionador* al módulo encargado de construir dicha solución.

Un marco general adecuado para resolver dicho planteamiento es utilizar el esquema de los problemas de búsqueda (ver Tabla 3.1).

Tabla 3.1. Búsqueda como tarea genérica en IA

Dados:

- conjunto de estados
  todas las configuraciones posibles de las situaciones que puedan plantarse en el dominio
- uno o más estados desde los cuales el *solucionador* comienza a actuar
  (llamados *estados iniciales*)
- uno o más estados que serán aceptados como soluciones del problema
  (llamados *estados finales*)
- un conjunto de operadores o reglas que describen las acciones
  (operadores o transformaciones) posibles

Encontrar:
- una secuencia de operadores que permitan pasar del estado inicial al estado final o a uno de éstos, en el caso de que haya varios

Este esquema es ciertamente un marco general bajo el cual se puede especificar el problema. Quedan bastantes cuestiones por definir. Por ejemplo: ¿cuáles son las suposiciones que están implícitas en la descripción inicial del problema?, ¿cuán generales tienen que ser las reglas?, ¿qué parte del trabajo realizado en la búsqueda puede ser calculado y expresado de antemano en las reglas?

La gran mayoría de los problemas planteados en el campo de la IA pueden formularse especificando los elementos de la tabla 3.1. Por ejemplo, en el campo de la robótica un problema puede ser el siguiente: dada la descripción de una situación que refleja un caso concreto del modelo del mundo en el que se desenvuelve el robot, encontrar la secuencia de acciones que debería realizar dicho robot para llegar a su objetivo (generalmente descrito por una condición que debe satisfacerse). Cada vez que al robot se le plantea el problema de encontrar una determinada meta se realiza un proceso de búsqueda.

El marco general de búsqueda permite representar esquemas de solución de problemas que van más allá de los problemas de planificación (p.ej., de las acciones de un robot [Fikes & Nilsson, 1972]). Los problemas de diagnóstico, que tienen un carácter abductivo[1] (este tipo de razonamiento se tratará en el apartado 10.3.2), se pueden resolver aplicando un método de clasificación. En este caso hay que determinar a qué clase pertenecen los síntomas apreciados. El problema de encontrar la clase adecuada a las "entradas" también puede resolverse con el esquema de búsqueda propuesto.

La mejor forma de resolver el esquema planteado en la tabla 3.1. es utilizar —como veremos en el siguiente apartado— un grafo para representar el proceso de búsqueda realizado. Para clarificar el tratamiento requerido en este tipo de planteamientos vamos a estudiar un caso concreto; un problema de integración simbólica resuelto mediante diferentes técnicas de búsqueda. Para ello tendremos que introducir los elementos implicados en la representación del problema.

[1] Se entiende por abductivo aquel proceso de razonamiento que busca determinar la causa a partir de los efectos observados.

3.2 ESPACIOS DE REPRESENTACIÓN

Antes de abordar la representación de los problemas de búsqueda conviene repasar un conjunto de nociones sobre la herramienta de representación más conveniente utilizada en los problemas de búsqueda: *los grafos.*

3.2.1 Nociones Básicas sobre Grafos

En esta sección vamos a definir brevemente y con poco rigor matemático algunos conceptos que nos ayudarán posteriormente a describir tanto los algoritmos de búsqueda utilizados en este tema, como los distintos modelos de redes (capítulo 7).

Un *grafo* viene dado por un conjunto de nodos y un conjunto de arcos entre los nodos. Cada arco se define formalmente como un par de nodos. En este libro vamos a estudiar solamente grafos en que cada enlace une dos nodos diferentes. (Obsérvese el contraste con los grafos que se utilizan en la teoría de autómatas, en los cuales puede existir un arco que enlace un nodo consigo mismo.)

En el caso de pares ordenados, es decir, cuando (A, B) se considera diferente de (B, A), tenemos un grafo *dirigido*; en caso contrario, el grafo es *no dirigido*. Un arco o enlace de un grafo no dirigido se representa gráficamente trazando una línea entre los dos nodos que forman el par; un arco dirigido se representa mediante una flecha desde primero hasta el segundo de sus nodos.

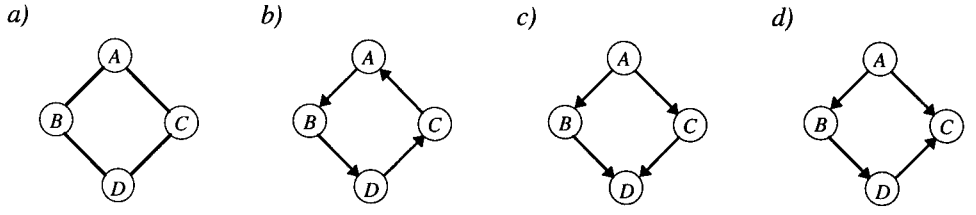


Fig. 3.1. Grafos: a) no dirigido; b), c) y d), dirigidos.

Dado un arco dirigido entre dos nodos, se dice que el primero de ellos es *predecesor* del segundo y que el segundo es *sucesor* del primero. Así, en el grafo *c)* de la gráfica anterior (Fig. 3.1), B y C son sucesores de A y predecesores de D. Un nodo X es *antepasado* de Y si X es predecesor de Y o si existe un nodo Z

que sea sucesor de *X* y antepasado de *Y* (observar que se trata de una definición recursiva). La definición de *descendiente* es la recíproca de la anterior. En el ejemplo citado, *A* es antepasado de *D* y *D* es descendiente de *A*.

Se define un *camino* como una sucesión de nodos tales que entre dos de ellos consecutivos existe un arco. Nosotros supondremos que se trata de caminos simples, es decir, de caminos que no pasan dos veces por el mismo nodo. Si el último nodo del camino coincide con el primero, diremos que se trata de un *camino cerrado*; si no, se denomina un *camino abierto*. En cada uno de los grafos de la Fig. 3.1, hay un camino cerrado {*A, B, D, C, A*}, y además, entre dos nodos cualesquiera existen dos caminos (abiertos). Conviene recalcar que en la definición de camino no se tiene en cuenta la dirección de los arcos.

Se dice que un grafo es *conexo* cuando entre dos nodos cualesquiera existe al menos un camino. Cuando entre dos nodos cualesquiera existe un único camino, se dice que el grafo es *simplemente conexo*; si para algún par de nodos existen dos o más caminos entre ellos, es decir, si hay algún camino cerrado, el grafo es *múltiplemente conexo*. Los cuatro ejemplos de la Fig. 3.1 son conexos; precisando más, múltiplemente conexos. En este libro vamos a trabajar siempre con grafos conexos.

En un grafo no dirigido, cualquier camino cerrado recibe el nombre de *ciclo*; por ejemplo, el grafo *a)* está formado por un ciclo de longitud 4. Si no contiene ciclos se denomina *árbol [no dirigido]*.

Supongamos que en un grafo dirigido tenemos un camino cerrado; si podemos recorrer el camino siguiendo la dirección de los arcos y volvemos al punto de partida, tenemos un *ciclo* (por ejemplo, en el grafo *b*); si no, tenemos un *bucle* (grafos *c* y *d*)[2]. Los *grafos dirigidos acíclicos* (GDA) son aquéllos que no contienen ciclos. En este contexto, *padre* es sinónimo de predecesor e *hijo* lo es de sucesor. A los nodos que no tienen descendientes se les llama *nodos terminales*, *hojas* o *extremos*.

Una de las propiedades más importantes de los GDA es que cada uno de ellos define una relación de orden entre los nodos, generalmente de orden parcial; y recíprocamente, una relación de orden puede representarse mediante

[2] Insistimos de nuevo en que en la definición de camino no se tiene en cuenta la dirección de los arcos, mientras que en la definición de ciclo y bucle sí es necesario considerarla. La distinción entre ciclos y bucles se introduce con el fin de caracterizar los grafos dirigidos acíclicos, cuya importancia irá quedando clara a lo largo de este libro.

un GDA. Un grafo con ciclos no podría representar una relación de orden por incumplir la propiedad transitiva.

Los GDA conexos que no contienen bucles se denominan *poliárboles* (ver la Fig. 3.2). Un *árbol [dirigido]* es un GDA conexo en que cada nodo tiene como máximo un padre; de esta definición se deducen cuatro propiedades:

(1) un árbol no puede contener ciclos ni bucles y es, por tanto, un caso particular de poliárbol;
(2) en todo árbol existe un único nodo, denominado *raíz*, que no tiene antepasados y que es antepasado de todos los demás;
(3) cada nodo, excepto el nodo raíz, tiene exactamente un padre;
(4) como consecuencia de la primera propiedad, para cada nodo existe un camino y sólo uno que lo conecta con el nodo raíz (excepto para el nodo raíz, naturalmente); este camino se conoce como *rama* y cumple la propiedad de que se puede recorrer desde la raíz hasta el nodo buscado siguiendo siempre el sentido de los arcos. Por ejemplo, para el nodo *D* de la gráfica Fig. 3.2, la rama es {*A, C, D*}.

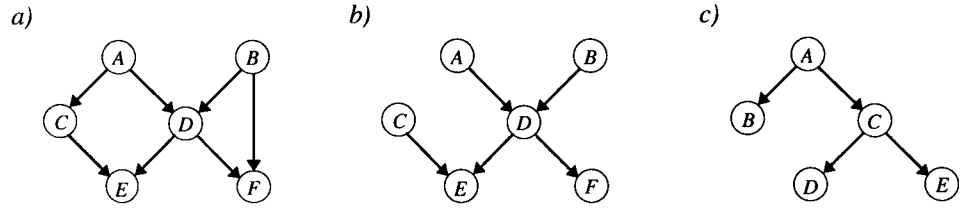


Fig. 3.2. *a)* GDA múltiplemente conexo. *b)* Poliárbol. *c)* Árbol.

En el caso concreto de los **problemas de búsqueda** deberemos tener en cuenta además las siguientes nociones:

- *Nodos:* elementos en el *espacio de estados* (ver 3.2.2) que representan situaciones válidas en el dominio (*nodo* y *estado* son sinónimos en este contexto). Un nodo terminal que satisfaga las condiciones del objetivo recibe el nombre de *nodo meta* (o simplemente *meta*).

- *Expansión de un nodo:* obtener todos sus posibles sucesores en el grafo de búsqueda a través de la aplicación de todos los operadores disponibles relacionados.

- *Nodo o estado "cerrado":* aquél que ya ha sido expandido.

- *Nodo o estado "abierto":* aquél en el que todavía queda por aplicar algún operador.

- *Coste de un arco:* es un valor numérico que refleja el tiempo requerido para aplicar un operador a un estado en el proceso de búsqueda. Aunque no se indique explícitamente siempre debe considerarse un coste implícito, que por defecto puede tomar el valor 1.

- *Coste de un nodo:* es una medida del tiempo consumido en alcanzar el nodo en cuestión desde la raíz a lo largo del mejor camino encontrado hasta un momento dado.

- *Factor de ramificación:* número medio de descendientes de un nodo, o lo que es lo mismo, el número medio de operadores que pueden aplicarse a un estado.

- *Longitud de una trayectoria:* número de nodos generados en un camino, que es igual al número de operadores aplicados en dicho camino.

- *Profundidad:* longitud del camino más corto desde el estado inicial a una *meta*. Dicho de otra forma, la longitud de secuencia más corta de operadores que resuelven el problema. La profundidad del nodo "raíz" es 0 y la de cualquier otro nodo es la de su antecesor inmediato + 1. Estrictamente hablando habría que distinguir la búsqueda en aquellos grafos donde puede haber más de un antecesor inmediato; en estos casos el concepto de profundidad se define recursivamente como "1 + la profundidad de su antecesor menos profundo".

3.2.2 Representación de los Problemas de Búsqueda

El planteamiento general del problema de búsqueda se ha realizado siguiendo dos esquemas generales de funcionamiento. El *esquema de producción* (también conocido como método de búsqueda en el *espacio de estados*) y el *esquema de reducción* (que responde al enfoque basado en la *descomposición del problema*, y se estudiará en la sección 3.5). Cada uno de estos planteamientos conlleva una determinada forma de realizar la construcción de la solución y un control específico sobre aquella. Para poder entender dicho proceso vamos a centrarnos en este apartado en distinguir y describir cuáles son los elementos implicados en cada uno de los dos enfoques.

El entorno en el cual se desarrolla el proceso de búsqueda es lo que se conoce como **espacio del problema** (o *espacio de representación*). Está formado por el conjunto de estados y el conjunto de operadores que permiten avanzar en el proceso de obtención de la solución. Una *instancia del problema* es un espacio del problema donde se señala cuál es el estado inicial y cuál el final. Este último, como ya hemos señalado anteriormente, suele venir especificado por las condiciones que debe cumplir un estado para ser considerado como meta. Cada uno de los esquemas de resolución del problema señalados anteriormente, producción y reducción, realizan el proceso de búsqueda en espacios del problema diferentes, que se denominan respectivamente *espacio de estados* y *espacio de reducción*.

En el **espacio de estados**, durante el proceso de búsqueda, cada estado representa una situación que se debe considerar como candidata a pertenecer a la solución del problema. Es decir; partiendo del estado llamado *estado inicial* (descripción del problema) se plantea la aplicación de alguno de los operadores disponibles. Por supuesto, no todos los operadores disponibles tienen que ser aplicables a un determinado estado (cada uno describirá sólo un subconjunto de situaciones posibles, a las que se le puede aplicar la transformación indicada en su definición). El proceso de comprobación de la aplicabilidad de los operadores —en este contexto— es lo que también se conoce con el nombre de *equiparación*. Una vez aplicado el primer operador se obtiene una nueva descripción *derivada* de la situación inicial a la que también se le considera *estado*. En este momento; si todavía no se ha obtenido la solución ni se han consumido los recursos disponibles de cálculo, caben dos opciones. Seguir intentando aplicar otros operadores al estado inicial o bien centrarse en el nuevo estado y aplicar alguno de los operadores disponibles sobre aquél. De esta forma se van generando nuevos estados que pueden ser seleccionados para intentar aplicarles operadores de tal forma que los estados generados sean cada vez más parecidos a la meta.

La **estrategia de control** (EC) es la encargada —utilizando información del dominio del problema cuando ésta sea conocida— de optimizar el proceso de búsqueda. Para ello, partiendo de las condiciones iniciales que definen el problema, se realiza un proceso continuo de selección de estados (señalados por la EC) y de generación de sucesores (aplicando los operadores elegidos por EC) hasta obtener un estado que cumpla las condiciones que definen el objetivo del problema. Bajo este esquema los estados también se denominan *pasos* en la búsqueda de la solución. La EC es el proceso encargado de seleccionar en cada momento el estado que será expandido y el operador que será aplicado, de

forma que la solución obtenida sea óptima teniendo en cuenta los recursos disponibles. *La búsqueda surge cuando hay más de un operador aplicable o existe más de una "instanciación" para un operador concreto* (diferentes formas de aplicarlo a un mismo estado); en este caso se hace necesario analizar las diversas alternativas existentes. La EC avanza hacia el objetivo considerado de una forma global. Una decisión local (aplicación de un operador concreto a un determinado estado) puede no ser acertada, pero el proceso en su conjunto debe ser un método sistemático (ajustado a un procedimiento) que explore las distintas alternativas que permitan acercarse a la meta.

Para ello debe:

(1) Seleccionar los nodos a lo largo del grafo de búsqueda generado.

(2) Determinar las reglas aplicables al nodo elegido en (1), considerando que pueden existir diferentes formas de aplicación de alguna de ellas.

(3) Seleccionar, dentro del conjunto obtenido en (2), una determinada regla y aplicarla.

(4) Comprobar que el estado generado en (3) cumple las condiciones asociadas al *estado meta*.

Una particularidad que conviene resaltar es que, en la representación del *espacio de estados*, los operadores, al ser aplicados a un determinado estado, producen un único nuevo estado (en el *esquema de reducción* esto ya no se cumple; ver el apartado 3.5).

La siguiente gráfica (Fig. 3.3) muestra la estructura genérica que tendría un árbol de búsqueda como herramienta para solucionar problemas según el esquema del espacio de estados.

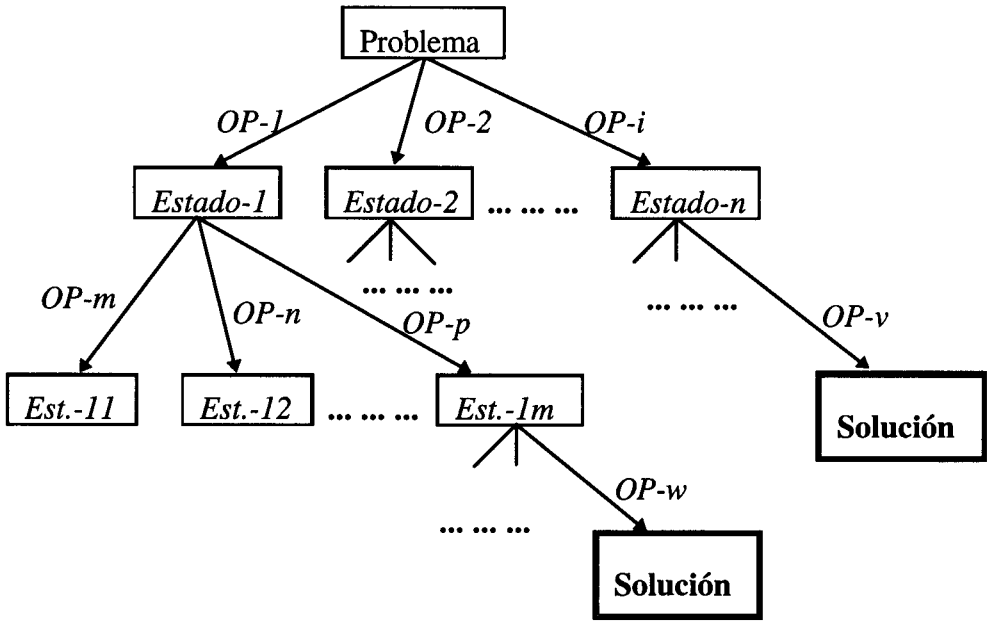


Fig. 3.3. Árbol genérico de búsqueda

El conjunto de estados obtenidos en el proceso de búsqueda descrito se denomina *grafo de estados* (en el caso más sencillo es un árbol; ver apartado 3.2.1). El *espacio de estados* está formado por todos aquellos estados que podrían obtenerse si se aplicaran todos los operadores posibles a todos los estados que se fueran generando (puede, por lo tanto, ser infinito o "intratable"; un ejemplo clásico es el del ajedrez con 10^{120} diferentes configuraciones posibles [Shannon 1950, 1956]). Por lo tanto, el *grafo* de estados es una "medida explícita" (refleja los caminos explorados) del proceso realizado y el *espacio de estados* es una "medida implícita" (el número de estados generados en dicho *espacio* se utiliza como referencia) que da una idea de la complejidad del problema. *El mejor método de búsqueda será aquél que necesite hacer explícita la menor parte del grafo implícito (entendiendo por tal la menos costosa de obtener) para obtener una solución del problema.*

La *complejidad de un cálculo* es la cantidad de recursos necesarios para efectuarlo. Hablaremos de complejidad espacial y temporal. La complejidad *temporal* se refiere al tiempo necesario para realizar un cálculo y refleja el almacenamiento requerido. Ambos parámetros de medida dependen de varios factores; por ejemplo, el tipo de máquina y el sistema de codificación empleados. Para evitar ambigüedades, en este contexto consideraremos la complejidad

temporal como una medida del número de operaciones realizadas y la complejidad espacial como el conjunto de datos que es necesario recordar en un momento dado.

El razonamiento antes descrito también se conoce como *guiado por los datos* o *encadenamiento hacia adelante*, ya que la dirección del proceso de búsqueda parte de los datos suministrados (datos iniciales del problema) y avanza en el camino de búsqueda hacia un estado *meta* (aquél que coincida con la meta declarada explícitamente o aquél que cumpla las condiciones exigidas para ser considerado meta). El estado inicial y todos los que vayan obteniéndose sucesivamente a partir de aquél formarán parte de la *base de datos de trabajo* (BD) (este es un término muy utilizado en el contexto de los *sistemas de producción*; capítulo 6).

En todos los planteamientos descritos presuponemos que en cada paso del proceso de búsqueda se considera la aplicación de un único operador (acción, movimiento, regla ...) a un único estado (paso, nodo, situación ...) y no la de varios operadores a distintas situaciones.

En la siguiente sección utilizaremos un árbol de búsqueda como el mostrado en la Fig. 3.3 para plantear la solución de problemas de integración simbólica. El problema que hay que resolver es el siguiente: dada una cierta integral indefinida expresada en un formalismo simbólico concreto (p.ej., "$\int x \sin(x)\, dx$"), encontrar la secuencia de operadores "apropiada" que permite obtener la solución (en este caso una expresión sin el símbolo de integración "$\int$").

3.2.3 Cómo Limitar el Espacio de Búsqueda

El problema de hallar un estado que satisfaga una condición puede formularse como la búsqueda de un grafo que posea un nodo cuya descripción asociada coincida con el estado objetivo. La búsqueda consiste en obtener un grafo explícito suficientemente grande para que contenga la solución del problema inicial.

El problema crítico son las cantidades de tiempo y espacio necesarias para encontrar la solución. Por ejemplo, volviendo al problema del ajedrez; para examinar todas las secuencias de n movimientos se debe trabajar con un espacio de búsqueda en el que el número de nodos crece exponencialmente con n, convirtiéndose en un problema intratable mediante técnicas de búsqueda

exhaustivas. Este fenómeno se conoce con el nombre de *explosión combinatoria.*

Para realizar una búsqueda eficiente en grafos, un método muy interesante y ampliamente extendido es el uso de *conocimientos heurísticos* (reflejan información dependiente del dominio) que ayudan a dirigir la búsqueda. En algunos tipos de problemas, el empleo de heurísticas evita que se produzca la explosión combinatoria (p.ej., en el problema del ajedrez analizando sólo las jugadas más prometedoras; es decir, ganan más piezas, amenazan más piezas, etc.). Este método es una de las contribuciones clave de la IA a la solución eficiente de problemas.

Vamos a utilizar el término *búsqueda heurística* según la acepción más ampliamente extendida del mismo, basada en [Nilsson, 1971]. Para Nilsson, a diferencia de una *búsqueda ciega* en el espacio de estados, basada en generar estados para luego comprobar si satisfacen o no las condiciones del objetivo, la búsqueda heurística no consiste en excluir parte de ese espacio "por definición", sino en añadir información, basándose en el espacio estudiado hasta ese momento, de forma que se restringe drásticamente dicha búsqueda. Dado que este tipo de procesos serán ampliamente estudiados en el próximo capítulo, vamos a dedicarnos en este tema a analizar los procesos de búsqueda sin considerar posibles heurísticas aplicables.

En el próximo apartado introducimos la descripción de un sistema real con un doble propósito: facilitar la comprensión de los ejemplos que se presentarán en posteriores secciones dedicadas a ilustrar estrategias de búsqueda y resaltar algunos aspectos relativos a la realización de un sistema que realiza procesos de búsqueda (por consiguiente se ajusta al esquema mostrado anteriormente en la tabla 3.1).

3.3 BÚSQUEDA EN INTEGRACIÓN SIMBÓLICA

Los problemas de integración simbólica que vamos a utilizar para ilustrar las diferentes estrategias de búsqueda se han generado a partir del trabajo [Boticario, 1988] (un sistema real basado en una serie de trabajos relativos al sistema LEX [Mitchell *et al.*, 1983; Utgoff, 1986]). En dicho programa se parte de integrales indefinidas especificadas mediante un lenguaje concreto de expresiones matemáticas que se comentará más adelante. El proceso de búsqueda podría definirse completamente especificando los elementos incluidos en la tabla 3.2.

Tabla 3.2. Búsqueda como tarea en LEX

Dados:

Estado inicial: p.ej. $\int x \sin(x)\, dx$

Estado final o meta: Un estado que no contenga el símbolo de integración, "$\int$ "

Espacio de estados: todas las expresiones que resulten de aplicar distintas secuencias de operaciones junto con todos los operadores disponibles

Operadores (se muestra un ejemplo de cada uno de los tipos existentes):

(donde: "fr" indica "función de real"; *u*, *v* y *x* son variables; *sin* y *cos* responden a la notación interna del programa para las funciones trigonométricas de "seno" y "coseno")

- Métodos generales de integración (p.ej., Integración por partes, operador OP12)

  OP12: "$\int udv \Rightarrow uv - \int vdu$, u=fr_1, dv=$fr_2 dx$"
- Integrales inmediatas (p.ej., "$\int cos(x)dx \Rightarrow sin(x)$ ")
- Relaciones trigonométricas

  p.ej. "$tan(x) \Rightarrow sin(x)/cos(x)$ "
- Transformaciones entre expresiones algebraicas equivalentes

  p.ej. "$fr_1 * (fr_2 +- fr_3) \Rightarrow (fr_1 * fr_2) +- (fr_1 * fr_3)$ "

Encontrar:

- una secuencia de transformaciones (aplicación de operadores) que permita resolver la integral planteada

Como puede apreciarse en la tabla precedente, el estado meta es aquél que se quiere alcanzar, sin considerar "cómo alcanzarlo". El planteamiento mostrado sirve como paradigma genérico para describir los elementos implicados en el proceso —en este caso de búsqueda— utilizado para resolver las integrales propuestas. Sin embargo, quedan muchas cuestiones por especificar. Analizaremos algunas de ellas en los siguientes apartados.

3.3.1 Descripción General del Sistema

La tarea consiste en, a partir de ejemplos de integración propuestos, aprender y refinar las heurísticas que indican cuándo un operador debe aplicarse a un estado de un problema en la búsqueda de la solución. Se parte de una serie de operadores, cada uno de los cuales posee un conjunto de condiciones previas que caracterizan el tipo de estados a los cuales puede aplicarse dicho operador. Por ahora nos limitaremos a describir aquellos aspectos que permiten resolver las integrales, dejando para más adelante aquéllos relativos al aprendizaje de las heurísticas que ayudan a optimizar el control de la búsqueda realizada. Nos

circunscribiremos por lo tanto al módulo del sistema que recibe el nombre de *solucionador*.

Para resolver el problema de búsqueda tal y como se ha planteado en la tabla 3.2. es necesario tener en cuenta los siguientes elementos.

- Representación adecuada de los elementos que participan en el proceso de búsqueda.
- Estrategia de búsqueda más eficiente (utiliza heurísticas).
- Restricciones en los recursos (tiempo y espacio) asignados a la solución.
- Eliminación de:
  - nodos ya examinados.
  - nodos que no pueden aportar nueva información.
  - subárboles que no puedan contener la solución.

Los ejemplos resueltos son integrales indefinidas descritas en un lenguaje concreto de expresiones matemáticas. Los operadores utilizados (unos 50 en [Boticario, 1988]), se corresponden con reglas generales de integración (como puede ser la integración por partes), integrales inmediatas, relaciones trigonométricas y transformaciones entre expresiones algebraicas equivalentes (como las leyes asociativa y distributiva).

Cada operador consta de un *dominio de estados* a los cuales puede aplicárseles dicho operador, una "regla de reescritura" y un rango de estados que pueden producirse por aplicación del operador. Por ejemplo, la regla general de integración por partes se corresponde con el operador OP12 mostrado en la tabla 3.2. Un estado en el que pueda utilizarse este operador debería tener la forma: "$\int f_1(x)\, f_2(x)\, dx$". Cualquier operador indica que si un estado o una parte de un estado posee una expresión equiparable a la condición de aplicabilidad del operador (p.ej., "$\int f_1(x)\, f_2(x)\, dx$" en el operador OP12), entonces esta integral puede reescribirse según la expresión señalada en dicho operador (de nuevo en OP12 sería; "$uv - \int v du,\ u = f_1(x), dv = f_2(x)$"). Por consiguiente, los operadores describen "subdominios" conocidos del problema a los que se les puede aplicar una transformación que permite avanzar en el proceso de búsqueda de la solución.

3.3.2 Lenguaje de Descripciones

Cada uno de los operadores define un espacio de aplicabilidad que, como veremos en el capítulo 10, puede llegar a ser aprendido por el propio sistema.

Dicho espacio debe definirse para que pueda ser recorrido de una forma eficiente. La definición *en extenso* (ver apartado 7.4.1) del espacio la constituyen todos los estados posibles a los que se les pueda aplicar el operador (los que sean "equiparables" con éste). La comprobación eficiente de si un determinado estado en el proceso de búsqueda puede ser objeto de la transformación indicada por un determinado operador debe evitar el tratamiento *en extenso* del conjunto de descripciones que abarca dicho operador. Dicho de otra forma, no se trata de comprobar si un estado pertenece al conjunto de estados a los que se puede aplicar un operador, sino de verificar si la expresión de dicho estado tiene una forma semejante a la del operador. Para ello son necesarios los siguientes elementos:

1. Un lenguaje para describir los estados.
2. Un lenguaje para describir los operadores.
3. Un proceso de equiparación que sirva para comprobar si la descripción asociada a un operador cubre toda o parte de la descripción asociada a un estado.

En el sistema LEX se utiliza un lenguaje (basado en las clases estándares de funciones presentadas en el libro de [Swokowski, 1975] dedicado al cálculo de Freshman) generado mediante una gramática que sirve para describir los estados, los operadores y las reglas utilizadas por la estrategia de control para guiar la búsqueda. En dicho lenguaje todas las formas de *sentencias terminales* (son las expresiones válidas en el lenguaje; p.ej., "$\int 3\cos(x)dx$") y no terminales (cada regla de la gramática; p.ej., TRIG ← TAN COS SIN COT SEC, refleja los símbolos que pueden sustituir al símbolo TRIG en una expresión) sirven como descripciones. Así por ejemplo, el símbolo de la gramática que representa las funciones trigonométricas *trig*; describe el conjunto de símbolos terminales (*tan*, *cos*, *sin*, *cot*, *sec*). Recorriendo las distintas reglas que forman el árbol que representa la estructura de la gramática se obtienen todas las sentencias válidas en el lenguaje.

Sólo como curiosidad para aquellos conocedores del lenguaje LISP, en el que se ha implementado el sistema, mostramos a continuación algunos ejemplos de integrales expresadas en el lenguaje de descripciones utilizado (sujeto a una notación funcional, primero aparece la función y luego sus argumentos).

Expresiones:	*Representación interna*:
$\int x\ cos(x)\ dx$	(INT ((* id cos) x))
$\int x^2/2\ cos(x)\ dx$	(INT ((/ (^ id 2) cos) x))

$\int f(x)\, Ln(g(x))\, dx$	(INT ((* fr_1 (~ log fr_2)) x))
$\int 1/tan(x)\, dx$	(INT ((^ tan -1) x))

3.3.3 Lenguaje de Operadores

Los diferentes operadores disponibles son los que permiten ir modificando la situación inicial del proceso (problema planteado) hasta alcanzar el objetivo (estado en el que no aparece el signo de integración).

El espacio de estados representaría todas las transformaciones posibles implícitas en la descripción de los operadores del sistema. Cada uno de éstos puede considerarse como una regla de la forma "*C* —> *A*" (que nos recuerda al esquema más genérico de los sistemas basados en reglas (capítulo 6): "SI *Condición* ENTONCES *Acción*"), lo cual quiere decir que si un estado del problema contiene una descripción equiparable (por equiparable debemos entender que todos los términos son iguales o pertenecen a una misma categoría, p.ej., *sin* y *cos* están dentro de la categoría *trig*) con *C*, entonces se aplica la regla asociada al operador, "reescribiendo" en dicho estado la descripción de *C* por *A*.

Los operadores poseen una estructura que los caracteriza.

1. Parte Izquierda (PI o LHS "Left Hand Side" en terminología sajona).
Descripción del *dominio* del operador. Es una expresión, lo más general posible, según las descripciones del árbol de la gramática, de aquellos estados a los que podría aplicárseles dicho operador.

2. Parte Derecha (PD o RHS "Right Hand Side").
Representa el *rango* del operador. Es la expresión que va quedar en aquella porción del estado sobre la que se ha podido aplicar el operador.

3. Parte Avanzar (PA).
Para poder sustituir PI por PD es necesario asignar valores a aquellos símbolos de PI que no estén en PD. Por ejemplo, uno de los operadores más sencillos del sistema es el operador OP36, que tiene la estructura:
PI ← (+ C_1 C_2)
PD ← C_3
PA ← (C_3 . [+ C_1 C_2])
(La notación utilizada en PA expresa que C_3 se reescribe por el resultado de la suma de C_1 y C_2, cuando esta pueda calcularse; por ejemplo, [+ 2 *cos*(*x*)] no daría lugar a ningún cambio, [+ 3 5] haría que C_3 tuviera el valor 8.)

4. Parte Comentario (PC).
Describe en lenguaje matemático la acción asociada al operador
Para el OP36 sería:
PC ← "(+ C_1 C_2) => calcular (C_1 + C_2) "

Este operador indica que, si la suma puede realizarse, en lugar de la expresión con el símbolo "+", se pondría el valor resultante de intentar evaluar dicha suma.

La forma en que un operador se aplica a un estado consiste básicamente en; extraer la descripción del dominio del operador (PI); comprobar la semejanza (equiparar) de éste con alguna porción del estado y realizar la transformación asociada al operador (indicada por PD y apoyada por PA), reescribiendo la parte del estado afectada (el resto de la descripción del estado permanecerá invariable). Por ejemplo, la expresión (+ 3 5) está dentro del dominio de aplicabilidad del operador OP13: (+ C_1 C_2), ya que C_1 se equipara con 3 y C_2 se equipara con 5. Por lo tanto, teniendo en cuenta la acción del operador reflejada en PA y PD se sustituye la expresión en la que aparece el operador suma por su valor, en este caso 8.

3.3.4 Equiparación de Descripciones

El lenguaje generado permite definir una relación de orden parcial entre las expresiones válidas en el lenguaje. En la siguiente gráfica (Fig. 3.4) se ilustra esta circunstancia. En ella se muestran diferentes descripciones a las que se les podría aplicar el operador OP3, encargado de "sacar" las constantes situadas bajo el signo de integración ("$\int$ c f(x) dx => c $\int$ f(x) dx"). La forma más genérica que puede adoptar una integral de este tipo sería la que aparece en el nodo raíz del grafo. En el extremo inferior aparece la descripción asociada a un estado en el proceso de búsqueda de la solución. Siguiendo el grafo en orden ascendente vamos encontrando diferentes expresiones que muestran descripciones cada vez más generales. Cada una de las ramas en la Fig. 3.4 representa una de las posibles "derivaciones" que permiten comprobar si la descripción del operador OP3 se equipara con un estado dado: "$\int -1 \cos(x)dx$" (en su sentido gramatical *derivar* significa obtener los símbolos alcanzables siguiendo las reglas de la gramática; p.ej., si unimos la regla TRANS ← TRIG con la anteriormente mostrada sobre TRIG, podremos decir que COS se deriva de TRANS —siendo este último el símbolo que representa las funciones transcendentes—).

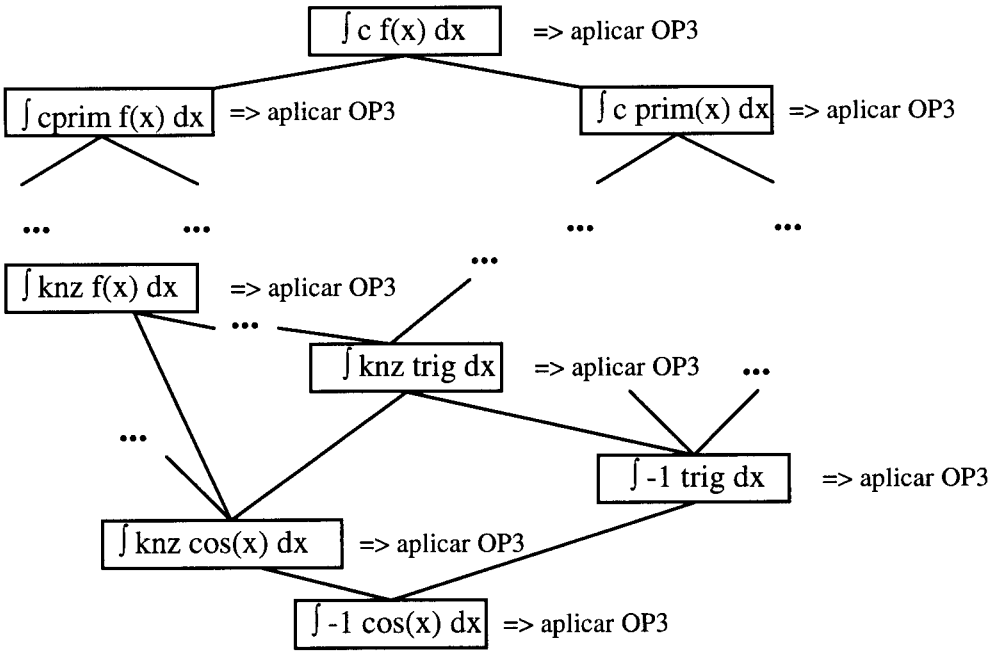


Fig. 3.4. Ordenamiento parcial según el lenguaje de descripciones

Cuando la precondición (ver el siguiente apartado) asociada a un operador es *más general* que la descripción de un estado, entonces se le puede aplicar el operador al estado. (La relación *más general* establece un ordenamiento parcial en el espacio de descripciones de aplicabilidad de cada operador.)

3.3.5 Solucionador

Es la parte del sistema que realiza la búsqueda de la solución del problema planteado. La estrategia seguida para alcanzar el objetivo es el "encadenamiento hacia delante". El problema se representa mediante un "grafo de búsqueda", que en cada momento contiene el conjunto de todos los nodos que ya se han alcanzado partiendo del nodo raíz (el problema que se quiere solucionar) por sucesivas aplicaciones de operadores y heurísticas disponibles.

El procedimiento SOLUCIONADOR recibe como argumentos un problema expresado según el lenguaje mencionado previamente, junto con unos "recursos-asignados" para su solución. Estos recursos se refieren al tiempo y al espacio de almacenamiento que se le permite emplear al solucionador para resolver dicho problema. Es importante darse cuenta de que, para el sistema, un

problema sin solución es equivalente a un problema para el que no se ha encontrado una solución con los recursos inicialmente asignados para su cálculo. Se ha introducido esta limitación para evitar que se llegue a intentar establecer soluciones con caminos excesivamente largos, que suelen producirse por la escasez de conocimiento (heurísticas) que el sistema tiene acerca del problema planteado. En estos casos es mejor plantear un problema relacionado más sencillo, del cual se puedan obtener nuevas heurísticas (capítulo 4) que sean utilizadas para alcanzar una solución más directa al primer problema. (Las heurísticas encauzarían la búsqueda por los caminos "más prometedores".)

Acabamos de introducir un aspecto determinante en todo proceso de búsqueda. Los enlaces que constituyen los caminos que conducen a la solución del problema llevan asociados implícitamente un coste de cálculo. Se denomina *coste real* de un nodo a la suma de los tiempos de cálculo consumidos en cada paso a lo largo del camino que conduce de la raíz al nodo considerado. Por ahora supondremos que el coste de un nodo es igual a su coste real.

La estrategia de control (EC) debe tener como principal objetivo la eficiencia (además, por supuesto, de la eficacia[3]). El coste (normalmente es una medida de distancia o de cantidad de recursos computacionales) es un factor de medida necesario para dirigir adecuadamente el proceso de búsqueda. El coste depende del dominio del problema. Existen otros parámetros generales, independientes del dominio, que determinan la eficiencia deseada. Éstos son la *profundidad* y el *factor de ramificación* (ver 3.2.1).

Partiendo de la descripción vista hasta ahora del problema de integración simbólica, vamos a presentar diferentes estrategias de búsqueda alternativas. La siguiente gráfica muestra el esquema del grafo que definiría el espacio completo de búsqueda de un problema concreto de integración.

[3] La *eficacia* es la capacidad de realizar una tarea, mientras que la *eficiencia* consisten en realizar dicha tarea con el menor coste posible (en tiempo o espacio utilizados).

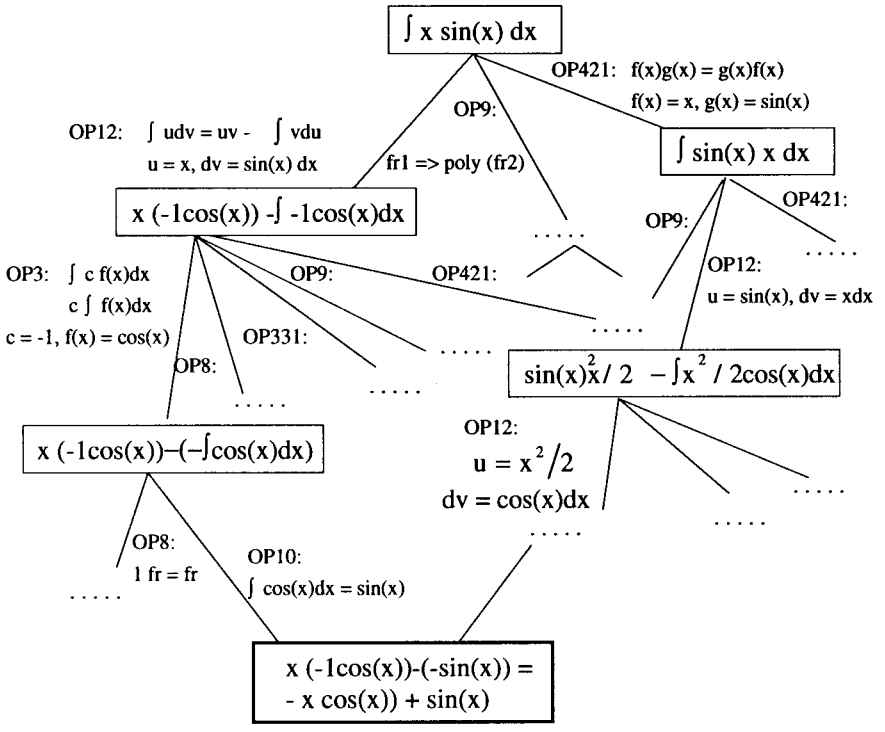


Fig. 3.5. Espacio de estados del problema ∫ x sin(x)dx

Como puede apreciarse en el grafo de la Fig. 3.5, completar el espacio de estados de un problema supone realizar muchas acciones "poco prometedoras". Así, por ejemplo, se observa cómo la aplicación del operador OP421 (que corresponde a la propiedad conmutativa con respecto al producto de funciones) seguida de la aplicación del operador OP12 (que realiza la integración por partes), en lugar de aproximar la solución complica la integral considerada. En este caso concreto podríamos aplicar sucesivamente el operador OP12 sin avanzar en modo alguno hacia la solución del problema. Por tanto, *el primer requerimiento que debe tener EC es que ayude a avanzar en la obtención de la meta.*

En el caso de que se eligieran —a lo largo del proceso de búsqueda— los estados arbitrariaremente y se aplicaran los operadores de forma gratuita, sería muy probable que se recorrieran caminos alternativos —muchas veces inútiles— para llegar a estados ya generados previamente en el proceso. Sin embargo, si fueran seleccionados de una forma ordenada (sistemática) estados y operadores, lograríamos avanzar más eficientemente en la solución del problema. Por

consiguiente, *el segundo requerimiento de las EC es que sigan un procedimiento organizado de recorrer el espacio de búsqueda.*

En el siguiente apartado vamos a ilustrar diferentes EC generales que no utilizan información del dominio. Todas ellas, precisamente por no utilizar ninguna información del dominio, realizan un recorrido poco eficiente del grafo de búsqueda. Una alternativa a la utilización exclusiva de la definición de los operadores consiste en aplicar un conjunto de reglas que indican cuándo un operador es útil en un determinado problema (el aprendizaje de reglas de control en los problemas de búsqueda se trata en la sección 10.3.3). Así, para el caso de la integración por partes, una de dichas reglas podría indicar que si el estado tiene una expresión "equiparable" con "$\int \text{poly}(x)\,\text{transc}(x)\,dx$", entonces debería aplicarse el operador OP12 (de integración por partes), haciendo que "$u = \text{poly}(x)$" y "$dv = \text{transc}(x)dx$" (siendo *poly* el indicativo de función polinómica y *transc* el de transcendente).

Conviene resaltar que si se conoce un método más directo de obtención de la solución del problema, no es conveniente aplicar alguna de las estrategias que estudiaremos seguidamente, debido a que son poco eficientes. Sin embargo, los métodos de búsqueda que estudiaremos aquí, y sobre todo los del capítulo 4, han inspirado la mayoría de las planteamientos finalmente aplicados en sistemas reales.

3.4 BÚSQUEDA SIN INFORMACIÓN DEL DOMINIO

Estudiaremos en primer lugar las formas básicas de control que permiten obtener, a través de un proceso exhaustivo, un estado meta. Consideramos aquéllas que realizan una búsqueda sistemática y *objetiva.* Se consideran objetivas por que el control del proceso de búsqueda no depende del problema concreto que se está resolviendo. (También se denominan técnicas de búsqueda *ciega.*)

En todos los métodos que presentaremos suponemos que partimos de la descripción del problema planteado. Damos por supuesto que dicha descripción no cumple la condición de ser *meta.* El problema que se está intentando resolver es encontrar el camino óptimo entre la descripción del problema y la *meta.* También puede ocurrir que en un determinado dominio de aplicación no sea necesario conocer todo el camino y baste con devolver el nodo meta.

Mostraremos gráficos que ayuden a percibir a primera vista el recorrido realizado en el espacio de estados. Igualmente describiremos detalladamente las especificaciones de los algoritmos que llevan a cabo cada uno de los tipos de control asociados a cada estrategia.

El problema de integración simbólica introducido en la sección 3.3 lo utilizaremos para ilustrar los diferentes grafos de búsqueda asociados a las distintas estrategias de control. A este respecto queremos aclarar que no es nuestro objetivo explicar detalladamente el significado de las expresiones matemáticas generadas en los distintos estados, más bien queremos llamar la atención del lector en el mayor o menor número de estados generados por dichas estrategias.

Antes de pasar al estudio de cada método conviene señalar que los principios que deben cumplir las estrategias sistemáticas de control son: no dejar —a priori— ningún nodo sin explorar y no explorar más de una vez el mismo nodo.

3.4.1 Búsqueda en Amplitud

• **Descripción:** es aquél procedimiento de control en el que se revisan todas las trayectorias de una determinada longitud antes de crear una trayectoria más larga. Es decir; no se genera ningún nodo de nivel N hasta que no se hayan obtenido todos los del nivel $N-1$. La Fig. 3.6 muestra esquemáticamente el recorrido realizado en un árbol cuando la estrategia de control aplicada es la búsqueda en amplitud.

Para llevar a cabo este método de búsqueda se crea una lista de "nodos por expandir" —generalmente conocida por el nombre de ABIERTA—, cuya manipulación se realiza en forma de *cola* (denominado en terminología sajona FIFO, «*first-in-first-out*»); es decir, se da una mayor prioridad a los nodos que llevan un mayor tiempo esperando.

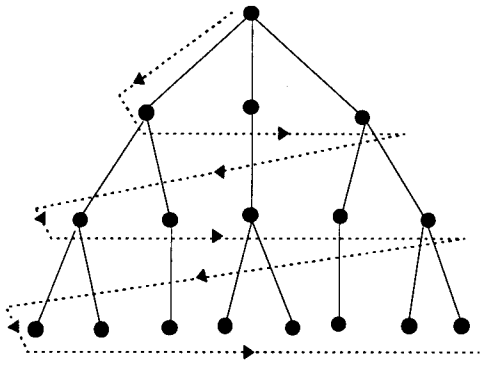


Fig. 3.6. Recorrido *en amplitud* de un árbol

- **Procedimiento Búsqueda en Amplitud:**

1. Crear una lista de nodos llamada ABIERTA e "inicializarla" con un único nodo raíz, al que se le asigna el estado inicial del problema planteado.

2. Hasta que ABIERTA esté vacía o se encuentre una meta realizar las siguientes acciones:

2.1. Extraer el primer nodo de ABIERTA, llamar a este nodo *m*.

2.2. Expandir *m* (generar todos sus sucesores)

- Para cada operador aplicable y cada forma de aplicación:

(1) Aplicar el operador a *m*, obtener un nuevo estado y crear un "puntero" que permita saber que su antecesor inmediato es *m*.

(2) Si el nuevo estado generado es *meta* salir del proceso iterativo inicializado en 2.2. y devolver dicho estado.

(3) Incluir el nuevo estado al final de ABIERTA. (Una vez completado este proceso para todos los sucesores de *m* —cuando no se haya encontrado antes una meta— se continua el proceso iterativo en el paso 2.)

Observación: si el algoritmo termina con una *meta*, el camino de la solución puede obtenerse recorriendo los punteros creados desde el

nodo *meta* al nodo raíz. En caso contrario el proceso terminaría sin haber encontrado la solución.

- **Ejemplo:**

La siguiente figura (Fig. 3.7) muestra el esquema del grafo que definiría el espacio de búsqueda de un problema concreto de integración siguiendo la estrategia de búsqueda en amplitud. Volvemos a recalcar que, dada su extensión, no vamos a detallar el significado de todas las expresiones del grafo de búsqueda; nuestro objetivo es llamar la atención sobre el número y el orden en que dichos estados van generándose.

En el grafo de la Fig. 3.7 aparecen los estados del espacio de búsqueda etiquetados con un identificador PASO*n* (siendo *n* el número que representa el orden en que fue generado dicho nodo en el proceso de búsqueda). También aparecen unos círculos con un número junto a algunos de los estados generados. Éstos representan los estados que han sido expandidos en el proceso de búsqueda, indicando el número el orden en que estos fueron expandidos. Como puede apreciarse, en un principio se expande el nodo raíz, que es la descripción del problema planteado: "$\int x \sin(x)\,dx$", generando todos sus sucesores (PASO2 y PASO3). Siguiendo lo indicado en el apartado 2.2 del "procedimiento", se expande el estado PASO2 obteniéndose sus sucesores (PASO5, ..., PASO7). Como sigue existiendo un nodo (PASO3) de nivel inferior, que todavía no se ha expandido, se procede a la generación de sus sucesores (PASO8). Una vez completado el primer nivel de expansión se comienza con el siguiente nivel. En el proceso de creación de los sucesores de los nodos del segundo nivel (recordar que la raíz se considera que tiene nivel 0) se obtiene como primer nodo obtenido a partir del PASO5 (primer nodo del segundo nivel) un estado (PASO9) que cumple las condiciones de ser un nodo meta; en este caso, un nodo que no contenga el símbolo de integración simbólica en su descripción. En este momento se detiene el "procedimiento" según lo señalado en la condición de finalización del bucle iniciado en el paso 2.

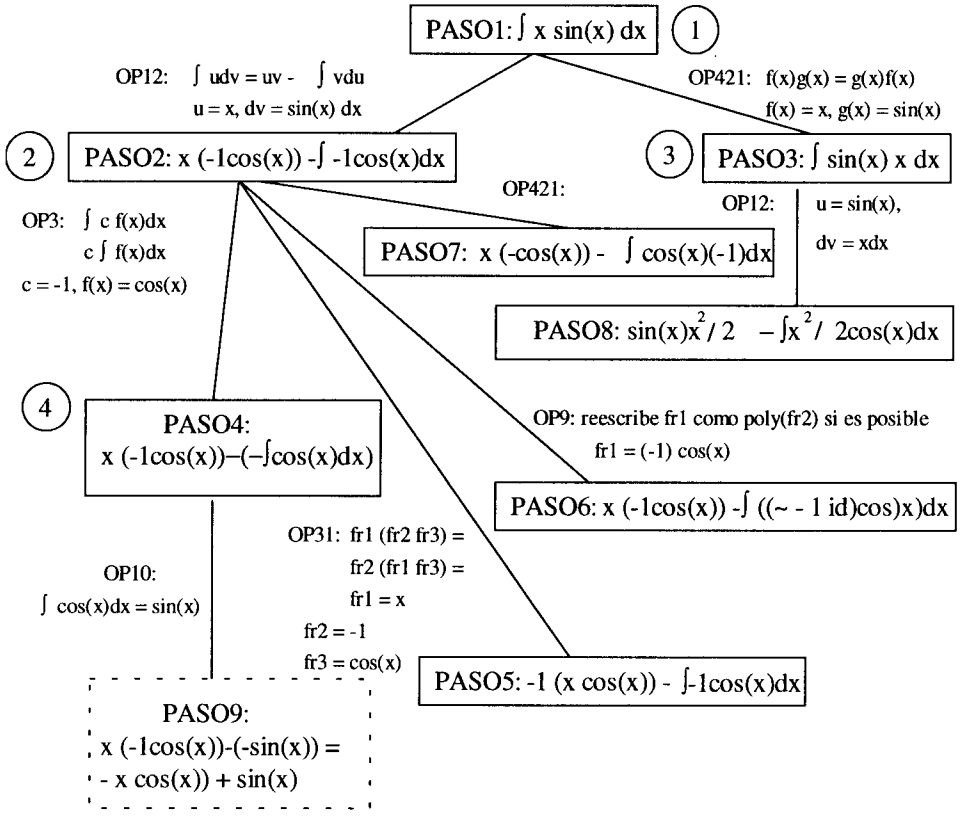


Fig. 3.7. Grafo de búsqueda en amplitud para un problema de integración simbólica

- **Complejidad**

Temporal: la complejidad de este método depende en gran medida del factor de ramificación y de la profundidad de la solución. Si el número medio de sucesores es n (se utiliza el valor medio ya que el árbol de búsqueda no tiene por qué ser uniforme; es decir, no todos los nodos tienen igual número de sucesores) y la solución se alcanza en el nivel p el tiempo empleado es $1 + n + n^2 + ... + n^p$. Siendo p un valor generalmente grande, la complejidad será del orden $O(n^p)$. (El

Espacial: Dado que antes de abandonar la generación de todos los sucesores de un nivel se deben recordar (almacenar en ABIERTA) todos los nodos de dicho nivel, la complejidad espacial es también del orden $O(n^p)$.

• **Análisis**

Ventajas: si el problema tiene una solución este procedimiento garantiza el encontrarla. Si hubiera varias soluciones, se obtiene la de menor coste (*la óptima*); es decir, la que requiere menor número de pasos (si consideramos un coste uniforme de aplicación de los operadores).

Desventajas: si el nivel de profundidad asociado a la solución es significativamente menor que el factor de ramificación se expandirían demasiados nodos inútilmente. Por otro lado, la principal desventaja de este método es el espacio de almacenamiento requerido. Esto lo hace prácticamente inviable para problemas complejos, como suelen ser los del mundo real.

3.4.2 Búsqueda en Profundidad

• **Descripción:** en cada nivel, a lo largo del proceso de búsqueda, se selecciona un único nodo para ser expandido. Por consiguiente, al contrario que en la búsqueda en amplitud (ver 3.4.1), no se tienen un conjunto de caminos que se van expandiendo según avanza el proceso de búsqueda, sino que sólo se considera un único camino (una sola rama del árbol). En esta estrategia son prioritarios los nodos de niveles superiores (más profundos). Es decir; el funcionamiento responde al modelo conocido como pila (en inglés, LIFO, «*last-in-first-out*»). Dicho de otra forma; *el objetivo es recorrer el grafo siguiendo el camino de máxima pendiente con respecto al parámetro «profundidad»*. El nodo más profundo (el último elemento de la pila) es el único que en cada momento está siendo considerado.

Hay que considerar el caso de que el camino pueda tener una longitud infinita, o demasiado larga (un camino se considera poco útil cuando el número de operaciones necesarias para generarlo lo hacen prácticamente inviable), necesitándose entonces establecer un *límite de profundidad* (*lp*). También debe estudiarse la factibilidad del camino seguido hasta el momento. En muchos problemas se puede establecer una prueba de viabilidad que determine si se ha alcanzado una "vía muerta" o "punto sin retorno" (también llamado "callejón sin salida"). Esto se produce cuando se viola alguna restricción que deben cumplir todos los nodos que formen el camino de la solución (p.ej., en el juego de "tres en raya" no puede ocurrir que el adversario complete una diagonal). Un punto sin retorno en el problema de integración simbólica sería aquél en el que no quedan más operadores aplicables.

La siguiente gráfica (Fig. 3.8) muestra esquemáticamente el recorrido de un árbol de búsqueda cuando la estrategia de control aplicada es la de búsqueda en profundidad.

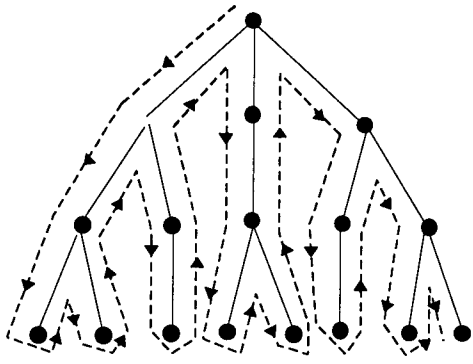


Fig. 3.8. Recorrido *en profundidad* de un árbol

- **Procedimiento de Búsqueda en Profundidad:**

1. Crear una lista de nodos llamada ABIERTA e "inicializarla" con un único nodo raíz, al que se le asigna el estado inicial del problema planteado.

2. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:

(1) Si ABIERTA está vacía, terminar con *fallo*; en caso contrario, continuar.

(2) Extraer el primer nodo de ABIERTA y denominarlo *m*.

(3) Si la profundidad de *m* es igual a *lp*, regresar a 2; en caso contrario, continuar.

(4) Expandir *m* creando "punteros" hacia *m* en todos sus sucesores, de forma que pueda saberse cuál es su antecesor. Introducir dichos sucesores al principio de ABIERTA siguiendo un *orden arbitrario*. (La "falta de orden" refleja el carácter "no informado" de este procedimiento.)

(4.1) Si algún sucesor de *m* es *meta*, abandonar el proceso iterativo señalado en 2 devolviendo el camino

de la solución, que se obtiene recorriendo los punteros de sus antepasados.

(4.2) Si algún sucesor de *m* se encuentra en un *callejón sin salida* eliminarlo de ABIERTA.(Se continua el proceso iterativo en el paso 2.)

Observación: en el paso (4), en lugar de generar todos los sucesores de *m*, se podría haber generado uno solo. En este caso, (4.1) y (4.2) se particularizarían para ese único nodo generado. El procedimiento se modificaría para convertirse en *búsqueda con retroceso* (en inglés, "Backtracking"), como se estudiará más adelante en 3.4.3.

En el caso de grafos, para evitar duplicados, habría que comprobar, para cada nuevo nodo generado, si ya existía en el grafo de búsqueda. Para reducir la complejidad de esta comprobación sólo se evitan los duplicados a lo largo del camino de la solución construido hasta un momento dado.

Cabe la posibilidad de que a un determinado nodo se pueda llegar por varios caminos. En estos casos, para que el procedimiento anterior garantice que siempre se "expande" el más "profundo" del grafo de búsqueda, habría que considerar la profundidad del resto de los caminos existentes hasta dicho nodo. La profundidad de un nodo es uno más la que tuviera su "padre" menos profundo. Para reducir el coste de dicha comprobación, se adopta una solución de compromiso que consiste en elegir, como nodo considerado más profundo, el último nodo de la pila, siendo la profundidad del nodo considerada en el paso (4) "la del antecesor *m* (que en ese momento se esté expandiendo) + 1".

- **Ejemplo:**

Se vuelve a mostrar el árbol generado para el mismo problema de integración simbólica resuelto previamente mediante la búsqueda en amplitud.

Como puede apreciarse en la gráfica siguiente (Fig. 3.9) —suponiendo que en cada iteración se generan todos los sucesores del nodo elegido (paso "(4)" en el algoritmo)—, se obtiene un grafo de búsqueda muy parecido al ya presentado (Fig. 3.7). La principal diferencia observable consiste en que no se han generado los sucesores del nodo PASO3 (que había resultado ser un trabajo inútil en el proceso de búsqueda en amplitud, ver Fig. 3.7). La eficiencia aparentemente mayor del método depende del orden de inclusión de los sucesores en ABIERTA (paso "(4)" del procedimiento). Por lo tanto, si a partir de la raíz se hubiera elegido el mencionado PASO3, el problema se habría complicado y la

solución obtenida en ningún caso hubiera sido tan buena como la encontrada (óptima) en la búsqueda en amplitud.

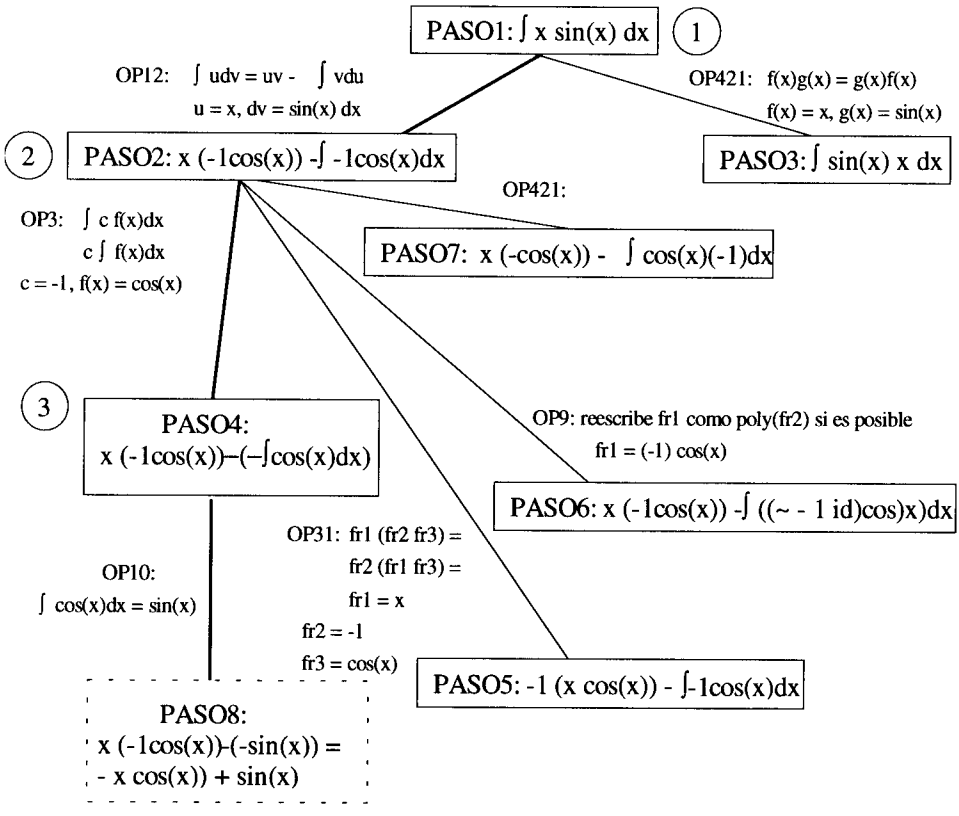


Fig. 3.9. Grafo de búsqueda en profundidad para un problema de integración simbólica

- **Complejidad**

Temporal: es la misma que la complejidad temporal del procedimiento de búsqueda en amplitud, ya que para un grafo de una determinada profundidad *p* se generan los mismos nodos, aunque en diferente orden. (Las complejidades de las estrategias de búsqueda se calculan considerando "el peor caso".)

Espacial: a lo largo del grafo de búsqueda sólo es necesario guardar constancia del camino construido hasta el momento, luego la complejidad para un camino de longitud *p* será dicho valor más el *factor de ramificación* (*r*), ya

que cada vez es necesario guardar constancia de todos los sucesores del nodo que se está expandiendo; es decir $O(p + r)$.

• **Análisis**

Ventajas: la principal ventaja de este método radica en el reducido valor de su complejidad espacial. Cuando existen múltiples soluciones posibles, la eficiencia del método aumenta.

Desventajas: la dificultad estriba en el tiempo requerido. El algoritmo puede dedicarse a recorrer un camino demasiado largo que no conduzca a ninguna solución. Es más, si no se guarda constancia de los nodos que forman el camino recorrido, se podría caer en ciclos y el proceso no acabaría. El problema, por lo tanto, es determinar cual debe ser *lp*. Si éste es inferior a la longitud real del camino de la solución, ésta nunca se encontraría y si es mucho mayor sería ineficiente. Ésta es la razón por la que *lp* debería más bien llamarse *límite de exploración*.

3.4.3 Búsqueda con Retroceso

• **Descripción:** hemos visto que la búsqueda en amplitud y en profundidad expanden nodos en el espacio de búsqueda, es decir, consideran todos sus sucesores. Vamos a estudiar ahora un procedimiento que sólo considera en cada iteración un único sucesor, restringiendo por lo tanto el espacio de estados considerado. Igualmente se eliminan de dicho espacio los nodos que sean una "vía muerta". Cuanto mejor (más informado; ver siguiente capítulo) sea el criterio utilizado para limitar el número de estados considerados más eficiente será el proceso de búsqueda.

Dicho en pocas palabras, esta estrategia realiza un recorrido en profundidad del grafo de búsqueda en la que en lugar de hablar de expansión de nodos (obtención de todos los sucesores) se habla de generación de aquellos. Es decir, cada vez que un nodo se selecciona, si éste no es meta ni es un "punto sin retorno", sólo se genera uno de sus sucesores, ahorrándose por lo tanto el coste que supone la creación simultánea de todos ellos (ver 3.4.2). El camino recorrido sigue avanzando a través de dicho sucesor y si en algún momento se encuentra el mencionado "callejón sin salida", entonces se retrocede hasta el primer antepasado del que todavía partan caminos no explorados. Por consiguiente, el rasgo que define este tipo de estrategia es que utiliza un subconjunto del espacio de estados que se va actualizando a lo largo de todo el

proceso, a partir del cual tiene que cumplirse que sea posible alcanzar la solución buscada.

Conviene precisar que en este apartado estamos describiendo los *procesos de búsqueda con retroceso "cronológico"*. Dado el carácter introductorio de este capítulo, no analizaremos detalladamente otras formas que podrían denominarse de "retroceso informado", en las que no se retorna al "estado anterior" sino a uno determinado por otros criterios (ver comentarios más adelante).

La importancia de considerar un espacio reducido de estados que pueden formar parte de la solución ("candidatos") va más allá del ámbito de los procesos de búsqueda que estamos analizando en este capítulo y constituye en sí una forma de enfocar la solución de problemas, en la que a dicho conjunto de candidatos se le denomina *solución parcial del problema.*

- **Procedimiento Búsqueda con Retroceso:**

  1. Crear una lista de nodos llamada ABIERTA e "inicializarla" con un único nodo raíz que contiene el estado inicial del problema planteado.

  2. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:

     (1) Si ABIERTA está vacía terminar con *fallo*; en caso contrario, continuar.

     (2) Seleccionar el primer nodo de ABIERTA y denominarlo *m*.

     (3) Si la profundidad de *m* es igual a *lp* o si *m* ya no tiene más sucesores posibles (que no hayan sido examinados anteriormente), eliminarlo de ABIERTA y regresar a 2; en caso contrario, continuar.

     (4) Generar un "nuevo" sucesor *m´* de *m* e introducirlo al principio de ABIERTA, creando un puntero a *m*, y señalar que dicha rama ya ha sido considerada.

        (4.1) Si *m´* es *meta*, abandonar el proceso iterativo iniciado en el paso 2. devolviendo el camino de la solución, que se obtiene recorriendo los punteros de sus antepasados.

(4.2) Si *m´* se encuentra en un *callejón sin salida* eliminarlo de ABIERTA. (Se continua el proceso iterativo en el paso 2.)

Observación: para evitar repetir los nodos generados en (4) pueden adjuntarse a *m* los operadores que se le van aplicando en cada iteración del bucle señalado en 2.

- **Ejemplo:**

Una vez más utilizaremos el ejemplo de integración propuesto (ver Fig. 3.7 y Fig. 3.9) para comprobar las diferencias con las estrategias de control anteriores.

Supongamos, según el proceso ilustrado en la gráfica siguiente (Fig. 3.10), que la estrategia de búsqueda con retroceso selecciona como primer operador aplicable la propiedad conmutativa con respecto a la multiplicación (OP421); en dicho caso el camino lleva, a través de la aplicación sucesiva del operador de integración por partes (OP12) a un punto de retroceso (PASO4), que se produce al comprobar la coincidencia con un paso previo (PASO2). La búsqueda realizada a partir del primer antecesor de dicho estado (PASO4) que tenga sucesores todavía no explorados (PASO3) complica aún más el problema inicial, sobrepasando un cierto límite de profundidad (PASO8) (dicho límite debe ser un parámetro inicialmente establecido en el procedimiento). Se produce entonces un retroceso escalonado hasta la raíz, que permite iniciar (PASO9) la reconstrucción del camino correcto de la solución. (Las flechas punteadas en la figura indican que no se han mostrado todos los estados generados por no ser relevantes; por ejemplo, entre el PASO3 y el PASO8 hay un estado que resulta simplemente de la aplicación del operador OP3 que extrae el valor −1 situado bajo el signo de integración.)

- **Complejidad**

Temporal: el número de operaciones para un factor de ramificación *n* y para una solución supuestamente situada en un cierto nivel *p* vuelve a ser del orden $O(n^p)$.

Espacial: en este caso se reduce el espacio necesario, basta con recordar el camino recorrido hasta un momento dado, esto es, $O(p)$.

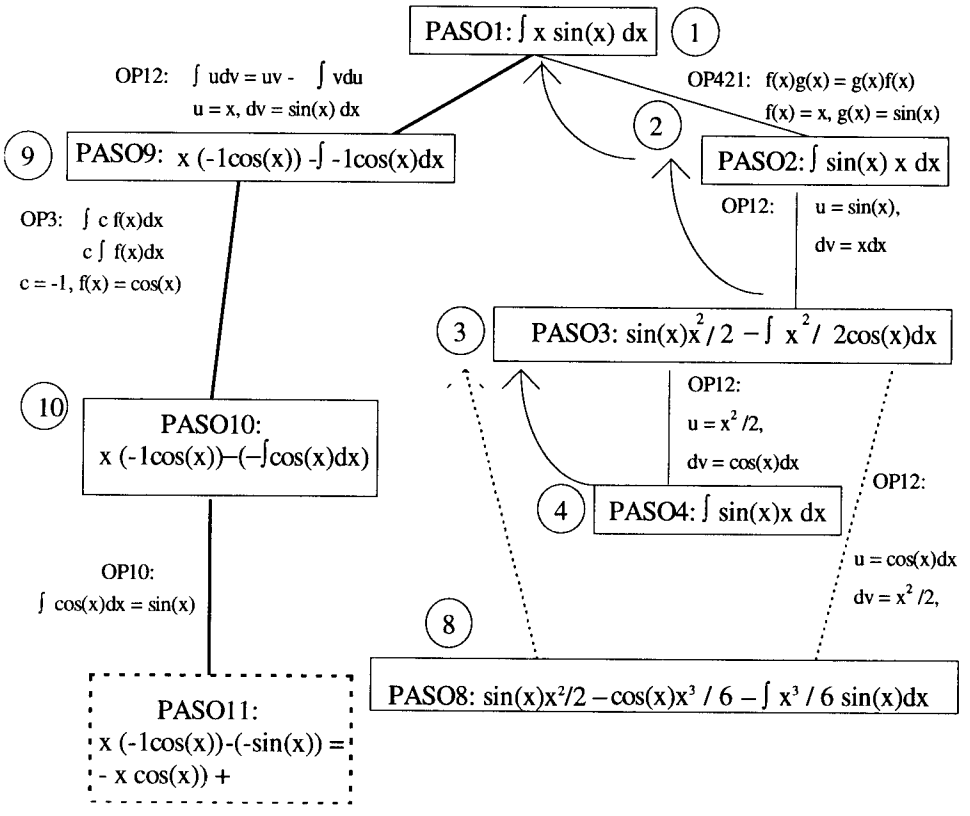


Fig. 3.10. Búsqueda con retroceso en un problema de integración simbólica

- **Análisis**

Ventajas: la gran ventaja de la búsqueda con retroceso frente a la búsqueda en profundidad es la de necesitar todavía un menor espacio de almacenamiento. Sólo hay que recordar en cada instante un sucesor del nodo seleccionado. Otra ventaja es que no se generan las ramas del árbol de búsqueda que se encuentren después del camino de la solución. Si recorremos el árbol de izquierda a derecha, no se explorarían las trayectorias situadas a la derecha de la solución.

- **Desventajas:** vuelve a ser un inconveniente saber cuál debe ser el límite de exploración *lp*. Para determinar su valor habría que conocer de antemano en qué nivel se encuentra la solución. Otra desventaja fundamental es el no establecer ningún orden previo de recorrido de los sucesores de un nodo (generar

primero los mejores sucesores en el punto "(4)" del algoritmo). Por tanto, la eficiencia temporal del método se queda limitada por su carácter fortuito.

Existen otras formas de volver sobre los pasos ya realizados. El nodo elegido en la etapa de retroceso no tiene por que ser el nuevo origen del camino de búsqueda. Por ejemplo, puede ser conveniente retroceder varios niveles. Para ello habrá que determinar en qué momento del camino ya recorrido se establecieron las condiciones que produjeron finalmente un *callejón sin salida*. En dichos casos se aplica el procedimiento general denominado **Búsqueda con Retroceso Dependiente y Dirigido**. Básicamente consiste en establecer cuándo dos pasos se consideran contradictorios, retrocediéndose entonces hasta el nivel en que pueda ser reemplazada la elección que causó la contradicción. En este caso habrá que tener presente si la reducción del grafo de búsqueda supone un mayor ahorro que el coste asociado al mantenimiento de las anotaciones adicionales (registros de dependencias) que habrá que realizar para determinar el origen de la incompatibilidad. Este método tiene su origen en la aplicación de la propagación de restricciones al análisis de circuitos eléctricos.

3.4.4 Otros Métodos Derivados

Búsqueda en Profundidad Progresiva

Hay un planteamiento alternativo que no es más que la combinación adecuada de los métodos anteriores que se puede denominar **Búsqueda en Profundidad Progresiva**. Consiste en realizar una búsqueda en profundidad por niveles, de forma que el límite de exploración aumenta una unidad por cada nivel analizado.

La gráfica siguiente (Fig. 3.11) pretende reflejar el proceso realizado en este método. En el primer cuadro (aparece con el fondo oscuro) se realiza una búsqueda en profundidad (apartado 3.4.2), considerando exclusivamente los nodos del primer nivel (lp=1). Si entre dichos nodos no se encontrara un nodo meta, entonces se procede a realizar un segundo recorrido en profundidad (el cuadro más grande con el fondo claro) que en esta segunda ocasión llegará hasta los nodos situados en el segundo nivel (lp=2). Dicho límite de profundidad se va incrementando en una unidad sucesivamente hasta encontrar la meta buscada. La consideración de cada vez más niveles para ser explorados es una técnica muy utilizada también en los problemas de búsqueda con adversarios que analizaremos en el próximo capítulo.

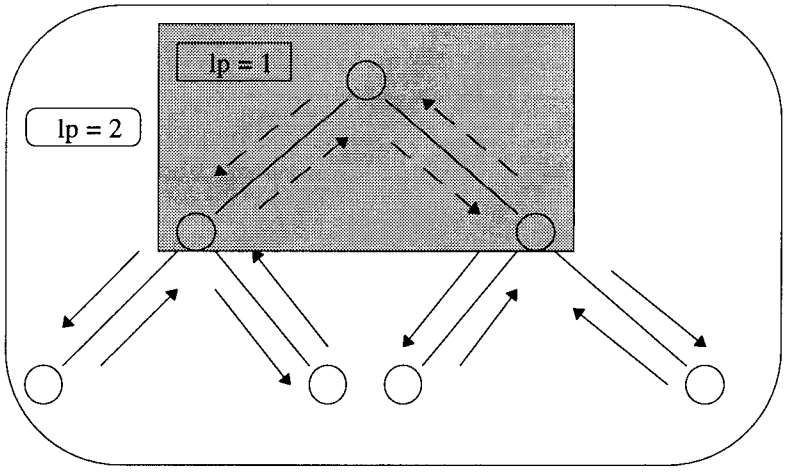


Fig. 3.11. Búsqueda en Profundidad Progresiva.

Esta estrategia es la más eficiente dentro de las estrategias "no informadas" que son capaces de encontrar la solución de menor coste existente (óptima). Se puede comprobar que su *complejidad temporal* es del orden $O(n^p)$ y la *espacial* es sólo del orden O(p).

Búsqueda Bidireccional

Antes de pasar a describir propiamente este método conviene aclarar algunas cuestiones relacionadas.

En los grafos de búsqueda mostrados hasta ahora para los problemas de integración simbólica, se parte de una integral indefinida expresada en un formalismo simbólico concreto (por ej., "$\int x\sin(x)\,dx$"). El proceso de búsqueda consiste en encontrar la secuencia de operadores "apropiada" que permite encontrar la solución (en este caso una expresión sin el símbolo de integración "$\int$"). Este tipo de razonamiento también se conoce como **dirigido por los datos** o **encadenamiento hacia delante**, ya que la dirección del proceso de búsqueda parte de los datos suministrados (planteamiento del problema) y avanza en el camino de búsqueda hacia un estado meta. El espacio de representación utilizado recibe el nombre de **espacio de estados**.

Existe un planteamiento alternativo denominado **encadenamiento hacia atrás** o **dirigido por las metas**. En éste se distinguen dos situaciones. Una, en la

que la aplicación de un operador al problema origina exactamente un único nuevo estado (problema), cuyo tamaño o dificultad es normalmente menor que el problema previo; en estos casos también se utiliza la representación del espacio de estados. Otra situación, más compleja, ocurre cuando al aplicar un operador se divide el problema en un conjunto de "subproblemas"; se dice entonces que emplea una representación basada en la **reducción del problema** (que analizaremos en el apartado 3.5).

• **Descripción:** cuando además de la descripción del problema, se parte de una meta explícita, existe un planteamiento alternativo que resulta de la combinación de dos grafos de búsqueda diferentes.

En el planteamiento bidireccional se realiza simultáneamente la búsqueda de la meta a partir de la descripción del problema (*encadenamiento hacia adelante*) y la búsqueda del estado inicial a partir de un estado solución (*encadenamiento hacia atrás*) hasta que ambos procesos confluyan en algún estado. La solución es el camino que conduce desde la situación inicial a la meta pasando por dicho estado.

Para poder llevar a cabo esta búsqueda se necesitan algunos elementos adicionales. Los operadores deben ser invertibles (o tiene que ser factible el recorrido hacia atrás) y se debe establecer expresamente —no bajo una serie de condiciones— cuál es la meta que se quiere alcanzar.

En el procedimiento que presentamos a continuación se presupone que los dos procesos de búsqueda realizados responden a una estrategia de búsqueda en amplitud. Realmente bastaría con que uno de ellos respondiera a tal estrategia para garantizar la convergencia total del proceso.

• **Procedimiento de Búsqueda Bidireccional en Amplitud:**

1. Inicializar dos grafos de búsqueda. En el primero, que llamaremos (A.1), el nodo raíz será el estado inicial del problema planteado, y en el segundo (A.2) tendrá como raíz la meta de dicho problema.

2. Inicializar el límite de exploración $lp=1$.

3. Continuar con la realización de la búsqueda (A.1) hasta el nivel lp.

4. Continuar con la realización de la búsqueda (A.2) hasta el nivel lp.

5. Comprobar si alguno de los nuevos estados generados en 3 y en 4 coinciden. En tal caso; devolver la trayectoria de la solución formada

por la concatenación de los caminos obtenidos en (A.1) y en (A.2) hasta el nodo encontrado. En caso contrario, volver al paso 2.

Observación: la comparación realizada en el paso 5. es un aspecto crítico que debe ser tenido en cuenta. Dado que se realizan dos procesos de búsqueda en amplitud y dado que se parte del problema y la meta, la solución está garantizada en alguno de los niveles marcados por *lp*.

En la práctica, la estrategia más eficiente consiste en cambiar el sentido de exploración en cada nuevo nivel, en vez de cambiarlo después de un cierto número de iteraciones. En este caso vuelve a ser necesario el tener un valor preconcebido de dicho número que dependerá del valor real de la profundidad asociada a la solución buscada.

En esta estrategia puede ocurrir que alguno de los dos procesos de búsqueda no responda a un esquema de búsqueda en amplitud.

- **Complejidad**

Temporal: suponiendo que la comprobación de la obtención de un estado común en el proceso de búsqueda puede realizarse en el mismo tiempo para todos los nodos del grafo, el número de operaciones para un factor de ramificación *n* y para una solución supuestamente situada en un cierto nivel *p* es del orden $O(n^{p/2})$. Cada uno de los procesos de búsqueda realizados sólo tiene que recorrer la mitad del camino.

Espacial: la condición necesaria impuesta para garantizar la convergencia permite afirmar que el espacio de almacenamiento necesario es igualmente del orden $O(n^{p/2})$.

- **Análisis**

Ventajas: la gran ventaja, con respecto a otras estrategias exhaustivas, de realizar este tipo de búsqueda es la división por dos del factor exponencial asociado tanto al número de operaciones realizadas como al espacio requerido. Por lo tanto, se explorarán muchos menos nodos que en un proceso realizado en un sólo sentido. Se ha podido comprobar que en ausencia de otras fuentes de información adicionales, la reducción en el número de nodos que cada vez constituyen la frontera de exploración (nodos de ABIERTA) es un método que logra una gran eficiencia.

Desventajas: vuelve a ser un inconveniente saber cuál debe ser el límite de exploración *lp* cuando se quiere realizar un planteamiento de *búsqueda redirigida*. El número de iteraciones antes de cambiar el sentido de la búsqueda es un parámetro crítico para la eficiencia de este método. Otro inconveniente es el no disponer de algún criterio adicional para poder ordenar la selección de nodos meta a lo largo del proceso.

3.5 REDUCCIÓN DEL PROBLEMA

Dentro del *razonamiento dirigido por las metas* se distinguen dos situaciones. Una en el que cada operador aplicado a un estado produce un único nuevo estado. En este caso se puede adoptar la representación ya vista del *espacio de estados*. Y otra, en la que al aplicar un operador se generan una serie de "subproblemas", cada uno de los cuales será en general más sencillo que el problema original. Entre los operadores aplicables en los problemas de integración simbólica, el operador OP11: "$\int (f(x) \pm g(x))\,dx \Rightarrow \int (f(x)\,dx \pm \int (g(x)\,dx$" realiza la descomposición de la integral de una suma (resta) en la suma (resta) de las integrales. Estos problemas se representan mediante el esquema conocido como **reducción del problema**. El objetivo es dividir el problema en subproblemas hasta alcanzar estados que tengan solución conocida (baste aplicarles un operador conocido).

Los problemas que pueden descomponerse tienen dos tipos de operadores aplicables, aquéllos que propiamente realizan la descomposición y el resto, que permite, por ejemplo, reconocer los estados meta. A diferencia del esquema de producción (ver el resto de los planteamientos del capítulo presente), se supone que la descomposición de un problema permite tratar los subproblemas en los cuales se divide de una forma independiente (y por lo tanto simultánea). Cada movimiento genera elementos en la *BD* que pueden tratarse a parte, como si cada uno de ellos fuera una nueva *BD*. La forma de representación más adecuada para resolver estos problemas son los llamados **grafos Y/O** o **árboles Y/O**, dependiendo del planteamiento del problema adoptado.

Un **árbol Y/O** tiene como raíz del árbol el problema completo que intenta resolverse. Los nodos serán los subproblemas en que aquél puede dividirse o las submetas que deben alcanzarse. Los nodos se conectan entre sí por medio de dos tipos de enlaces. Los enlaces "O" representan problemas que pueden resolverse de varias formas alternativas. Los enlaces "Y" reflejan situaciones que pueden descomponerse en una serie de etapas o subproblemas; de forma que el

problema inicial (el nodo "padre" en el árbol) queda resuelto cuando se ha obtenido la solución de todas sus partes (nodos hijos en el árbol). Por lo tanto los árboles ya vistos relativos al problema de integración simbólica eran árboles donde todos sus enlaces eran del tipo O, ya que se trataba de encontrar un sólo camino hasta la meta (veremos más adelante que realmente se trataba de encontrar el mejor camino). En dichos árboles "O" cada rama representa un camino alternativo. En la figura (Fig. 3.12) se muestra un esquema de la topología implicada en este tipo de árboles.

Los enlaces del tipo Y se representan por medio de una línea arqueada que conecta todos los nodos implicados. Los nodos terminales (nodos hoja en el caso del árbol) representan un problema básico (soluble directamente) o un subproblema que no puede descomponerse más. El primero se "etiqueta" como resuelto y el segundo como irresoluble.

Conviene aclarar que el concepto de "enlace" es diferente al de arco. El primero puede conectar más de dos nodos entre sí, son los llamados enlaces Y. Por el contrario, los enlaces O establecen una relación tan sólo entre dos nodos (une el antecesor en el grafo con su inmediato sucesor). Por tanto, en este tipo de grafos hablaremos de enlaces en lugar de arcos.

Como muestra la figura siguiente (Fig. 3.12), la solución encontrada en los grafos Y/O no es un simple camino a lo largo del grafo de búsqueda, sino un grafo denominado *grafo solución* (o *árbol solución* en el caso de los árboles). Dado que en este tipo de grafos hay nodos que requieren la terminación de todos sus hijos (unidos por enlaces Y), la solución puede estar formada por varios nodos terminales (nodos hoja) y no por un único nodo (grafos tipo O, todos los vistos hasta ahora). Las ramas discontinuas representan aquéllas que no pertenecen al árbol solución y los nodos terminales son los que aparecen rellenos; se observa cómo, en el árbol solución marcado en el dibujo A, aparecen líneas arqueadas que conectan enlaces del tipo Y, que requieren la terminación de todos los nodos conectados (es decir, estos enlaces reflejan la descomposición de un problema en un conjunto de subproblemas). Por otro lado, en el esquema B el árbol solución sólo contiene enlaces del tipo O, que sólo necesitan para ser "satisfechos" que alguno de los caminos implicados lleve a un nodo meta.

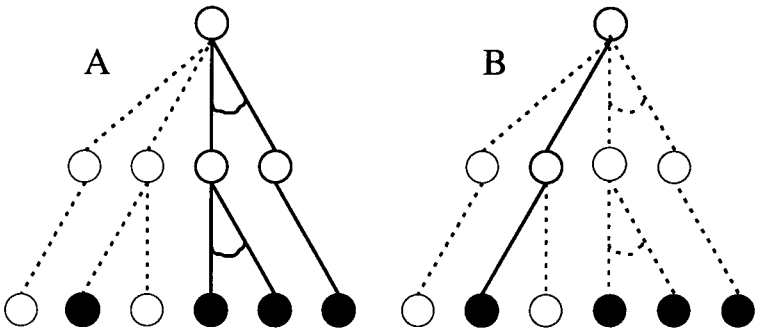


Fig. 3.12. Soluciones en árboles Y/O

Este tipo de estructuras son útiles en una gran diversidad de áreas. Por ejemplo en juegos (como se estudiará en el capítulo 4), un área clásica que ha servido para establecer algunos de los pilares de la IA. Los grafos Y/O son especialmente adecuados en las situaciones en las que hay una pareja de adversarios que van alternando sus movimientos. Los movimientos de uno de los jugadores se podrán representar mediante enlaces O —reflejan las alternativas posibles—, las jugadas del adversario se traducen en enlaces Y, ya que habrá que considerar todas las alternativas posibles de éste para intentar contrarrestar su efecto.

Otro área de aplicación de los grafos Y/O son los problemas en los que la solución es un conjunto no ordenado de acciones. Por ejemplo, en los problemas de integración simbólica que estamos presentando en este capítulo, pueden aplicarse operadores como la integración por partes o la descomposición en suma de integrales. Este tipo de operadores dividen un problema en subproblemas que no requieren ser resueltos en ningún orden determinado. En la siguiente gráfica (Fig. 3.13) se ilustra el caso de una solución obtenida aplicando el operador OP11 (descomposición de suma o resta de integrales). La aplicación de dicho operador se representa mediante enlaces Y, en los que todos los nodos necesitan ser resueltos (llegar a nodos meta) para obtener el correspondiente árbol solución. En la figura se han marcado dicho árbol y los nodos meta alcanzados (estados PASO6 y PASO9).

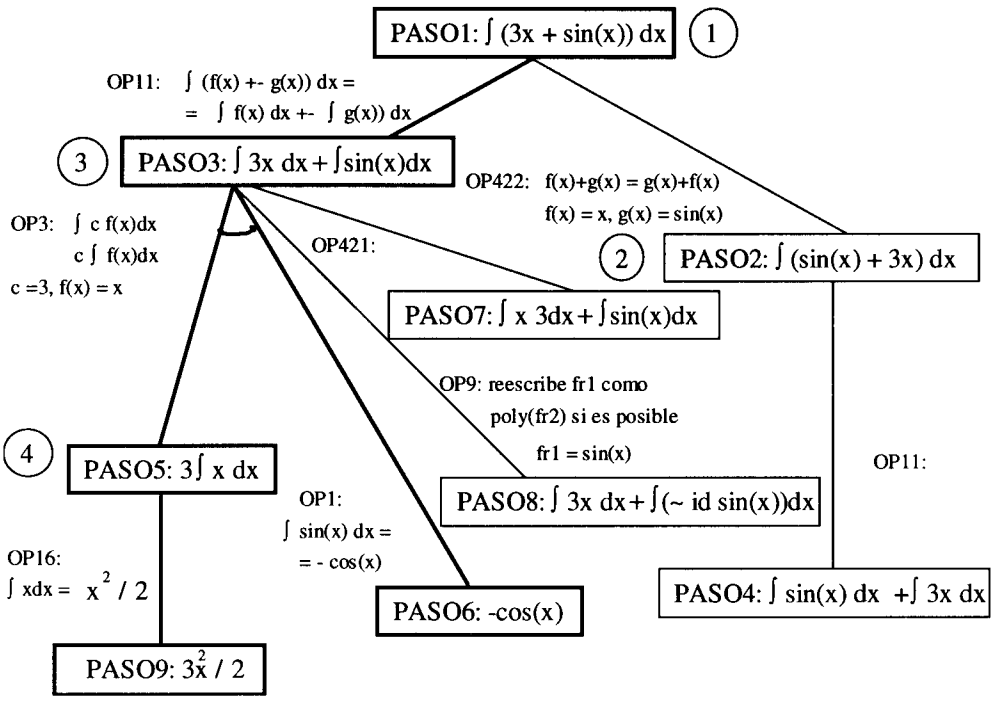


Fig. 3.13 Árbol de búsqueda Y/O para el problema: ∫ (3x + sin(x)) dx

Este tipo de representación también se utiliza en los problemas de razonamiento lógico y en los de planificación por etapas. Es decir, aquéllos en los que se pueda establecer una secuencia de pasos determinados por una serie de condiciones, no por el orden en que éstas se vayan cumpliendo. Por ejemplo, en los problemas de demostración automática (ver el capítulo 5 dedicado a la lógica), el orden en el que se generen el conjunto de premisas de una determinada inferencia no es relevante, lo importante es que llegue un momento en que todas ellas estén presentes, entonces se aplica la inferencia y se sigue avanzando en la demostración hasta que finalmente se alcance la conclusión deseada.

3.6 ALGORITMO GENERAL DE BÚSQUEDA EN GRAFOS

Este capítulo se ha dedicado a presentar las estrategias más básicas y genéricas de búsqueda en grafos, aquéllas que realizan una búsqueda sistemática

en el espacio de estados. El siguiente capítulo lo dedicaremos a métodos más complejos y útiles que se basan en el uso de información adicional del dominio para guiar la búsqueda. Vamos a presentar en este apartado la estructura que podría tener un procedimiento general de búsqueda en grafos. Conviene recordar que, aunque se puede aplicar al caso particular de los árboles, este método está diseñado para el caso más general de grafos, donde cada nodo del grafo puede tener varios antecesores.

Un método útil para resolver el problema de la búsqueda en grafos es mantener a lo largo del proceso dos listas, llamadas ABIERTA y CERRADA respectivamente. La primera permite reanudar en cualquier momento un camino abandonado previamente por existir un camino alternativo que entonces se consideró más favorable (con la información de la que se disponía, esto es, el grafo explícito o grafo de búsqueda). La lista CERRADA sirve como registro de los nodos elegidos a lo largo de todo el grafo de búsqueda.

Vamos a discutir y presentar brevemente el método clásico que sirve para recorrer estas estructuras [Nilsson, 1981]. La estrategia de control que proponemos aquí realiza un "bucle" (proceso iterativo) básico de exploración que permite obtener el camino de menor coste desde el nodo inicial a la meta buscada.

- **Procedimiento General de Búsqueda en Grafos[4]:**

  1. Crear un grafo de exploración G que consista en un único nodo que contiene la descripción del problema. Crear una lista de nodos llamada ABIERTA e "inicializarla" con dicho nodo.

  2. Crear una lista de nodos llamada CERRADA que inicialmente estará vacía.

  3. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:

     (1) Si ABIERTA está vacía terminar con *fallo*; en caso contrario, continuar.

     (2) Eliminar el primer nodo de ABIERTA, llamar a este nodo *m* e introducirlo en la lista CERRADA.

[4] Como se analizará en el próximo capítulo, este método también se conoce con el nombre de **Primero el Mejor**, ya que continuamente se está eligiendo a lo largo del proceso de búsqueda el nodo que reciba una mejor valoración con respecto al criterio de ordenación aplicado.

(3) Expandir *m*:

(3.1) Generar el conjunto *M* de todos sus sucesores que no sean antecesores e introducirlos como sucesores de *m* en G.

(3.2) Si algún miembro de *M* es *meta*, abandonar el proceso iterativo señalado en 3. devolviendo el camino de la solución, que se obtiene recorriendo los punteros de sus antepasados.

(3.3) Poner un puntero a *m* desde los nuevos nodos generados (aquellos que no pertenezcan a G). Incluir dichos elementos en ABIERTA.

(3.4) Para cada nodo de *M* que ya estuviese en ABIERTA o en CERRADA, decidir si se *redirige* o no su puntero a *m*.

(3.5) Para cada nodo de *M* que ya estuviese en CERRADA, decidir para cada uno de sus descendientes si se *redirigen* o no sus punteros.

(4) **Reordenar** la lista ABIERTA aplicando algún criterio previamente establecido.

Observaciones:

1. A lo largo del camino se crean dos grafos de exploración, uno explícito, que coincide con el grafo de búsqueda, y otro implícito, que se obtiene recorriendo los punteros que conducen desde la meta al estado inicial. Este último mantiene una estructura de árbol, ya que cada nodo tiene un puntero a un único antecesor inmediato.

2. El paso (3.1) garantiza que ningún nodo es antecesor de sí mismo. El coste de lograr esta condición es muy elevado, sobre todo considerando que el grafo de búsqueda puede ser muy grande en problemas reales, por lo que muchas veces se toma la solución de compromiso de garantizar este extremo sólo para los nodos que formen parte del mejor camino encontrado hasta el momento. Este camino siempre puede reconstruirse siguiendo los punteros establecidos.

3. En sentido figurado, si consideramos el proceso de expansión del proceso de búsqueda como el desbordamiento de un río, ABIERTA representaría la orilla que puede seguir creciendo (la frontera de expansión) y CERRADA lo constituirían los nodos "bajo la superficie" y aquellos que aun estando en la superficie no están en la orilla. En otras palabras, *abiertos* son los nodos frontera no expandidos y *cerrados* son todos los expandidos, independientemente de que hayan tenido sucesores o no.

4. Los punteros se "reasignan" en función del criterio aplicado en (3.4) y (3.5), que en principio debe intentar garantizar que señalen el camino mejor encontrado hasta el momento.

5. Si un nodo "cerrado" cambia el puntero de su antecesor (3.5), el camino "menos costoso" puede ser parte del camino "menos costoso" de alguno de sus sucesores.

6. La realización eficiente de este procedimiento requiere establecer en el paso (4) el **criterio de ordenación** de los nodos en ABIERTA, que depende del problema en concreto que se está intentando resolver. Hasta ahora en este capítulo se han presentado métodos que, dada su independencia del dominio de aplicación, realizan un ordenamiento arbitrario de los nodos generados. En el próximo capítulo, dedicado a los métodos informados de búsqueda, analizaremos diferentes formas de llevar a cabo dicho ordenamiento.

3.7 DISCUSIÓN

Hemos presentado a lo largo de este capítulo los elementos básicos necesarios para establecer un marco genérico de resolución de problemas en IA, de los cuales podemos extraer al menos las siguientes conclusiones.

1. Los elementos básicos introducidos en este capítulo no son suficientes para resolver problemas complejos, como suele ocurrir en la mayoría de los problemas del mundo real; incluso el problema de integración simbólica utilizado como referencia, requiere una cantidad considerable de información adicional que no hemos tenido en cuenta.

El formalismo de representación utilizado limita y determina en gran parte los posibles resultados. Así, en el problema de integración, la especificación del lenguaje de descripciones tiene un gran impacto sobre la eficiencia total del proceso de búsqueda. Habría que definir:

1. Un lenguaje para describir los estados.
2. Un lenguaje para describir los operadores.
3. Un proceso de "equiparación" que sirva para comprobar si la descripción asociada a un operador cubre toda o parte de la descripción asociada a un estado; en otras palabras, para comprobar si un operador es aplicable a un estado.

2. En los métodos de búsqueda sistemáticos explicados hasta ahora, se ha omitido una cuestión que puede llegar a ser determinante para decidir cuál es el camino de búsqueda más adecuado. Nos referimos al hecho de haber presupuesto un coste de aplicación homogéneo de los operadores. Esta circunstancia sólo suele ocurrir en los "problemas de laboratorio" (perfectamente definidos).

A este respecto, en el problema de integración simbólica, la aplicación del operador de integración por partes OP12 requiere realizar una "llamada recursiva" al *solucionador* del sistema. Esto implica mucho mayor coste computacional que, por ejemplo, la aplicación del operador que descompone la integral de la suma en la suma de las integrales (OP11). Este último sólo realiza la "reescritura" de la expresión asociada al estado en cuestión.

Por lo tanto, en el integrador se podría diferenciar el *coste real* de un nodo (suma de los tiempos de cálculo consumidos en cada paso a lo largo del camino que conduce de la raíz al nodo considerado) del coste finalmente utilizado, que como veremos en el capítulo 4 es un valor que está en función de la propia información dependiente del dominio introducida en el proceso.

Todas estas cuestiones reflejan la influencia que tiene el dominio de aplicación en la resolución de problemas, incluso utilizando los métodos de búsqueda más generales, como los que hemos descrito a lo largo de este capítulo.

4 BÚSQUEDA HEURÍSTICA

J. G. Boticario

El contenido de este tema es la continuación natural del anterior. Allí vimos la búsqueda sin información del dominio y aquí vamos a estudiar la búsqueda basada en conocimiento. El principal problema de las estrategias de búsqueda estudiadas en el capítulo previo es su carácter exhaustivo; tratan de llegar a la solución del problema recorriendo todos los nodos del espacio de búsqueda según un criterio de ordenación independiente del problema que se esté intentando resolver.

En este capítulo analizaremos cómo puede aprovecharse la influencia —sin lugar a duda positiva— que puede tener el conocimiento dependiente del dominio. Los métodos que veremos, conocidos como «estrategias de búsqueda heurística», utilizan dicho conocimiento para guiar el control del proceso de búsqueda. Analizaremos un marco en el cual se puede explotar dicho conocimiento de una forma efectiva y estudiaremos algunas de las estrategias básicas que hacen uso del mismo.

La búsqueda heurística es una de las contribuciones clave de la IA a la solución eficiente de problemas. Dentro de los métodos de control heurístico distinguiremos los problemas y las soluciones aportadas en algunas de las técnicas utilizadas en la teoría de juegos. Los métodos utilizados en los juegos son especialmente interesantes por diversos motivos: por razones históricas, este área ha recibido un especial interés en la IA dado que en su planteamiento se pueden apreciar comportamientos inteligentes; por razones experimentales, los juegos ofrecen un marco sencillo de actuación en el que pueden analizarse metodologías complejas de resolver problemas, de planificación y de la teoría de la decisión, que pueden ser útiles en campos tan diversos como el mundo de los negocios o el de la experimentación científica; por razones formales, las

decisiones de un juego pueden evaluarse en modelos perfectamente definidos en los que, además, pueden contraponerse las respuestas del sistema con las respuestas de origen humano. Una vez hayamos revisado las estrategias básicas de juego, terminaremos este capítulo mostrando una estrategia general de resolución de problemas que utiliza el conocimiento dependiente del dominio.

4.1 ELEMENTOS IMPLICADOS

Se pueden distinguir dos marcos de referencia diferentes, el dominio del observador (DO) y el dominio propio (DP), que ayudan a concretar los problemas y las soluciones aportadas. Creemos conveniente realizar un especial hincapié en diferenciar las atribuciones que le corresponden a cada uno de éstos.

- **Dominio del Observador**

Los métodos heurísticos tratan de resolver problemas difíciles —poco estructurados— eficientemente, obteniendo soluciones satisfactorias en el dominio de aplicación. Estos métodos no garantizan encontrar la solución óptima del problema, pero sí obtienen buenas aproximaciones con mucho menor esfuerzo.

El origen de estos métodos está en el análisis del comportamiento humano. Muchas veces resolvemos problemas, a veces muy complejos, utilizando criterios que carecen de una sólida fundamentación analítica, aunque resulten muy útiles.

Los problemas que van a ser objeto de estas técnicas pueden ser de dos tipos:

1. Problemas de *solución parcialmente conocida.* El campo de la medicina es un claro exponente. Por ejemplo, el diagnóstico precoz de algunas enfermedades sigue siendo una cuestión no resuelta. Sin embargo, existen parámetros determinantes establecidos a través de la experiencia que ayudan a solucionar muchos casos. En el diagnóstico del cáncer de próstata la edad del paciente es un factor decisivo.

2. Problemas *intratables de solución conocida.* Aquéllos que no pueden resolverse completamente por la complejidad implicada, pero que sí se conoce el

método para resolverlos. En estos casos se utiliza algún criterio extraído de una simplificación tratable del problema original.

En el problema del ajedrez el objetivo es realizar el movimiento más oportuno teniendo en cuenta la disposición de las fichas en el tablero en un momento dado. El método conocido consiste en prever los posibles movimientos propios y del adversario con vistas a ganar la partida. Dado que no se pueden anticipar todas las posibles alternativas, un criterio heurístico válido consiste en intentar eliminar el mayor número de piezas del adversario.

Los métodos heurísticos son adecuados para problemas en los que para alcanzar un nivel de principiante no es necesario realizar un proceso excesivamente complejo. La razón está en que los criterios aplicados suelen ser sencillos y prácticamente inmediatos.

Las metas que quieren alcanzarse se entienden en un sentido genérico. Es decir, el objetivo del modelo no tiene por que ser exactamente obtener una determinada meta: ganar, perder o empatar una partida en un juego, también puede ser reducir el coste o el tamaño de la solución.

- **Dominio Propio:**

Los métodos heurísticos se utilizan para obtener soluciones útiles en problemas sujetos al fenómeno de la *explosión combinatoria*; es decir, problemas en los que el número de operaciones realizadas crece de forma incontrolada a lo largo del proceso de búsqueda de la solución.

Cuando hablamos de problemas difíciles conviene recordar que suelen presentarse dos tipos de problemas: los denominados tipo *P*, que son los que pueden ser resueltos mediante un procedimiento de complejidad polinómica (el grado del polinomio no suele ser elevado) y los problemas *NP* —objeto de las presentes técnicas—, en los que todos los algoritmos conocidos requieren un tiempo exponencial en el peor caso, aunque se conocen algoritmos eficientes no deterministas.

Recordemos algunos de los ejemplos más clásicos. El número de operaciones que hay que realizar para determinar la estrategia ganadora en el problema del ajedrez crece exponencialmente con el número de jugadas posibles que quieren predecirse antes de realizar cada movimiento (el número de operaciones necesarias es 10^{120}). Otro problema ampliamente comentado es el del viajante de comercio. En éste se pretende encontrar la ruta de viaje óptima

para recorrer un conjunto de ciudades conectadas entre sí, de forma que sólo se visite una ciudad una sola vez. La solución del problema depende en forma factorial del número de ciudades implicadas. Por otro lado, incluso problemas realmente sencillos como el del juego del 8-puzzle (con un tablero de 3x3, ver Fig. 4.1 y discusión posterior), también llegan a ser intratables cuando se aumenta el número de casillas consideradas; por ejemplo, para el caso de 4x4 el espacio de estados lo forman 16! nodos distintos.

Desde el punto de vista computacional, los objetivos antes declarados en el DO suelen traducirse en establecer las llamadas **funciones de evaluación heurística** que ayuden a seleccionar el estado más prometedor en el espacio de búsqueda.

4.1.1 Conocimiento de Control

La clave de los procesos heurísticos de búsqueda es el ahorrar tiempo y/o espacio de almacenamiento evitando recorrer muchos caminos inútiles. No consiste en eliminar una parte importante del espacio de búsqueda, sino en introducir información adicional que guíe el recorrido realizado.

- **Reglas de Control Heurístico**

Una forma de reducir la complejidad del proceso realizado es introducir reglas de control heurístico en el proceso de búsqueda. Éstas pueden ser de dos tipos en función del conocimiento reflejado. Dado que en el capítulo previo no tuvimos en cuenta la información del dominio, hasta ahora no hemos mencionado el hecho de que en la aplicación de cualquier procedimiento de búsqueda a un problema concreto existen dos tipos de conocimiento de control claramente diferenciados. El primero se refiere al conocimiento heurístico dependiente del dominio, utilizado para seleccionar alguna de las situaciones alcanzadas y también para seleccionar la siguiente acción aplicable sobre aquella (p.ej., los operadores aplicados en el problema de integración simbólica descrito en el capítulo previo). La segunda fuente utilizada en el control del proceso de búsqueda es el conocimiento general disponible sobre el funcionamiento del solucionador (p.ej., un operador es útil si al aplicarlo resuelve el problema).

En cada etapa del proceso, al *expandir* un nodo (ver capítulo 3) se comprueba si alguno de sus sucesores es meta y entonces se devuelve el camino de la solución. Este conocimiento es independiente del dominio de aplicación. Si

tradujéramos dicho conocimiento en una regla de control podríamos obtener la forma siguiente:

R1: EXITO-OPERADOR **SI** (AÑADE *objetivo operador*) **Y**
(APLICABLE *operador nodo*)

Por otro lado, existe un conocimiento dependiente del dominio de aplicación (heurístico), que se utiliza para determinar cuándo se ha alcanzado la situación objetivo. En el ejemplo de integración simbólica (ver capítulo 3), una regla de control dependiente del dominio tendría la forma indicada en R2.

R2: (AÑADE objetivo operador) **SI**
(CONDICION-OPERADOR *op* "∫") **Y**
(NO (ACCION-OPERADOR *op* "∫"))

La regla R2 indica que si el operador aplicado elimina el símbolo de integración de la expresión asociada al estado, entonces se alcanza el objetivo buscado.

En este tema nos vamos a centrar en explicar las mencionadas *funciones de evaluación heurística* y dejaremos los aspectos relativos a las reglas para más adelante (el tema 6 se dedica enteramente a las reglas y el tema 10 trata el aprendizaje de reglas de control).

- **Funciones de Evaluación Heurística**

Estas funciones son una aplicación del espacio de estados sobre un espacio numérico.

$$f(estado_j) = n_j$$

siendo n_j el valor numérico que refleja en qué grado se considera prometedor un estado en la búsqueda de la solución.

El conocimiento del dominio reflejado por las funciones de evaluación heurística (de ahora en adelante también denotadas por ***fev***) se obtiene a través del establecimiento de *modelos simplificados del modelo original.* Suelen ser *funciones de evaluación estáticas* que miden factores que determinan la proximidad al objetivo. Por ejemplo, en el problema del 8-puzzle —que consiste en ocho fichas numeradas que pueden moverse sobre un tablero de 3x3 cuadros gracias a que una de las casillas está vacía—, un método heurístico que ayuda a seleccionar el mejor movimiento consiste en comprobar cuál es el número de

piezas situadas correctamente, ya que el objetivo es alcanzar una determinada situación *meta* a partir de una situación dada. En la Fig. 4.1 se presenta un problema sencillo resuelto de forma prácticamente inmediata.

reglas aplicables en el problema del 8-puzzle

R_1: si b $\neq$ $Fila_1$ $\Rightarrow$ Mover el blanco hacia arriba
R_2: si b $\neq$ $Fila_3$ $\Rightarrow$ Mover el blanco hacia abajo
R_3: si b $\neq$ $Columna_3$ $\Rightarrow$ Mover el blanco a la derecha
R_4: si b $\neq$ $Columna_1$ $\Rightarrow$ Mover el blanco a la izquierda
(*b* señala la posición del blanco en el tablero)

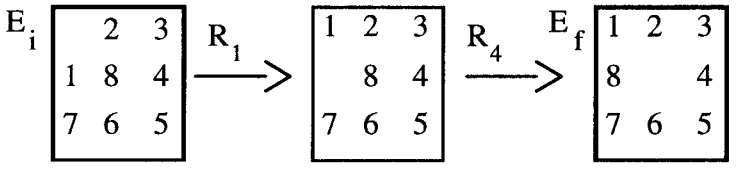


Fig. 4.1 Problema del 8-Puzzle.

Las reglas que causan el movimiento en el 8-Puzzle deben garantizar dos restricciones: una casilla sólo puede moverse a un cuadro lindante y una casilla sólo debe moverse a un cuadro vacío. Si eliminamos la segunda restricción, es decir, **simplificamos el modelo**, podemos establecer una medida heurística que se utiliza para determinar la distancia de un nodo cualquiera a la meta. Esta medida es la suma de las distancias, vertical y horizontal, que hay desde cada una de las casillas mal situadas con respecto a sus posiciones correspondientes en la situación meta. Dicho valor se conoce con el nombre de **distancia de Manhattan** (o distancia entre los bloques en una ciudad). Por ejemplo, en la Fig. 4.1 el valor de dicha medida para el estado inicial es $H_m(E_i)=2$. En cada etapa del proceso se elige como siguiente nodo en el grafo de búsqueda aquél que tenga un menor valor para dicha función.

El grafo de búsqueda se recorre en función de un objetivo que consiste en maximizar o minimizar el valor indicado por dichas funciones, teniendo en cuenta que dicho valor debe reflejar la cercanía relativa con respecto a la meta buscada. Por ejemplo, en el problema del mapa de carreteras que veremos a continuación (ver Fig. 4.2), se busca obtener la mejor ruta entre dos ciudades dadas, para ello se aplica una función heurística que tiene en cuenta la *distancia aérea* (en línea recta) entre las distintas ciudades que componen el mapa. De

esta forma, se elige una ciudad como siguiente en el camino cuando la suma de la distancia a la ciudad actual más la distancia aérea a la meta sea la menor.

Supongamos el caso de una persona que desea elegir la mejor ruta para viajar de Madrid a Santander considerando el mapa de carreteras de la Fig. 4.2. En la primera etapa del proceso existen cuatro alternativas: Cáceres, Palencia, Zaragoza y Valencia. Si la única información disponible es la distancia a dichas ciudades, la primera elección será Cáceres, dado que es la más próxima. Sin embargo, en el caso de considerar la distancia aérea, se podría haber tomado una decisión más acertada. Veamos la definición de la función de evaluación heurística que podría utilizarse para aprovechar dicho conocimiento. Por ejemplo, la distancia entre Madrid y Santander a través de Palencia sería:

Distancia (Madrid, Santander) = Distancia (Madrid, Palencia)
+ Distancia-aérea (Palencia, Santander)

que es más pequeña que la obtenida a través de Cáceres:

Distancia (Madrid, Santander) = Distancia (Madrid, Cáceres)
+ Distancia-aérea (Cáceres, Santander)

La utilización de la función *distancia aérea* en la valoración de cada alternativa permite acelerar el proceso de búsqueda gracias a la consideración de información especialmente relevante para el problema que se está intentando resolver.

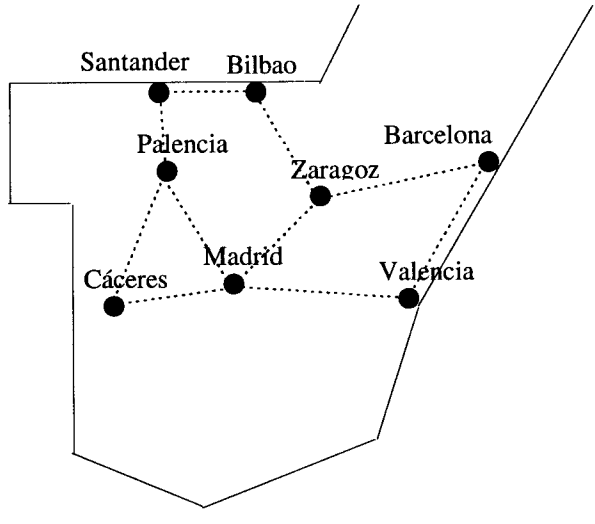


Fig. 4.2 Problema del mapa de carreteras

Evidentemente, *cuanto mejor indique la función heurística la cercanía real a la meta, menor será el grafo de búsqueda expandido.* Una condición que deben cumplir las *fev* es que su valor máximo (o mínimo, por ejemplo, en el problema del 8-Puzzle el objetivo era tener el menor número de piezas mal colocadas) debe alcanzarse al ser aplicada a un estado *meta* (comentaremos de nuevo esta condición más adelante al estudiar el algoritmo *A**).

Aunque el objetivo último sea encontrar funciones de evaluación heurística lo más exactas posibles, hay que valorar si el coste asociado al cálculo de dichas funciones compensa al coste de realizar el proceso de búsqueda sin dicho conocimiento o con otras funciones menos exactas pero más sencillas. El equilibrio, como ya hemos comentado, suele estar en obtener funciones en modelos simplificados del problema inicialmente planteado.

Estas funciones son óptimas cuando son invariantes con respecto a un movimiento óptimo desde cualquier estado a lo largo del proceso de búsqueda. Es decir, el valor de un estado es aproximadamente igual al valor del estado surgido por aplicación del mejor movimiento sobre el primero. En otras palabras, el valor de la función permanece invariante a lo largo del camino óptimo (aconsejamos la lectura del apartado dedicado al método *A** para comprender esta cuestión).

Por último, antes de pasar a describir métodos heurísticos concretos, conviene señalar que las técnicas que analizaremos proporcionan un marco general independiente del dominio concreto de aplicación. Sin embargo, su efectividad depende fundamentalmente de la identificación correcta de los aspectos más relevantes en el problema que se está intentando resolver. El carácter general de estos métodos amén de su gran dependencia del dominio de aplicación son las razones por las que en la literatura disponible también conocen con el sobrenombre de *métodos débiles.*

4.1.2 Planteamiento del Problema

Al igual que en el apartado 3.1 del capítulo previo, el planteamiento del problema se realiza utilizando el esquema de la tarea genérica de búsqueda definido en la Tabla 3.1. La diferencia —explicada en el apartado previo— radica en la utilización del conocimiento del dominio para guiar la selección de los operadores disponibles.

Hemos analizado en la sección 3.6 un procedimiento general de búsqueda en grafos[1] que permite aplicar directamente la información proveniente del dominio. Dicho algoritmo, como veremos más adelante, se conoce más precisamente con el nombre de método de **búsqueda primero el mejor**. Recordando lo visto, la eficiencia del problema depende del criterio de ordenación de los nodos en ABIERTA (estructura de datos, generalmente una lista, que contiene los nodos que todavía no se han expandido). Una forma inmediata de aprovechas las *fev* estudiadas anteriormente es aplicarlas sobre los nodos generados a lo largo del proceso de búsqueda, de manera que se ordenen los nodos en ABIERTA según el valor de la función heurística utilizada.

La función de evaluación heurística mide la **cercanía estimada** al nodo meta. Es un valor inexacto ya que la información disponible hasta el momento en que se toma la decisión no es completa. Por ejemplo, en el problema del 8-Puzzle, la *distancia de Manhatann* no consideraba las interacciones existentes entre las posiciones de las distintas fichas en el tablero; así, cuando se está intentando aproximar una pieza a su posición deseada (meta), el resto de los elementos pueden verse alejados de sus *metas* correspondientes. Por tanto, el control realizado no determina con seguridad cuál es el mejor nodo pero sí que ayuda a seleccionar un buen nodo teniendo en cuenta dicha información, y por tanto, sirve para reducir el número de nodos considerados (complejidad espacial) y el tiempo empleado (complejidad temporal).

4.2 UNA ESTRATEGIA IRREVOCABLE

Aunque no lo hemos mencionado explícitamente, hasta ahora sólo hemos presentado estrategias de búsqueda *tentativas*. Es decir, aquéllas en las que, en cualquier momento a lo largo del proceso de búsqueda, puede abandonarse un determinado camino de exploración para seguir estudiando otros caminos más prometedores.

Existe también la posibilidad de encontrar problemas para los que una estrategia *irrevocable* sea adecuada. La dificultad de aplicar este tipo de estrategias está en la dependencia absoluta existente con respecto al criterio de ordenación seguido en el espacio de búsqueda. Éste debe ser tan robusto que

[1] Hemos utilizado el nombre originalmente dado en [Nilsson, 1980] debido al tipo de notación y de explicación genérica que hemos preferido adoptar en el capítulo 3 de este libro. Sin embargo, en el contexto de la aplicación de las funciones de evaluación heurística consideramos más acertado denominarlo método de búsqueda *primero el mejor* (denominación utilizada en [Pearl, 1984]).

nunca sea necesario reconsiderar una decisión ya tomada. Dicho de otra forma, nunca se puede abandonar el camino de exploración elegido.

Método del Gradiente

• **Descripción:** también conocido como **búsqueda en escalada** por su semejanza con el recorrido realizado por un montañero en una escalada. Tanto el montañero como el método tienen como principal objetivo alcanzar la cima. La diferencia está en que el montañero no siempre elige el trazado de máxima pendiente, el método del gradiente sí. Es decir, se sigue el recorrido a través de los nodos en los que el valor de dicha función sea máximo.

El parámetro que define el sentido de la máxima pendiente es el valor de la *fev* aplicada, que llamaremos *f*.

• **Procedimiento del Método del Gradiente:**

1. Denominar *m* al estado inicial del problema planteado y asignar *m* a una variable llamada *elegido*.

2. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:

2.1. Expandir *m* creando el conjunto de todos sus sucesores.

- Para cada operador aplicable y cada forma de aplicación[2]:

(1) Aplicar el operador a *m* generando un estado *nuevo*.

(2) Si *nuevo* es *meta*, salir del proceso iterativo iniciado en el paso 2 y devolver dicho estado.

(3) Si *f*(*nuevo*) es mejor que *f*(*elegido*), cambiar el valor de la variable *elegido* asignándole el valor *nuevo*..

2.2. Si *f*(*elegido*) ≠ f(*m*) asignar *m* = *elegido*; en caso contrario, devolver *fallo*.

Observación: en el paso 2.2, se comprueba que alguno de los sucesores del nodo expandido tiene un mejor valor de la *fev*

[2] En el capítulo previo describimos un problema de integración simbólica en el que un mismo operador se puede aplicar a un estado concreto de varias formas diferentes; por ejemplo, el operador de integración por partes.

aplicada. Esta es una condición que debe cumplirse en todo momento, ya que no hay posibilidad de reconsiderar decisiones anteriores. En algunos problemas puede ocurrir que existan un conjunto de nodos situados al mismo nivel (con respecto al valor de la función *f*), en el borde de una *meseta* elevada. Estas situaciones se conocen también con el nombre de *altiplanos*. Para evitarlas, se podría haber creado una lista CERRADA que guardara constancia de los nodos expandidos. Si no se caería en la posibilidad de estar recorriendo constantemente los nodos situados en el borde de la meseta.

- **Campos de aplicación**

Se deben definir procesos de búsqueda en escalada en todos los casos en que se pueda identificar una función que tenga la propiedad de ir creciendo (o decreciendo) hasta el valor que tenga la función en el nodo meta. Los casos más representativos son:

- La teoría de juegos: algunos problemas de este tipo pueden afrontarse bajo este esquema. Un caso sencillo ya comentado es el problema del 8-Puzzle. También pueden definirse estas funciones para guiar la búsqueda en algunas de las situaciones tipo identificables en una partida de ajedrez; por ejemplo, al final de una partida. Supongamos un jugador con un rey y una torre que quiere dar un jaque a otro al que sólo le queda un rey. El primero debe reducir el espacio de movimiento del segundo evitando *ahogar* al contrario que supondría terminar en *tablas*.

- Los problemas de búsqueda de máximos y mínimos locales en el campo del análisis numérico.

- Situaciones en las que siempre (en cualquier estado) es posible aplicar cualquiera de los operadores disponibles. Esta propiedad se ha denominado *conmutatividad* [Nilsson, 1980]. En dichos casos siempre se puede llegar a la meta, independientemente del estado del que se parta, en el peor de los casos sólo se habrá producido un retardo en su obtención.

- **Análisis**

Ventajas: la principal ventaja de este método es su sencillez. La única memoria realmente necesaria consiste en guardar constancia de cuál es el estado alcanzado (*elegido*) en el proceso de búsqueda. En cada iteración, el estado

elegido pasa ser aquél que tenga un mejor valor de la función *f*. En otras palabras, no se necesita recordar cuál es el camino recorrido hasta un estado dado.

Desventajas: claramente el principal problema es la dependencia con respecto a la definición correcta de la función *f*. Este método exige que la función aplicada sea lo más *informativa* posible. Es decir, debe tener un valor que sea máximo en un nodo meta, que sea relativamente sencillo de calcular y que no incurra en máximos o mínimos locales (es decir, nodos en los que todos sus sucesores tienen un valor peor para la función *f*). Una forma de evitar los máximos locales es iniciar de nuevo el proceso en algún nodo que no estuviera en esa hipotética meseta, pero entonces podríamos dejarnos nodos sin explorar o repetir la exploración de un nodo expandido previamente.

La gran pregunta es, por tanto, ¿cómo obtener funciones tan informadas como las que aquí son necesarias?. La respuesta ya la hemos apuntado anteriormente. Se establecen **modelos simplificados** de tal forma que la función que obtenga una solución adecuada en éstos también la obtenga en el modelo real. Para que dicha simplificación sea posible una propiedad que tienen que tener los problemas es que sean *descomponibles* (ver 3.5). Por ejemplo, la *distancia de Manhattan* en el 8-Puzzle se obtiene considerando cada casilla independientemente sin estudiar las interacciones surgidas en el modelo real.

El resto de los métodos que vamos a tratar en este tema no dependen de la existencia de máximos locales.

4.3 ESTRATEGIAS DE EXPLORACIÓN DE ALTERNATIVAS

Muchas veces es muy difícil encontrar funciones tan informadas como las requeridas por el método del gradiente; sin embargo, sí es posible identificar criterios que ayudan a dirigir adecuadamente el proceso de búsqueda, aunque no sea de una forma infalible (precisamente por la información parcial de la que parte el criterio utilizado).

Las estrategias de búsqueda heurística más conocidas y utilizadas son precisamente las que realizan una exploración de alternativas en el espacio de búsqueda. El procedimiento que describe el funcionamiento al que responden estos métodos ya se ha descrito en el último apartado del capítulo 3. Vamos ahora a realizar un mayor hincapié en algunos aspectos no comentados entonces,

sobre todo aquéllos que se refieren a la utilización de las funciones de evaluación heurística para seleccionar los nodos que son expandidos a lo largo del proceso de búsqueda.

4.3.1 Búsqueda Primero el Mejor

• **Descripción:** (esta estrategia se ha descrito de forma genérica en el apartado 3.6) consiste en recorrer el grafo de búsqueda eligiendo en cada momento el nodo que tenga un mejor valor para una determinada función heurística que llamaremos *f*. A diferencia del *método del gradiente* (apartado 4.2), cuando el camino actual se aleje de la meta, se pueden retomar caminos de exploración abandonados anteriormente. La finalidad del proceso es encontrar el camino de menor coste, por lo que la función *f* debe medir la distancia a la meta. Una de las situaciones resueltas en este planteamiento es la existencia de *mesetas* o puntos de *estancamiento*, debido a la posibilidad de abandonar dichas situaciones en cuanto son detectadas. Esta estrategia trata de combinar las ventajas de los métodos ya vistos de Búsqueda en Profundidad y Búsqueda en Amplitud (capítulo 3).

• **Procedimiento de búsqueda Primero el Mejor (PM):**

Aunque ya hemos descrito este algoritmo en el apartado 3.6 volveremos a detallar los pasos implicados en la reordenación de los elementos en ABIERTA, teniendo en cuenta la utilización de las funciones de evaluación heurística.

Expansión y ordenación de nodos:

(3) Expandir *m* creando "punteros" a *m* en todos sus sucesores, de forma que pueda saberse que su antecesor inmediato es *m*.

(4) Si algún sucesor de *m* es *meta*, abandonar el proceso iterativo establecido en el paso 2, devolviendo el camino de la solución, que se obtiene recorriendo los punteros de sus antepasados.

(5) Para cada sucesor *n* de *m* calcular *f(n)* teniendo en cuenta las siguientes consideraciones:

(5.1) Si *n* es nuevo (no estaba en ABIERTA ni en CERRADA) asociar a *n* el valor *f(n)* e introducirlo en ABIERTA.

(5.2) En caso contrario (ya existía en el grafo):

a) - Si $f(n)$ es mayor que el que ya tenía asociado n, descartar el nuevo nodo.

b) - Si no (el nuevo camino es mejor), sustituir el valor que tuviera asociado el viejo nodo por el nuevo $f(n)$, y cambiar el puntero a m; si el nodo estaba en CERRADA, *actualizar el coste de sus descendientes que ahora podrían ver reducido su valor.*

- **Observaciones**

CERRADA es una estructura de datos necesaria para evitar el repetir la expansión de un nodo.

Otra consideración significativa es que la evaluación de la función en un nodo puede depender del camino seguido para alcanzarlo. Podríamos hablar de copias diferentes del nodo para cada uno de los valores posibles de f en función del camino elegido para llegar al nodo en cuestión desde la raíz. Los diferentes caminos pueden recrearse mediante el seguimiento de los punteros establecidos a lo largo del proceso.

Por tanto, la única situación conflictiva de este procedimiento surge cuando dos caminos confluyen en un mismo nodo n, esto puede ocurrir ya que la búsqueda es en un grafo. En estos casos, si el camino nuevo es de menor coste (paso 5.2 apartado b), habrá que determinar si alguno de los descendientes de n se ve afectado por dicha reducción. Es una comprobación muy laboriosa y hasta cierto punto discutible, ya que muchos de los descendientes analizados dejarán de tener interés debido precisamente al cambio efectuado. Aún así, en general, su coste es menor que el que supondría volver a expandir dicho nodo. Para esto último se podría cambiar la última etapa del procedimiento, trasladando n de CERRADA a ABIERTA. Es como si dicho nodo fuera nuevo en el camino, por lo que se generarán sólo los descendientes que a partir de entonces se consideren necesarios.

En el proceso se distinguen, un grafo de búsqueda G y un árbol de búsqueda implícito T. Éste último se puede dibujar siguiendo los punteros creados en el camino que en cada momento esté siendo explorado. G está formado por la unión de todos los árboles que han sido T a lo largo del proceso de búsqueda.

• **Ejemplo:**

Para entender mejor la diferencia entre el grafo y los árboles de búsqueda considerados, vamos a analizar de forma esquemática un caso concreto (ver Fig. 4.3). Como acabamos de señalar, *G* resulta de la unión de todos los árboles de búsqueda construidos hasta alcanzar la solución T_S. Los valores que aparecen en *G*, son los devueltos por la función *f* aplicada a cada uno de los nodos del grafo. Esta función indica la *distancia estimada a la meta desde cada nodo.* Por consiguiente, el mejor valor de la función *f* será el nodo con el valor mínimo. Así, en un principio se eligió el nodo cuya distancia estimada era 3, se generaron sus descendientes con una distancia estimada de 7 y 6 respectivamente (los punteros creados forman el árbol correspondiente T_1). Ante la posibilidad de obtener la solución a una distancia de sólo 4 se retomó dicha rama llegando a un único nodo de una distancia de 7 (T_2). Finalmente, se volvió a elegir el mejor camino hasta el momento, volviendo así al camino que condujo al nodo de distancia 6 (en ese instante el resto de los caminos tienen una distancia estimada de 7), desde el cual se terminó de construir el camino de la solución (T_S).

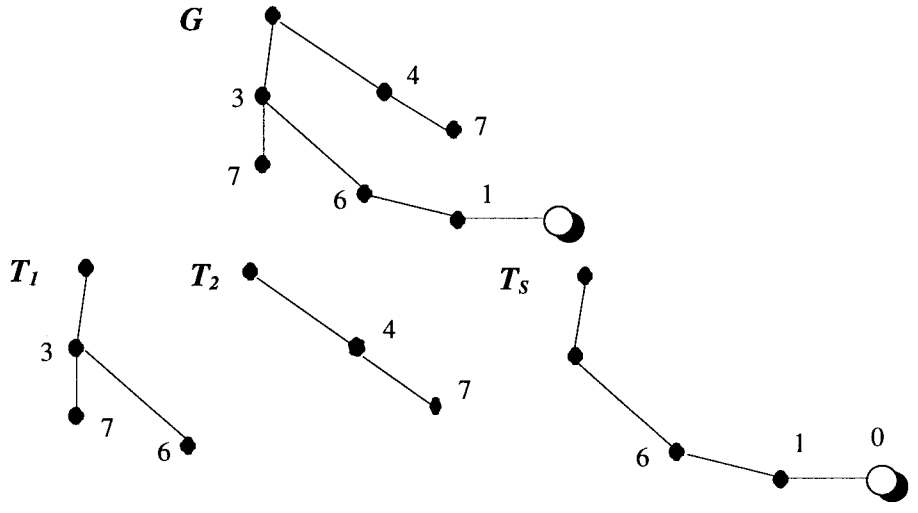


Fig. 4.3 Búsqueda *Primero el Mejor*

• **Análisis**

Ventajas: este método permite soslayar el problema de los *mínimos locales*, además tiene la garantía de que, tarde o temprano (depende de lo buena

que sea función heurística *f*), encontrará el mínimo global (suponiendo que el planteamiento sea reducir las distancias con respecto a la situación considerada *meta*).

Desventajas: la principal desventaja es que en el procedimiento no se considera el camino recorrido hasta el momento. Es decir, si un nodo se encuentra muy cerca del origen de la búsqueda y en otro ya se ha realizado un camino considerable, si el segundo tiene una distancia estimada a la meta menor se elegiría éste, pudiendo ocurrir que el primero tuviera un valor de *f* ligeramente superior al segundo. Esta es la razón por la que no se abandonan las *mesetas* hasta que no se ha alcanzado el borde de las mismas, recorriéndose inútilmente muchos caminos infructuosos.

Existe una variación del *PM* cuya utilidad se ha comprobado en diversos campos como el reconocimiento del habla, la visión automatizada y el aprendizaje (p.ej., [Dietterich & Michalski, 1981]). Nos referimos al conocido como **búsqueda en haz** (en inglés, "beam search"). Este método pretende acelerar el proceso de búsqueda ampliando el rango de estados que son considerados simultáneamente como mejores estados. De esta forma, en lugar de considerar uno sólo como *mejor* estado, se consideran una *frontera* de éstos. En este caso se establecen dos funciones: una con el mismo propósito que *f*, para evaluar la posibilidad de que un nodo determinado nodo pertenezca al camino óptimo y otra para *eliminar aquéllos menos prometedores* (realmente es un umbral del valor de *f* por debajo del cual los nodos no se incluyen en el haz de búsqueda). En cada momento habrá que mantener dos estructuras de datos, una que contenga los estados que pertenezcan al haz de búsqueda actual y otra para agrupar los posibles sucesores; es decir, todos los que surjan al expandir los nodos del haz. De nuevo, cuanto más informada sea la función que determina la validez de un nodo, más estrecho será el haz y más eficiente será el proceso de búsqueda realizado. En general este método requiere menos recursos ya que se consideran menos nodos en el espacio de búsqueda (se ahorra bastante espacio de almacenamiento si se compara con los métodos que utilizan ABIERTA).

4.3.2 Algoritmo A*

• **Descripción:** éste es el mejor método de búsqueda del tipo *primero el mejor* que sirve para resolver el problema que conlleva el no considerar el camino recorrido hasta un momento dado (el procedimiento original data de 1968 [Hart *et al.*, 1968]. Es una estrategia con una serie de propiedades formales significativas que discutiremos a continuación.

Hasta ahora la función f era un estimador heurístico (la hemos considerado "distancia estimada a la meta"). La clave de este método está en ampliar la definición de dicha función para que tenga en cuenta el coste del camino recorrido, resultando entonces la expresión siguiente:

$$\boxed{f(n)=g(n)+h(n)}$$

siendo,

$g(n)$: coste real del camino de menor coste encontrado hasta el momento desde la raíz del grafo de búsqueda G hasta el nodo en cuestión n. Es por lo tanto un estimador del coste real del camino óptimo al que identificaremos por $g^*(n)$.

$h(n)$: estimador heurístico del coste del camino óptimo desde el nodo n a una *meta*. El coste real lo calcularía la función $h^*(n)$.

$f(n)$: es un estimador del coste total del camino óptimo de una solución que pase por el nodo n. El valor real de dicho camino lo devuelve la función inicialmente desconocida que llamaremos $f^*(n)$.

- **Procedimiento A^*:**

1. Crear una lista de nodos llamada ABIERTA e "inicializarla" con un único nodo raíz que contiene el estado inicial del problema planteado. Llamar a este elemento r y asignarle $g(r) = 0$.

2. Crear una lista de nodos llamada CERRADA que inicialmente estará vacía.

3. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:

3.1. Si ABIERTA está vacía terminar con *fallo*; en caso contrario, continuar.

3.2. Eliminar el nodo de ABIERTA que tenga un valor mínimo de f, llamar a este nodo m e introducirlo en la lista CERRADA.

3.3. Si m es *meta*, abandonar el proceso iterativo señalado en 3. devolviendo el camino de la solución, que se obtiene recorriendo los punteros de sus antepasados (creados en 3.5.).

3.4. En caso contrario, expandir m generando todos sus sucesores.

3.5. Para cada sucesor n' de m:

(1) Crear un puntero de n' a m.

(2) Calcular $g(n') = g(m) + c(m, n') \mid c(a, b)$: coste de pasar de a a b.

(3) Si n' está en ABIERTA, llamar n al nodo encontrado en dicha lista, añadirlo a los sucesores de m y realizar (3.1.).

(3.1.) Si $g(n') < g(n)$ entonces "redireccionar" el puntero de n a m y cambiar el camino de menor coste encontrado a n desde la raíz; $g(n) = g(n')$ y $f(n) = g(n') + h(n)$

(4) Si n' no cumple (3), comprobar si está en CERRADA, llamar n al nodo encontrado en dicha lista y realizar las siguientes acciones:

Si (3.1.) no se cumple, abandonar (4), en caso contrario propagar el nuevo menor coste $g(n')$ (por lo que también se actualizarán sus los valores de f correspondientes) a sus descendientes (que llamaremos $n_i \mid i = 1, 2,...,$ siendo sus costes anteriores $g(n_i)$) realizando un **recorrido en profundidad** (apartado 3.4.2) de éstos, empezando en n' y teniendo en cuenta las siguientes consideraciones:

(4.1.) Para los nodos descendientes n_i cuyo puntero (que debe apuntar siempre al mejor antecesor hasta ese momento) conduzca hacia el nodo n' actualizar $g(n_i) = g(n_i')$ y $f(n_i) = g(n_i') + h(n_i)$ y seguir el recorrido hasta que se encuentre un n_i que no tenga más sucesores calculados o se llegue a un nodo en que ya ocurra que $g(n_i) = g(n_i')$; en cuyo caso se habría producido un ciclo y también habría que terminar la propagación.

(4.2) Para los nodos descendientes n_i cuyo puntero no conduzca hacia el nodo n' comprobar si $g(n_i') < g(n_i)$, en cuyo caso se debe actualizar el puntero para que conduzca hacia el nodo n' (mejor camino desde la raíz encontrado hasta ese momento) y se continúa el proceso de propagación.

(5) Si n' no estaba en ABIERTA o en CERRADA, calcular $h(n')$ y $f(n') = g(n') + h(n')$, introducirlo en ABIERTA y añadirlo a la lista de sucesores de m.

- **Relaciones con otros métodos de búsqueda:**

1. Si $\forall n\ \forall m\ (h(n) = 0) \wedge (c(m, n) = 1)$ entonces A^* se convierte en un proceso de **búsqueda en amplitud** (ver apartado 3.4.1).

2. Si $\forall n\ h(n) = 0$ entonces A^* se convierte en el método denominado **procedimiento de coste uniforme**; que es una variación de la búsqueda en amplitud donde lo que se busca no son caminos de la menor longitud sino de menor coste.

- **Propiedades formales:**

En ocasiones se han criticado los métodos de búsqueda heurística por su *impredecibilidad.* Son capaces de aportar soluciones buenas la gran mayoría de las veces, pero no garantizan encontrar el óptimo. Sin embargo, la método A^* posee una serie de propiedades que pueden confirmarse de antemano. Vamos a comentar las más relevantes.

1. Es un *método completo* de búsqueda incluso para grafos infinitos. Es decir, termina encontrando la solución cuando ésta exista para cualquier tipo de grafos.

2. Es **admisible**. Es decir, no sólo encuentra una solución sino que la que encuentra es la óptima.

Una función heurística se dice que es admisible si:

$$\forall n\ h(n) \leq h^*(n) \tag{4.1}$$

Es decir, si la función heurística que estima la distancia a la meta nunca supera la distancia real existente, entonces el algoritmo A^* garantiza encontrar la solución óptima.

Por ejemplo, si en el problema del 8-puzzle se aplica la función *distancia de Manhattan* a la que podríamos llamar h_1, siempre se encontrará el camino óptimo, ya que dicha función proviene de una simplificación del problema que garantiza la condición (4.1). Otra función heurística que se podría haber aplicado en dicho problema es simplemente el número de piezas mal colocadas con

respecto al objetivo, a la que podríamos denominar h_2. Siempre que se garantice la propiedad de admisibilidad, cuanto mayor sea el valor devuelto por la función h, más cerca del valor correcto h^* estará y más eficiente será la búsqueda. Bajo estas consideraciones llegamos a la siguiente definición:

Una función heurística h_1 se dice que está **más informada** que otra h_2 si ambas son admisibles y además:

$$h_1(n) \leq h_2(n) \; \forall n \mid \; n \text{ no es nodo meta} \tag{4.2}$$

En este caso el algoritmo que utilice la función h_1 (A_1) también se dice que está más informado que el que utilice h_2 (A_2). Se puede demostrar que todo nodo expandido por A_1 es también expandido por A_2; es decir, A_1 es más eficiente que A_2. En otras palabras, la efectividad del algoritmo se mide en función del número de nodos que se ahorra expandir.

Desgraciadamente hay que decir que la bondad del algoritmo depende de las funciones h que normalmente están basadas en el conocimiento estratégico sobre el problema concreto que está intentando resolverse. Dicha información no aparece de forma explícita en el grafo de búsqueda.

3. Si la función heurística h es *monótona* y *consistente* el algoritmo encuentra un camino óptimo para todos los nodos expandidos (son dos propiedades equivalentes, ya que aunque la *consistencia* se refiere a la propiedad (4.3) entre dos sucesores inmediatos, se puede generalizar por inducción sobre la profundidad del grafo).

Una función es monótona si se cumple:

$$h(n) \leq k(n,n') + h(n') \;\; \forall (n,n') \tag{4.3}$$

siendo $k(n, n')$ el coste del camino óptimo entre n y n'.

En otras palabras, si el valor de la función nunca decrece a lo largo de un camino desde la raíz, entonces se puede afirmar que cuando el algoritmo expande un nodo objetivo, el camino encontrado es el de menor coste a un objetivo. Por supuesto, la función h^* es monótona cumpliendo la propiedad (4.3).

La consecuencia inmediata en el funcionamiento del procedimiento descrito anteriormente es que deja de ser necesario considerar la revisión de los valores de f a lo largo del grafo de búsqueda expandido.

4. Entre todos los algoritmos que utilizan una función heurística consistente $h(n)$ y que encuentran una solución óptima, A^* expande el menor número de nodos.

Ya hemos comentado que el número de nodos expandidos depende de lo informada que sea la función $h(n)$. Dado que no siempre es posible encontrar funciones que cumplan la condición de *admisibilidad*, se plantea la cuestión de como se comporta el proceso con funciones menos *informadas*.

Para funciones cuyo error esté limitado por una cierta cota ε (es decir, $h(n) - h^*(n) \leq \varepsilon$), el coste de la solución nunca sobrepasa el coste óptimo en más de $1 + \varepsilon$. Desgraciadamente si la cota de error en lugar de ser absoluta es relativa, el número de nodos expandidos ya no guarda una relación lineal sino exponencial con respecto a la longitud de la solución.

- **Análisis**

Ventajas: recogiendo las propiedades anteriores se puede decir que el algoritmo A^* es un método completo, admisible y que encuentra la solución óptima para funciones de evaluación heurística consistentes con respecto a la longitud de la solución.

Desventajas: la propiedad de admisibilidad hace que el algoritmo gaste mucho esfuerzo en discriminar entre caminos prácticamente iguales, por lo que es más realista buscar soluciones dentro de unos márgenes acotados de error (como se ha comentado en la propiedad anterior 4). Al igual que ocurría en el algoritmo *PM*, un inconveniente es el espacio requerido (resultado de tener que mantener ABIERTA).

Una opción para reducir el espacio es realizar un planteamiento semejante al del *método de búsqueda en haz*. Igualmente existe una versión del algoritmo A^* llamada ***A**** **en profundidad progresiva** (*A*-P*). En este método cada iteración es una búsqueda en profundidad completa en la que se calcula si cualquier nodo supera un umbral de máximo coste ($f(n) > \delta \ \forall n$), en cuyo caso se elimina dicha rama y se vuelve al nodo generado más reciente. Dicha cota empieza siendo $h(r)$ y se va incrementando en cada iteración con el valor mínimo que supere el valor de la iteración anterior. La gran ventaja de *A*-P* es que la complejidad espacial es lineal con respecto a la profundidad de la solución, no necesitando gestionar la lista ABIERTA.

4.3.3 Búsqueda Heurística en Integración Simbólica

Vamos a presentar un ejemplo del proceso de búsqueda realizado en el sistema de integración simbólica introducido en el capítulo 3 (ver apartado 3.3).

Para resolver realmente un problema de integración simbólica se deben tener en cuenta las siguientes consideraciones:

- La representación adecuada de los elementos que participan en el proceso de búsqueda.
- La estrategia de búsqueda más eficiente (utilizar heurísticas disponibles).
- Las restricciones en tiempo y en coste de la solución.
- La eliminación del proceso de búsqueda de:
  - nodos ya examinados.
  - nodos que no pueden aportar nueva información.
  - subárboles que no puedan contener la solución.

El problema de la representación ya lo abordamos en el capítulo 3. Estudiaremos seguidamente la posibilidad de utilizar heurísticas que aceleren el proceso de búsqueda de la solución.

La procesos de búsqueda heurística pueden tener dos tipos de objetivos diferentes. Uno sería la búsqueda de la solución óptima (se sobreentiende de mínimo coste). En otras ocasiones se busca la solución que tenga el menor número posible de nodos explorados, esto es, la solución que se obtiene con el menor esfuerzo. Lo que realmente se realiza en la práctica es buscar soluciones que combinen ambos aspectos. El objetivo es encontrar un camino de coste reducido de la solución que a su vez haya requerido poco esfuerzo de búsqueda. Éste es precisamente el planteamiento de la búsqueda heurística del problema de integración simbólica que estamos comentando.

Recordemos que el *solucionador* del sistema busca obtener una expresión, obtenida a partir de la descripción inicial del problema dado, en la que no exista el símbolo de integración "$\int$". Para ello aplica —en principio— los operadores disponibles según la definición general de los mismos. Supongamos sin embargo, que además de la información sobre la descripción de las condiciones más generales de aplicabilidad de un operador, hubiera reglas heurísticas que determinan cuáles son las condiciones de utilidad del operador. Es decir, cuando

una determinada forma de aplicar un operador se ha comprobado que es útil en base al estudio a posteriori del grafo de búsqueda expandido.

Vimos en el ejemplo mostrado dentro del apartado dedicado a la *búsqueda con retroceso* (apartado 3.4.3) como la aplicación de forma equivocada del operador de integración por partes provocaba el abandono de una de las ramas del árbol de búsqueda.

$$\int sin(x)\,x\,dx = sin(x)\,x^2/2 - \int x^2/2\,cos(x)\,dx \mid OP12:\ u = sin(x),\ dv = x\,dx \qquad (4.4)$$

La aplicación del operador *OP12* según lo indicado en (4.4), conduce a una complicación innecesaria del integrando.

$$\int x\,sin\,(x)\,dx = -\,x\,cos\,(x) + \int cos(x)\,dx \mid OP12:\ u = x,\ dv = sin\,(x)\,dx \qquad (4.5)$$

Sin embargo, aplicando *OP12* según lo indicado en (4.5), conduce a una situación en la que la única expresión que queda por integrar es una integral inmediata. Por lo tanto, debería crearse una regla heurística que señalara la conveniencia de aplicar dicho operador de esta última forma. Aunque no es el objetivo de este capítulo, podemos anticipar que la expresión que tienen dichas reglas heurísticas permiten su utilización en situaciones análogas. Siguiendo con el ejemplo previo, también se podría haber aplicado a la expresión:

$$\int x\,cos\,(x)\,dx = x\,sin\,(x) + \int sin\,(x)\,dx \mid OP12:\ u = x,\ dv = cos\,(x)\,dx \qquad (4.6)$$

Bajo estas condiciones es razonable que se "premie" la utilización de las reglas heurísticas en el proceso de búsqueda. La "bondad" de una solución se mide en función del coste —tiempo de cálculo— empleado en su obtención. Por lo tanto, suponiendo que el proceso busca obtener la solución de mínimo coste (recordar el *procedimiento de coste uniforme* señalado anteriormente) y teniendo en cuenta que el coste real de un nodo (calculado por la función *coste_real*(*n*)) es la suma de los tiempos reales de cálculo consumidos en cada paso a lo largo del camino que conduce de la raíz al nodo, se plantea la reducción del coste del camino cuando sean aplicadas las reglas heurísticas disponibles.

De esta forma, cuando una regla heurística se aplica en el proceso de búsqueda de la solución, el coste del camino recorrido desde la raíz al nodo en el cual se realice dicha aplicación se ve reducido en un 33%. Es decir:

$$coste(n) = coste\_real(n) \times 0.33$$

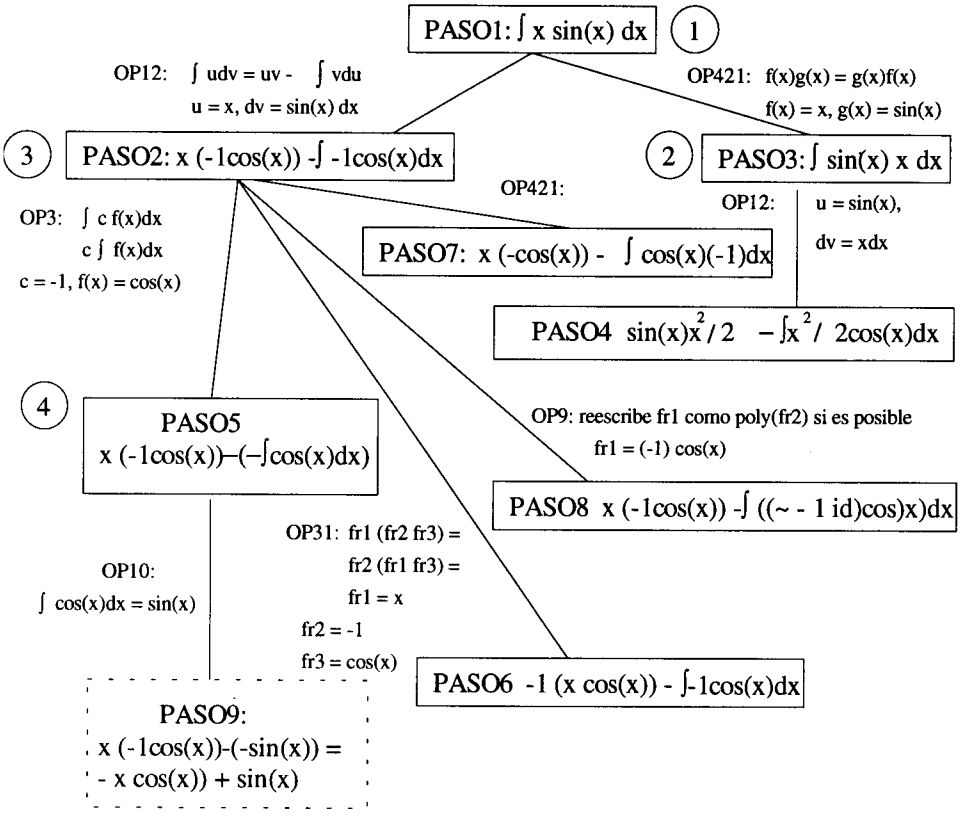


Fig. 4.4. Grafo de búsqueda *sin heurísticas* para un problema de integración simbólica

El proceso de búsqueda de la solución de coste mínimo, sin tener en cuenta las reglas heurísticas disponibles, aparece en la Fig. 4.4. La secuencia de nodos expandidos se indica mediante los círculos en los que aparece el orden dentro de la misma. Comparando dicha secuencia con las mostradas en el capítulo previo, se observa que se parece bastante a la de la figura 3.7. Realmente, sin la reducción de los costes correspondientes a las heurísticas aplicadas, el algoritmo realizado es exactamente un procedimiento de *búsqueda en amplitud* en el que el parámetro *longitud del camino* se ha cambiado por *coste del camino* (es decir, un *procedimiento de coste uniforme*).

Por el contrario, si se habilita la utilización de las heurísticas existentes (entre las que se encuentran las correspondientes al operador *OP12* y al operador *OP3*), se obtiene un proceso de búsqueda —que en este problema concreto resulta ser óptimo— de la solución buscada como se muestra en la siguiente ilustración (ver Fig. 4.5.).

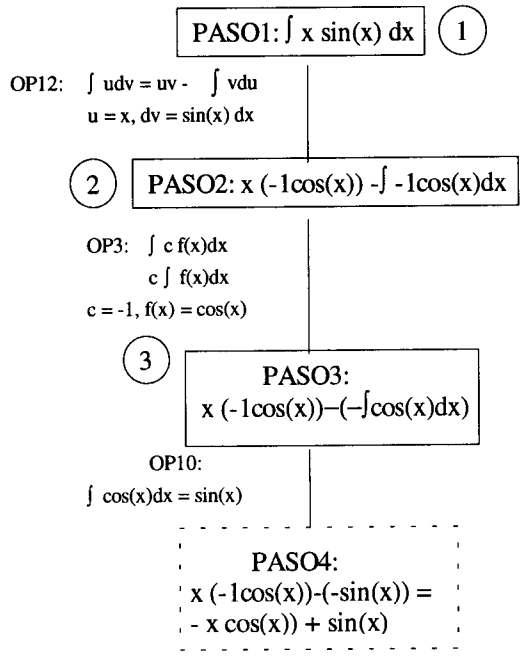


Fig. 4.5. Grafo de búsqueda *con heurísticas* para un problema de integración simbólica

Una de las heurísticas utilizadas define las condiciones de aplicabilidad del operador OP3:

H2-UTILIZA-OP3:

MG: $\int c\,f(x)dx \Rightarrow c\int f(x)dx$

MP: $\int -1\cos(x)dx \Rightarrow -1\int \cos(x)dx$

Esta heurística es la primera versión del conocimiento aprendido (el sistema aprende la definición correcta de las heurísticas a la vez que se van resolviendo los problemas planteados; ver capítulo 9, apartado 10.3.1). MG representa la situación más general en que dicho operador es conveniente y MP el caso más específico; cualquier estado cuya descripción esté comprendida entre

MG y MP será objeto de la aplicación de dicha heurística. *H2-UTILIZA-OP3* es el resultado de haber analizado exclusivamente el problema ilustrado en la gráfica (Fig. 4.4).

La solución encontrada mediante este método de búsqueda heurística no siempre es óptima ya que depende de las heurísticas disponibles. Además existen otras consideraciones que no se han tenido en cuenta. Si el único objetivo es encontrar una solución de menor coste —considerando los elementos implicados en el planteamiento hasta ahora descrito—, habría que reconsiderar la reducción igualitaria de éste por cada regla heurística aplicada. Más correcto sería una reducción equitativa del coste, ya que las reglas heurísticas dependen de los problemas resueltos hasta el momento. Por lo tanto, no todas las reglas tienen el mismo valor. Habrá algunas más fiables que otras[3].

4.3.4 Método AO*

• **Descripción:** este procedimiento es una generalización del algoritmo *A** para el caso de grafos *Y/O*. Sigue cumpliendo la propiedad de ser un algoritmo admisible. Es decir, encuentra una solución óptima siempre y cuando la función heurística *h* de estimación de la distancia al objetivo siga un criterio optimista de valoración (devuelva un valor menor o igual que la distancia real existente).

Todos los aspectos vistos hasta ahora en esta sección para la búsqueda heurística en grafos *O* deben generalizarse para que el objetivo del proceso de búsqueda de la solución intente encontrar un grafo óptimo (o uno suficientemente bueno), en lugar de un camino óptimo. Para obtener un grafo solución de coste mínimo hay que tener en cuenta que su coste depende de la expansión de los nodos que lo forman. Para calcularlo es necesario distinguir los enlaces del tipo *O* de los enlaces del tipo *Y*, ya que en estos últimos hay que sumar el coste de la obtención de cada uno de los nodos conectados por el enlace.

Suponiendo un coste uniforme de valor 1 asociado a cada arco del árbol, el grafo de menor coste desde la raíz será aquél en que la suma de los enlaces que lo forman sea menor (recordar que un enlace Y está constituido por todos los arcos conectados a través de dicho enlace; ver el capítulo 3, apartado 3.5). Por ejemplo, en la siguiente ilustración (ver Fig. 4.6), el nodo que finalmente

[3] Estos y algunos otros asuntos relacionados serán estudiados más ampliamente en el capítulo dedicado al *aprendizaje* (capítulo 10).

tiene el valor 14 es el formado por el enlace Y en el que hay conectados dos nodos de valores 9 y 3 respectivamente, a los que habrá que sumar el coste supuesto de sus correspondientes arcos, que en total suman 14, siendo este valor menor que 16, que sería el encontrado siguiendo enlace Y correspondiente al subárbol de la derecha. Una vez se han obtenido nuevos valores de un nodo (por asignación del valor de la función heurística o por haberse calculado el valor de sus descendientes) se comprueba si se debe actualizar el valor de algún antecesor suyo (ver el procedimiento *AO** más adelante). De esta forma la raíz pasa a tener finalmente el valor de 15.

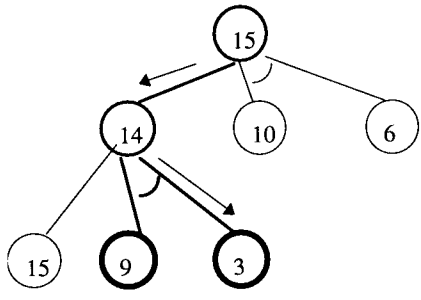


Fig. 4.6. Cálculo de costes en grafos Y/O

Por tanto, la solución ya no es un único nodo meta, sino que puede ser una colección de éstos, que se corresponden con cada uno de los subproblemas en los que puede haberse dividido el problema inicial (raíz del grafo).

En lugar de considerar estados para ser explorados, ahora habrá que trabajar con grafos (en realidad *subgrafos* con respecto al grafo de búsqueda completo). Por ello, en lugar de trabajar con estructuras de datos para estados, habrá que trabajar con estructuras que manejen directamente grafos de búsqueda. Los grafos que tengan nodos que puedan ser expandidos serán aquellos que deberán ser estudiados como candidatos para ser explorados en el proceso de búsqueda. En otras palabras, para avanzar en el proceso de búsqueda se expanden los nodos todavía no expandidos (nodos en la *frontera* del grafo parcialmente desarrollado).

En el procedimiento de búsqueda del grafo de la solución, cada vez que se genere un nodo o un grupo de éstos, el algoritmo debe comprobar si sus antecesores se han convertido en nodos resueltos. Esta circunstancia se produce debido a que el valor inicial asignado a dichos antecesores era el resultado de la

estimación realizada por aplicación de la función heurística h, como queda reflejado en el procedimiento siguiente.

La variable *coste-máximo* refleja el mayor valor que puede llegar a tener el coste del cálculo de una determinada solución.

- **Procedimiento AO*:**

1. Crear un grafo G que en un principio está formado exclusivamente por el nodo raíz m. Estimar la distancia heurística a la meta de ésta mediante $h(m)$.

2. Realizar hasta que m se considere *resuelto* o hasta que su coste supere un cierto valor *coste-máximo* (incluye el caso de que no queden más sucesores por obtener; ver paso (2)).

(1) Seguir los enlaces marcados que señalan el mejor grafo parcial de la solución obtenido hasta el momento hasta alcanzar los extremos de dicho grafo. Escoger alguno de dichos extremos que llamaremos n.

(2) Expandir n generando todos sus sucesores s. Si no tiene ninguno, asignarle el valor *coste-máximo*.

(2.1.) $\forall s$: Si s es un nodo terminal marcarlo como *resuelto* asignarle $h(s) = 0$; en caso contrario asignarle $h(s)$.

(3) Crear un conjunto S sólo con los nodos s.

(4) Realizar hasta que S se vacíe.

(4.1.) Sacar de S algún nodo s' que no tenga descendientes en S.

(4.2.) Actualizar el coste de s':

- Calcular el coste de todos los enlaces que parten de s' y asignarle como $h(s')$ el menor de éstos marcando dicho enlace (y borrando si existiera una marca de otro enlace previamente elegido saliente de s'). Si todos los nodos del enlace marcado son terminales, marcar también s' como *resuelto*.

(4.3.) Si (4.2.) ha hecho que s' sea resuelto o si su $h(s')$ ha cambiado, añadir todos sus antecesores a S y continuar el proceso recursivo de actualización de valores de los nodos (que pueden llegar hasta la raíz del grafo G).

Observación:

La clave del algoritmo AO^* está en la búsqueda (hacia adelante) de formas de expandir el mejor grafo parcial de la solución encontrado hasta el momento; para lo cual, se expanden los nodos *frontera* del mismo y se realiza una revisión (hacia atrás) de los valores $h(n)$ de los nodos antecesores de los nodos *frontera* actualizados.

- **Propiedades**

Si se cumplen:

1. $\forall n\ h(n) \leq h^*(n)$

2. Cada vez que se actualiza el coste de $h(n)$ en función de sus sucesores el valor previo de $h(s)$ nunca supera al calculado en función de aquéllos

entonces el algoritmo es *admisible* y *encuentra el camino óptimo.*

4.4 BÚSQUEDA CON ADVERSARIOS

Los juegos son un área especialmente adecuada para abstraer los elementos dependientes del dominio de aplicación y poder centrarse en los mecanismos que guían el comportamiento. La teoría de juegos ha sido ampliamente estudiada en diversas áreas. Por ejemplo, en economía se utiliza para representar la interacción entre los distintos agentes que influyen en la evolución del mercado. La gran ventaja de los juegos es que la validez de una decisión puede medirse objetivamente.

En IA se han estudiado principalmente aquellas situaciones en las que participan varios jugadores (generalmente dos) perfectamente informados. Es decir, cada jugador conoce todas las reglas aplicables y la disposición de los elementos implicados en el juego; por ejemplo, la disposición de todas las piezas en un tablero de ajedrez.

Nuestro objetivo es mostrar modelos generales aplicables. Recordemos que en el capítulo 3 introdujimos la conveniencia de la utilización del esquema de *reducción del problema* para resolver este tipo de situaciones. Vimos como la representación más adecuada son los llamados *grafos Y/O* (o *árboles Y/O*, dependiendo de la complejidad del problema). Éstos tienen la particularidad de poder tener *enlaces Y*, que representan la relación existente entre un problema y los subproblemas en los que éste puede descomponerse, y *enlaces O*, que representan las distintas alternativas para resolverlo.

En los problemas de juegos, los *enlaces O* se utilizan para representar los distintos movimientos realizables en un instante dado; alternativamente, los *enlaces Y* son utilizados para señalar los movimientos del adversario. Los dos jugadores participan por turnos en el juego. Las jugadas no se deciden sólo en función de la evaluación de un único estado, sino que se tiene en cuenta que éste no es más que uno dentro del conjunto de los posibles. El resultado de cada movimiento sitúa a cada jugador en un nuevo *conjunto de estados*. Esta forma de interpretar el árbol que representa el juego, permite considerar simultáneamente todas las respuestas posibles ante una jugada seleccionada. Por tanto, cada nodo, además de ser un estado del juego, representa una clase de respuestas alternadas a las acciones del contrario. El árbol obtenido es una representación explícita de todas las posibles partidas dentro del juego.

Los *nodos hojas* representan las tres situaciones posibles: ganar, perder o empatar la partida representada por la rama que conduce de la raíz a dicho nodo.

El objetivo de los contrincantes es ganar la partida. El número de movimientos posibles que deben considerarse para garantizar una estrategia ganadora es generalmente *intratable*. Esto es debido a que para cada jugada propia deben estudiarse todas las del oponente. Surge por lo tanto el problema de la *explosión combinatoria* en la mayoría de los problemas (10^{120} en el juego del ajedrez, 10^{40} en el juego de las damas [Samuel, 1959]). Esta dificultad hace que se plantee la necesidad de aplicar **funciones de evaluación heurística** que evalúen la ventaja relativa que supone el alcanzar una determinada situación.

Dada la imposibilidad real de generar el árbol que represente todas las jugadas posibles, se realiza una simplificación del proceso en la que se determina el estado (ganador, empate o perdedor) de los nodos hasta un determinado **límite de profundidad** en el que el valor del nodo es precisamente el valor devuelto por la aplicación de la función de evaluación heurística utilizada. Llamaremos al nivel más profundo analizado *frontera de exploración* o simplemente *frontera*.

Para que el problema converja a la situación deseada, las funciones de evaluación aplicadas deben medir los parámetros más relevantes observables en una determinada situación. Es más, su valor se toma como si de verdad fueran los resultados reales y no como estimaciones heurísticas. El principal escollo que tiene la definición de estas funciones en este contexto es que deben determinar la "deseabilidad" de un estado, basándose en una valoración estática de las condiciones del problema, en un entorno donde prima la variabilidad de la situación a lo largo del tiempo. Es por ello que se estudia hasta un determinado nivel las distintas jugadas posibles de los distintos adversarios, de tal forma que el resultado de la valoración estática (aplicación de *fev*) se vea soportado por un análisis dinámico del juego (exploración del árbol).

A continuación explicaremos más detenidamente todas las cuestiones anteriormente señaladas mediante el análisis de las estrategias básicas aplicadas en este tipo de problemas.

4.4.1 Estrategia MINIMAX

• **Descripción:** dada la imposibilidad práctica de obtener la estrategia ganadora en un juego —aun estando perfectamente informados de las jugadas posibles del adversario—, se plantea la necesidad de establecer métodos generales que permitan aproximarse a dicha estrategia ganadora. MINIMAX es en realidad un esquema genérico de búsqueda para situaciones en las que el objetivo es ganar una partida en la que participan dos adversarios que realizan movimientos alternativos en el juego. La efectividad última de esta estrategia reside en la función de evaluación heurística aplicada, encargada de medir la ventaja relativa de realizar las distintas jugadas.

Básicamente el método consiste en prever el mayor número de jugadas posibles —tanto propias como del otro jugador— y actuar en consecuencia. Para ello se recorre un árbol *Y/O* llevando a cabo una política conservadora de evaluación de las distintas situaciones, es decir, se considera siempre que el adversario va a realizar la mejor de las opciones posibles cuando a él le toque mover. Los nodos del árbol son de dos tipos: nodos *MAX* y nodos *MIN*, cada uno de éstos toma las decisiones uno de los dos contrincantes. El objetivo del jugador *MAX* es maximizar el valor de la función *f* de evaluación heurística que mide la proximidad estimada a la *meta* de una situación dada (recordar que la meta es ganar el juego). Por otro lado, los nodos *MIN* reflejan el modo de actuar del adversario con respecto al objetivo del nodo *MAX*; es decir, realizar aquel movimiento que haga más pequeño el valor de *f*. Debido a la alternancia en el

juego, cada nivel dentro del árbol está formado sólo por nodos *MAX* o sólo por nodos *MIN* (se supone que se comienza en un nodo MAX, por lo que estos ocupan los niveles pares del árbol, recordar que a la raíz se le asigna el nivel 0), siendo el nivel inmediatamente posterior de nodos del signo contrario.

(1) Cada nodo terminal (es decir, aquéllos situados en la frontera de exploración en un momento dado) puede representar el ganar, perder o empatar la partida correspondiente (depende del valor de *f*). Para saber lo que reflejan el resto de los nodos (no terminales), se recorre el árbol en sentido inverso (desde el último nivel explorado hacia atrás), comprobando a cual de las tres situaciones mencionadas se puede llegar desde dicho nodo (siempre suponiendo que el objetivo es que MAX gane la partida). El estado de un nodo **MAX** será **ganador** si alguno (carácter disyuntivo) de sus sucesores conduce a un terminal ganador, será **perdedor** si ningún sucesor conduce a uno ganador (carácter conjuntivo) y se corresponderá con el **empate** si ningún sucesor conduce a la meta y al menos uno conduce al empate. El estado de un nodo **MIN** será **ganador** si todos sus sucesores llegan a un final ganador, será **perdedor** si alguno conduce a la pérdida del juego y será **empate** si ningún sucesor es perdedor y alguno conduce a un empate.

Por lo tanto, una **estrategia ganadora para *MAX*** sería un subárbol en el que todos sus nodos terminales son ganadores, de esta forma se garantiza el éxito sea cual sea la respuesta del adversario (ver subárbol en negrita en Fig. 4.7). En dicha estrategia, cada vez que juega MAX basta escoger una rama en la que todas las jugadas de *MIN* hacen que *MAX* sea el vencedor. Contrariamente, una **estrategia ganadora para *MIN*** será el subárbol en el que todos sus nodos terminales sean *perdedores*, teniendo en cuenta que cada vez que juega *MAX* habrá que considerar todas sus movimientos y cuando juega *MIN* sólo es necesario considerar una de las alternativas.

En la siguiente figura (Fig. 4.7), ganar se representa por un nodo con valor +1, perder con valor −1 y empatar con valor 0. Como puede apreciarse, hay dos formas de etiquetar el árbol de búsqueda: por una lado la que se deriva de lo expresado en el párrafo previo (1), a la que hemos llamado *MAX*, y por otro, una estrategia única (desaparece la distinción entre nodos *MAX* y *MIN*) de asignación de valores a los nodos del árbol, que hemos denominado *MMvalor*(*n*) y que se calcula como sigue:

$$MMvalor(m) = \begin{cases} f(m) \mid m \text{ no tiene sucesores} & (4.7) \\ \max_n \{-MMvalor(n) \mid n \text{ sucesor de } m\} & (4.8) \end{cases}$$

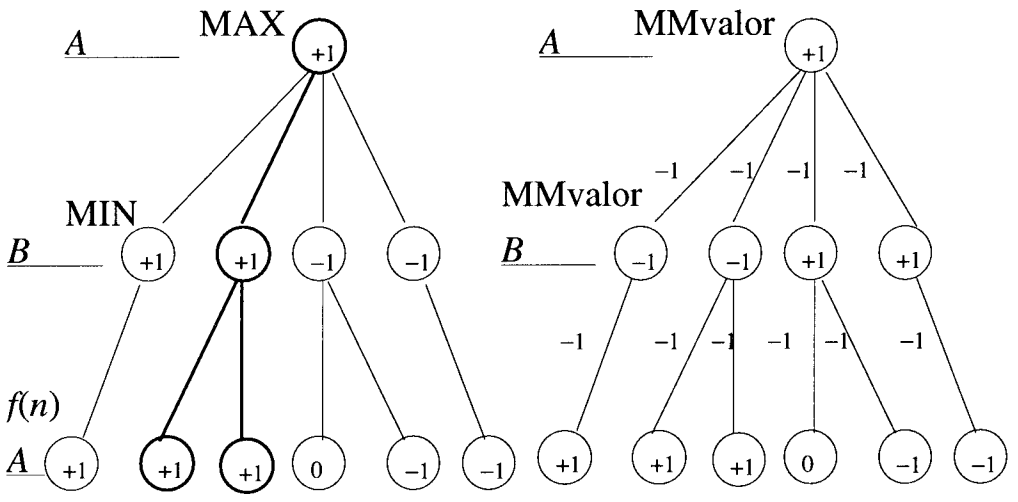


Fig. 4.7. Resultados de la evaluación de nodos en la estrategia MINIMAX

Como ya hemos comentado, *MAX* etiqueta *todos* los niveles del árbol de la figura pensando en ganar él la partida, contrariamente *MMvalor(n)*, como su propio nombre refleja, *depende del nodo n* que en ese momento esté siendo valorado. Por ejemplo, en el nivel *B* del árbol (nivel *MIN*) de la izquierda aparece etiquetado con valor +1 un nodo en el que todos sus sucesores (los que indican el turno de *MAX*) tienen valor +1, ya que todas las opciones de *MIN* hacen que *MAX* gane la partida. Sin embargo, aplicando la fórmula (4.8) se obtiene un valor −1 para ese mismo nodo, ya que ahora se está calculando su valor desde la perspectiva del jugador que le toca mover en dicho nodo. Es decir, para el jugador *MIN* que tiene que decidir en dicho nodo, cualquier rama que elija (hemos dicho que todas tienen valor +1) le lleva a un situación en la que el jugador que mueve a continuación gana la partida, o lo que es lo mismo, en la que él va a perder. Por lo tanto, para MIN dicho nodo tiene un valor −1. Si retrocedemos en el árbol al nivel anterior *A* (en la figura corresponde al nodo raíz) y calculamos *MMvalor(raíz)*, obtendremos el mismo valor que el que ya obtuvimos con el etiquetado del árbol de la izquierda, ya que haciendo consideraciones análogas, ahora es el turno del jugador *MAX*.

La ventaja que tiene el etiquetado de *MMvalor* es que no hay que establecer un procedimiento con reglas específicas para *MAX* y para *MIN*, además proporciona la perspectiva del jugador que se encuentra en cada nivel del árbol.

Realmente, los valores devueltos por la función estática de evaluación heurística f de la gráfica anterior, simplemente +1 o −1, son poco útiles. Normalmente evaluar la conveniencia de una situación supone tener en cuenta una gran diversidad de factores [Samuel, 1959], por lo que no es nada fácil determinar si en una situación dada se puede ganar o perder claramente, sino que más bien se devuelve algún valor numérico dentro de un rango más amplio que indique hasta que punto se puede considerar *ganador* a un estado determinado.

El método MINIMAX se comporta como un procedimiento de *búsqueda con retroceso* (ver apartado 3.4.3) con una frontera de exploración calculada. Es decir, si el proceso llega a un punto en que no hay sucesores o se ha alcanzado un nivel máximo de profundidad para dicho camino, entonces se continua con el antecesor más cercano. El nivel de exploración depende de diversos factores entre los que pueden destacarse: las características del camino recorrido hasta ese momento y el coste computacional de alcanzar dicha profundidad.

El algoritmo que se muestra a continuación es un proceso recursivo de exploración del árbol que refleja el etiquetado realizado por *MMvalor*(n).

- **Procedimiento MINIMAX:**

  MINIMAX(m, *profundidad*, *jugador*)

  1. Si m no tiene sucesores o si se considera que m ha alcanzado el límite de profundidad en *profundidad*, devolver: $mmv(m) = f(m)$

  2. Generar los sucesores de m:

  2.1. Inicializar la variable *mejor* con el valor mínimo que esta pueda tener (puede ser un valor de referencia previamente establecido).

  $$mejor = \min_j \{f(j)\ \forall j\}$$

  2.2. Para cada sucesor n de m:

  (1) $mmv(n) = MINIMAX(n, profundidad + 1, C(jugador))$;

  siendo $C(jugador)$ una función que cambia de jugador.

  (2) $mejor = max[-MMv(n), mejor]$

3. Una vez se han analizado recursivamente todos los sucesores de un determinado nivel (indicado por la variable *profundidad*), se devuelve el que ha obtenido un mejor valor.

mmv = mejor

Observación: a la vez que se devuelven los valores mejores se podría considerar necesario el haber devuelto el camino que pasa por ellos (p.ej., utilizando una variable *camino* a la cual se van añadiendo nodos en el paso (2), cuando haya un cambio de la variable *mejor*). Esto no es realmente necesario, dado que depende de las respuestas del otro jugador, por lo que en la versión más sencilla de este procedimiento basta con saber cual es la siguiente mejor jugada.

- **Complejidad:**

Un árbol de coste uniforme de una profundidad p y de un factor de ramificación n contiene n^p nodos terminales que serán analizados por el procedimiento MINIMAX descrito anteriormente. De todas formas, conviene tener en cuenta las siguientes consideraciones.

Resolver un juego es, al fin y al cabo, encontrar al menos dos estrategias, una para *MAX* y otra para *MIN*, que sean compatibles; es decir, que la intersección de ambas tenga un nodo terminal en común, que será la posición final del juego. Esto ocurre siempre que ambos se ajusten a la estrategia predeterminada.

Dado que una estrategia, por ejemplo para *MAX*, se expande en niveles alternativos del árbol, el número de nodos de una estrategia es alrededor de $n^{p/2}$. Si de antemano se pudiera saber que dos estrategias son compatibles, la complejidad del procedimiento sería $2n^{p/2}$. Desgraciadamente, habrá que realizar un número indeterminado de intentos antes de encontrar dos estrategias compatibles, por lo que dicho límite inferior rara vez es alcanzable.

- **Análisis**

Ventajas: es un método general y sencillo que permite representar y buscar estrategias ganadoras en los problemas de juegos en los que hay dos adversarios. El etiquetado de los nodos en el árbol permite identificar fácilmente las distintas estrategias posibles.

Desventajas: es un método que peca por ser demasiado exhaustivo; es decir, poco eficiente debido a su complejidad. Por ejemplo, si reconsideramos el árbol de la gráfica anterior (ver Fig. 4.7.) y suponiendo que el objetivo es encontrar una estrategia ganadora para el nodo *MAX* en la raíz del árbol, nada más estudiar el primer sucesor (rama de más a la izquierda) se podría haber abandonado el proceso de búsqueda del resto de sus sucesores, ya que al estar en un nodo *MAX* y haber encontrado su valor máximo posible (+1) no hace falta continuar. Este análisis tan simplista sólo sería válido para un árbol como el presentado en la figura anterior. En el siguiente procedimiento veremos una generalización de este tipo de consideraciones para cualquier tipo de árboles de juegos.

4.4.2 Estrategia de Poda α-β

• **Descripción:** es una modificación del procedimiento MINIMAX que permite ahorrarse el recorrido de caminos inútiles del árbol de búsqueda. Dada su efectividad es el método más común en los problemas de juegos.

Esta estrategia abandona el estudio de subárboles que no pueden conducir a soluciones mejores que las ya encontradas en un cierto nodo. Esto permite que se elija el valor de un nodo poco después de empezar el estudio de los subárboles que nacen en él, ahorrándose un tiempo considerable.

Para que el recorrido de un cierto subárbol no sea rechazado de antemano se tienen en cuenta las siguientes consideraciones. Si se está bajo un nodo *MAX*, se tiene que garantizar por lo menos el valor máximo ya alcanzado. Si se está bajo un nodo *MIN* se tiene que cumplir que se está buscando por debajo del valor mínimo encontrado hasta el momento.

Aplicando estos criterios en el ejemplo de la gráfica (ver Fig. 4.8.). Si estamos en un nodo *MIN* (p.ej., el más cercano a la raíz de valor 8) y ya hemos analizado alguno de sus hijos (nodo *MAX*) que ha devuelto un cierto valor (el mencionado 8) que llamaremos β, si en el estudio del siguiente hijo encontramos un nieto con mayor valor que dicho β (en este caso el valor es 9), entonces podemos abandonar el estudio del resto de los nietos provenientes de dicho hijo y por lo tanto de dicho hijo también. La razón es que en el nivel de este segundo hijo (nodo *MAX*) cogeremos como poco el valor encontrado, ya que se busca el máximo, por lo que al devolver dicho valor al nodo padre *MIN* nunca será escogido, ya que previamente existía un valor β inferior.

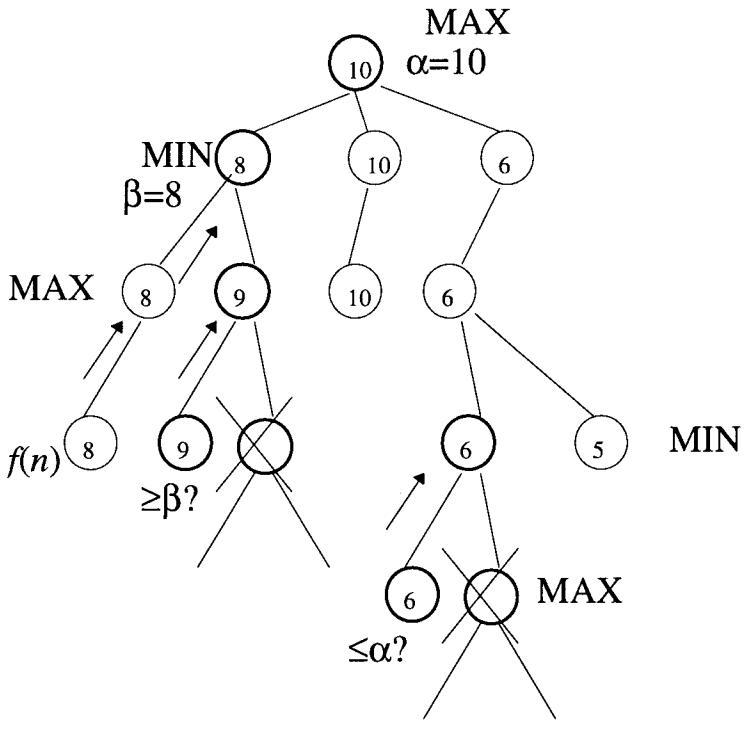


Fig. 4.8. Podas α y β

Según lo mostrado en el ejemplo de la Fig. 4.8. Si estamos en un nodo *MAX* (p.ej., la raíz del árbol) y alguno de sus hijos ya ha devuelto un cierto valor (en este caso tomemos el valor 10) que llamaremos α, si al analizar el subárbol de los demás hijos encontramos que alguno de éstos tiene un nieto (el que finalmente tiene valor 6) que a su vez tiene un hijo (del que obtiene el valor 6) que está ya por debajo de dicho α entonces se puede abandonar el estudio de del resto de sus hijos y olvidarse también del valor devuelto por él mismo. La razón es que al ser el biznieto de la raíz un nodo *MIN*, dicho valor será devuelto a través de su padre (que como poco tendrá dicho valor ya que es un nodo *MAX*) hasta la raíz que es un nodo *MAX* y que ya tenía el valor superior α indicado (valor 10).

Los valores de umbral inferior para nodos *MAX* y de cotas superiores nodos *MIN* se propagan hacia abajo en el proceso recursivo de exploración del árbol. Por lo tanto, sólo pueden afectar al tomar una decisión en un cierto nivel, el umbral y la cota de cualquiera de sus antecesores.

Expresado en forma operativa (ver el siguiente procedimiento) **α es el valor mínimo que debe tener un nodo MIN** para que éste siga siendo explorado. Este valor es igual al mayor valor encontrado hasta el momento de todos sus antecesores MAX. Por otro lado, **β es el valor es el valor máximo que debe tener un nodo MAX** para que este siga siendo explorado. Este valor es igual al valor más pequeño encontrado de todos sus antecesores MIN.

- **Procedimiento de Poda α-β:**

  alfabeta(*m*, *profundidad*, *jugador*, *limite-actual*, *límite-siguientes*)

  1. Si *m* no tiene sucesores o si se considera que *m* ha alcanzado el límite de profundidad en *profundidad*, devolver: $\alpha\beta v(m) = f(m)$

  2. Generar los sucesores de *m*:

  2.1. Para cada sucesor *n* de *m*:

  (1) $\alpha\beta v(n)$ = *alfabeta*(*n*, *profundidad* + *1*, *C*(*jugador*), $-$*límite-siguientes*, $-$*límite-actual*,);

  siendo *C*(*jugador*) una función que cambia de jugador.

  (2) *límite-siguientes* = *max*[$-\alpha\beta v(n)$, *límite-siguientes*]

  (3) Si *límite-siguientes* $\geq$ *límite-actual* entonces abandonar este nivel y devolver *límite-siguientes*.

  3. Una vez se han analizado recursivamente todos los sucesores, se devuelve el que ha obtenido un mejor valor.

  $\alpha\beta v(m)$ = *límite-siguientes*

  Observación: la primera llamada al procedimiento se realiza tomando como α el valor máximo que podría tener $f(n)$ y como valor β el valor mínimo de la misma. Una vez más hay que decir en todos los pasos en que se devuelvan valores nuevos (pasos 1. y 3.) para el resto de los niveles, puede construirse el camino mejor añadiendo a una cierta variable (p.ej., camino-recorrido) el nuevo nodo encontrado.

La idea básica que guía el proceso recursivo de búsqueda es seguir explorando un determinado camino siempre y cuando se mantenga dentro de los límites $\alpha < \alpha\beta v < \beta$.

Como hemos eliminado la distinción entre niveles MAX y MIN, en lugar de utilizar α y β utilizamos *límite-actual* y *límite-siguientes*. El segundo sirve para determinar cuál es el mejor valor que deben considerar los descendientes en el árbol y el primero es para comprobar que en el nodo actual no se viola la restricción impuesta por un límite de un nivel superior. En otras palabras, dentro de un nivel, la variable *límite-siguientes* refleja el mejor valor encontrado hasta el momento en dicho nivel y si dicho valor no supera el valor ya alcanzado por niveles superiores reflejado en la variable *límite-actual*, entonces el nivel debe abandonarse ya que no supera el valor por el cual se está realizando la presente búsqueda —paso (3) del procedimiento—.

- **Complejidad:**

Si suponemos que el árbol de búsqueda está perfectamente ordenado (es decir, si en las ramas de más a la izquierda se obtienen los mejores valores de α y de β para así discriminar el resto de los caminos) entonces el número de nodos que son evaluados es (partiendo de que la longitud del árbol es "p" y el factor de ramificación "n"):

1. Si p es un número par: $2n^{p/2} - 1$

2. Si p es un número impar: $n^{(p+1/2)} + n^{(p-1)/2} - 1$

En el peor de los casos habría que examinar todos los nodos terminales y tendría la misma complejidad que ya se ha comentado en el procedimiento MINIMAX.

- **Análisis**

Ventajas: es un método eficiente que evita el recorrido exhaustivo de un árbol de búsqueda para una situación en la que se enfrentan dos adversarios. Reduce considerablemente el número de nodos expandidos y permite afrontar problemas supuestamente intratables.

Desventajas: a pesar de la mejora realizada con respecto al método básico MINIMAX, tiene el principal inconveniente de la dependencia excesiva del ordenamiento realizado de los nodos del árbol.

Además del coste se pueden aplicar otros criterios heurísticos (independientes como caminos que supongan muy poca mejora o dependientes

del dominio de aplicación) que pueden ayudar a discriminar caminos *explorables* en el árbol.

4.5 ANÁLISIS DE MEDIOS-FINES

Esta es la estrategia de búsqueda más general en la resolución de problemas según el esquema planteado en la tabla 3.1. Como su propio nombre indica, consiste básicamente en distinguir cuáles son los *medios* disponibles en dicha formulación, es decir, los operadores, y cuáles los *fines*, es decir, los que hemos llamado estados *meta*. Este método se definió inicialmente en el contexto del llamado **Solucionador General de Problemas** (en inglés, *General Problem Solver*, GPS [Newell *et al.*, 1960]), así denominado por ser capaz de plantear de una forma genérica la solución de problemas según el modelo mencionado.

El planteamiento es bastante intuitivo y responde a modelos de actuación provenientes del campo de la *psicología cognitiva*. Básicamente consiste en analizar constantemente las diferencias que hay entre la descripción inicial del problema planteado y la meta deseada. Una vez aquéllas son detectadas, se accede a una tabla en la que aparecen ordenadas en función de su importancia en el dominio. En cada fila de la tabla se indica cuáles son los operadores disponibles para reducir la diferencia correspondiente a dicha fila. Una vez se ha seleccionado un operador para llevar a cabo la reducción, puede ocurrir (sobre todo si acabamos de iniciar el proceso de reducciones) que dicho operador no se pueda aplicar directamente en la situación inicial; en cuyo caso, se estable como *submeta* el alcanzar un estado en que dicho operador sea aplicable. Por otro lado, la aplicación de dicho operador garantiza la reducción de algunas diferencias, pero no todas; esto supone tener que iniciar otro proceso de reducción de diferencias desde el estado que se haya producido por aplicación de dicho operador.

El proceso consiste en establecer primero las diferencias consideradas más significativas en el dominio de aplicación y luego ir reduciéndolas sucesivamente en función de la información disponible sobre los operadores existentes.

Si analizamos la explicación anterior podemos sacar una serie de consecuencias:

1. Es necesario establecer claramente cuales son las condiciones previas, a las que llamaremos *precondiciones*, que permiten aplicar cada uno de los operadores del sistema.

2. También deben ser fácilmente accesibles los resultados causados por la utilización de cada uno de los operadores.

3. La tabla de diferencias depende fuertemente del dominio de aplicación.

4. El recorrido realizado del grafo de búsqueda de la solución es una combinación de *búsqueda con retroceso*, para resolver las diferencias, y para encontrar estados en los que los operadores que resuelven las diferencias puedan ser aplicados, y de *encadenamiento hacia adelante*, cada vez que se aplica un operador y se alcanza un nuevo estado que vuelve a ser comparado con la *meta*.

La solución efectiva a este problema consiste en tener dos tablas: una de operadores, con dos columnas claramente diferenciadas: la de *precondiciones* y la de *resultados* (en otros capítulos de este libro también denominados *acciones*) y otra, la de las diferencias ya mencionadas.

Desde el punto de vista metodológico lo más interesante de este método es el poder diferenciar claramente en un problema el conocimiento dependiente del dominio: los operadores, las metas y las diferencias, del conocimiento independiente relativo al funcionamiento del *solucionador*, es decir, de la forma efectiva de llevar a cabo la reducción sucesiva de las diferencias encontradas.

Una variación especialmente significativa de este método denominada STRIPPS [Fikes & Nilsson, 1971], se ha aplicado en la resolución de problemas en el campo de la robótica. La variación más notable con respecto a GPS es que en lugar de tener una tabla ordenada de diferencias, dicho concepto está implícito en la propia definición de los operadores. De esta cada operador, en lugar de tener *precondiciones* y *resultados*, pasan a estar formados por dos listas: *adiciones*, con los elementos introducidos como resultado de la aplicación del operador, y *eliminaciones*, con el conjunto de condiciones que dejan de cumplirse una vez se haya aplicado dicho operador. Las diferencias son ahora los elementos del estado *meta* o de las *precondiciones* del operador que no son satisfechos por el estado del cual se parta.

4.6 COMENTARIO

En este capítulo hemos introducido métodos que permiten utilizar el conocimiento disponible del dominio para llevar a cabo una búsqueda eficiente de la solución. Al final del capítulo 3 señalamos la importancia que tenía el dominio en el problema de la representación. Una vez estudiados los métodos de búsqueda heurística, podemos afirmar con mayor motivo, que la utilización

efectiva del conocimiento relevante del dominio es una condición básica para la solución de los problemas de IA. Esta argumentación se verá apoyada a lo largo del resto de los capítulos de este libro, según se vayan presentando los diferentes mecanismos existentes para resolver los problemas de representación e inferencia en cualquier sistema.

5 LÓGICA

J. G. Boticario

En los capítulos previos hemos abordado fundamentalmente cuestiones de tipo metodológico. De ahora en adelante vamos a tratar cuestiones más relacionadas con la construcción de sistemas. Empezaremos describiendo cada una de las principales técnicas de representación del conocimiento. En este capítulo nos centraremos en la utilización de la lógica.

Como ya se ha señalado al principio de este libro, un objetivo de la IA es hacer computable el conocimiento no analítico. Quizás la forma más intuitiva de afrontar dicho problema es analizar el medio más utilizado para expresarlo; es decir, el lenguaje. La lógica tiene como objeto de estudio los procesos de razonamiento expresados a través del lenguaje.

La lógica como ciencia ha pasado por diversas etapas. En un principio —desde su origen en los tiempos de Aristóteles (denominada lógica clásica)— se dedicó principalmente a la identificación de las formas de razonamiento humano. En una segunda etapa se definieron un conjunto de cálculos formales, sobre todo a raíz de la formulación matemática de Boole y Fregge inspirada en el planteamiento de Leibniz de formalizar la lógica como un cálculo para manipular ideas, que finalmente culminó en los Principia Mathematica de Whitehead y Russell en 1913. El estudio de las limitaciones del cálculo (Gödel en 1936 demostró la incompletitud del enfoque axiomático), produjo un cierto desencanto hasta que la aparición de los ordenadores, junto con el abandono de los objetivos de formulación universal de la matemática, ha incentivado nuevos avances en las formulaciones existentes. De las que destacamos —dados los objetivos del presente libro— la regla universal de resolución

[Robinson, 1965] y el tratamiento de la incertidumbre realizado en la «lógica difusa o borrosa»[1] *[Zadeh, 1978].*

Vamos a presentar en este capítulo la lógica no como una ciencia[2]*, sino como un formalismo básico de representación e inferencia en los problemas de IA.*

5.1 EL USO DE LA LÓGICA EN LA REPRESENTACIÓN DEL CONOCIMIENTO

Ya hemos comentado en los capítulos dedicados a la búsqueda que el conocimiento del dominio juega un papel determinante en la resolución de problemas. El formalismo elegido para su representación ha sido una de las principales áreas de investigación dentro de la IA. La riqueza expresiva de los formalismos de representación del conocimiento y su realización computacional determinan la *eficiencia* y la *corrección* de las soluciones obtenidas.

Las tres formas más generales de representar el conocimiento en IA son las estructuras orientadas a la representación de los objetos/situaciones mediante sus características (fundamentalmente los llamados *marcos*, ver capítulo 7), las fórmulas lógicas y las reglas de producción (capítulo 6). La combinación de estas técnicas en lugar de la utilización exclusiva de una de ellas, es la opción que se ha mostrado más efectiva hasta el momento en la solución de problemas en IA. Es evidente, por ejemplo, que no todo el razonamiento humano es de tipo lógico.

Para valorar la conveniencia de un método se debe tener en cuenta dos capacidades del mismo: la de representar todo el conocimiento requerido y la de disponer de un método de obtención de las conclusiones requeridas de dicho conocimiento (también llamada capacidad de *razonamiento* o capacidad de *inferencia*). La utilización de la lógica como formalismo de representación del conocimiento (de ahora en adelante RC) cubre ambos aspectos. Por un lado supone que los lenguajes lógicos se utilizarán como esquema de representación y por otro los problemas se resolverán siguiendo el principio de deducción lógica.

---

[1] Como se comentará en el capítulo 7, consideramos más preciso denominar a este tipo de cálculo *lógica difusa*, por tanto, de ahora en adelante utilizaremos este término.

[2] Para profundizar en los cálculos lógicos básicos sugerimos la lectura de libros dedicados enteramente a su explicación [Cuena, 1986; Fdez. & Sáez, 1987; Deaño, 1991; Garrido, 1992; Aranda *et al.*, 1993].

La utilización de la lógica en la RC tiene una serie de ventajas. Los diferentes lenguajes lógicos tienen una gran riqueza expresiva que iremos descubriendo a lo largo de este capítulo. Por otro lado, al contrario de muchos otros formalismos, la lógica tiene una semántica perfectamente definida; dada una expresión y una interpretación de la misma, se puede determinar inequívocamente su significado. Dicho de otra forma, el valor de verdad de la expresión se obtiene asignándoles valores de verdad concretos a los componentes de la misma (en principio, verdadero y falso, aunque iremos matizando esta bivalencia a lo largo del capítulo). Además, es un formalismo que tiene una serie de propiedades formales claramente especificadas. Aunque veremos las limitaciones de algunas de éstas (p.ej., la *semidecibilidad* del cálculo de predicados), lo importante —con respecto a otros formalismos— es que estas propiedades son conocidas.

Vamos a presentar las ventajas y las limitaciones de la utilización de los lenguajes lógicos más comúnmente utilizados en IA. Para ello iremos analizando, siguiendo un grado creciente de complejidad, la capacidad expresiva de los distintos formalismos.

5.2 LÓGICA DE PROPOSICIONES

El conocimiento disponible, o bien es el resultado de la declaración de hechos o ideas, o bien se ha generado mediante algún proceso de **inferencia**: consecución de nuevos elementos de conocimiento a partir de otros siguiendo un proceso mental de naturaleza deductiva (en el capítulo 10 veremos formalismos lógicos orientados a los procesos de naturaleza inductiva). Un razonamiento de este tipo es el siguiente:

La lógica es un formalismo de representación (5.1)

Si la lógica es un formalismo de representación **entonces** la lógica de proposiciones también lo es (5.2)

La lógica de proposiciones es un formalismo de representación (5.3)

La lógica de enunciados (también llamada de *proposiciones* o *cálculo proposicional*) permite estudiar la validez formal de los razonamientos realizados mediante frases formadas por enunciados simples. Estas unidades mínimas de información también se conocen con el nombre de **proposiciones**. Las proposiciones tienen un significado sobre el que es posible pronunciarse. El

significado puede ser verdadero ("V") o falso ("F"), pero no ambos simultáneamente. Más adelante mostraremos frases difusas que no pueden ser analizadas correctamente en este formalismo (p.ej., "ser alto" no es exactamente cierto o falso, depende de la medida de referencia considerada).

No vamos a describir exhaustivamente todos los elementos (símbolos elementales + reglas de formación + reglas de transformación) que hacen que la lógica de proposiciones constituya un cálculo, vamos sin embargo a recordar algunos de ellos para entender cuáles son los aspectos del conocimiento que pueden manipularse adecuadamente en dicho cálculo.

Los símbolos básicos utilizados para representar los enunciados simples son las llamadas *variables proposiciones* (p, q, r, etc.). El razonamiento del ejemplo anterior se puede representar como:

$$1.\ p \tag{5.4}$$

$$2.\ p \rightarrow q \tag{5.5}$$

$$3.\ q \tag{5.6}$$

Esto representa una forma abstracta de plantear el razonamiento prescindiendo de su contenido, lo cual nos permite centrarnos en la forma que debe tener dicho proceso deductivo para que otros razonamientos análogos sean igualmente válidos. Es decir, la lógica nos proporciona el conjunto de símbolos y reglas para plantear procesos válidos de razonamiento independientemente de la interpretación concreta que dichos símbolos tengan en el mundo real. Un esquema de razonamiento con la misma representación de la estructura deductiva reflejada en (5.4), (5.5) y (5.6) es el siguiente:

Se ha detectado fuego (5.7)

Cuando se detecte fuego, activar la alarma de incendios (5.8)

Activar la alarma de incendios (5.9)

Los enunciados en los que aparecen varias proposiciones (enunciados compuestos) se representan enlazando las variables proposicionales mediante las llamadas *conectivas*. Este es el caso de la conectiva "$\rightarrow$", llamada *condicional o implicación material*, que aparece en (5.2), (5.5) y (5.8). El resto de las conectivas básicas son: "$\neg$" (no), "$\wedge$" (y), "$\vee$" (o).

Cada conectiva representa varias expresiones del lenguaje natural. Así por ejemplo la expresión "$p \vee q$" indica p o q o ambas a la vez. Para el caso de la "o" exclusiva se utiliza el símbolo "$\oplus$". La conjunción "$p \wedge q$" sirve para representar expresiones como: "p y q", "p sin embargo q", "p a pesar de q", ...etc. La expresión del condicional "$p \rightarrow q$" se puede asociar a "si p entonces q", "p sólo si q", "q si p", "p es suficiente para q", "q necesario para p", ...etc. También se puede utilizar la expresión "$p \leftrightarrow q$" para representar "p si y sólo si q". Las expresiones así formadas también se conocen como fórmulas correctamente formadas (abreviadamente *fcf*).

Ya hemos señalado que una de las ventajas del cálculo lógico es que puede obtenerse el significado de una determinada expresión dentro del formalismo. En principio (ya hemos anticipado que más adelante matizaremos esta afirmación) un enunciado descriptivo puede ser verdadero (representado con el valor "V" o "1") o falso (valor "F" o "0"). De esta forma, para una expresión con n variables existen 2^n combinaciones posibles de valores de verdad.

Todas las expresiones con conectivas tienen una forma preestablecida de calcular su valor de verdad. El caso más peculiar es el de la *implicación material*, que no se corresponde —en general— con el valor causal asignado a la *implicación* en el lenguaje común. Los valores de verdad de cualquier expresión se agrupan en las llamadas *tablas de verdad* como la mostrada en la Tabla 5.1.

Tabla 5.1.

p	q	$p \rightarrow q$
V	V	V
V	F	F
F	V	V
F	F	V

Los valores de verdad indican que el condicional es cierto siempre que q (también llamado *consecuente*) lo sea y siempre que p (también llamado *antecedente* sea falso. Por lo tanto tiene la misma interpretación que la expresión "$\neg p \vee q$". Teniendo en cuenta las dos últimas filas de la tabla 5.1., podría afirmarse que del hecho de que "Madrid sea la capital de Bélgica" —lo cual es evidentemente falso— se puede implicar materialmente cualquier cosa. Cuando todas las combinaciones de valores de verdad de cada una de las proposiciones hacen que la expresión formada en base a ellas sea siempre verdad se dice que dicha expresión es una *tautología* (p.ej., "$p \vee \neg p$"). Por el contrario siempre que sean falsos se habla de una *contradicción* (p.ej., "$p \wedge \neg p$").

Una fórmula cualquiera puede ser: **válida** (cierta en todas las interpretaciones posibles), **satisfacible** (cierta en al menos una interpretación), o

insatisfacible (falsa en todas las interpretaciones). Por lo tanto, según lo dicho anteriormente las *tautologías* son *fórmulas válidas* y las *contradicciones* son *fórmulas insatisfacibles.*

Recordemos que la lógica tiene como principal objetivo el estudio de las argumentaciones formalmente válidas. En otras palabras, se intenta formular un procedimiento de razonamiento abstracto que permita obtener conclusiones a partir de los enunciados iniciales (llamados *premisas*). De esta forma, una *demostración en el sistema axiomático* consiste en, dadas una serie de fórmulas válidas de las cuales se parte (llamadas *axiomas*), obtener un conjunto de nuevas fórmulas igualmente válidas aplicando para ello las denominadas *reglas de transformación.* Este proceso se realiza sucesivamente hasta que se genere la conclusión buscada. La regla de transformación básica de los procesos deductivos de la lógica clásica es el llamado *modus ponens*, que es la que se ha aplicado en el proceso deductivo mostrado en (5.4), (5.5) y (5.6). Esta regla expresa lo siguiente: "Si S y $S \rightarrow R$ son fórmulas válidas entonces R también es una fórmula válida". Todo razonamiento tiene —o puede reducirse a— la forma de un condicional; recordar (5.2), uno de los primeros enunciados vistos en este apartado.

Modus Ponens:

$$\frac{\vdash S,\ \vdash S \rightarrow R}{\vdash R} \qquad (\vdash, \text{símbolo de fórmula válida}) \qquad (5.10)$$

Los axiomas lógicos son fórmulas o esquemas de fórmulas válidas tautológicas, que se han elegido por ser especialmente adecuados para deducir el resto de las fórmulas válidas del cálculo.

Además del *modus ponens*, existen otra serie de reglas que expresan las distintas formas válidas del razonamiento. Estas reglas muestran los procesos de razonamiento del lenguaje ordinario, por lo que también se conocen como el método de **deducción natural** [Gentzen, 1930]. La diferencia con el **método axiomático** comentado hasta ahora está en que las inferencias aplicadas para poder avanzar en las secuencias deductivas de éste requerían que tanto las fórmulas de las que se partía, como aquéllas que se generaban, fueran enunciados formalmente válidos, es decir, verdaderos en todos los casos posibles. Sin embargo, en deducción natural los enunciados están indeterminados en su valor de verdad. Éste es un supuesto mucho más acorde con la forma en que normalmente razonamos los seres humanos (donde muchas

veces razonamos partiendo de premisas falsas). Es decir, las reglas de inferencia aplicadas sólo garantizan la veracidad de las conclusiones obtenidas cuando las premisas también sean verdaderas, pero dichas conclusiones no tienen por qué ser verdaderas por sí mismas. Ambos métodos son equivalentes en el sentido de que en ambos son correctas las mismas estructuras deductivas. En *deducción natural* existe también una regla de inferencia que se corresponde con la regla básica de transformación del cálculo axiomático (*modus ponens*) conocida igualmente como regla de eliminación del condicional (denotada por "RE $\rightarrow$"), ya que realmente elimina el condicional de la secuencia deductiva. Existen otras reglas de inferencia para la introducción y la eliminación del resto de las conectivas. Destacamos la regla de introducción del condicional ("RI $\rightarrow$") también llamada *Teorema de la Deducción*, el cual nos dice que si en un cálculo existe una demostración de una fórmula R (conclusión) a partir de una fórmula S (premisa), entonces también existe una demostración de $S \rightarrow R$. Por tanto $S \rightarrow R$ es una fórmula válida (tautología).

Un ejemplo muy sencillo de una secuencia deductiva aplicando el método de deducción natural se muestra a continuación.

1. $p \rightarrow (\neg q \rightarrow r)$	Premisa	(5.11)
2. $\neg q$	Premisa	(5.12)
3. p	Premisa	(5.13)
4. $\neg q \rightarrow r$	RE $\rightarrow$, 1, 3	(5.14)
5. r	RE $\rightarrow$, 2, 4	(5.15)

Existe una definición semántica de **estructura deductiva correcta**. Una estructura deductiva es correcta cuando no existe ninguna interpretación que satisfaga todas las premisas y no satisfaga la conclusión. Por lo tanto, según el teorema de la deducción, dadas una serie de premisas y una conclusión, la fórmula $P_1 \wedge P_2 \wedge \ldots P_n \rightarrow C$ es una tautología.

Incluso en el método más intuitivo de deducción natural, el problema sigue siendo el tener criterios que simplifiquen el proceso de construcción de la secuencia deductiva que permite pasar de la situación inicial a la buscada. El hecho de plantear un método sistemático de deducción permitiría resolver el problema computacionalmente. Afortunadamente, la *regla de resolución* en el cálculo de proposiciones y su extensión en el *método general de resolución y unificación* en el cálculo de predicados [Robinson, 1965] —que veremos más

adelante— permiten realizar el planteamiento de los llamados **métodos de deducción automática**. Consiste básicamente en la aplicación de procedimientos que detectan si la negación de una fórmula es insatisfacible; o lo que es lo mismo, si la fórmula es válida.

Para verificar si una fórmula es deducible habrá que generar una secuencia deductiva que permita pasar, aplicando las reglas de inferencia disponibles, de las premisas a la fórmula buscada (conclusión, C). Teniendo en cuenta lo dicho anteriormente; habrá que comprobar que si las premisas P_1, P_2, P_3, ..., P_n son ciertas, entonces la conclusión C también es verdad. Una forma alternativa de verificar que una fórmula es satisfacible es suponer que la conclusión buscada es falsa y comprobar que $(P_1 \wedge P_2 \wedge P_3 \ldots \wedge P_n \wedge \neg C)$ es una contradicción (esto se conoce genéricamente como resolver por **refutación**).

El cálculo de proposiciones basado exclusivamente en la regla de resolución (utilizando sólo dicha regla de inferencia) permite comprobar la deducibilidad de una fórmula cualquiera a partir de unas premisas dadas. Para aplicar la regla de resolución a un conjunto de premisas éstas tienen primero que ponerse en *forma clausulada*. Una cláusula es una disyunción de variables proposicionales, negadas o sin negar. Se dice que una sentencia está en forma clausulada cuando consiste en una conjunción de cláusulas. A continuación (5.16) se presenta una fórmula en forma clausulada constituida por una serie de premisas (cláusulas) y la negación de la conclusión (que también tiene forma de cláusula).

$$\underset{P_1}{(\neg p \vee q)} \wedge \underset{P_2}{(\neg q \vee s \vee r)} \wedge \underset{P_3}{\neg s} \wedge \underset{\neg C}{\neg r} \tag{5.16}$$

La regla de resolución transforma dos cláusulas cualesquiera (que se denominan *generatrices*) en las que tiene que aparecer una misma variable proposicional, negada en una y sin negar en la otra, en una sola cláusula (denominada *resolvente*) que contendrá la disyunción del resto de los literales (variables proposicionales negadas o sin negar) que formaban las *generatrices*. Por ejemplo aplicada a las cláusulas P_1 y P_2 de la sentencia (5.16) se obtendría:

$$\neg p \vee s \vee r \tag{5.17}$$

Una estrategia posible para comprobar la **validez de un proceso deductivo** como el formado en (5.16) y (5.17), por las premisas y la conclusión, es realizar un proceso exhaustivo de aplicación de la regla de resolución entre todas las cláusulas de las cuales se parte y entre éstas y las nuevas resolventes que

vayan generándose por aplicación de la regla de resolución, hasta encontrar la cláusula vacía (cuándo las generatrices tienen una misma y única variable proposicional; por ejemplo, si P_1 fuera q y P_2 fuera $\neg q$), que se representa mediante el símbolo λ (ver apartado 5.4.3). En otras palabras, cuando se comprueba que el único caso que haría el razonamiento inválido es una contradicción. Es decir, si $(P_1 \wedge P_2 \wedge P_3 \ldots \wedge P_n \wedge \neg C)$ es una contradicción (al intentar satisfacer dicha expresión se ha llegado a la cláusula vacía), nunca podrá ocurrir que las premisas sean verdad y la conclusión falsa, que es precisamente el único caso que invalidaría el razonamiento.

Se puede comprobar que el cálculo de proposiciones basado en la regla de resolución como única regla de inferencia cumple las tres propiedades básicas de un formalismo lógico. Es decir; es **consistente** (no se puede demostrar simultáneamente una fórmula y su negación), es **completo** (cualquier fórmula definida como válida según un criterio prefijado es demostrable a partir de los axiomas y del conjunto de reglas de inferencia) y es —como se ha señalado hasta ahora— **decidible** (existe un procedimiento efectivo de comprobar si cualquier fórmula es válida en el sistema). Por todo ello se dice que la lógica de proposiciones es un formalismo de representación muy *robusto*.

Uno de los problemas claves a la hora de elegir el formalismo de representación adecuado para un problema dado es la determinación de la *granularidad* del formalismo. Dicho de otra forma; la capacidad expresiva del formalismo o lo que también es lo mismo; su **adecuación representacional**. Es decir, que se pueda expresar todos los elementos relevantes para la solución del problema.

Supongamos que se nos presenta el siguiente razonamiento:

Los estudiantes de informática aman la lógica (5.18)

Juan es estudiante de informática (5.19)

Juan ama la lógica (5.20)

Si intentamos representarlo en la lógica de proposiciones sólo podríamos optar por una estructura como la siguiente:

$1.\ p \rightarrow q$ (5.21)

2. r (5.22)

3. s (5.23)

Con lo cual, no inferiríamos adecuadamente la conclusión del razonamiento —que parece ser intuitivamente correcto—, ya que no expresaríamos la relación existente entre *Juan* y el conjunto de los *estudiantes de informática.* Vamos a estudiar a continuación un cálculo lógico que permite representar adecuadamente este tipo de situaciones.

5.3 LÓGICA DE PREDICADOS

La lógica de predicados abarca —superándola— a la de proposiciones. Esta última es, por lo tanto, un subsistema de aquélla, y todas las reglas del cálculo proposicional siguen siendo aplicables en el cálculo de predicados.

En las declaraciones previas (5.18), (5.19) y (5.20), la estructura deductiva se basa en la relación existente entre los componentes de los enunciados simples. Tanto *Juan* como los *estudiantes* señalados cumplen la propiedad de estar cursando los estudios de informática. Es decir, es necesario introducir un formalismo que permita distinguir **qué se afirma** (propiedades o relaciones) y **de quién se afirma** (individuos). El razonamiento previo quedaría expresado como:

$$\forall x \left(Estudiante\_Infor(x) \rightarrow Ama\_log(x)\right) \quad (5.24)$$

$$Estudiante\_Infor(Juan) \quad (5.25)$$

$$Ama\_log(Juan) \quad (5.26)$$

En el cálculo de predicados, las *propiedades* (*Estudiante_Infor*) y los nombres de *relaciones* entre individuos (*Amigo*, *Ama_log*) se denominan **predicados**, para expresarlos se utiliza una notación funcional (un predicado con una serie de *argumentos* o *términos*). Las letras de variables, las constantes y las funciones —que comentaremos en seguida— representan los argumentos de dichos predicados (p.ej., variables: *x*, *y*, ..., constantes: *Juan*, *Pedro*, ..., funciones: *máximo*, *sucesor*, ...). El símbolo "$\forall$" representa el cuantificador universal y se utiliza para señalar a todos los individuos que cumplen una cierta propiedad o tienen una cierta relación (p.ej., "todos los objetos que pueden servir de soporte son rígidos"). El cuantificador existencial, también llamado *particular*, se expresa mediante "$\exists$" y sirve para personificar propiedades en

individuos indeterminados (p.ej., "alguien tiene que ser el responsable"). Para indicar el alcance de los cuantificadores se hace uso de los paréntesis; en la expresión (5.27) y es una variable *libre*.

$$\exists x \left(Estudiante\_Infor(x) \wedge Amigo(x, y)\right) \tag{5.27}$$

Veamos los ejemplos siguientes:

1. "Algunos estudiantes de informática sólo son amigos de los aficionados a la lógica":

$$\exists x \left(Estudiante\_Infor(x) \wedge \forall y \left(Amigo(x, y) \rightarrow Ama\_log(y)\right)\right) \tag{5.28}$$

2. "todos son amigos de todos":

$$\forall x \forall y \, Amigo(x, y) \tag{5.29}$$

Los predicados que tienen un único argumento se denominan predicados *monádicos* (éstos tienen algunas propiedades específicas que comentaremos más adelante) y los que tienen dos o más argumentos se denominan predicados *poliádicos* (dos argumentos: *diádicos*, tres argumentos: *triádicos*, etc.). Hay expresiones que exigen la utilización de predicados poliádicos; en otras ocasiones éstos son opcionales. Por ejemplo, supongamos que en la expresión anterior (5.28) en lugar de tratarse de los aficionados a la lógica se hablara de los aficionados al cine. En estos casos sería mejor haber planteado un enunciado más general como se muestra a continuación.

$$\exists x \left(Estudiante\_Infor(x) \wedge \forall y \left(Amigo(x, y) \rightarrow Ama(y, cine)\right)\right) \tag{5.30}$$

Las funciones, al igual que los predicados, pueden tener términos que sean a su vez variables o constantes. La diferencia con los predicados estriba en que las funciones son aplicaciones de $D^n \rightarrow D$, siendo D el dominio de referencia. Por lo tanto, no tienen ningún valor de verdad asociado ("V" o "F"). La razón de su utilización es la *simplificación notacional* a la que dan lugar.

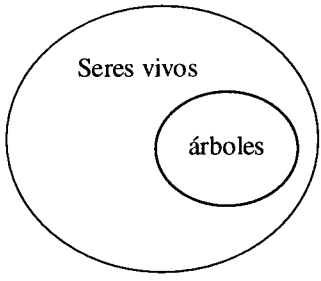
Por ejemplo, supongamos que queremos representar la frase "Todo cociente de dos números primos es un número fraccionario". Si tomamos como dominio de referencia el de los números primos, cabrían dos opciones:

1. $\forall x \forall y \forall z$ (Cociente(x, y, z) $\rightarrow$ fraccionario(z)) (5.31)

Si en su lugar hubiéramos utilizado la función *cociente C*:

2. $\forall x \forall y$ fraccionario(C(x, y)) (5.32)

Puede resultar bastante útil (p.ej., en la verificación de razonamientos mediante diagramas de Euler, ver a continuación la Fig. 5.1) interpretar las expresiones de la *lógica de predicados monádicos* en la **lógica de clases**[3], tomando las expresiones $\forall x\ P(x)$ como la definición de la clase P. Es decir, la clase de todos los x tales que se hace verdadero el enunciado $P(x)$. De esta forma la expresión "todos los árboles son seres vivos" representaría la relación de inclusión entre dos conjuntos.



$\forall x(árbol(x) \subset ser\_vivo(x))$

$árbol \subset ser\_vivo$

Fig. 5.1. Interpretación en lógica de clases

Para los predicados poliádicos existe una interpretación equivalente en la **lógica de relaciones**[4]. Una relación, binaria por ejemplo, la podemos definir como el conjunto de pares ordenados (x, y) tales que xRy es verdadero, o también como la clase de pares ordenados que hacen verdadero el enunciado $R(x, y)$[5].

$$R = \{(x,y) \mid xRy\} \equiv \{(x,y) \mid R(x,y)\} \tag{5.33}$$

Aunque hemos hablado de la existencia de dos cuantificadores, en realidad podría haberse utilizado uno sólo. Decir, por ejemplo, que "no todos los x tienen la propiedad P" equivale a decir que "hay algún x que no tiene la propiedad P".

[3] El modelo matemático de la *lógica de clases* es el *álgebra booleana de clases*.

[4] El modelo matemático de la *lógica de relaciones* es el *álgebra de relaciones*.

[5] El *álgebra de relaciones* se ha utilizado como modelo matemático para el diseño de base de datos relacionales.

Igualmente decir que "todos los x carecen de la propiedad P" equivale a decir que "no hay ningún x que posea la propiedad P". También el decir que "no todos los x carecen de la propiedad P" es tanto como decir: "hay algún x que tiene la propiedad P". Finalmente, "no hay ningún x que no sea P" equivale a "todos los x son P". Estas equivalencias se representan a continuación.

$$\neg\forall x\ P(x) \leftrightarrow \exists x\ \neg P(x) \tag{5.34}$$

$$\forall x\ \neg P(x) \leftrightarrow \neg\exists x\ P(x) \tag{5.35}$$

$$\neg\forall x\ \neg P(x) \leftrightarrow \exists x\ P(x) \tag{5.36}$$

$$\neg\exists x\ \neg P(x) \leftrightarrow \forall x\ P(x) \tag{5.37}$$

Para poder interpretar el cálculo de predicados tenemos que hacer referencia a individuos concretos. Una expresión no es verdadera ni falsa hasta que no la particularizamos. Por ejemplo, *discípulo*(x, y) es cierta para el caso de *discípulo*(*Pedro*, *Jesús*). Por lo tanto para validar sentencias en el cálculo de predicados habrá que referirse a un determinado *universo de discurso*, es decir, al conjunto de objetos que constituye el marco de referencia de nuestro lenguaje en un momento dado. Por ejemplo, si consideramos que el universo de discurso es el conjunto $\{a, b, c\}$, para establecer el valor de verdad de los predicados cuantificados se tienen en cuenta las equivalencias siguientes.

$$\forall x\ P(x) \leftrightarrow P(a) \wedge P(b) \wedge P(c) \tag{5.38}$$

$$\exists x\ P(x) \leftrightarrow P(a) \vee P(b) \vee P(c) \tag{5.39}$$

Para validar una sentencia podría intentarse, al igual que en el cálculo de proposiciones, construir la tabla de verdad de la expresión y comprobar que es una tautología. Dado que la validación ahora depende del universo de discurso, y este cálculo puede ser muy complejo (para verificar una expresión con n predicados monádicos habría que definir un universo de discurso que tenga 2^n elementos), es necesario plantear métodos efectivos de validación.

Vimos que el cálculo de proposiciones es un cálculo muy robusto que cumple las tres propiedades básicas: *consistencia*, *completitud* y *decidibilidad*. Sobre todo esta última tiene una gran importancia desde el punto de vista computacional, ya que el método de resolución descrito permite establecer un proceso sistemático de validación de fórmulas. En el cálculo de predicados sólo se puede establecer un procedimiento para la prueba de un *teorema* (una fórmula que se ha obtenido a partir de los axiomas y las reglas de inferencia del cálculo;

por lo tanto verdadera si el cálculo es consistente); sin embargo, para el resto de las fórmulas el procedimiento no garantiza su finalización. Esta característica, conocida como la **semidecibilidad** del cálculo de predicados, no es tan importante teniendo en cuenta que los métodos de IA buscan métodos plausibles (no necearíamente óptimos) de solucionar problemas y en muchos casos la *semidecibilidad* del cálculo de predicados es una condición suficiente.

En el cálculo de predicados es necesario introducir nuevas reglas de inferencia para tratar convenientemente las expresiones cuantificadas. Ya señalamos la existencia de reglas en *deducción natural* que permiten introducir y eliminar los cuatro signos lógicos básicos del cálculo —negación, conjunción, disyunción condicional—. Ahora se añaden las siguientes.

En todas las reglas de inferencia se presupone que:

– P es una variable metalingüística que representa cualquier predicado.
– $a_1, a_2, \ldots, a_n$ son símbolos cualesquiera de nombres de individuo (también llamados *parámetros*).

• Regla de eliminación del cuantificador universal (RE ∀),

$$\forall x_1, x_2, \ldots, x_n \; P(x_1, x_2, \ldots, x_n) \tag{5.40}$$

$$\overline{\qquad\qquad}$$

$$P(a_1, a_2, \ldots, a_n) \tag{5.41}$$

Interpretación: es evidente que lo que se cumple genéricamente para todos los individuos se cumple para un conjunto cualquiera de estos.

• Regla de introducción del cuantificador universal (RI ∀),

$$P(a_1, a_2, \ldots, a_n) \tag{5.42}$$

$$\overline{\qquad\qquad}$$

$$\forall x_1, x_2, \ldots, x_n \; P(x_1, x_2, \ldots, x_n) \tag{5.43}$$

Observación: para que esta última regla sea válida, los parámetros $a_1, a_2, \ldots, a_n$ se consideran constantes cualesquiera; es decir, aunque señalen a elementos concretos de una especie, su valor es indeterminado y por tanto puede ser ocupado por cualquier individuo de dicha especie. Este tipo de generalizaciones también ocurren en el mundo real. Por ejemplo, si al comprar por primera vez un número de una revista nos cuesta una determinada cantidad, suponemos que

los demás números también tendrán el mismo precio. Esta regla requiere que ninguno de los parámetros a_i se vea afectado por ninguna restricción (p.ej., esto ocurriría si algún parámetro surgío previamente como resultado de aplicar la regla "RE $\exists$", ver la siguiente explicación).

- Regla de eliminación del cuantificador existencial (RE $\exists$),

$$\exists x_1, x_2, \ldots, x_n \; P(x_1, x_2, \ldots, x_n) \tag{5.44}$$

$$\overline{\qquad\qquad\qquad\qquad}$$

$$P(a_1, a_2, \ldots, a_n) \tag{5.45}$$

Interpretación: el hecho de afirmar que hay algún individuo que cumple una determinada propiedad no nos da licencia para representar tal individuo por la constante a_i que nosotros elijamos por conveniencia (p.ej. un a_i que haya aparecido previamente en la secuencia deductiva). Es decir, el parámetro generado debe considerarse un individuo concreto, aquél que cumple la propiedad P. Por tanto, tampoco puede ser considerado un individuo cualquiera que luego pueda ser generalizado (aplicando RI $\forall$). Por ejemplo, del conocimiento de que alguién ha inventado la penicilina, no puede afirmarse que Cervantes (el que arbitrariamente hemos elegido) sea dicho inventor.

- Regla de introducción del cuantificador existencial (RI $\exists$),

$$P(a_1, a_2, \ldots, a_n) \tag{5.46}$$

$$\overline{\qquad\qquad\qquad\qquad}$$

$$\exists x_1, x_2, \ldots, x_n \; P(x_1, x_2, \ldots, x_n) \tag{5.47}$$

Interpretación: es obvio que si un individuo cumple una propiedad (o un conjunto satisface una relación) entonces se puede afirmar que hay alguno que cumple dicha propiedad (o hay una serie de individuos que satisfacen dicha relación).

Conviene por tanto, tener muy presente las restricciones de las reglas (RI $\forall$) y (RE $\exists$) para no llegar a conclusiones erróneas en el cálculo de predicados.

Supongamos, por ejemplo, que queremos demostrar la validez del siguiente esquema de inferencia:

$$\forall x\left(Estudiante\_Infor(x) \rightarrow Ama\_log(x)\right) \quad (5.48)$$

$$\forall x\left(Ama\_log(x) \rightarrow conoce\_formalismo(x)\right) \quad (5.49)$$

$$\forall x\left(Estudiante\_Infor(x) \rightarrow conoce\_formalismo(x)\right) \quad (5.50)$$

Para demostrar su validez se puede establecer la siguiente secuencia deductiva.

1. $\forall x\left(Estudiante\_Infor(x) \rightarrow Ama\_log(x)\right)$ Premisa (5.51)

2. $\forall x\left(Ama\_log(x) \rightarrow conoce\_formalismo(x)\right)$ Premisa (5.52)

3. $Estudiante\_Infor(a) \rightarrow Ama\_log(a)$ RE $\forall$, 1 (5.53)

4. $Ama\_log(a) \rightarrow conoce\_formalismo(a)$ RE $\forall$, 2 (5.54)

5. $Estudiante\_Infor(a) \rightarrow conoce\_formalismo(a)$ RTr$\rightarrow$, 3,4 (5.55)

6. $\forall x\left(Estudiante\_Infor(x) \rightarrow conoce\_formalismo(x)\right)$ RI $\forall$, 5 (5.56)

En el paso 5. se ha aplicado la Regla de Transitividad del Condicional. En 6., teniendo en cuenta que *a* es un individuo cualquiera (proviene de una especificación universal), se puede aplicar la Regla de Introducción del Cuantificador Universal.

Dado que nuestro objetivo es centrarnos en la expresividad asociada al cálculo de predicados, vamos a resumir algunas cuestiones relevantes en cuanto a la representación realizada en dicho formalismo.

1. El cuantificador universal no implica necesariamente la noción de existencia. Por ejemplo, en la expresión "todos los ingenieros son prácticos", parece indicarse que se presupone la existencia de "ingenieros". Sin embargo, en el enunciado "El que piense lo contrario será considerado un desertor" no implica la existencia de "personas que piensen lo contrario"; en todo caso, habla de su posibilidad. Por tanto, los siguientes enunciados sólo son ciertos si se excluye el universo de discurso vacío, ya que si no, su antecedente podría ser

verdadero siendo falso su consecuente ("$V \rightarrow F$") y sabemos por la definición del condicional ("$\rightarrow$") que esto invalidaría las expresiones:

$$\forall x\, Pensar\_contrario(x) \rightarrow \exists x\, Pensar\_contrario(x) \tag{5.57}$$

$$\forall x \neg Pensar\_contrario(x) \rightarrow \exists x \neg Pensar\_contrario(x) \tag{5.58}$$

2. Al igual que en la lógica de proposiciones, el símbolo "$\vee$" tiene carácter inclusivo. Por consiguiente, el sentido exclusivo de una expresión como "todos los seres humanos son mujeres o son hombres" se tendría que representar explícitamente como:

$$\forall x\,(Humano(x) \rightarrow (Mujer(x) \wedge \neg Hombre(x)) \vee (Hombre(x) \wedge \neg Mujer(x))) \tag{5.59}$$

o lo que es lo mismo:

$$\forall x\,(Humano(x) \rightarrow (Mujer(x) \vee Hombre(x)) \wedge \neg(Hombre(x) \wedge Mujer(x))) \tag{5.60}$$

3. Un problema muy importante al representar los hechos que reflejan el conocimiento de un dominio es la elección de la *granularidad* de los enunciados, es decir, su nivel de detalle o, lo que es lo mismo, su capacidad expresiva. Teniendo en cuenta que —como veremos más adelante al explicar el método de resolución— el proceso de validación de un razonamiento depende del número de variables de los predicados implicados, cuanto menor sea el número de éstas, más eficiente será el proceso. Sin embargo, hay que considerar las necesidades expresivas requeridas; recordemos el caso ya comentado de los predicados "*Ama_lógica*(x)" frente al más expresivo "*Ama* (x, *lógica*)".

4. Supongamos que queremos expresar el siguiente enunciado "todos los mamíferos poseen una propiedad en común". Necesitaríamos utilizar una expresión como la siguiente:

$$\exists P\ \forall x\ (Mamífero(x) \rightarrow P(x)) \tag{5.61}$$

Por lo tanto, puede existir la necesidad de cuantificar los predicados. Para ello se han definido **lógicas de predicados de orden superior**, que permiten además cuantificar las funciones y utilizar tanto éstas como los predicados como términos a su vez de otros predicados o funciones. Se denominan lógicas de orden superior, ya que todo lo visto hasta ahora en el presente apartado realmente se conoce como **lógica de predicados de primer orden**.

5. Un problema inherente a la lógica de predicados es que la robustez del razonamiento realizado se basa en la suposición de que se ha representado todo

el conocimiento necesario en el dominio. Ocurre que generalmente hay muchas suposiciones válidas tanto en un dominio concreto como en situaciones dispares que es muy difícil plasmar exhaustivamente. Por ejemplo, si alguien va al médico y le recetan una caja de aspirinas, se supone que el afectado debe conseguir una caja de aspirinas "con aspirinas dentro". Este tipo de razonamiento se denomina razonamiento basado en el sentido común y lo comentaremos más adelante en el apartado dedicado a introducir las **lógicas no monótonas**.

6. Hemos indicado la utilidad de estudiar la lógica de predicados como lógica de clases. Bajo dicha perspectiva, un predicado determina el conjunto de elementos que cumplen la propiedad especificada. ¿Qué pasaría si quisiéramos representar el conjunto de los días en que hace frío, o el conjunto de la gente alta, o el de los pasteles ricos?. En todos estos casos definir las fronteras del conjunto es un aspecto que depende de la subjetividad de los individuos que participen en la conversación. Así, para un aficionado al baloncesto, una persona con una estatura de 1'85ms no pertenecerá al conjunto de personas altas, sin embargo, un amante de la gimnasia rítmica si que lo considerará de dicho conjunto. Es decir, la imprecisión es un elemento intrínseco al lenguaje natural. Para poder representar este tipo de expresiones introduciremos más adelante la **lógica difusa**.

5.3.1 Ampliaciones de la Lógica de Predicados

- **Lógica de predicados de orden superior**

Muchas veces se presentan en el lenguaje común frases del tipo: "Las personas que luchan por la justicia tienen al menos una cualidad en común". Esta frase podría representarse como:

$$\exists P(\forall x\,(Personas(x) \wedge Lucha\_justicia(x) \rightarrow P(x))) \tag{5.62}$$

Supongamos que quisiéramos representar la frase siguiente: "Algunas funciones aplicables a ciertos números enteros son complejas"

$$\exists x\,(Entero(x) \wedge \exists f\ (Compleja(f(x))) \tag{5.63}$$

Para poder representar adecuadamente las expresiones (5.62) y (5.63) es necesario introducir la posibilidad de representar dominios de referencia tanto para predicados como para funciones. En este marco, los nombres de predicado y de función tienen las mismas atribuciones que los nombres de variable, es decir, pueden ser cuantificados. Este tipo de lógica recibe el nombre de **cálculo**

de predicados de segundo orden. Además de este caso, en el que se requiere expresar propiedades de propiedades de individuos, cabría plantearse casos más complejos en que tuvieran que representarse propiedades de propiedades de propiedades de individuo (*lógica de tercer orden*) y así sucesivamente. Todas estas lógicas se agrupan bajo el nombre de **lógicas de orden superior**.

La razón fundamental de estudiar estas lógicas como un cálculo aparte es que en ellas no se plantea la *semidecidibilidad* del cálculo tal y como se ha hecho en la lógica de predicados. El cálculo de predicados sólo es *decidible* si los predicados son monádicos y bajo ciertas restricciones en el caso de los predicados poliádicos. Sin embargo, no es posible construir una lógica en la que puedan ser demostradas todas las expresiones válidas expresables en la lógica de predicados de orden superior (incompletitud) [Gödel, 1931].

- **Lógica de predicados con identidad**

Existe otro tipo de expresiones que todavía no hemos mencionado. Veamos algunos ejemplos:

1. El rey de España es Juan Carlos I (5.64)
2. Cervantes fue el autor del Quijote (5.65)
3. $2 + 3 = 5$ (5.66)

En todas ellas estamos usando una relación de *identidad*. Cuando expresamos la frase (5.65) estamos señalando precisamente aquella persona que escribió el Quijote y sólo a dicha persona. Las expresiones de igualdad o de identidad tienen una serie de peculiaridades dentro del cálculo de predicados que estamos analizando.

Se establece una nueva forma "$x = y$" en el cálculo de predicados que permite expresar la relación de identidad, siendo en este caso x e y las variables que ocupan las plazas del nuevo predicado diádico definido. La interpretación semántica de dicho predicado es que "$x = y$" es verdad cuando ambos se refieren al mismo elemento del *universo de discurso*. A menudo se utiliza "$x \neq y$" como forma abreviada de $\neg (x = y)$. La inclusión de la relación de identidad permite crear nuevos elementos de la teoría axiomática del cálculo de predicados, que sirven para reflejar la validez de las inferencias realizables bajo la riqueza expresiva de este nuevo formalismo. Por lo tanto, la *lógica de predicados con identidad* es una extensión de la lógica de predicados. Dado el carácter introductorio de este apartado, simplemente vamos a mostrar como ejemplo una de estas leyes, que refleja que si dos entidades son idénticas, tienen las mismas

propiedades (la siguiente expresión define dicha relación en lógica de primer orden, esta es la razón por la que no se incluye $\forall P$).

Ley de *indiscernibilidad* de los idénticos:

$$\forall x \forall y ((x=y) \rightarrow (P(x) \leftrightarrow P(y))) \tag{5.67}$$

Además de la extensión *inferencial* del cálculo, el nuevo predicado permite ampliar la riqueza expresiva asociada a los cuantificadores hasta ahora vistos. Así, el cuantificador universal permitía señalar a todos los elementos de un conjunto, mientras que el existencial nos permitía nombrar a algunos elementos indeterminados; supongamos sin embargo que deseamos precisar un poco más dicha cuantificación. ¿Qué pasaría con frases como: "Hay al menos 3 personas que han estado en la Luna", "Hay exactamente tres cantantes de ópera españoles con un gran prestigio universal", o "Hay a lo sumo dos partidos políticos con posibilidad de ganar las próximas elecciones"?. Veamos a continuación la representación de algunas frases de este tipo.

1. "Hay al menos dos porteros en la selección de fútbol" (suponemos que el dominio de referencia son los jugadores de fútbol):

$$\exists x \exists y ((Portero(x) \wedge Portero(y)) \wedge (x \neq y)) \tag{5.68}$$

2. "Hay a lo sumo un capitán en el equipo":

$$\forall x \forall y ((capitan(x) \wedge capitan(y)) \rightarrow (x = y)) \tag{5.69}$$

Hay exactamente un entrenador:

$$\exists x (Entrenador(x) \wedge \forall y (Entrenador(y) \rightarrow (x=y))) \tag{5.70}$$

La primera parte de la fórmula (5.70) indica que hay al menos uno que cumple la propiedad y la segunda que no hay más de uno que la tenga.

5.4 DEDUCCIÓN AUTOMÁTICA: RESOLUCIÓN

Vamos a mostrar dentro del *cálculo de predicados de primer orden* un método sistemático de validación de fórmulas basado en la *expresión general de la regla de resolución.* Aunque el procedimiento que se describe puede no terminar, existe un número importante de *fórmulas decidibles* que justifica el desarrollo e implantación de este método, enmarcado dentro de los denominados de *deducción automática.* Estas técnicas son aplicables a la *verificación formal*

de programas y por otro lado, han servido de base para la formulación de intérpretes de sistemas de programación lógica (PROLOG). Se han utilizado "programas de razonamiento automático" en el diseño y validación de circuitos lógicos, así como en la resolución de algunos teoremas, tanto en el cálculo lógico como matemático, que hasta entonces no tenían una formulación conocida.

La **expresión general de la regla de resolución** permite, al igual que ocurría con la *regla de resolución* del cálculo de proposiciones, sistematizar el procedimiento de búsqueda asociado a la comprobación de la *satisfacibilidad* de una determinada conclusión (método garantizado para conclusiones válidas).

5.4.1 Forma Clausulada

Una condición básica para poder aplicar el método de resolución (más exactamente *resolución binaria* [Robinson, 1965]) es que las premisas y la conclusión estén en **Forma Clausulada** (conjunción de disyunciones; también llamada Forma Normal de Skolem o Forma Normal Conjuntiva [Davis & Putnam, 1960]; abreviadamente **FNC**). Esta forma se traduce en las siguientes condiciones:

- Todos los cuantificadores están a la cabeza de la fórmula.
- Sólo existen cuantificadores universales.
- El resto de la fórmula —todo lo que no es *cabeza*—, se denomina *matriz* y está formado por una conjunción de fórmulas, cada una de ellas está consituida a su vez por una disyunción de fórmulas atómicas, negadas o sin negar.

La siguiente fórmula es la FNC de la expresión (5.30) vista anteriormente, para obtenerla se aplica el procedimiento que veremos a continuación.

$$\forall y \left(Estudiante\_Infor(a) \wedge \left(\neg Amigo(a, y) \vee Ama(y, cine) \right) \right) \qquad (5.71)$$

La uniformidad de la *forma clausular* facilita su representación en el ordenador y la estructuración de los programas que la manipulan (ver más adelante el apartado dedicado a la *Programación Lógica*).

A continuación mostramos un procedimiento aplicable para la obtención de la forma clausulada de cualquier expresión de la lógica de predicados de primer orden. Para entender mejor los pasos que hay que realizar, conviene

observar su efecto en el ejemplo que se muestra más adelante a partir de la expresión (5.93).

- **Procedimiento de conversión de sentencias en forma clausulada:**

  1. Eliminación de "$\rightarrow$" y "$\leftrightarrow$", aplicando la ley de equivalencia del condicional con respecto a la disyunción y la de la relación entre ambos:

  $$(P \rightarrow S) \leftrightarrow (\neg P \vee S) \tag{5.72}$$

  $$(P \leftrightarrow S) \leftrightarrow (P \rightarrow S) \wedge (S \rightarrow P) \tag{5.73}$$

  2. Eliminación de "$\neg$" de las fórmulas compuestas. Dependiendo de las fórmulas presentadas, recurrir al uso de alguna de las reglas siguientes.

  2.1. Utilizar las leyes de *De Morgan.*

  $$\neg(P \vee S) \leftrightarrow \neg P \wedge \neg S \qquad \text{(De Morgan)} \tag{5.74}$$

  $$\neg(P \wedge S) \leftrightarrow \neg P \vee \neg S \tag{5.75}$$

  2.2. Aplicación de la ley de la doble negación.

  $$\neg\neg P \leftrightarrow P \tag{5.76}$$

  2.3. Supresión de las negaciones en las fórmulas afectadas por cuantificadores aplicando las leyes (5.34) a (5.37) vistas anteriormente.

  3. Cambio de variable en las fórmulas cuantificadas para que cada una tenga un nombre de variable distinto. Para ello se tiene en cuenta que siempre puede cambiarse el nombre de las variables de una expresión *cerrada* (sin variables *libres*):

  $$\forall x\ P(x) \leftrightarrow \forall y\ P(y) \tag{5.77}$$

  $$\exists x\ P(x) \leftrightarrow \exists y\ P(y) \tag{5.78}$$

  Esto permite transformar expresiones como:

  $$R(x) \vee \forall x\ P(x) \leftrightarrow R(x) \vee \forall y\ P(y) \tag{5.79}$$

En lugar de cambiar siempre el nombre de las variables iguales, para evitar crear un número excesivo de cuantificadores diferentes al comienzo de la fórmula (ver paso 4.), se pueden aplicar también las siguientes reglas (no son válidas las equivalencias correspondientes a las ley de distribución del $\forall$ por la $\vee$, ni la ley de contracción del $\exists$ por la $\wedge$).

$$\forall x\,(R(x) \wedge P(x)) \leftrightarrow \forall x\,R(x) \wedge \forall x\,P(x) \tag{5.80}$$

(*ley de distribución del* $\forall$ *por la* $\wedge$)

$$\exists x\,(R(x) \vee P(x)) \leftrightarrow \exists x\,R(x) \vee \exists x\,P(x) \tag{5.81}$$

(*ley de distribución del* $\exists$ por la $\vee$)

No son válidas:

$$\forall x\,(R(x) \vee P(x)) \not\leftrightarrow \forall x\,R(x) \vee \forall x\,P(x) \tag{5.82}$$

(*ley de distribución del* $\forall$ *por la* $\vee$, p.ej., de "Todas las páginas de un libro son pares o impares" no se puede deducir "O todas las páginas de un libro son pares o todas las páginas de un libro son impares")

$$\exists x\,R(x) \wedge \exists x\,P(x) \not\leftrightarrow \exists x\,(R(x) \wedge P(x)) \tag{5.83}$$

(*ley de contracción del* $\exists$ *por la* $\wedge$, p.ej., de "Hay alumnos que son buenos estudiantes" y "Hay alumnos que son malos estudiantes" no se puede deducir "Hay alumnos que son buenos y malos estudiantes")

4. Colocar todos los cuantificadores a la *cabeza* de la fórmula. Para ello se aplican las siguientes reglas.

$$R \vee \forall x\,P(x) \leftrightarrow \forall x\,(R \vee P(x)) \tag{5.84}$$

$$R(x) \vee \forall y\,P(y) \leftrightarrow \forall y\,(R(x) \vee P(y)) \tag{5.85}$$

(existen otras 6 reglas que resultan de sustituir en las anteriores $\forall$ por $\exists$ y $\vee$ por $\wedge$)

5. Supresión de los cuantificadores existenciales, actuando según uno de los dos casos siguientes posibles.

5.1. Si el $\exists$ no tiene $\forall$ a su izquierda (no se encuentra bajo su alcance) se elimina el $\exists$ y se introduce una *constante*. Por ejemplo:

$$\exists x \forall y \forall z \forall u\,((R(x) \vee P(x,u)) \wedge S(x, y, z)) \tag{5.86}$$

se transformaría en:

$$\forall y \forall z \forall u \, ((R(a) \vee P(a,u)) \wedge S(a, y, z)) \tag{5.87}$$

Dichas *constantes* son en realidad *parámetros* cuyo nombre no puede coincidir con ninguna otra constante ya existente en la fórmula (recordar la explicación dada para la regla RE $\exists$). Estas *constantes* se denominan **constantes de Skolem**.

5.2. Si el $\exists$ tiene uno o varios $\forall$ a su izquierda, se elimina la variable afectada por el $\exists$ y se introduce en su lugar una **Función de Skolem**.

Supongamos que queremos representar la frase "todos los hombres tienen un padre".

$$\forall x \exists y \, (Hombre(x) \rightarrow Padre(x, y)) \tag{5.88}$$

Para eliminar el $\exists$ hay que tener en cuenta el hecho de que el "padre" está en función del "hombre" concreto del que se trate (debemos crear un nombre de función específico que refleje dicha dependencia; normalmente se elige alguno como f, g, h, ...). La expresión anterior se reescribiría por:

$$\forall x \, (Hombre(x) \rightarrow Padre(x, f(x))) \tag{5.89}$$

Las funciones así creadas reciben el nombre de *funciones de Skolem*. Estas funciones dependerán de tantas variables como cuantificadores $\forall$ haya a la izquierda del $\exists$ que en cada caso se esté intentando eliminar. Por ejemplo:

$$\forall y \forall z \exists x \, (R(x) \vee P(x, y, z)) \tag{5.90}$$

Se transformaría en:

$$\forall y \forall z \, (R(g(y,z)) \vee P(g(y,z), y, z)) \tag{5.91}$$

6. Eliminar todos los $\forall$. En esta etapa ya sólo quedan cuantificadores universales en la cabeza de la fórmula que afectan a cada una de las variables existentes. Por lo tanto, éstos pueden simplemente omitirse y suponer de ahora en adelante que todas las variables pueden utilizarse en sentido genérico.

7. Convertir la fórmula en una conjunción de disyunciones. Para ello se aplica la propiedad distributiva de la disyunción con respecto a la conjunción.

$$R \vee (P \wedge S) \leftrightarrow (R \vee P) \wedge (R \vee S) \tag{5.92}$$

8. Considerar cada una de las disyunciones de 7 como una cláusula aparte; algunas de ellas puede estar formada por un único literal.

9. Diferenciar el nombre de las variables entre las distintas cláusulas surgidas en el paso 8, de tal forma que no haya dos que hagan referencia al mismo nombre de variable. Para ello se cambian los nombres de dichas variables —esto es factible por la regla (5.80)—.

Como ejemplo ilustrativo del proceso de conversión vamos a utilizar la expresión que resulta de negar la fórmula (5.30) presentada anteriormente. Mostramos en cada paso el resultado final de las modificaciones realizadas en dicha etapa.

$$\neg\exists x \left(Estudiante\_Infor(x) \wedge \forall y \left(Amigo(x,y) \rightarrow Ama(y, cine) \right) \right) \tag{5.93}$$

Conversión en forma clausulada (se muestra el resultado de haber considerado cada paso):

1. $\neg\exists x\ (Estudiante\_Infor(x) \wedge \forall y(\neg Amigo(x,y) \vee Ama(y,cine)))$ (5.94)
2. $\forall x\ (\neg Estudiante\_Infor(x) \vee \exists y\ (Amigo(x,y) \wedge \neg Ama(y, cine)))$ (5.95)
3. $\forall x\ (\neg Estudiante\_Infor(x) \vee \exists y\ (Amigo(x,y) \wedge \neg Ama(y, cine)))$ (5.96)
   (cada una de las variables cuantificadas en el paso 2 tiene un nombre distinto, por lo que en el paso 3 no es necesario realizar ningún cambio)
4. $\forall x \exists y\ (\neg Estudiante\_Infor(x) \vee (Amigo(x,y) \wedge \neg Ama(y, cine)))$ (5.97)
5. $\forall x\ (\neg Estudiante\_Infor(x) \vee (Amigo(x, f(x)) \wedge \neg Ama(f(x), cine)))$ (5.98)
6. $\neg Estudiante\_Infor(x) \vee (Amigo(x, f(x)) \wedge \neg Ama(f(x), cine))$ (5.99)
7. $(\neg Estudiante\_Infor(x) \vee Amigo(x, f(x))) \wedge (\neg Estudiante\_Infor(x) \vee \neg Ama(f(x), cine)))$ (5.100)
8. C_1: $\neg Estudiante\_Infor(x) \vee Amigo(x, f(x))$ (5.101)
   C_2: $\neg Estudiante\_Infor(x) \vee \neg Ama(f(x), cine))$
9. C_1: $\neg Estudiante\_Infor(x) \vee Amigo(x, f(x))$ (5.102)
   C_2: $\neg Estudiante\_Infor(z) \vee \neg Ama(f(z), cine))$

Al igual que ocurría en lógica de proposiciones, una vez comprobado cómo cualquier expresión en lógica de predicados puede ponerse en su *forma clausular* equivalente, el objetivo final sigue siendo el encontrar un método sistemático de búsqueda que sirva para comprobar la validez del proceso deductivo.

El método de resolución (como veremos más adelante) se basa en aplicar la regla de resolución generalizada para el cálculo de predicados. En realidad dicha regla es —como su homónima en el cálculo de proposiciones— una generalización de la regla de *modus ponens* (para verlo sólo hay que transformar $p \rightarrow q$ por su equivalente $\neg p \vee q$). El problema en el cálculo de predicados estriba en que para poder identificar la existencia de una fórmula atómica negada en una cláusula y sin negar en otra hay que considerar la posible existencia de variables, constantes y funciones en los términos de los predicados comparados.

5.4.2 Unificación

Uno de los procesos básicos más comunes en los problemas de IA es la denominada **equiparación** o *equiparación de patrones*. En general, la equiparación consiste en poner en relación de igualdad dos cosas. Para ello se comprueba que su estructura es semejante y que cada uno de los componentes que forman dicha estructura son "compatibles". La condición de *compatibilidad* depende del contexto de aplicación (veremos en los siguientes capítulos ejemplos de *equiparación* de reglas, marcos, guiones, redes, etc.).

En la lógica de predicados de primer orden el problema de la *equiparación* de fórmulas tiene una gran trascendencia en la eficiencia general de todo el proceso. Básicamente consiste en comparar predicados con el mismo nombre y en comprobar si sus términos (argumentos) se pueden **unificar**. Para que dos términos sean *unificables* debe existir una sustitución que los haga idénticos. Teniendo en cuenta que las variables que aparecen en los predicados se suponen cuantificadas universalmente (ver 5.4.1.), éstas pueden adoptar cualquier valor. Por tanto, en principio las variables pueden sustituirse por cualquier constante, función o incluso otra variable diferente (su significado en una sentencia en FNC es meramente *notacional*).

Una sustitución es un conjunto de pares del tipo $[t_k / v_k]$; donde v_k es una variable que ocupa el lugar *k-ésimo* dentro de los argumentos del predicado y t_k es un término cualquiera del mismo conjunto (otras variables, constantes o funciones que ocupen el mismo lugar) en el que la variable v_k no está presente

(es decir, no puede ocurrir que $v_k = t_k(v_k)$). Las dos expresiones siguientes (ver 5.103) pueden unificarse a través de la sustitución s_1.

$$\left.\begin{matrix} P(u,b) \\ P(f(x),v) \end{matrix}\right\rangle s_1\colon \{f(x)/u, b/v\} \Rightarrow P(u,b)s_1 = P(f(x),v)s_1 = P(f(x),b) \quad (5.103)$$

Las dos expresiones del predicado P han podido unificarse teniendo en cuenta las consideraciones siguientes:

1. u y $f(x)$ ocupan el primer lugar o plaza de los argumentos del predicado P, además en $f(x)$ no aparece u, por tanto puede crearse el par $\{f(x)/u\}$.
2. b y v ocupan el segundo lugar, siendo b una constante, por tanto puede crearse el par $\{b/v\}$.

La sustitución s_1 debe interpretarse como: "donde haya una u poner $f(x)$" y "donde haya una v poner una b".

En (5.103) es posible realizar una sustitución menos general:

$$\left.\begin{matrix} P(u,b) \\ P(f(x),v) \end{matrix}\right\rangle s_2\colon \{f(b)/u, b/v\} \Rightarrow P(u,b)s_2 = P(f(x),v)s_2 = P(f(b),b) \quad (5.104)$$

Como puede apreciarse, la forma $P(f(b),b)$ resultante de aplicar la sustitución s_2 a $P(u, b)$ tiene un menor número de variables. Si ahora se intentaran unificar dicha expresión con $P(f(c),b)$ no sería posible, ya que se ha eliminado la variable u; por lo tanto, tal y como se refleja en (5.105), debería haberse aplicado la sustitución más general $P(u, b)s_1 = P(f(x), b)$.

$$\left.\begin{matrix} P(f(x),b) \\ P(f(c),b) \end{matrix}\right\rangle s_3\colon \{c/x\} \Rightarrow P(f(c), b) \quad (5.105)$$

Debido a que vamos a definir un proceso de búsqueda sistemática de aplicación de la regla de resolución entre cláusulas distintas, interesa aplicar unificadores lo más generales posibles, de tal forma que no se frustre ningún camino de búsqueda que pueda llevar a la comprobación final de la validez del proceso deductivo propuesto (ver epígrafe siguiente).

En el ejemplo mostrado en (5.105) se ha definido implícitamente la operación de **composición de sustituciones** (también llamada *producto*). La composición de s_1s_3 consiste en sustituir en s_1 todos los términos t_k que tengan variables que también aparezcan en s_3 por su valor sustitutorio, añadiendo además a s_1 todas los demás cambios de variables que no aparecieran en s_1. La

composición de sustituciones no cumple la propiedad conmutativa, ya que se realiza aplicando primero los pares de las sustituciones más a la izquierda (ver la operación de composición mostrada en 5.106).

$$\left.\begin{array}{c} P(u,b) \\ P(f(x),v) \\ P(f(c),b) \end{array}\right\rangle s_1 s_3 : \{f(c)/u,\ b/v\} \Rightarrow P(f(c),b) \qquad (5.106)$$

El método de resolución que veremos a continuación requiere que la unificación entre los predicados de las cláusulas se realice utilizando el denominado **unificador mínimo** o **de máxima generalidad** (UMG). El UMG se define como aquél que cumple la propiedad de que cualquier otro unificador puede obtenerse a partir de él por aplicación de alguna otra sustitución. Este unificador contiene el menor número de pares y por tanto deja un mayor número de variables sin sustituir (en el ejemplo anterior s_1 era más general que s_2 ya que este último se puede obtener del primero aplicando la sustitución $\{b/x\}$). Para crear un UMG se resuelven sucesivamente las discordancias entre los términos que ocupan las mismas plazas entre los predicados que están siendo unificados. En cada paso se toma como referencia una discordancia en un término cualquiera k y se crea un par t_k/v_k en el que t_k no contenga v_k. Dicho par se añade —mediante la operación de composición anteriormente descrita— a la sustitución formada por los pares encontrados en etapas anteriores. Se termina el proceso iterativo cuando se comprueba que el unificador formado hace que se resuelvan todas las discordancias. Veamos dicho proceso mediante un ejemplo.

Partimos de dos expresiones del predicado *Padre*, mostradas en (5.107). El proceso de creación del UMG consiste en:

1. Se busca la primera discordancia entre los términos homólogos de las expresiones de los predicados implicados. El par econtrado *Juan*/x se incluye como primer elemento de la sustitución ω buscada (la sustitución específica de la etapa j se denota por ω_j).

$$\left.\begin{array}{c} Padre(Juan,\ mayor(Antonio,u)) \\ Padre(x,\ mayor(y, Hermano(y, Hijo(x)))) \end{array}\right\rangle \omega_1 = \{Juan/x\} \qquad (5.107)$$

2. Se aplica ω_1 a ambas expresiones y se busca el siguiente par que permita resolver otra discordancia. Se encuentra el par *Antonio*/y y se añade

mediante la operación de composición a la sustitución encontrada en la etapa anterior creando la nueva sustitución $\omega_2 = \{Juan/x, Antonio/y\}$.

$$\left.\begin{array}{c} Padre(Juan, mayor(Antonio, u)) \\ Padre(Juan, mayor(y, Hermano(y, Hijo(Juan)))) \end{array}\right| \omega_2 = \omega_1 \{Antonio/y\} \qquad (5.108)$$

Se aplica la nueva sustitución a ambas expresiones y se busca el siguiente par $Hijo(Juan)/u$ que se añade a ω_2 creando la sustitución $\omega_3 = \{Juan/x, Antonio/y, Hijo(Juan)/u\}$.

$$\left.\begin{array}{c} Padre(Juan, mayor(Antonio, u)) \\ Padre(Juan, mayor(Antonio, Hermano(Antonio, Hijo(Juan)))) \end{array}\right| \omega_3 = \omega_2 \{Hijo(Juan)/u\} \qquad (5.109)$$

Al aplicar ω_3 se obtienen dos expresiones iguales, dándose por concluido el proceso. Por lo tanto el UMG encontrado es $w = \omega_1\omega_2\omega_3$, siendo el resultado de la unificación:

$$Padre(Juan, mayor(Antonio, Hermano(Antonio, Hijo(Juan)))) \qquad (5.110)$$

5.4.3 Método General de Resolución

Este método establece un procedimiento sistemático (por lo tanto computable) de búsqueda de la demostración de un teorema propuesto. Dicha demostración se realiza siguiendo el método lógico de la refutación, es decir, demostrando que si el teorema fuera falso se llegaría a una contradicción.

La única regla de inferencia considerada para avanzar en el proceso de búsqueda es la expresión general de la regla de resolución que esencialmente consiste en lo siguiente: una vez convertidas las premisas y la negación de la conclusión en su correspondientes formas clausuladas, se seleccionan sucesivamente parejas de literales en cláusulas distintas (llamadas *generatrices*) con el mismo nombre —uno negado y otro sin negar—, se comparan —mediante su *unificador mínimo*— y si existe la unificación se aplica la regla básica de resolución, que consiste en crear una nueva cláusula (llamada *resolvente*) formada por la disyunción del resto de los literales de las *generatrices*. Dicha cláusula se añade al conjunto de cláusulas existente y, tomando de nuevo aquél como referencia, se intenta una vez más aplicar el

proceso descrito. La búsqueda termina cuando la cláusula recién formada está vacía.

A continuación se describe detalladamente el método de resolución. Para comprender mejor los pasos implicados conviene observar el ejemplo presentado después del método.

- **Procedimiento General de Resolución:**

1. Convertir todas las premisas P_1, P_2, ..., P_N en su forma clausulada (FNC) e *inicializar* un conjunto de cláusulas P con dichas expresiones.

2. Convertir la expresión resultante de negar la conclusión $\neg C$ en forma clausulada y añadirla al conjunto de cláusulas P.

3. Realizar (3.1 y 3.2) hasta encontrar la cláusula vacía o hasta que no se pueda seguir avanzando (no se haya podido generar ninguna resolvente nueva en 3.2.) o hasta que se hayan agotado los recursos de cálculo (en tiempo o en espacio de memoria fijados inicialmente).

3.1. Inicializar, asignar un valor de referencia, por ejemplo *nulo*, a la variable que contendrá el unificador mínimo encontrado s (ver 3.2.)

3.2. Seleccionar parejas nuevas (no elegidas en etapas anteriores) de **átomos** (*literales negados o sin negar*) con el mismo nombre de predicado en cláusulas distintas. Denominar dichas cláusulas G_1 y G_2 (se analizará si pueden ser *generatrices*) y a la pareja de átomos seleccionada A_1 y A_2 respectivamente.

(1) Buscar el *unificador de máxima generalidad* de A_1 y A_2. Si se encuentra, guardar su valor en s.

(2) Si s tiene un nuevo valor aplicar dicha sustitución a G_1 y G_2 formando su *resolvente* de la siguiente forma:

(2.1.) Si existe en G_1s un elemento A_1 y en G_2s un elemento $\neg A_2$, o en G_1s un elemento $\neg A_1$ y en G_2s un elemento A_2; crear una nueva cláusula (*resolvente*) formada por la disyunción del resto de los elementos de G_1s y G_2s (no se deben eliminar nuevas ocurrencias de la contradicción encontrada entre el resto de elementos; por ejemplo, si hay algún otro A_1 y $\neg A_2$, éstos sí aparecerán en la *resolvente*).

(2.2.) Si la nueva *resolvente* está vacía (las generatrices sólo tenían el literal seleccionado) salir del bucle iniciado en el paso 3. (se ha encontrado la contradicción buscada). En caso contrario añadir la nueva resolvente al conjunto *P*.

Observación: La clave de la eficiencia del proceso de búsqueda realizado está en cuál sea el criterio para seleccionar las cláusulas *generatrices* en el paso 3.2. De las diferentes formas de hacerlo surgen las llamadas Estrategias de Resolución [Nilsson, 1981].Cuando el número de cláusulas es elevado la búsqueda ciega en el espacio de cláusulas puede llegar a ser intratable.

- **Estrategias de Resolución:**

Al igual que hicimos en los capítulos de búsqueda, es útil analizar el proceso de resolución como una búsqueda en un espacio de cláusulas donde en cada momento se tiene un grafo de cláusulas explícito en el que el objetivo es encontrar la cláusula vacía, que representaremos por λ. Las estrategias pueden ir encaminadas bien a limitar el número de cláusulas consideradas en dicho espacio o bien a dirigir el proceso de búsqueda en el mismo. Las primeras que surgieron fueron las orientadas a dirigir el proceso de búsqueda, pero éstas se mostraron claramente insuficientes para problemas reales donde hay que considerar un número elevado de cláusulas (problema de la *explosión combinatoria*). Vamos a discutir brevemente las estrategias básicas de que permiten acelerar el procedimiento de resolución anteriormente descrito:

1. La estrategia que se propuso inicialmente consiste en seleccionar aquellas cláusulas que tengan un único literal [Wos *et al.*, 1964]. Efectivamente, la cláusula vacía se obtiene cuando se resuelven dos cláusulas unitarias contradictorias. Por otro lado, siempre que se resuelva una determinada cláusula con otra unitaria, la resolvente obtenida hará que la primera quede reducida al menos en un literal.

2. Un primer enfoque tendente a disminuir el número de cláusulas consideradas es utilizar algunas de ellas como *conjunto de apoyo* [Wos *et al.*, 1965]. Éstas sirven de referencia para guiar el proceso de búsqueda de modo que siempre se utilice alguna de ellas o alguna de sus descendientes en la generación de nuevas resolventes. El proceso intenta evitar la generación de conclusiones que nada tengan que ver con el objetivo marcado. En realidad, el

método intenta completar un razonamiento sujeto al conjunto elegido incluyendo el resto de las cláusulas para completar los pasos de dicho razonamiento.

Normalmente el planteamiento de un problema concreto (teorema a demostrar, en este contexto) se realiza en un dominio en el cual puede existir una serie de axiomas relativos a una cierta teoría dominante. El principal objetivo del planteamiento realizado mediante el método del *conjunto de apoyo* es seleccionar un conjunto de cláusulas que ayuden a obtener la cláusula vacía en el menor número de pasos posibles. Por lo tanto, sería bueno que, aunque se puedan utilizar los axiomas del dominio como soporte de los argumentos realizados, el proceso de búsqueda intente centrarse en las cláusulas que definen el problema concreto planteado en dicho dominio (el teorema propuesto) y no en completar otros aspectos deducibles de la propia teoría. Por lo tanto, un planteamiento efectivo consiste en hacer que el conjunto de cláusulas elegido contenga hipótesis relativas al teorema que se intenta resolver así como aquéllas obtenidas al negar la conclusión propuesta. Este último planteamiento supone realmente realizar un *encadenamiento hacia atrás* —o *dirigido por las metas*— del proceso de búsqueda (ver capítulos dedicados a la búsqueda).

La estrategia del *conjunto de apoyo* sigue siendo la estrategia de búsqueda más efectiva en los programas de razonamiento automático; tiene además la gran ventaja de ser *completa*, es decir, con la utilización del conjunto seleccionado se garantiza encontrar la cláusula vacía —si esta existe—.

3. Las estrategias llamadas **de simplificación** reducen el número y la forma de las cláusulas consideradas. Pueden distinguirse las siguientes:

(1) Eliminar las tautologías (p.ej., $p \vee \neg p$), ya que el objetivo es llegar a obtener λ.

(2) Eliminar átomos repetidos en las cláusulas, dejando sólo uno de ellos. Esto es válido formalmente ya que se cumple la ley de *idempotencia*:

$$P(x_1, x_2, ..., x_n) \vee P(x_1, x_2, ..., x_n) \leftrightarrow P(x_1, x_2, ..., x_n) \qquad (5.111)$$

Esta estrategia se enmarca dentro de aquéllas que intentan normalizar la expresión de las fórmulas en lugar de borrar o añadir cláusulas al conjunto. Se intentan buscar diferentes formas de reducción de expresiones y de *reescritura* de las mismas. Estas estrategias en sentido general se han denominado *estrategias de demodulación* [Wos *et al.*, 1967].

(3) Se eliminan cláusulas que contienen a otras; es decir, si al aplicar una sustitución a una cláusula se convierte en otra o en parte de otra, se elimina ésta última. Se puede demostrar que el conjunto resultante de eliminar estas cláusulas también es completo. Un ejemplo de este tipo se muestra en (5.112).

$$C_1: \neg Estudiante\_Infor(x) \vee Amigo(x, hermano(x)) \quad (5.112)$$

$$C_2: \neg Estudiante\_Infor(Juan) \vee Amigo(x, hermano(Juan)) \vee Ama(hermano(Juan), arte) \quad (5.113)$$

La cláusula C_1 está incorporada en C_2 (5.113), por lo que ésta última se podría directamente eliminar del conjunto considerado. Es decir, se eliminan consecuencias triviales de la información existente.

- Existen otras estrategias que permiten centrar la búsqueda en las cláusulas más relevantes del dominio de aplicación. Una posibilidad es asignar un cierto peso o valor a los conceptos y a los símbolos manipulados, de tal forma que éstos sirvan para guiar la selección de cláusulas, e incluso para eliminar aquéllas que superen una cierto umbral. Así, se valoran más los predicados que a priori son más relevantes en el dominio. Esta estrategia la denominaremos estrategia de asignación de pesos (originalmente "weighting strategy" [McCharen *et al.*, 1976].

• **Ejemplo:**

Vamos a aplicar la estrategia de resolución para comprobar que una conclusión se deduce de un conjunto de premisas. Utilizaremos un caso sencillo con tan sólo dos premisas. La primera es la expresión ya vista en (5.93), su forma clausulada se obtuvo en (5.102) y su expresión se corresponde con:

$$P_1: \neg\exists x \left(Estudiante\_Infor(x) \wedge \forall y \left(Amigo(x, y) \rightarrow Ama(y, cine) \right) \right) \quad (5.114)$$

La segunda premisa consiste en:

$$P_2: \neg Estudiante\_Infor(Ana) \vee Amigo(Ana, f(Ana)) \vee Ama(f(Ana), cine) \quad (5.115)$$

La conclusión dada es:

$$C: Estudiante\_Infor(Juan) \rightarrow Amigo(Juan, f(Juan)) \quad (5.116)$$

Según el método de resolución, para verificar el razonamiento propuesto habrá que comprobar que $(P_1 \wedge P_2 \wedge \neg C)$ es una contradicción. Para ello, el primer paso será convertir tanto las premisas como la conclusión en sus correspondientes formas clausualadas. El proceso de conversión de P_1 ya se ha

analizado detalladamente y dió lugar a la expresión (5.102), que corresponde a las cláusulas C_1 y C_2 del conjunto que se muestra más adelante. La segunda premisa P_2 no necesita ningún tipo de transformación al estar ya en forma clausulada. *Por tanto, sólo resta* obtener la FNC que corresponde a la negación de la conclusión, cuyo resultado es:

$$\neg C: Estudiante\_Infor(Juan) \wedge \neg Amigo(Juan, f(Juan)) \quad (5.117)$$

Para expresar la negación de la conclusión en forma clausulada hay que generar dos nuevas cláusulas (C_4 y C_5). Por todo ello, el conjunto de cláusulas inicial que finalmente debe ser considerado es el siguiente:

$$C_1: \neg Estudiante\_Infor(x) \vee Amigo(x, f(x)) \quad (5.118)$$
$$C_2: \neg Estudiante\_Infor(z) \vee \neg Ama(f(z), cine)) \quad (5.119)$$
$$C_3: \neg Estudiante\_Infor(Ana) \vee Amigo(Ana, f(Ana)) \vee Ama(f(Ana), cine) \quad (5.120)$$
$$C_4: Estudiante\_Infor(Juan) \quad (5.121)$$
$$C_5: \neg Amigo(Juan, f(Juan)) \quad (5.122)$$

Antes de aplicar el método en sí, veamos si puede reducirse el conjunto de cláusulas inicial teniendo en cuenta las estrategias de simplificación descritas previamente. Lo primero que puede descubrirse en el conjunto de cláusulas anteior es que C_3 incorpora a C_1 (a través de la sustitución $\{Ana/x\}$), por lo que no consideraremos C_3 en el proceso de búsqueda.

Teniendo en cuenta la estrategia que prima las cláusulas unitarias, el algoritmo de búsqueda que refleja el procedimiento de resolución aplicado sería el mostrado en la Fig. 5.2.

La gran ventaja de del método de resolución es que permite simplificar en gran medida el planteamiento computacional del problema de la deducción, ya que se utiliza una notación canónica fácil de manejar, como es la forma clausulada.

La tendencia actual para llevar a cabo una solución eficiente de los problemas de deducción automática es utilizar heurísticas dependientes del dominio de aplicación que ayuden a reducir el espacio de búsqueda realizado.

En los próximos capítulos veremos cómo las reglas y las representaciones estructuradas (redes y marcos) tratan de reducir la complejidad del proceso de búsqueda realizado. Hay que tener presente que el número de declaraciones que definen la teoría de un dominio puede ser muy elevado.

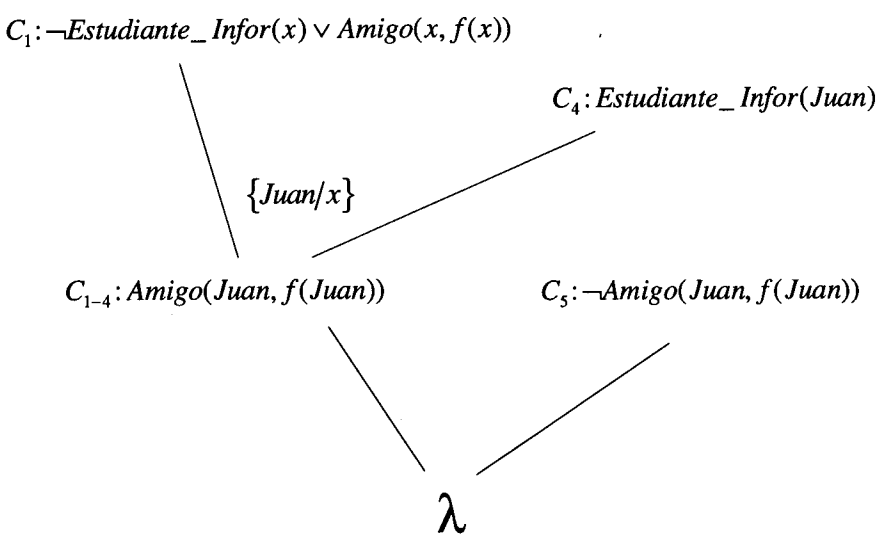


Fig. 5.2. Ejemplo de aplicación del método de resolución

5.5 EXTENSIONES DE LA LÓGICA CLÁSICA

Vamos a introducir nuevos formalismos que permiten ampliar la expresividad (y por consiguiente, las inferencias derivadas de aquélla) de los cálculos lógicos hasta ahora presentados.

Hemos analizado los dos formalismos básicos que forman el eje de las llamadas *lógicas clásicas*—el cálculo de proposiciones y el de predicados—. Estas lógicas responden a una serie de supuestos utilizados en su construcción. Algunos de éstos son: no considerar enunciados de los que no se pueda afirmar su verdad o falsedad, no admitir más que dos valores de verdad y operar *extensionalmente* en las expresiones consideradas. La operación en **extenso**[6] o en **intenso**[7] (comentado en el apartado 7.4.1), se refiere a si se consideran exclusivamente los elementos que forman un determinado conjunto y su valor de verdad (*extensión*) o si se tiene en cuenta el concepto que designa (*intensión*: en

[6] La acepción castellana del término *extenso* (y también de *extensión*) es "conjunto de individuos abarcados en una idea", que coincide plenamente con el sentido generalmente empleado en IA.

[7] El sentido *computacional* de la palabra *intenso* (y también de *intensión*) es un neologismo con respecto a su significado en castellano. Una definición *intensional* de un concepto se puede traducir en una serie de reglas que comprueban si un elemento dado está *abarcado* o no por dicho concepto.

un predicado sería la propiedad significada por el nombre del predicado). Es decir; desde el punto de vista *intensional* dos predicados pueden ser diferentes aun cuando estén definidos sobre los mismos elementos del dominio. Por lo tanto, la teoría intensional de tipos (*conceptos*) estará basada en la distinción entre predicados monádicos y clases.

La distinción entre *intensión* y *extensión* es esencial a la hora de valorar adecuadamente las capacidades de cualquier sistema computacional. La manipulación sintáctica a la que está sujeto un sistema que pretenda realizar algún tipo de razonamiento (la lógica como sistema formal de cálculo también lo cumple) requiere —en último término— la traducción sintáctica de la definición intensional implícita de los conceptos manipulados[8].

En los siguientes epígrafes vamos a introducir algunas de las principales lógicas no-clásicas centrándonos —una vez más— en las capacidades de representación e inferencia aportadas por dichos formalismos.

5.5.1 Lógica Modal

Los enunciados —y por tanto los predicados utilizados en las lógicas clásicas— sólo podían ser interpretados como verdaderos o falsos. No se permitían expresar matices o modalidades de la verdad o falsedad adscrita a un enunciado. La lógica modal surge para satisfacer la necesidad de manipular adecuadamente los conceptos de *necesidad* y *posibilidad.* Cualifica los enunciados señalando si estos son *posibles*, *imposibles*, *necesarios* o *contingentes* (pueden suceder o no).

La conveniencia de realizar inferencias sujetas a nociones modales ya fue señalada por Aristóteles. Lewis realizó una primera formalización matemática [Lewis & Langford, 1932] y puso de relieve la necesidad de refinar la semántica de la regla del condicional "→". Kripke en [1959] propuso una semántica para los conceptos modales. Más recientemente, Manna y Pnueli [1980] han aplicado la semántica de Kripke a la teoría de la programación. Igualmente se ha aplicado en inferencia no monótona y en representación del conocimiento [Moore, 1984].

[8] En el tema dedicado al aprendizaje veremos formas efectivas de tratar la *definición intensional* de conceptos.

La simbolización básica de la lógica modal es una generalización de la sintaxis del cálculo clásico, se introducen los siguientes símbolos:

$\Box P$ "es necesario que P" (5.123)
$\Diamond P$ "es posible que P" (5.124)

Los nuevos operadores "$\Diamond$" y "$\Box$" tienen el mismo nivel (ámbito de aplicación) que tiene el operador negación "$\neg$" del cálculo de la lógica de primer orden en las expresiones en que éstos aparecen. Al igual que ocurría en dicha lógica, cada uno de los operadores añadidos puede definirse en función del otro —por lo que podría haberse definido un único operador (la distinción se hace por claridad)— ya que se cumplen las siguiente ley de equivalencia:

$\Diamond P \leftrightarrow \neg\Box \neg P$ "es posible que P ≡ no es necesario que no-P" (5.125)

(5.125) también puede expresarse como:

$\Box \neg P \leftrightarrow \neg\Diamond P$ "es necesario que no-P ≡ es imposible que P" (5.126)

En este formalismo, Lewis define la implicación en sentido estricto (lo representaremos por "$\Rightarrow$").

$$R \Rightarrow S \equiv \neg\Diamond (R \wedge \neg S) \quad (5.127)$$

La definición (5.127) expresa —mediante la noción de posibilidad— una relación *más fina* de *deducibilidad*, donde se quiere romper con la definición más amplia hasta ahora vista de *implicación material* con la que se cumplía la tautología siguiente:

$$\vdash (R \rightarrow S) \vee (R \rightarrow \neg S) \quad (5.128)$$

En otras palabras, un objetivo de la lógica modal es llegar a expresar la relación de necesidad entre premisas y conclusiones en el razonamiento deductivo. Es decir, de una premisa R se sigue necesariamente una conclusión S. He aquí, por lo tanto, un caso que demuestra que la lógica requiere matizaciones modales.

Otro caso donde aparece la modalidad es en el tratamiento del *tiempo*. Por ejemplo, frases como "es posible que vayamos al cine mañana" o "siempre habrá buenas personas" expresan la necesidad de distinguir cuándo una determinada afirmación se considera posible. Cada una de las etapas en que se *discretice* el tiempo tendrá un valor de verdad para una determinada afirmación.

Para poder realizar la interpretación de fórmulas modales se introduce la idea de **mundos** (dominios parciales). Los mundos son subdivisiones del universo de discurso, de tal forma que en un determinado mundo un predicado puede ser verdad y en otro falso. Dado que el universo se divide en mundos, habrá que considerar una *relación de accesibilidad* entre éstos. Para que la necesidad de un predicado sea verdad tendrá que ser verdad en todos los mundos accesibles desde el mundo en que se esté evaluando.

Dependiendo de que la relación entre los mundos sea sólo reflexiva, reflexiva y transitiva o de accesibilidad total, se definen leyes básicas correspondientes a diferentes sistemas axiomáticos en lógica modal proposicional. Por ejemplo, el primer axioma del sistema en el que la relación es reflexiva es el siguiente:

$$\vdash \Box P \rightarrow P \tag{5.129}$$

La demostración de la validez de (5.129) se basa en comprobar que no puede ocurrir que P sea falso y a su vez $\Box P$ sea verdadero, ya que esto supondría que P sería falso en algún mundo y como en dicho mundo la relación de accesibilidad es reflexiva esto implicaría que $\Box P$ también sería falso.

De forma análoga a lo visto en el cálculo proposicional clásico se pueden demostrar las propiedades de *consistencia*, *completitud* y *decidibilidad*.

En el caso del cálculo de predicados modal —un ejemplo de expresión en este cálculo podría ser $\Box \exists x P(f(x, y), z, u)$— la interpretación consiste en suponer un conjunto de mundos, m_1, m_2, ..., m_n, para un mismo conjunto de objetos[9] que identifican el dominio de referencia. Al igual que en el cálculo de predicados clásico, las letras de constante y variable se corresponden con elementos del dominio; sin embargo, las letras de función y los valores de cada predicado dependen del mundo considerado. La inclusión de operadores modales permite igualmente ampliar el sistema axiomático del cálculo de predicados (su estudio queda fuera de los objetivos del capítulo presente[10]).

Otro campo donde se ha aplicado la lógica modal ha sido en la programación. Para representar la *secuencialidad* a la que están sujetas las

[9] Se puede eliminar esta restricción. Dado un dominio —por ejemplo el de los ordenadores— habría que añadir nuevos objetos al dominio (p.ej., portátiles) a lo largo de los años (mundos en nuestro esquema de representación)

[10] Un texto donde se puede obtener una introducción al cálculo modal —además de los generales de lógica señalados al principio de este capítulo— es [Hughes & Cresswell, 1968].

etapas por las que puede pasar un determinado algoritmo se establecen mundos diferentes que reflejan los estados de las variables del programa en cada una de dichas etapas. La relación de accesibilidad la definiría la secuencia del flujo del proceso.

5.5.2 Lógica Difusa

El valor de verdad de un razonamiento puede depender de enunciados como "los *más altos* de la clase juegan al baloncesto" o como "*es bueno* ser *adulto*" o como "cuando sean las 5 *aproximadamente*" o como "algunos niños son *muy avispados*". La lógica, por lo tanto, no puede permanecer ajena al hecho de que la imprecisión es algo intrínseco a una gran parte del lenguaje ordinario[11].

En las *lógicas polivalentes* es posible distinguir más de dos valores de verdad. Por ejemplo, en la lógica trivalente de Lukasiewicz[12], los enunciados pueden ser verdaderos "1", falsos "0" e indeterminado "1/2". Esto hace que en dichas lógicas se varíen algunas propiedades formales esenciales; por ejemplo, todos los teoremas llamados de *reducción al absurdo* dejan de ser válidos, ya que se basan en la validez de la ley del *tercio excluso* "$p \vee \neg p$", que deja de ser una tautología fuera de la lógica bivalente. Más allá de estas lógicas existe la denominada *lógica de conjuntos difusos* o *lógica difusa*, o mejor, *de enunciados vagos*, ya que la imprecisión no está en la lógica sino en los enunciados tratados. En las lógicas polivalentes cada enunciado puede tener un valor de verdad matizado, pero ese valor tiene que ser uno concreto y no puede depender del contexto. La lógica difusa [Zadeh, 1965, 1975] permite afirmar que los enunciados son más o menos verdaderos en un contexto determinado y más o menos falsos en otro contexto diferente.

En el lenguaje natural, lo normal es que los conjuntos —y los conceptos que estos representan[13]— no estén perfectamente definidos. La lógica difusa trata esta imprecisión definiendo dichos conjuntos mediante la indicación del grado de pertenencia de sus miembros. Así, una persona con 20 años puede

---

[11] En los próximos temas veremos como los llamados *sistemas expertos* tampoco escapan a la presencia de la *imprecisión*.

[12] Lukasiewicz generalizó su lógica a *n* valores de verdad {0, 1/n-1, 2/n-2, ..., n-2/n-1, 1}; incluye el planteamiento de lógicas con infinitos valores de verdad.

[13] En el capítulo dedicado al aprendizaje veremos técnicas que se basan en el paralelismo existente entre la noción de concepto y de conjunto.

tener un grado de pertenencia al conjunto difuso de los jóvenes de 0.8 y una de 30 de 0.5. Por lo tanto, para definir formalmente un conjunto difuso se define el valor de la llamada **función de pertenencia** que determina el grado de pertenencia de cada elemento del universo de discurso a dicho conjunto.

La definición matemática de un conjunto difuso P es la siguiente:

$$P = \left\{ x \mid \mu_P(x), \forall x \in U \wedge \mu_P(x) \neq 0 \right\} \tag{5.130}$$

Por ejemplo, sea el conjunto difuso "exceso de trabajo" (ver 5.131) definido sobre el universo de discurso de las horas trabajadas diariamente.

horas-por-día = {7, 8, 9, 10, 11, 12}

exceso-trabajo = {7|0.25, 8|0.5, 9|0.75, 10|1, 11|1, 12|1} (5.131)

Muchas veces es útil representar gráficamente este tipo de conjuntos, sobre todo para distinguir como afecta a su definición la aplicación de los llamados *modificadores difusos* que además del modificador "no" de la lógica clásica incluye "muy, más o menos, aproximadamente, bastante, ... etc." (veremos más adelante algún ejemplo).

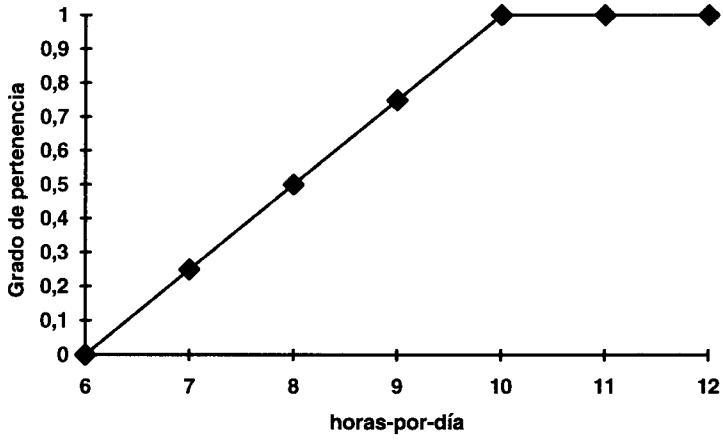


Fig. 5.3. Representación gráfica de un conjunto difuso

Las funciones de pertenencia también pueden ser continuas. Por ejemplo, se puede definir el conjunto difuso determinado por los números reales y por una función de pertenencia que permita trazar el grado de pertenencia de un rango

continuo de valores (ver 5.132). La representación gráfica de dicho conjunto sería una campana de Gauss con un campo de convergencia en el intervalo (-1, +1).

$$\mu(x) = \frac{1}{1 + x^2} \tag{5.132}$$

Todas las operaciones de interpretación e inferencia realizadas en el cálculo difuso se reducen a operaciones aritméticas definidas sobre los valores de la función de pertenencia de los elementos del dominio considerado (o de los dominios, si se trata de una relación).

Una relación difusa permite representar el grado de pertenencia de los elementos de dos o más conjuntos a una determinada asociación o interconexión. Por ejemplo; sean los conjuntos no difusos A = {Santander, Oviedo} y B = {Santander, Cuenca} sobre los que se ha definido la relación difusa $R(x, y)$ = "x está lejos de y", expresada mediante la matriz siguiente (las matrices son una forma útil de representar y manipular relaciones binarias).

		Santander	Cuenca
R:	Santander	0	0.5
	Oviedo	0.2	0.6

(5.132)

A continuación mostramos la notación correspondiente como conjunto difuso:

$$R(A, B) = \{(\text{Santander, Cuenca})|0.5, (\text{Oviedo, Santander})|0.2, (\text{Oviedo, Cuenca})|0.6\} \tag{5.133}$$

Al igual que señalamos que los predicados poliádicos representaban relaciones, las relaciones difusas son conjuntos difusos definidos sobre el producto cartesiano de los universos de discurso de los dominios que participan en la relación.

Aunque los valores de verdad de los conjuntos difusos se toman como referencia en el intervalo [0, 1], la suma de los valores de verdad no está sujeta a ningún tipo de restricciones. Por lo tanto, esto no debe confundir dicha interpretación con el cálculo probabilista, donde la suma de las probabilidades tiene que ser igual a la unidad.

Para interpretar sentencias se tiene en cuenta que el significado de un predicado viene definido por el conjunto difuso correspondiente, que en realidad refleja el grado de pertenencia de los elementos del universo de discurso (o de

los universos de discurso si es una relación) al significado señalado en tal predicado. Si quisiéramos interpretar el significado de la expresión "Juan tiene exceso de trabajo" primero tendríamos que saber el número de horas que trabaja Juan, es decir; tendríamos que establecer una correspondencia entre el sujeto del que se afirma algo y el universo de discurso sobre el que se define la propiedad afirmada. Por lo tanto, si Juan trabaja 8 horas el valor de verdad de la expresión señalada es 0.5 ya que es el valor de la función de pertenencia para dicho número de horas.

La interpretación de los enunciados que aquí vamos a utilizar está referida a la interpretación verdadera de las sentencias. De esta forma, la utilización de los modificadores difusos nos va a permitir afirmar si una proposición puede ser muy cierta, cierta, algo cierta, etc. Dado que el significado de un predicado se obtiene de los valores correspondientes de la función de pertenencia, los modificadores —cuya función es variar en un cierto grado la definición del conjunto de referencia— se interpretan mediante la aplicación de modificaciones (funciones numéricas) a dichas funciones de pertenencia (significados). Por ejemplo, la interpretación (que mencionaremos como "I") del modificador "algo" es la indicada en (5.134).

$$I(\text{algo } P(x)) = DIL\ (\mu_P(x)) = 2\mu_P(x) - \mu_P{}^2(x) \qquad (5.134)$$

En la expresión anterior, el modificador "algo" se define en base a una operación más básica denominada *dilatación* (operador "DIL"). Existen otras operaciones básicas; por ejemplo la negación (operador "NEG") es la operación básica utilizada para interpretar el modificador "no".

$$I(\neg P(x)) = NEG\ (\mu_P(x)) = 1 - \mu_P(x) \qquad (5.135)$$

Por lo tanto, la interpretación del enunciado "Juan tiene algo de exceso de trabajo" se podría obtener representando la función de pertenencia del conjunto difuso resultante de aplicar el modificador "algo" (se considera sinónimo de "más o menos") a la función de pertenencia del conjunto *exceso-trabajo* (ver Fig. 5.4.).

Como puede apreciarse, sobre todo en la Fig. 5.4., la operación de dilatación amplía ligeramente —como era de esperar— el grado de pertenencia de los elementos del dominio al conjunto difuso original (señalado mediante una línea discontinua).

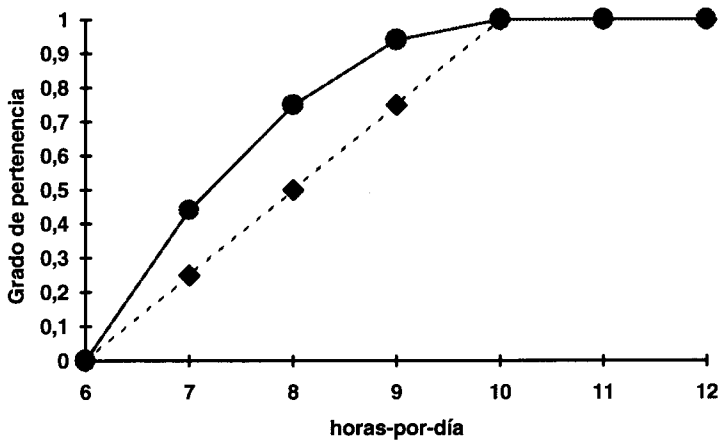


Fig. 5.4. Representación de I("algo de exceso-trabajo")

El conjunto difuso mostrado en la figura anterior puede expresarse como:

$$I(\text{algo } \mu_{\text{exceso-trabajo}}(x)) = \{7|0.44, 8|0.75, 9|0.94, 10|1, 11|1, 12|1\} \quad (5.136)$$

La interpretación de sentencias compuestas se traduce igualmente en la realización de operaciones sobre las funciones de pertenencia. Por ejemplo, si quisiéramos interpretar la frase disyuntiva siguiente: "Juan tiene exceso de trabajo" *o* "dispone de poco tiempo libre" (suponemos que en su definición sólo se contabilizan las horas de esparcimiento), habría que considerar las definiciones de las funciones de pertenencia de los dos conjuntos difusos implicados.

$$\text{poco-tiempo-libre} = \{1|1, 2|1, 3|0.25, 4|0.15, 5|0.05\} \quad (5.137)$$

Para poder interpretar la frase señalada se tiene que definir operativamente la interpretación de la conectiva "o" en el cálculo difuso (si los universos de discurso son distintos —que es el caso más general— equivale a la definición de una relación difusa).

$$I(P(x) \vee S(y)) = \max(\mu_P(x), \mu_S(y)) \; \forall x \in U, \forall y \in V \quad (5.138)$$

Suponiendo que Juan tiene 3 horas al día libres, la interpretación sería:

$$I = \max(\mu_{\text{exceso-trabajo}}(8), \mu_{\text{poco-tiempo-libre}}(3)) = \max(0.5, 0.25) = 0.5 \quad (5.139)$$

Antes de introducir los procesos de inferencia conviene explicar la interpretación del condicional —sobre todo dado el paralelismo existente en la lógica clásica entre la definición de deducción y la del operador condicional (por el teorema de la deducción)—. Como otra conectiva cualquiera, la interpretación de "$\rightarrow$" consiste en la aplicación de una operación sobre las funciones de pertenencia de los conjuntos difusos implicados (aunque existen varias definiciones, tomamos la más utilizada). Ocurre además que el valor de esta interpretación coincide con la definición del producto cartesiano de conjuntos difusos ($P \times S$, ver 5.140).

$$I(P(x) \rightarrow S(y)) = \min (\mu_P(x), \mu_S(y)) \; \forall x \in U, \forall y \in V \qquad (5.140)$$

Otra operación básica que aparece en la definición de algunas de las reglas de inferencia es la de *composición* (representada por el símbolo "$\circ$"); que es una operación propia de las relaciones difusas. Para definirla, suponemos que tenemos dos relaciones difusas *P* y *B* definidas sobre los universos (U,V) y (V, W) respectivamente; la *relación binaria compuesta R* entre *U* y *W* se expresa según lo señalado en (5.141).

$$R(U, W) = P(U, V) \circ B(V, W) = \left\{(x,z) \middle| \mu_{P \circ B}(x,z), \forall x \in U, \forall y \in V, \forall z \in W\right\} \qquad (5.141)$$

$$\text{siendo } \mu_{P \circ B}(x, z) = \max_{y \in V} \left[\min \left(\mu_P(x,y), \mu_B(y,z)\right)\right] \qquad (5.142)$$

La forma más sencilla de manipular las operaciones entre relaciones es representarlas por medio de matrices. De ahí que la composición se conozca igualmente por el sobrenombre de *producto matricial max-min*. Es decir, como puede observarse si se desarrolla la definición (5.142), la composición no es más que el producto de matrices donde se ha cambiado la operación *suma* por la operación *máximo* y el *producto* por el *mínimo* (su cálculo responde intuitivamente al valor del flujo máximo de líquido que puede circular entre dos puntos unidos por varias tuberías de diferentes diámetros).

Para comprender mejor la operación de *composición* planteemos un caso concreto. Supongamos que además de la relación *R* = "lejanía entre ciudades" mostrada anteriormente en (5.132) disponemos de la relación difusa *S* = "tipo de tráfico en ciudad" definida a partir del conjunto no difuso *C* = "tipo de tráfico" = {congestionado, normal, fluido}.

		congestionado	normal	fluido	
S:	Santander	0.8	0.6	0.2	(5.143)
	Cuenca	0.3	0.4	0.7	

La operación de *producto matricial max-min* entre las matrices mostradas en (5.132) y (5.143) respectivamente es la siguiente:

		congestionado	normal	fluido	
$R \circ S$:	Santander	0.3	0.4	0.5	(5.144)
	Oviedo	0.3	0.4	0.6	

La relación $R \circ S$ se podría interpretar como "lejanía a una ciudad de un determinado tipo de tráfico"

Al igual que en las lógicas clásicas existen una serie de reglas de inferencia que permiten obtener el valor de verdad de una determinada conclusión a partir de unas premisas en el cálculo difuso. Una de dichas reglas es la *regla de intersección*, que se expresa mediante el esquema de inferencia (5.145):

$$\begin{array}{l} P(x) \\ S(x) \\ \hline (P \cap S)(x) \qquad \forall x \in U, P \subset U, S \subset U \\ \text{siendo } \mu_{P \cap S}(x) = \min(\mu_P(x), \mu_S(x))^{14} \end{array} \qquad (5.145)$$

La regla anterior debe interpretarse igual que hacíamos en el el cálculo de predicados (p.ej., disponíamos de una regla de inferencia que indicaba que de $P(x)$ y $S(x)$ se podía inferir $P(x) \wedge S(x)$). La inferencia realizada consiste en afirmar que la conclusión obtenida a partir de la existencia de las premisas $P(x)$ y $S(x)$ (conjuntos difusos definidos sobre el mismo universo de discurso) es $(P \cap S)(x)$ y su valor es siempre menor o igual al de dichas premisas, cumpliéndose además esta condición para toda estructura deductiva correcta.

Como puede apreciarse, las reglas de inferencia de la lógica difusa utilizan operaciones max/min para la propagación de evidencias en los caminos existentes en una deducción.

[14] En lógica difusa la conjunción se interpreta mediante la intersección "$\cap$" y la disyunción mediante la unión "$\cup$" (el valor de ésta última sería el máximo de los valores de las funciones de pertenencia involucradas)

Otro ejemplo representativo de dichas reglas es la *regla de composición*, cuya definición responde al esquema de inferencia (5.146).

$$\begin{array}{l} P(x) \\ S(x, y) \\ \hline (P \circ S)(y) \qquad \forall x \in U, \forall y \in V, P \subset U, S \subset U \times V \end{array} \tag{5.146}$$

$$\text{siendo } \mu_{PoS}(y) = \max_{x \in U} \left[\min \left(\mu_P(x), \mu_S(x, y)\right)\right]$$

La regla de composición indica que de un conjunto difuso y de una relación difusa se puede obtener un conjunto difuso asociado a la operación de composición entre ambos. Retomando el ejemplo de las ciudades (5.143) y utilizando la regla de *composición* (5.146), de las afirmaciones "Santander es una ciudad cara" (habría que definir el conjunto difuso "caro" sobre el universo de ciudades) y "Cuenca está lejos de Santander", se podría concluir que "Cuenca está lejos de una ciudad cara" con un valor calculado según la definición anterior de $\mu_{PoS}(y)$.

Siguiendo con la disminución progresiva de las restricciones impuestas por los distintos formalismos lógicos, vamos presentar en el siguiente apartado un tipo de lógicas especialmente relevantes para los sistemas de IA.

5.5.3 Lógicas No Monótonas

Los sistemas de IA cuya representación del conocimiento y cuyos métodos de inferencia sean fundamentalmente lógicos se enfrentan a una serie de condicionamientos desde el punto de vista práctico y metodológico.

Una de las dificultades —ya señaladas— del cálculo de predicados es la excesiva rigidez a la que están sujetos los razonamientos realizados. Para que el conocimiento del domino pueda ser *efectiva* y *eficientemente* formulado en forma de una teoría —descrita mediante leyes que reflejan las relaciones válidas entre los elementos del dominio— es necesario que ésta sea *completa*, *consistente* y además *tratable*. Para garantizar la completitud, toda fórmula válida debe ser demostrable; es decir, no pueden haberse cometido ningún tipo de omisiones (por falta de conocimiento, por olvido, o por una formulación incorrecta) en el conjunto de leyes inicialmente formuladas en el sistema. La consistecia requiere que todo aquello que se añada al sistema no debe contradecir a las leyes existentes; es decir, no puede ocurrir que se lleguen a conclu-

siones diferentes bajo las mismas condiciones iniciales —leyes y premisas—. Finalmente, la tratabilidad se traduce en que la complejidad del cálculo derivado de la manipulación de dichas leyes y premisas en los procesos de inferencia no sea excesiva (requerimiento para poderse implementar que depende del dominio y la tarea). Parecen —en principio— demasiados condicionamientos que restringen la cantidad de problemas resolubles mediante una formulación basada exclusivamente en el cálculo de predicados.

Quizás sea el requerimiento de *consistencia* una de las limitaciones que más contrastan con las situaciones existentes en la mayoría de los problemas del mundo real. Suele ocurrir que nuestra percepción de muchos problemas varía con el tiempo —surgen nuevas evidencias— y muchos de los matices —a veces incluso aspectos fundamentales— que inicialmente no percibíamos pueden llevarnos a reconsiderar algunos de los principios o leyes que inicialmente creíamos —y utilizábamos como— correctos. Esta misma circunstancia ocurre en el *razonamiento basado en el sentido común*, donde se aplican principios que generalmente se suponen válidos en el dominio, pero que pueden dejar de serlo en situaciones concretas. Este tipo de razonamiento es el atribuido a las inferencias que se realizan teniendo en cuenta el conocimiento básico del mundo que poseen hasta los niños más pequeños; se aplica fundamentalmente en los siguientes campos: *razonamiento espacial*, *razonamiento temporal*, *fenomenología física*, *estudios sobre magnitudes* (datos físicos: temperatura, masa, ..., datos espaciales: longitudes, ángulos, ..., datos temporales: duraciones, etc.), *agentes autónomos* (capítulo 10) e *interacciones interpersonales* (entre personas y entre grupos de personas).

Para resolver estos problemas en un formalismo lógico, se plantea una formulación del problema menos rígida. No se impone la restricción de que nada de lo que se añada al sistema puede entrar en contradicción con aquél, sólo se exige que las conclusiones que en un momento dado se hayan obtenido sí sean consistentes con las condiciones de los problemas tratados y con las leyes aplicables en dichos problemas. Por lo tanto, el problema al que nos estamos enfrentado es realmente la *consideración de la temporalidad* implícita en muchos de los razonamientos lógicos.

La raíz del problema estriba en las carencias existentes en la teoría del dominio inicialmente conocida. Para resolver problemas para los que se carece de una teoría sólida y contrastada, se crean reglas o leyes que permiten realizar un tipo de razonamiento denominado **razonamiento por defecto** (traducción del término original *default reasoning*). Se razona por defecto cuando se utilizan

leyes que expresan *suposiciones* o *conjeturas*, de carácter más o menos genérico, que permiten obtener conclusiones de carácter condicionado o incierto. Es decir, al contrario de lo visto en las lógicas clásicas; las conclusiones de los razonamientos ya no son verdaderas o falsas "per se", sino que *dependen del resto de hechos y leyes.*

Este tipo de razonamientos es fácilmente percibible en situaciones de la vida cotidiana. Por ejemplo, supongamos la existencia de una ley que establece un supuesto generalmente válido; "los seres humanos andan". Si apareciera "Gonzalo" —un ser humano—, podría concluirse el hecho "Gonzalo anda". Pero existen multitud de excepciones que pueden hacer inconsistente dicha conclusión establecida a priori. Supongamos que Gonzalo tiene sólo unos meses de vida. Si modificamos dicha ley diciendo "la mayoría de los seres humanos andan" o "hay una probabilidad del 85% de que los seres humanos anden" también podríamos caer en inconsistencias. ¿Qué pasaría si estuviéramos aplicando las leyes de nuestro sistema en una población de bebés, o en un hospital, o en una residencia de ancianos, o en un colegio de educación especial?. Lo mismo podríamos decir de frases como "las aves vuelan", "en invierno hace frío", ...etc. Es decir, dependiendo de los casos concretos, la generalización inicialmente asumida deja de ser cierta. La raíz del problema está en partir de una información incompleta.

Una de las posibles soluciones al razonamiento por defecto es la aplicación de las denominadas **lógicas no monótonas** (Minsky introdujo este término en el contexto del *razonamiento basado en el sentido común* [Minsky, 1975], que comentaremos más adelante).

En las lógicas clásicas comentadas anteriormente, se cumple el siguiente esquema de razonamiento:

$$\frac{\vdash P,\ \vdash P \rightarrow C}{\vdash P \vee W \rightarrow C}$$

($\vdash$, símbolo de fórmula válida)

Es decir, si es válido un enunciado P y de P se puede obtener C; o lo que es equivalente: si $P \rightarrow C$ es válido, entonces, de $P \vee W$ también se llegaría a obtener C (recordar el significado no excluyente de la contectiva "$\vee$"); es decir, $P \vee W \rightarrow C$ también es válido.

En las *lógicas no monótonas* el surgimiento de una nuevo evento puede invalidar el esquema anterior de razonamiento, es decir:

$$\frac{\vdash P,\ \vdash P \rightarrow C}{\vdash P \vee W \rightarrow C} \qquad (\vdash, \text{símbolo de fórmula válida}) \qquad (5.147)$$

La inconsistencia surgida en (5.147) es debida al hecho de que la relación de implicación lógica inicialmente supuesta entre P y C: $P \rightarrow C$, no era rigurosamente cierta, sino que respondía más bien a una suposición *plausible* teniendo en cuenta los datos disponibles en el dominio en el momento en que ésta fue establecida. Es decir, era una afirmación que debería haberse considerado con una vigencia limitada o restringida a la existencia de nuevos hechos que pudieran entrar en contradicción con ella. En otras palabras, las *lógicas clásicas* partían del carácter no excluyente de los nuevos axiomas añadidos a los ya existentes. Por el contrario, las *lógicas no monótonas* tienen en cuenta la necesidad de detectar posibles inconsistencias con los nuevos axiomas. Por tanto, el rasgo definitorio de estas últimas es que se tiene en cuenta *lo que no se conoce*, o lo que es lo mismo *asume los límites de su propio conocimiento.*

Un primer formalismo para abordar este tipo de problemas es el que originalmente fue simplemente denominado **lógica no-monótona** [McDermott & Doyle, 1980]. El objetivo de este cálculo es poder representar leyes como "Si x es un ser humano, entonces x puede andar, a menos que haya algo que lo contradiga". Para ello se amplía la lógica de primer orden introduciendo el *operador modal M* (es modal ya que indica una modalidad de verdad). Así, la fórmula Mp se considera una fórmula correcta (*fcf*). Veamos un ejemplo.

1. $verano \wedge \mathrm{M}\, hacecalor \rightarrow hacecalor$ (5.148)
2. $verano$ (5.149)
3. $hay\_aireacondicionado \rightarrow \neg hacecalor$ (5.150)

El operador modal Mp expresa "p es consistente", es decir, si $\neg p$ no es cierto, entonces se puede concluir p. Por lo tanto, de (5.148) y (5.149) se puede concluir *hacecalor*, pero el sistema debe considerar y tratar adecuadamente la circunstancia asociada a dicho supuesto. Es decir, es necesario establecer un mecanismo de **mantenimiento de coherencia** —generalmente traducido como *mantenimiento de verdad* ("truth maintenance")— que permita eliminar el supuesto en cuanto se presente un hecho que lo invalide. Por ejemplo, *hay_aireacondicionado* podría invalidar *hacecalor*.

4. *hay_aireacondicionado* (5.151)

Debido a que (4) no estaba en la teoría disponible (formada por los hechos, las leyes y las reglas de inferencia existentes en un momento dado), el supuesto *hacecalor* se consideraba como una conjetura válida. A partir del momento en que (4) se conoce, deja de ser cierta la posibilidad de considerar como válido dicho supuesto. Esta es la razón por la que otros autores [Kowalski, 1979; Israel, 1980] han criticado el enfoque de Doyle argumentando que realmente, dado que incluso (5.148) deja de ser ya cierto a raíz de la existencia de (5.151), lo mejor sería afrontar el problema como si de dos lógicas diferentes se tratase. La segunda lógica es más completa (*informada*) que la primera, ya que matiza una ley de aquélla (tendría en cuenta que "si hay aire acondicionado, aun siendo verano, no hace calor").

Un segundo formalismo que ha intentado abordar el problema de la *no monotonicidad* (cuando la adición de nuevos hechos provoca inconsistencias con los hechos ya existentes) es la llamada **lógica por-defecto** (originalmente *default logic*, abreviadamente DL) [Reiter, 1980]. Es un enfoque muy parecido al anterior, pero el operador *M* ya no hace la función de un operador modal capaz de formar sentencias M*p* supuestamente válidas en el sistema; en su lugar dicho operador sólo aparece en las reglas de inferencia denominadas *reglas por-defecto* definidas al efecto.

$$\frac{verano(x):\ M hacecalor(x)}{hacecalor(x)} \qquad (5.152)$$

Al contrario de la *lógica no-monótona*, Reiter no amplía la lógica de primer orden, sino que se limita a crear reglas que permiten establecer una relación de dependencia entre los supuestos inferidos (*hacecalor*(*x*)) y las condiciones bajo las que han sido inferidos. Al igual que en lógica de predicados, en la regla de inferencia (5.152) se considera cierta la conclusión cuando son ciertos los antecedentes, para validar el razonamiento se define un mecanismo de prueba que permite averiguar si un determinado hecho proviene o no de *reglas por-defecto* como la que acabamos de presentar.

Otros formalismos donde se manifiesta el problema del *razonamiento por defecto* son el *razonamiento por herencia* (ver capítulo 7) y el *razonamiento abductivo* (ver capítulo 10).

Además de las lógicas *no monótonas* existen tratamientos numéricos, algunos de ellos basados en el cálculo estadístico (ver capítulo dedicado a los sistemas expertos) que se han aplicado con éxito para abordar el problema de la incertidumbre en los razonamientos.

Los enfoques que hemos presentado sobre el *razonamiento no monótono* se pueden tildar de **permisivos** en el sentido de que posibilitan la obtención de suposiciones cuando no existen evidencias que las contradigan. También puede realizarse un enfoque **restrictivo** del problema donde sólo se considera válido lo que se ha comprobado como cierto, siendo dicha definición de lo que se considera válido una definición que se va ampliando según aparezcan nuevas evidencias. Un ejemplo de este tipo es la consideración de la **circunscripción de un predicado** [McCarthy, 1980]. Circunscribir un predicado consiste en comprobar que si todas las ocurrencias del mismo tienen una propiedad, entonces todos los objetos que cumplan dicho predicado también tienen dicha propiedad. Por ejemplo, en el mundo de los bloques utilizado por McCarthy, circunscribir el predicado *es-bloque* a la sentencia "es-bloque(a) $\wedge$ es-bloque(b) $\wedge$ es-bloque(c)" consiste en comprobar que si todas las ocurrencias de dicho predicado "*a*, *b* y *c*" tienen la propiedad *ser-idéntico-a* (en este caso "*a*, *b* y *c*" cumplen el ser idénticos a ellos mismos), entonces todos los objetos que cumplan *es-bloque* deben también cumplir dicha propiedad. Es decir, la definición de bloque en tal caso sería "ser idéntico a *a*, *b* o *c*" pero si apareciera un nuevo bloque *d* —que también cumple dicha propiedad— habría que redefinir *es-bloque* diciendo que ser bloque es ser "*a*, *b*, *c*, o *d*". Esta forma de razonar proviene de utilizar el principio de la "navaja de Occam" (de uso frecuente en los sistemas de aprendizaje, ver capítulo 10) que consiste básicamente en evitar artificiosidades y cosas superfluas intentando siempre mantener la sencillez de la esencia de las cosas, en este contexto: "sólo las cosas requeridas para comprender la situación".

5.6 CONCLUSIONES

El objetivo de este capítulo no ha sido entrar en el debate clásico sobre la conveniencia o no de utilizar la lógica como herramienta básica en la representación del conocimiento e inferencia (ver [Newell, 1981; Moore, 1982; Brachman *et al.*, 1983]). Hemos optado por describir las virtudes *expresivas* e *inferenciales* de algunos de los formalismos lógicos más extendidos, pudiéndose extraer del análisis realizado las siguientes conclusiones genéricas:

- La lógica surge como intento de abstraer, formalizar y hacer calculables las propiedades inferenciales implícitas en el razonamiento expresado a través del lenguaje.

- Los diferentes cálculos lógicos intentan dar satisfación a los distintos grados de complejidad expresiva e inferencial requerida en los problemas del mundo real.

- Existen métodos de inferencia sistemáticos para un número significativo de problemas.

- La efectividad de los métodos de inferencia depende del grado de conocimiento del dominio y la tarea.

Finalmente, señalaremos que, la complejidad inherente a muchos de los sistemas de IA hacen aconsejable combinar las técnicas vistas en el capítulo presente con otras que analizaremos a lo largo de los próximos capítulos. Sin embargo, existen ejemplos de sistemas guiados por principios exclusivamente lógicos (p.ej., [Jackson *et al.*, 1989]).

6 REGLAS

F. J. Díez Vegas

El uso de las reglas de producción surgió dentro de la teoría de autómatas y lenguajes formales: cada gramática viene definida por un conjunto de reglas de reescritura capaces de generar todo un lenguaje, por lo que se conocen también como reglas de producción. También en los problemas de búsqueda que aparecen tradicionalmente en IA puede utilizarse un conjunto de reglas que generen los estados de un espacio de búsqueda (cap. 3). Dentro de este contexto, Newell y Simon [1972] utilizaron las reglas como modelo psicológico: el comportamiento inteligente puede describirse mediante un conjunto de reglas que indiquen en cada momento cómo actuar, en función de la información disponible. El programa DENDRAL, de Buchanan y Feigenbaum [1978], fue el primer sistema experto basado en este formalismo. Las reglas de producción se utilizaban para generar las estructuras químicas que pudieran explicar los resultados espectrográficos.

La utilización de las reglas, tal como las conocemos hoy en día, surge en el proyecto MYCIN [Buchanan & Shortliffe, 1984]. Las reglas se utilizan para examinar un conjunto de datos y solicitar nueva información hasta llegar a un diagnóstico. En este caso, el generar un espacio de estados ocupa un lugar secundario, aunque el nombre de reglas de producción *se ha mantenido principalmente por motivos históricos.*[1] *Hoy en día las reglas, ampliadas con el uso de marcos y con algunos conceptos propios de las redes semánticas, constituyen el método estándar para la construcción de sistemas expertos.*

---

[1] El encadenamiento hacia adelante (sec. 6.3.2) es similar al proceso de generación de un lenguaje a partir de una gramática; por eso en los primeros tiempos de la IA era frecuente hablar de *reglas de producción*. Sin embargo, en los sistemas que utilizan encadenamiento hacia atrás tiene menos sentido utilizar este término, y por este motivo las denominaremos simplemente "*reglas*".

Muchas de las características que hace pocos años eran patrimonio exclusivo de los prototipos de investigación se han convertido actualmente en métodos de uso común, y es posible encontrar en el mercado a un precio razonable herramientas potentes y flexibles que simplifican notablemente la construcción de un sistema experto de este tipo.

A pesar de que los distintos sistemas basados en reglas (SBR) coinciden en los puntos esenciales, existen también diferencias importantes entre ellos. La visión que vamos a ofrecer en este capítulo intentará ser suficientemente genérica para abordar todos los aspectos importantes sin restringirnos a un único modelo, y a la vez suficientemente concreta para que resulte útil a quien vaya a trabajar algún día con una de estas herramientas.

6.1 COMPONENTES BÁSICOS DE LOS SBR

Tal como adelantábamos en la sec. 1.3.4.2, un sistema basado en reglas consta de cinco elementos básicos, según muestra la Fig. 6.1. Vamos a describir a continuación cada uno de ellos y más adelante completaremos este esquema tan simple introduciendo otros elementos.

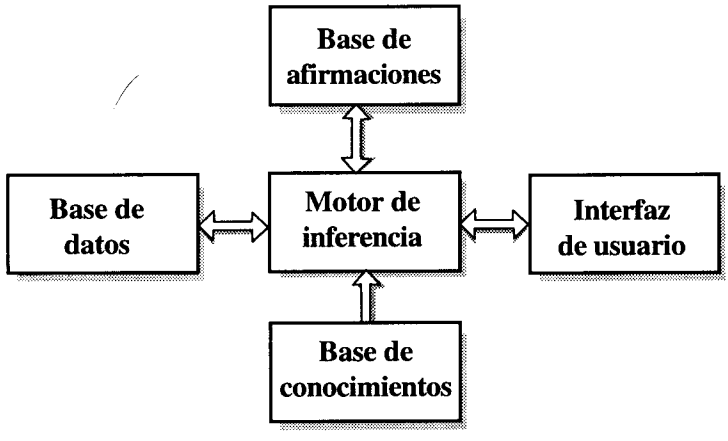


Fig. 6.1. Componentes básicos de un sistema basado en reglas

Motor de inferencia: A veces se le denomina también *intérprete de reglas*. Es el elemento central del sistema; se encarga de coordinar la información procedente de todos los demás y de enviar los resultados de la inferencia al lugar oportuno.

Base de conocimientos: Contiene las reglas y, a veces, también algunas afirmaciones iniciales. La base de conocimientos es específica del dominio en que el sistema puede considerarse experto: medicina, espectrografía, geología, diseño de circuitos, etc.

Base de afirmaciones: También recibe el nombre de *memoria de trabajo*. Contiene las afirmaciones iniciales (contenidas en la base conocimientos), las que se extraen de la base de datos, las que el usuario introduce como datos y todas las que el motor de inferencia infiere a partir de las anteriores aplicando las reglas.

Interfaz de usuario: se encarga de solicitar al usuario la información necesaria y de mostrarle los resultados de la inferencia; además, le ofrece explicaciones de cómo y por qué está funcionando el sistema. Más adelante ampliaremos este punto.

Base de datos: Aunque desde el punto de vista de la IA la base de datos tiene poco interés, en casi todas las aplicaciones prácticas se hace imprescindible tener acceso eficiente a toda la información relativa a los casos anteriores.

Conviene insistir en la diferencia entre *base de datos* y *base de conocimientos*. La primera almacena información puntual; por ejemplo, "el paciente tiene 36 años, pesa 75 Kg., mide 1'73, padeció fiebre reumática, etc.". La base de conocimientos, en cambio, contiene información sobre cómo relacionar los datos y los conceptos entre sí: "SI el área de la válvula mitral es mayor que 1'5 cm^2 y menor que 2 cm^2 ENTONCES hay estenosis mitral leve", "SI el paciente padeció un infarto ENTONCES aplicar tratamiento anti-coagulante".

En la Fig. 6.1 es importante la dirección de las flechas. Las que conectan el motor de inferencia con la base de datos, con la base de afirmaciones y con interfaz de usuario son dobles, lo cual indica que la información fluye en ambos sentidos. En cambio, hay una flecha que va de la base de conocimientos al motor de inferencia pero no a la inversa, lo cual indica que en los sistemas expertos convencionales éste no modifica aquélla. Si describiéramos un sistema más avanzado, con capacidad de aprendizaje, esta flecha debería ser bidireccional.

Ésta es la estructura más general de un SBR. En algunas aplicaciones puede ser innecesaria la base de datos. En aplicaciones de control y monitorización de procesos puede haber además un módulo que introduzca en el sistema la información procedente de los sensores y que aplique las conclusiones mediante un conjunto de actuadores. En el caso poco frecuente de control no

supervisado podría estar ausente el interfaz de usuario. Sin embargo, en la mayor parte de los sistemas expertos aplicados a problemas reales el interfaz de usuario desempeña un papel mucho más importante de lo que se pensó en los primeros años, pues en muchas ocasiones la aceptación del programa por parte de los usuarios finales dependerá de la calidad del interfaz (que sea fácil de aprender, cómodo y flexible) tanto como de la calidad de los resultados.

Es importante señalar que, en principio, el motor de inferencia es un programa de ordenador independiente del dominio de aplicación del sistema. Por tanto, es posible sustituir una base de conocimientos por otra diferente con la única condición de que tenga la misma sintaxis que la anterior. Si al motor de inferencia unimos las facilidades para la construcción y depuración de la base de conocimientos, obtenemos una **herramienta** para la construcción de sistemas expertos. Hablaremos de ellas en el capítulo 9.

6.2 ESTRUCTURA DE LAS REGLAS

6.2.1 Antecedente y Consecuente

Aunque en esencia todos los sistemas basados en reglas utilizan una misma estructura, existen también diferencias significativas en cuanto a la sintaxis y a la semántica implícita que utiliza cada uno de ellos. Dado que éste es un libro de texto, en la sintaxis que vamos a definir a continuación mostraremos las características básicas más comunes, y nos limitaremos a comentar ocasionalmente alguna de las variaciones específicas que las herramientas actuales introducen con el fin de aumentar la expresividad y la eficiencia.

Fundamentalmente, una regla consta de dos partes:

Antecedente: También llamado *parte izquierda*, debido a que la reglas pueden escribirse como $A \rightarrow C$. Contiene las *cláusulas* que deben cumplirse para que la regla pueda evaluarse o ejecutarse.

Consecuente: También llamado *parte derecha*. Indica las *conclusiones* que se deducen de las premisas (interpretación declarativa) o las *acciones* que el sistema debe realizar cuando ejecuta la regla (interpretación imperativa).

Para explicar mejor el funcionamiento de las reglas, vamos a introducir una sintaxis propia, que no coincide con ninguna herramienta en particular, sino que intenta recoger los rasgos más generales comunes a varias de ellas. A la hora de

poner ejemplos, tendremos en mente tres sistemas expertos basados en reglas: uno de ellos serviría para facilitar la conducción de un automóvil, incluyendo el diagnóstico de averías; el otro sería un sistema experto para medicina; y el tercero estaría destinado a organizar la actividad docente en un departamento universitario.

Los elementos que pueden intervenir en una regla son los siguientes:

- **Hipótesis:** Por ejemplo, `hielo-en-la-carretera`, `avería-eléctrica`, `signos-de-isquemia-en-ECG`, `diabetes`, `hay-catedráticos`, `hay-becarios`, etc. En un momento dado de una consulta, cada hipótesis tiene asociado un valor de verdad. Más adelante volveremos sobre este punto.

- **Dato:** Por ejemplo, `nivel-de-gasolina`, `temperatura-exterior`, `edad-del-paciente`, `sexo-del-paciente`, `dedicación`, `área-de-conocimiento`, etc.

  Cada dato puede tomar cierto tipo de valores. El nivel de gasolina, el voltaje de la batería y la edad del paciente toman valores numéricos; estamos dando por supuesto que existe una unidad de medida para cada uno de ellos: litros, grados centígrados, años, etc. Otros datos toman valores dentro de un conjunto discreto como `{encendido, apagado}`, `{varón, mujer}` o `{catedrático, titular, ayudante, becario}`.

  En el caso de utilizar marcos, los datos pueden corresponder a los campos de las instancias.[2] Así, tendríamos un marco `paciente`, con varias instancias, tales como `Pedro-García` y `María-López`, y varios campos como `edad`, `sexo`, etc. Representaremos cada uno de estos datos mediante la instancia y el campo, separados por un punto: `Pedro-García.edad`, `María-López.dedicación`, etc.

- **Relación de comparación:** Se establece entre dos datos o entre un dato y un valor. Algunos ejemplos pueden ser los siguientes:

```
nivel-de-gasolina = 8
temperatura-exterior < 0
```

[2] El término *instancia* se utiliza como neologismo para traducir el término inglés "*instance*", que significa "ejemplo, "caso particular". En los dos próximos capítulos explicaremos el significado de estos conceptos con más detalle. De momento, baste decir que los *marcos* representan clases y que las *instancias* de un marco son los elementos de una clase; cada *campo* corresponde a una propiedad. El motivo de anticiparnos a lo que explicaremos más adelante es que la mayor parte de las herramientas actuales destinadas a la construcción de sistemas expertos combinan reglas y marcos.

```
rueda-tras-izq.presión ≠ rueda-tras-dcha.presión
Pedro-García.edad < 60
María-López.categoría = titular
```

(Observar que los dos primeros ejemplos contienen datos simples, mientras que en los tres últimos aparecen campos de instancias.) Al igual que ocurre con las hipótesis, cada relación es cierta o falsa en un momento de la consulta, según la interpretación obvia del signo que determina la relación.

♦ **Relación de pertenencia:** Se establece entre una instancia y un marco:

```
rueda-tras-izq ES rueda
Pedro-García ES paciente
```

Una relación de pertenencia "`x ES y`" se considera cierta cuando `x` es una instancia del marco `y`.

♦ **Cláusula:** Consiste en una hipótesis o una relación (de comparación o de pertenencia), o bien en la negación, la conjunción o la disyunción de otras cláusulas (mediante los operadores `NO`, `Y` y `O`, respectivamente). Si es necesario pueden utilizarse paréntesis para indicar el alcance de cada operador.

Hemos hablado hasta ahora de las cláusulas, que forman el antecedente. Vamos a describir a continuación las conclusiones, que forman el consecuente. En nuestro caso consideraremos tres tipos de acciones: afirmar, retractar y actuar. El último de estos tres indica que el sistema enviará una orden a los actuadores con los que está conectado. Con ello podríamos tener reglas como:

```
SI  temperatura-interior > temperatura-deseada
    temperatura-deseada > temperatura-exterior
ENTONCES  ACTUAR encender-el-ventilador
          AFIRMAR ventilador = encendido

SI  temperatura-interior > temperatura-deseada
    temperatura-deseada ≤ temperatura-exterior
ENTONCES  ACTUAR encender-el-aire-acondicionado
          AFIRMAR aire-acondicionado = encendido
```

Observar que el antecedente va precedido de la palabra clave `SI` y el consecuente va precedido de `ENTONCES`. Entre las distintas condiciones y entre las acciones se supone que hay una conjunción `Y` implícita. La palabra clave `AFIRMAR` puede omitirse, de modo que cuando en una de las conclusiones de la regla no aparece ninguna de las tres acciones mencionadas, supondremos que hay un "`AFIRMAR`" implícito.

6.2.2 Uso de Variables en las Reglas

Hasta ahora hemos hablado de cláusulas en que los datos y las instancias aparecían explícitamente. De este modo las reglas constituirían una lógica de primer orden (cap. 5) y su capacidad expresiva sería mínima. Como consecuencia, habría que definir una regla particular para cada individuo o cada elemento, lo cual haría impracticable este método de representación. Por tanto, nos interesa poder representar mediante reglas afirmaciones equivalentes a las que aparecen en la lógica de predicados, y para ello se hace necesario introducir **variables** en las reglas.

Por ejemplo, una afirmación como "Todos los catedráticos son doctores", que en la lógica de predicados se expresa como

$$\forall x, \quad \text{CATEDRÁTICO}(x) \Rightarrow \text{DOCTOR}(x)$$

daría lugar a la siguiente regla

```
SI  ?x ES catedrático
ENTONCES  ?x ES doctor
```

o bien a esta otra

```
SI  ?x.categoría = catedrático
ENTONCES  ?x.titulación = doctor
```

La primera de estas reglas interpreta cada predicado como la pertenencia a una clase (un marco), mientras que la segunda lo trata como una propiedad (un campo). Observar que hemos introducido la convención de denotar las variables mediante la anteposición de un signo de interrogación.

6.2.3 Comentarios

• La sintaxis que hemos definido es sumamente flexible, a pesar de su sencillez. En realidad, para que estuviera completa habría que determinar si las variables van a representar solamente instancias o si pueden representar también clases y propiedades. En este caso, nos estaríamos aproximando a las lógicas de orden superior y, naturalmente, la complejidad del motor de inferencia que interpretase estas reglas debería ser mucho mayor. Del compromiso entre expresividad y tratabilidad hablaremos en la sección 6.8.

Los sistemas expertos y las herramientas comerciales suelen utilizar una sintaxis más rígida, con el fin de tener motores de inferencia más eficientes. Aunque, por otro lado, introducen otros operadores para asignar variables

dentro de la regla, para mostrar y solicitar información, para calcular expresiones matemáticas, para introducir comentarios, etc., que constituyen una ayuda valiosa en la práctica pero que alejan las reglas de la lógica de primer orden.

• Una de las características que más diferencia entre sí los sistemas basados en reglas es la forma de indicar la falsedad de una afirmación. Algunos sistemas, tales como Nexpert, indican explícitamente si una determinada hipótesis es cierta, falsa, no investigada o desconocida (cuando el sistema ha tratado de averiguar su valor de verdad pero no ha llegado a ninguna conclusión). En otros sistemas, en cambio, sólo almacenan las hipótesis confirmadas. El considerar falsa toda proposición que no se encuentre en la base de afirmaciones ni pueda deducirse de la información disponible, se conoce como *axioma del mundo cerrado*; el lenguaje PROLOG utiliza este axioma. Otros sistemas, como GoldWorks, son menos explícitos a la hora de establecer una semántica para las proposiciones no deducibles, y el diseñador del programa tiene una cierta libertad para determinar si las va a considerar como desconocidas o como falsas.

Dado que en la sintaxis que hemos definido existe una acción para `RETRACTAR`, el lector puede deducir que hemos optado por la segunda posibilidad; es decir, la forma de negar una hipótesis o relación consiste en eliminarla de la base de afirmaciones.

• Algunos sistemas —GoldWorks es un ejemplo— admiten la posibilidad de indicar que algunos datos son *univaluados*, como la temperatura exterior o la edad de un paciente, mientras que otros, como las enfermedades padecidas por una persona o las asignaturas que imparte un profesor, son *multivaluados*. Cuando un dato es univaluado, la asignación de un valor elimina el que pudiera estar asignado anteriormente, y de este modo podemos evitar que en la base de conocimientos se introduzca información contradictoria. Así, si afirmamos "`Pedro-García.edad = 63`" se eliminará de la base de afirmaciones cualquier otro valor que existiera para la edad de esa persona. En cambio, el afirmar "`Pedro-García.enfermedad-anterior = infarto-de-miocardio`" no eliminará una afirmación como "`Pedro-García.enfermedad-anterior = apendicitis`". (Observar la relación con la faceta *multivaluado* que mencionaremos en la sec. 8.2.1 al hablar de los marcos.)

• Las **variables** que hemos descrito aparecen dentro una regla; así es como suele entenderse el término "variable" en el contexto de los sistemas basados en reglas. Sin embargo, algunas herramientas comerciales, al hablar de "variables" se refieren a ciertos registros de la base de afirmaciones, tales como

`temperatura-exterior` o `edad-del-paciente` —nosotros hemos llamado "**datos**" para evitar confusiones— y se parecen más a las variables de los lenguajes de programación tradicionales que a las "variables de reglas" que hemos introducido en este capítulo. El sentido de este término quedará más claro en la próxima sección, al discutir el papel que las variables desempeñan en los patrones.

6.3 INFERENCIA

6.3.1 Comparación de Patrones

En general, se entiende por *patrón* un modelo que puede representar diferentes elementos. En el contexto de las reglas, este término se utiliza de forma restringida para denominar una *cláusula con variables*, y recibe este nombre porque equivale a un conjunto de afirmaciones. Así, el patrón "`?x ES profesor`" representa las afirmaciones "`Pedro-García ES profesor`", "`Antonio-Pérez ES profesor`", etc.

Una regla se ejecuta (se dispara) cuando se cumple su antecedente, y para ello es necesario que se cumplan cada una de las cláusulas que lo componen. Podemos tener tres situaciones:

1. *Claúsula sin variables.* Si la cláusula consta de una *hipótesis*, se satisface cuando en la base de afirmaciones (BA) hay una afirmación que coincide con ella. Si se trata de una *relación*, se satisface cuando los valores de los datos se ajustan a ella. Para las dos cláusulas de la regla siguiente, la discusión es trivial:

```
SI  hielo-en-la-carretera
    velocidad > 70
ENTONCES recomendación = reducir-la-velocidad
```

2. Cuando *una variable aparece por primera vez* en una regla, la cláusula correspondiente se satisfará si y sólo si hay una asignación de valor a dicha variable tal que el patrón coincida con uno de los elementos existentes en la base de afirmaciones. Por ejemplo, supongamos que tenemos la regla

```
SI  ?x ES catedrático
ENTONCES  el-director-de-dpto-ha-de-ser-catedrático
```

y que la BA contiene, entre otras, las siguientes afirmaciones: "`Pedro-García ES catedrático`" o "`Antonio-Pérez ES catedrático`". En este caso

hay dos asignaciones que pueden hacer que la regla se ejecute: '?x ≡ Pedro-García' y '?x ≡ Antonio-Pérez'. Si no hubiera ninguna afirmación de ese tipo en la BA, la regla no podría ejecutarse y diríamos que ha fracasado. Obsérvese que este proceso es similar a las *sustituciones* que se aplican en el caso de la lógica (sec. 5.4.2).

3. Cuando *una variable aparece por segunda vez* (o sucesivas veces) en una regla, necesariamente tiene ya un valor que le fue asignado la primera vez que apareció. La cláusula en que la variable aparece por segunda vez se satisfará solamente si, al sustituir dicha variable por el valor asignado, la cláusula coincide con una de las afirmaciones de la BA. Tomemos como ejemplo la regla siguiente:

```
SI  ?x ES profesor
    ?x.área-de-conocimiento = ccia
ENTONCES  ?x.puede-enseñar = teoría-de-autómatas
```

Si existe una afirmación como 'Antonio-Pérez ES profesor', la primera cláusula puede satisfacerse con la asignación '?x ≡ Antonio-Pérez'. La segunda cláusula se satisface si y sólo si el valor de Antonio-Pérez.área-de-conocimiento es ccia (Ciencias de la Computación e Inteligencia Artificial). En este caso, la regla se ejecuta y la afirmación 'Antonio-Pérez.puede-enseñar = teoría-de-autómatas' pasará a la BA. Naturalmente, la misma asignación de variables se mantiene en el antecedente y en el consecuente de la regla.

La regla se ejecutará una vez por cada asignación de variables capaz de satisfacer todas sus cláusulas. En este caso se habla de diferentes *"instanciaciones" de una regla*. Hablaremos más sobre este punto al tratar las agendas.

6.3.2 Tipos de Encadenamiento

Acabamos de ver cómo se realiza la comparación de patrones dentro de una regla, es decir, qué operaciones realiza el motor de inferencia para determinar si una regla está lista para ejecutarse. Ahora vamos a examinar un paso previo al anterior, que consiste en seleccionar cuáles de las reglas existentes va a examinar el motor de inferencia.

Supongamos que tenemos las dos reglas siguientes:

```
SI  director-del-dpto = ?x
ENTONCES  ?x.categoría = catedrático
```

```
SI  ?x.categoría = catedrático
ENTONCES  ?x.titulación = doctor
```

Observamos que el consecuente de la primera coincide con el antecedente de la segunda siempre que la variable `?x` tome el mismo valor en ambas reglas. Por tanto, una conclusión obtenida a partir de la primera regla puede servir para ejecutar la segunda. Esto es lo que se conoce como **encadenamiento de reglas**[3], y puede ser básicamente de dos tipos.

Encadenamiento hacia adelante o **basado en datos:** se da cuando la información introducida en el sistema hace que se ejecute una regla, y la conclusión obtenida permite que se ejecuten otras reglas. Así, al conocer que "Juan López es el director del dpto.", el motor de inferencia puede deducir que es catedrático y, a continuación, que es doctor.

Encadenamiento hacia atrás o **basado en objetivos:** consiste en buscar una regla que permita establecer cierta conclusión. Por ejemplo, para averiguar si "Juan López es doctor", buscaremos una regla que tenga esta afirmación en su consecuente. La segunda de las anteriores es una buena candidata: basta que se cumpla el antecedente con la asignación de valores adecuada. Si en la BA no se encuentra la afirmación "`Juan-López.titulación = doctor`", se puede establecer ésta como objetivo intermedio, y buscar una regla que nos lleve a esta conclusión. Y así sucesivamente.

En general, el encadenamiento hacia adelante utiliza solamente los datos disponibles, mientras que el encadenamiento hacia atrás suele solicitar al usuario la información que no ha podido deducir. Dada la importancia de este punto, todas las herramientas, incluso las más simples, incluyen algún modo de controlar cuándo y cómo se va a solicitar esta información.

Observar que el encadenamiento hacia adelante es menos específico, pues en principio ejecutará todas las reglas posibles en función de la información introducida. En cambio, el razonamiento hacia atrás lleva implícito un proceso de búsqueda, por lo que es más específico y, en consecuencia, más eficaz.

El sistema experto MYCIN y el lenguaje PROLOG sólo admiten el encadenamiento hacia atrás. OPS5, en cambio, utiliza sólo encadenamiento hacia adelante. Las herramientas comerciales avanzadas de hoy en día (KEE,

[3] Observar que este fenómeno se da solamente cuando el consecuente de la regla es del tipo `AFIRMAR`; los consecuentes como `RETRACTAR` y `ACTUAR` no introducen nueva información en la BA y por eso no provocan la ejecución de reglas.

GoldWorks, NEXPERT...) permiten usar uno u otro o una combinación de ellos. En GoldWorks, por ejemplo, cada regla puede definirse como "hacia adelante" (`:forward`), "hacia atrás" (`:backward`) o "bidireccional" (`:bidirectional`); la regla que aparece en la Fig. 6.2 está definida como "hacia adelante". Ésta es una de las formas más sencillas de controlar cuáles de las reglas van a ser aplicadas en cada momento. Más adelante mostraremos mecanismos de control más sofisticados.

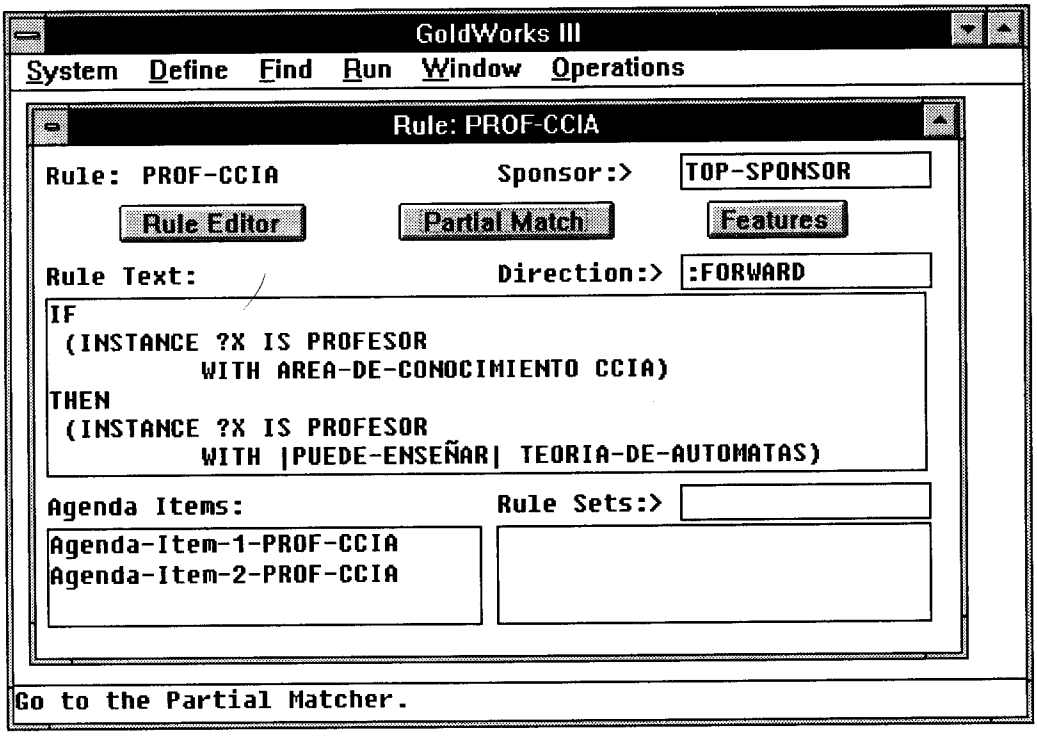


Fig. 6.2. Editor de reglas de GoldWorks

Antes de continuar, y para evitar confusiones, conviene insistir en que la dirección del encadenamiento no tiene nada que ver con la dirección en que se ejecuta una regla. En realidad, las reglas siempre se ejecutan "hacia adelante", es decir, ejecutando el consecuente cuando se confirma el antecedente. Cuando hablamos de encadenamiento hacia atrás nos referimos solamente a un proceso de búsqueda y selección de reglas.

6.3.3 Dependencia Reversible e Irreversible

En esta sección hemos hablado sobre todo de cómo la inferencia añade nueva información a la base de afirmaciones; es decir, hemos tratado el razonamiento monótono. Sin embargo, una de las ventajas de los sistemas basados en reglas frente a la lógica clásica es la capacidad de retractar afirmaciones anteriores como consecuencia de nuevas inferencias. Ahora bien, ¿qué ocurre si la información que ahora se retracta había sido utilizada para obtener otras conclusiones?

Las herramientas más avanzadas permiten que sea el diseñador del sistema quien tome esta decisión. Para ello, al definir una regla hay que indicar el *tipo de dependencia*. Comparemos las dos reglas siguientes:

```
SI  bombilla-encendida
ENTONCES habitación-iluminada

SI  bombilla-encendida
ENTONCES película-velada
```

En el primer caso, queremos indicar que la habitación está iluminada cuando y sólo cuando la bombilla está encendida. Si en un momento dado se cumple la premisa, el consecuente pasará a la BA, y si se retracta la primera afirmación deberá retractarse también la segunda. En este caso hablaremos de *dependencia reversible*.

La segunda regla plantea una situación muy distinta. La regla afirma que si en un momento dado se enciende la bombilla, la película fotográfica quedará velada. La diferencia con el caso anterior es que, aunque luego la bombilla se apague la película va a seguir estando velada. Nosotros hablaremos de *dependencia irreversible* (la terminología varía mucho de unos sistemas a otros), y en este caso la afirmación no debe retractarse.

De esta discusión se deduce la necesidad de que el SBR registre junto a cada afirmación la forma en que ha sido deducida, con el fin de poder mantenerla o eliminarla en función del tipo de reglas que han intervenido. Esta información es útil no sólo para mantener la consistencia lógica del sistema, sino también para generar explicaciones sobre el razonamiento, tal como mostraremos en la sección 6.5.

6.4 CONTROL DEL RAZONAMIENTO

Hemos hablado de cómo examina el motor de inferencia las condiciones de una regla para determinar si está lista para ejecutarse. También hemos hablado de los tipos de encadenamiento que determinan el flujo de la inferencia. Sin embargo, en esta discusión hemos omitido un paso intermedio: cómo seleccionar qué regla ejecutar primero cuando hay varias disponibles.[4] El control del razonamiento es importante por tres motivos:

1. Por el **contenido** de la inferencia: en algunos casos, las conclusiones pueden depender del orden en que se apliquen las reglas. Por eso, las reglas más específicas, y en particular las que tratan las excepciones, deben activarse primero para que no se apliquen otras reglas más generales.
2. Por **eficiencia**: el utilizar la regla adecuada llevará rápidamente a una conclusión, mientras que una elección equivocada puede hacer que se pierda mucho tiempo en un camino que no conduce a ninguna parte (recordar los capítulos sobre búsqueda).
3. Por el **diálogo** que genera: es importante que el sistema no pregunte al usuario la información que pueda deducir por sí mismo y, además, que el orden en que solicita la información sea razonable. (Una de las críticas de los médicos al sistema MYCIN en su primera versión es que generaba preguntas de forma desordenada, sin seguir una línea de razonamiento clara.)

Vamos a mostrar algunas de las soluciones más comunes que suelen aplicarse para controlar el razonamiento.

6.4.1 Mecanismos Sencillos

La solución más simple consiste en **ordenar las reglas** en la base de conocimientos, colocando en primer lugar las que deseamos que sean examinadas antes. Este método, además de ser poco elegante, tiene el inconveniente del mantenimiento, es decir, que en una base de conocimientos grande existe el riesgo de que la adición o eliminación de una regla modifique el flujo del razonamiento de modo imprevisto. Por otra parte, esta solución sólo es aplicable en sistemas simples que almacenan las reglas como una lista, según el

[4] Algunos autores llaman a este problema *resolución de conflictos*. Nosotros preferimos hablar de control del razonamiento o de selección de reglas, para no confundir esta cuestión con otros "conflictos", como los que surgen cuando diferentes reglas llevan a información contradictoria.

orden en que fueron introducidas por el usuario, y examinan esta lista de forma cíclica, hasta que la aplicación de un nuevo ciclo no introduce modificaciones en la BA. Frente a esta evaluación "estilo intérprete", otros sistemas, como OPS5, realizan una especie de compilación de las reglas para formar una red en que un mismo elemento presente en dos o más reglas viene representado por un solo nodo. El algoritmo de propagación que utiliza OPS5 se conoce como RETE, y consiste en examinar solamente el efecto de la información añadida o eliminada de la BA en cada ciclo. (Recordemos que OPS5 utiliza solamente encadenamiento hacia adelante.)

Otra forma sencilla de control, aplicable en sistemas que utilizan razonamiento hacia atrás, consiste en **ordenar** cuidadosamente **las cláusulas** dentro de cada regla, para que unas sean examinadas antes que otras. Cuando fracasa una de ellas, las cláusulas que aparecen después dentro de la misma regla no necesitan ser investigadas. Por eso es conveniente colocar primero las cláusulas que tienen más probabilidades de fallar, con el fin de optimizar la búsqueda.

Una solución más robusta[5] y con mejor fundamento teórico, válida para ambas direcciones del encadenamiento, se basa en añadir **nuevas cláusulas** relacionadas con el punto de inferencia en que nos encontramos, con el fin de controlar en cada momento qué reglas deben aplicarse y cuáles no. Por ejemplo, la primera regla de la sección 6.3.1 podría transformarse en

```
SI  situación = coche-en-marcha
    hielo-en-la-carretera
    velocidad > 70
ENTONCES recomendación = reducir-la-velocidad
```

si en un momento dado "`situación = motor-no-arranca`", ya no es necesario continuar evaluando la regla. Este mismo recurso sirve para dividir en varias etapas la resolución del problema; así, en un sistema experto para medicina, cada regla podría comenzar con una cláusula como "`objetivo = recoger-antecedentes`", "`objetivo = diagnóstico`", "`objetivo = recomendar-tratamiento`", etc.

Éstas son las dos soluciones más sencillas, disponibles incluso en herramientas rudimentarias. Veamos otros métodos más sofisticados.

[5] En este caso, la "robustez" consiste en que el orden de ejecución de las reglas no varíe sustancialmente a pesar de que se introduzcan otras nuevas.

6.4.2 Control de las Agendas

En los sistemas más simples, cada regla se ejecuta en cuanto se ha comprobado que se cumplen sus condiciones. En cambio, las herramientas más avanzadas realizan primero una búsqueda de todas las reglas que en un momento dado están listas para ser ejecutadas y las reúnen en una agenda. Más exactamente, cada asignación de variables que satisface una regla (cada "instanciación") da lugar a un elemento distinto en la agenda. La Fig. 6.2 muestra una regla que ha dado lugar a dos elementos elementos en la agenda (dos "instanciaciones").

Estas agendas pueden tener estructura de pila (el último elemento introducido es el primero en ser ejecutado) o de cola (lo contrario). En algunas herramientas es posible tener varias agendas, como comentaremos más adelante, y el usuario puede escoger un tipo de estructura para cada una de ellas.

El orden de pila o cola establecido para cada agenda puede alterarse mediante la asignación de una **prioridad** a cada regla, que en general consiste en un valor numérico dentro de un cierto rango. Cuando un elemento (una "instanciación" de una regla) entra en una agenda, se sitúa por delante de todos los que tengan una prioridad más baja y por detrás de los que la tengan más alta. En las herramientas más avanzadas es posible incluso modificar de forma dinámica la prioridad de una regla en función del proceso de inferencia.

Otro mecanismo para controlar el encadenamiento hacia adelante consiste en tener varias agendas, controladas por sus respectivos **patrocinadores** (en la Fig. 6.2 aparece un rectángulo que indica el "*sponsor*") . Cada regla está asignada a un patrocinador, en cuya agenda se almacenarán las "instanciaciones" de la regla. Los recursos disponibles, es decir, el número de elementos ejecutados en cada ciclo, se distribuye entre todos los patrocinadores, equitativamente o del modo que el diseñador del sistema haya determinado. Incluso es posible tener una jerarquía de patrocinadores con el fin de distribuir los recursos.

Naturalmente, cuando la información obtenida al ejecutar un elemento de la agenda hace que otro de los que estaban en espera deje de ser válido, éste no llegará a ejecutarse. En algunos sistemas es posible acceder directamente desde el motor de inferencia a las agendas con el fin de eliminar o reordenar sus elementos, aunque esta tarea suele resultar bastante compleja.

Otro mecanismo de control, independiente de las agendas, consiste en agrupar las reglas en **conjuntos de reglas** (los "*rule sets*" de la Fig. 6.2), lo cual permite activar o desactivar algunas de ellas en bloque, evitando así la necesidad de añadir en varias etapas premisas adicionales (ver la sección anterior). Los conjuntos de reglas, al igual que los patrocinadores, estarán desactivados cuando se supone que no van a aportar ninguna información relevante, y se activarán solamente cuando sea necesario. En consecuencia, estos dos mecanismos ayudan a estructurar de algún modo la base de conocimientos, facilitando así su construcción y su mantenimiento y mejorando la eficiencia.

6.4.3 Metarreglas

Las primeras reglas contenidas en el sistema experto MYCIN contenían información sobre enfermedades, microorganismos, fármacos... Es lo que a veces se denomina *conocimiento del nivel del objeto*. Sin embargo, cuando Randall Davis [1980; Davis & Buchanan 1984] trató de extender MYCIN y construir TEIRESIAS como sistema de adquisición automática del conocimiento mediante la interacción con un experto humano, observó la necesidad de que el sistema conociera las reglas ya introducidas y pudiera razonar sobre ellas. Este *conocimiento sobre el conocimiento* se denomina **metaconocimiento**. En TEIRESIAS venía representado de dos formas: mediante **modelos de reglas** y mediante **metarreglas**. Los primeros eran una forma estructurada de indicar los atributos comunes a un grupo de reglas, y se usaban para orientar la adquisición de conocimiento, sintetizando ideas de dos campos de la IA: la comprensión basada en modelos y el aprendizaje a partir de la experiencia. Las metarreglas, en cambio, eran *reglas destinadas a razonar acerca de otras reglas* (de ahí su nombre). Puesto que éstas están más orientadas que aquéllos al control del razonamiento, que es el tema que nos ocupa, vamos a estudiarlas aquí con más detenimiento.

Una metarregla sencilla podría ser la siguiente:

```
SI  objetivo = determinar-cualificación
    ?r1 ES regla
    ?r1.antecedente MENCIONA titulación
ENTONCES  r1.prioridad = 100
```

Como vemos, es una regla que se refiere a otras reglas, y nos lleva a examinar primero aquéllas que parecen ser más relevantes para el problema en cuestión. En este caso se trata de una regla *dependiente del dominio*, pues aparecen conceptos como `determinar-cualificación` y `titulación`, que son

propios de un campo concreto. La siguiente metarregla, en cambio, es *independiente del dominio*:

```
SI  objetivo = ?x
    ?r1.consecuente NO MENCIONA ?x
ENTONCES  r1.prioridad = -500
```

Los dos ejemplos que acabamos de mostrar son muy sencillos. El sistema TEIRESIAS era capaz de hacer afirmaciones sobre la *utilidad estimada* (la posibilidad de que nos lleve a una conclusión) y sobre el *orden de ejecución* (determinando, por ejemplo, que un conjunto de reglas debe ejecutarse antes que otro).

Comentarios

En primer lugar, podemos observar que el metaconocimiento es un paso más en la explicitación del conocimiento. En efecto, supongamos que tenemos un sistema de diagnóstico médico basado en un árbol de decisión. Obviamente tal programa posee conocimiento sobre la enfermedad (de otro modo no podría diagnosticar nada), pero se trata de un conocimiento implícito compilado en el código del programa. Al utilizar un sistema basado en reglas para resolver ese mismo problema, estamos representando explícitamente el conocimiento, y eso nos permite trabajar en un nivel superior a la hora de construir y modificar el sistema; también el propio sistema está trabajando en ese nivel superior y, más aún, pueda llegar a razonar sobre el conocimiento que posee y la forma en que lo utiliza; esta posibilidad se observa en la capacidad de explicación presente en muchos sistemas expertos.

Ahora bien, en un sistema basado en reglas, a parte de este conocimiento explícito, hay un conocimiento implícito que se halla "dentro" del motor de inferencia, pues éste "sabe" cuándo puede ejecutar una regla, cómo debe encadenar unas con otras, cómo debe tratar las prioridades y controlar las agendas, etc. El metaconocimiento, como decíamos, da un paso más en la explicitación del conocimiento, tratando de expresar en forma declarativa cómo tratar el conocimiento de nivel inferior. Una de las ventajas que se deducen es la posibilidad de modificar la estrategia de control sin tener que reprogramar el motor de inferencia. Otra ventaja es que un sistema dotado de metaconocimiento será capaz de explicar cuál es la estrategia que ha utilizado, por qué ha aplicado unas reglas antes que o en vez de otras, etc. y de ahí puede surgir más adelante la posibilidad de *reorganizar* el conocimiento, de *aprender* a partir de la experiencia o mediante un diálogo con el usuario (éste era el objetivo de Davis al

construir TEIRESIAS), de *enseñar* lo que conoce (la base de conocimientos de MYCIN fue utilizada por Clancey [1984a] para construir GUIDON, un programa tutor sobre enfermedades infecciosas), etc.

Estas capacidades se basan en que el metaconocimiento, de algún modo, puede considerarse como una forma rudimentaria de consciencia (el sistema no sólo conoce un campo, como la medicina, sino que también *conoce que conoce*) y en este sentido es un paso hacia adelante en el camino de la IA. De hecho, no puede hablarse plenamente de inteligencia sin que haya consciencia.

También puede enfocarse este tema desde el punto de vista de los procesos de búsqueda, considerando las reglas como las posibles transformaciones en cierto espacio. En ausencia de metaconocimiento tenemos una búsqueda ciega, en que el motor de inferencia examina la base de conocimientos hasta dar con el encadenamiento de reglas adecuado. El metaconocimiento, en cambio, permite realizar una búsqueda heurística, pues aporta información sobre cómo realizar la búsqueda. Recordemos que en el capítulo 4 se habló de la importancia de inyectar conocimiento para dirigir la búsqueda y evitar así la explosión combinatoria.

6.4.4 Otros Mecanismos

Los creadores del sistema OPS5 [Brownston *et al.*, 1985], al abordar el problema del control del razonamiento mediante la selección de reglas (ellos lo llaman *resolución de conflictos*), señalan dos características que debe poseer el sistema: la *sensibilidad* y la *estabilidad*, que en cierto modo son contrapuestas. La primera indica la capacidad de responder a los estímulos; en este caso, significa que el sistema debe procesar con prontitud la nueva información que recibe. La estabilidad, en cambio, exige que no haya una reacción desmesurada ante los estímulos, es decir, que la llegada de información poco relevante no cambie excesivamente la línea de razonamiento.

Entre los mecanismos que proponen para lograr un compromiso entre ambos objetivos se encuentran los siguientes:

1. **Refractariedad:** Significa que, después de haber respondido a un estímulo, existe un tiempo en que el sistema es incapaz de volver a responder. En los sistemas basados en reglas esto implica la existencia de mecanismos de control para impedir que una regla se ejecute repetidamente sin que se haya

introducido nueva información. (Estos mecanismos afectan sobre todo a la estabilidad.)

2. **Actualidad:** Consiste en examinar primero las reglas que se apoyan en *la información más reciente*, con el fin de seguir una línea de razonamiento estable y a la vez sensible a la nueva información que va llegando. El lenguaje OPS5 implementa este mecanismo anotando para cada afirmación de la BA el momento (el ciclo) en que se ha deducido.

3. **Especificidad:** Su objetivo es aplicar las reglas más específicas antes que las más generales, pues aquéllas tienen en cuenta más información que éstas. Comparemos las dos siguientes:

```
SI  prótesis-mitral-recién-implantada
    complicaciones-postoperatorias
    plazo-próxima-revisión = DESCONOCIDO
ENTONCES plazo-próxima-revisión = 3_semanas

SI  prótesis-mitral-recién-implantada
    plazo-próxima-revisión = DESCONOCIDO
ENTONCES plazo-próxima-revisión = 6_meses
```

Puesto que la primera es más específica, habrá que examinarla antes para que la segunda se ejecute solamente si ha fracasado la primera. Ésta es una forma de tratar las excepciones a las reglas generales. Si la herramienta con la que trabajamos no considera la especificidad de las reglas, deberemos recurrir a otro mecanismo —como la asignación de prioridades u otro que tengamos disponible—.

6.5 EXPLICACIÓN DEL RAZONAMIENTO

En el capítulo 9 hablaremos de la importancia de que un sistema experto sea capaz de explicar su razonamiento. Por ahora, nos limitaremos a describir algunos de los métodos utilizados en los sistemas basados en reglas para abordar este tema.

El primer sistema experto capaz de explicar su razonamiento fue MYCIN [Buchanan & Shortliffe, 1984]. Para comprender cómo se implementa esta función, conviene recordar que el sistema experto realiza la inferencia mediante el encadenamiento hacia atrás de reglas. Una regla sólo se ejecuta cuando se ha comprobado que se cumplen todas sus premisas. Si una de ellas no puede deducirse a partir de la información disponible, el sistema preguntará al usuario,

quien puede introducir la información solicitada por el sistema o bien pedir una explicación de por qué se ha planteado esa pregunta, mediante el comando POR_QUÉ (WHY). Supongamos que tenemos la regla siguiente

```
SI  ?x.gram = gram-negativo
    ?x.forma = bastón
    ?x.aerobicidad = anaeróbico
ENTONCES  ?x.identidad = bacteroide
```

y que se han confirmado las dos primeras premisas. Al tratar de confirmar la última, puede producirse el siguiente diálogo:

```
12) ¿Cuál es la forma de ORGANISMO-1?
** BASTÓN
13) ¿Cuál es la aerobicidad de ORGANISMO-1?
** POR_QUÉ
[es decir, POR_QUÉ es importante determinar la
  aerobicidad de ORGANISMO-1]
[1.0] Esto ayudará a determinar la indentidad de
  ORGANISMO-1.
    Se ha determinado ya que
       [1.1] La tinción de ORGANISMO-1 es gram-negativo
       [1.2] La forma de ORGANISMO-1 es bastón
    Por tanto, si
       [1.3] La aerobicidad de ORGANISMO-1 es anaeróbico
    entonces
       hay evidencia sugerente (0.6) de que
         la identidad de ORGANISMO-1 es bacteroide.
    [REGLA-005]
```

La otra forma en que el usuario puede solicitar explicaciones a MYCIN es mediante el comando CÓMO (HOW), que se utiliza para saber cómo ha llegado el sistema experto a una determinada conclusión. Otro fragmento de consulta podría ser el siguiente:

```
** CÓMO 2.3
[es decir, CÓMO se ha concluido que la identidad de
  ORGANISMO-1 es bacteroide]
He utilizado la REGLA-005 para deducir que la identidad
  de ORGANISMO-1 es bacteroide.
** CÓMO 1.2
[es decir, CÓMO se ha concluido que la forma de
  ORGANISMO-1 es bastón]
Usted lo dijo [pregunta 12].
```

Utilizando estos dos comandos repetidamente es posible seguir las reglas que el sistema ha encadenado o está tratando de encadenar. Aunque las reglas de MYCIN están codificadas en Lisp y contienen variables, el módulo de explicación se encarga de traducirlas al lenguaje natural y de sustituir las variables por sus correspondientes asignaciones, para que sea más fácil comprender el significado de cada regla en el contexto de la consulta actual.

Posteriormente los diseñadores de MYCIN incluyeron otras posibilidades de explicación, para que el programa pudiera responder a preguntas en lenguaje natural, tales como "¿Qué organismos suelen encontrarse en la garganta?", "¿Qué dosis de estreptomicina recomiendas generalmente?", "¿Qué te dice [tal dato] acerca de la identidad de un microorganismo?", "¿Has considerado que ORGANISMO-1 podría ser bacteroide?", "¿Por qué no has necesitado conocer la forma de ORGANISMO-2?", "¿Cómo ha ayudado la regla 178 a determinar la identidad de ORGANISMO-1?", etc.

La capacidad de explicación al estilo de MYCIN es de gran valor para el diseñador del sistema experto, pero para un usuario que no conozca a fondo la tecnología de los sistemas basados en reglas puede resultarle insuficiente y demasiado artificiosa, pues está demasiado ligada al encadenamiento de las reglas. Lo que un usuario final necesita, entre otras cosas, es una explicación de las estrategias de razonamiento en función de objetivos más generales que la confirmación de una cláusula. La capacidad de generar este tipo de explicaciones es una de las motivaciones que llevaron a Clancey [1984b] a diseñar NEOMYCIN, un programa en que las metarreglas contenían el conocimiento estratégico.

El tema de la explicación es una de las cuestiones más abiertas dentro de los sistemas basados en reglas —y de los sistemas expertos en general—. Una prueba de lo poco que se ha avanzado en este terreno es que las herramientas comerciales de la actualidad en general se limitan a ofrecer mecanismos similares a `WHY` y `HOW`, y ni siquiera alcanzan la capacidad de MYCIN, por rudimentaria que ésta pueda parecernos.

6.6 TRATAMIENTO DE LA INCERTIDUMBRE

En el mundo real existen numerosas fuentes de incertidumbre: la información puede ser imprecisa, incompleta e incluso errónea; igualmente, el modelo de que disponemos puede ser incompleto e inexacto, el dominio de aplicación suele ser no determinista... Por todos estos motivos, el tratamiento de

la incertidumbre es uno de los puntos claves a la hora de construir sistemas expertos. Los dos métodos más utilizados para abordar este problema mediante reglas son utilizar unos factores de certeza más o menos semejantes a los de MYCIN o basarse en la lógica difusa.

6.6.1 Factores de Certeza de MYCIN

MYCIN fue el primer sistema basado en reglas que se ocupó del tratamiento de la incertidumbre. En su antecesor inmediato, DENDRAL, no se había planteado este tema, pues en el campo de la espectrografía las fuentes de incertidumbre son incomparablemente menores que en cualquier especialidad médica. Ya antes de MYCIN se habían desarrollado sistemas de diagnóstico basados en técnicas bayesianas; sin embargo, los métodos probabilistas disponibles cuando se desarrolló MYCIN presentaban graves inconvenientes (ver la sec. 6.5.2.1), y por eso sus creadores desarrollaron un modelo que, aunque inspirado en la teoría de la probabilidad, se apartaba notablemente de ella.

El caso más sencillo se plantea cuando tenemos una regla que relaciona un solo hallazgo, que llamaremos *e* (evidencia), con una sola hipótesis, *h*:

SI *e* ENTONCES *h*

En este caso, puede definirse un **factor de certeza**, $FC(h,e)$, que indica la fiabilidad con que podemos aceptar la hipótesis *h* en el caso de tener la evidencia *e*. A pesar de que los autores de MYCIN definen el *FC* a partir de las probabilidades a priori, $P(h)$, y condicional, $P(h|e)$, en la práctica los factores de certeza se obtienen directamente a partir de estimaciones subjetivas de los médicos participantes en el proyecto, es decir, mediante preguntas como "Si Vd. observara *e* en un paciente, ¿con qué certeza pensaría que el diagnóstico es *h*?".

Cuando en una regla hay varias premisas, se utiliza una combinación de máximos y mínimos, dependiendo de si se trata de conjunciones o disyunciones, con el fin de obtener un factor de certeza $FC(h,e_1,...,e_n)$ semejante al del caso anterior.

Si hay varias reglas que aportan evidencia a favor o en contra de una hipótesis, hace falta una fórmula que indique cómo combinar los factores de certeza correspondientes. Buchanan y Shortliffe [1984, sec. 10.2] explican por qué la fórmula original utilizada en MYCIN resultaba insatisfactoria y cómo fue

sustituida por la fórmula de van Melle [van Melle *et al.*, 1984]: cuando tenemos dos reglas cuyos factores de certeza son x e y, su FC combinado viene dado por

$$FC_{comb}(x,y) = \begin{cases} x + y \cdot (1-x) & \text{si } x > 0 \text{ e } y > 0 \\ \dfrac{x+y}{1-\min(|x|,|y|)} & \text{si } x \cdot y < 0 \\ -FC_{comb}(-x,-y) & \text{si } x < 0 \text{ e } y < 0 \end{cases}$$

Ya desde los primeros tiempos del proyecto MYCIN, sus participantes estaban insatisfechos con el modelo de los factores de certeza, debido a que estaba basado más en intuiciones que en una teoría consistente. Las críticas se hicieron cada vez más severas, especialmente tras los estudios de Adams, Horvitz y Heckerman, quienes demostraron que el modelo contenían hipótesis implícitas muy estrictas, que en general no se cumplían y que, aunque se cumplieran, el modelo podía producir resultados contrarios al sentido común y a la teoría de la probabilidad. Para ampliar este punto y obtener las referencias bibliográficas, ver [Díez, 1994, cap. 2].

6.6.2 Lógica Difusa en SBR

En la sección 5.6.2 hemos estudiado la lógica difusa; en ésta, vamos a mostrar su aplicación al razonamiento mediante *reglas difusas*, entendiendo como tales las que contienen predicados difusos. Por ejemplo, en las claúsulas de la regla

```
SI  curva es muy cerrada
Y   velocidad es alta
ENTONCES  ACCIÓN frenar moderadamente
```

aparecen tres conceptos difusos. La regla siguiente, en cambio, está cualificada globalmente por un modificador difuso:

```
Generalmente (SI   ?X es deportista-alta-competición
                ENTONCES  ?X padece hipertrofia-v-i)
```

La estructura general de una regla difusa es

SI X_1 es A_1 Y ... Y X_n es A_n ENTONCES Y es B

En caso de que haya una sola premisa, "X es A", la regla puede interpretarse mediante una distribución de posibilidad condicional, $\pi_{B|A}(x,y)$, que es una relación difusa binaria del tipo $R(x,y)$. Algunas de las interpretaciones que han propuesto diferentes autores son las siguientes:

$$\pi_{B|A}(x,y) = \min(\mu_A(x),\mu_B(y))$$
$$\pi_{B|A}(x,y) = \min(1,\ 1-\mu_A(x)+\mu_B(y))$$
$$\pi_{B|A}(x,y) = \max(\min(\mu_A(x),\mu_B(y)),1-\mu_A(x))$$
$$\pi_{B|A}(x,y) = \max(1-\mu_A(x),\mu_B(y))$$

La inferencia se realiza a partir del *modus ponens difuso*, definido así:

$$\frac{X\ es\ A^* \qquad X\ es\ A \rightarrow Y\ es\ B}{Y\ es\ B^*}$$

donde B^* es un conjunto difuso que viene dado por

$$\mu_{B^*}(y) = \max_{x\in U} T(\mu_{A^*}(x),\pi_{B|A}(x,y))$$

La función que se suele utilizar como T es el mínimo, aunque algunos autores han propuesto otras funciones (más exactamente, otras normas triangulares). Trillas y Valverde [1985] indican cómo se debe calcular $\pi_{B|A}(x,y)$ a partir de T con el fin de que la inferencia cumpla ciertas propiedades y coincida así con los resultados que dicta la intuición.

Una de estas propiedades consiste en que, cuando $A^*\subseteq A$ entonces $B^*=B$. Por tanto, si A y B son conjuntos "nítidos" (es decir, conjuntos clásicos donde $\mu_A(x)=0$ o $\mu_A(x)=1$ para todo elemento x) y $A^*=A$, obtenemos el "modus ponens" clásico como caso particular del *modus ponens difuso*. Sin embargo, éste último es más general que el primero, pues permite obtener conclusiones incluso en el caso de que $A^*\neq A$.

De la regla "Si un tomate está rojo entonces está maduro" y la afirmación "El tomate está muy rojo" —observar que A={rojo}, $A^*=A^2$={muy rojo}, B={maduro} y que $A^*\subseteq A$[6]— puede deducirse que "El tomate está maduro", pues $B^*=B$. En lógica clásica este tipo de razonamiento no era posible, pues el principio de resolución sólo podía aplicarse cuando $A^*=A$.

[6] La propiedad $A^*\subseteq A$ se deduce de que $\mu_{A^*}(x)=\mu_A{}^2(x)=\mu_A(x)^2\leq\mu_A(x)$.

En caso de que tengamos una regla con varias premisas, habrá que combinar las $\mu_{Ai}(x_i)$ mediante los operadores mínimo o máximo según se trate de una conjunción o una disyunción, respectivamente, tal como explicábamos en la sección 5.6.2. También es posible combinar las reglas en serie (encadenando las conclusiones de una con las premisas de otra) o en paralelo (cuando varias reglas conducen a una misma conclusión) utilizando las propiedades de unión e intersección de conjuntos descritas en esa sección.

Para conocer con más detalle los distintos modelos de *modus ponens difuso* que existen, el lector puede consultar los artículos recopilados entre las págs. 478 y 574 de [Dubois *et al.*, 1993].

6.7 VALORACIÓN

6.7.1 Comparación con los Programas Basados en Comandos

El paradigma de las reglas como método computacional para representar el conocimiento y realizar inferencias da lugar a una forma de programar diferente de la tradicional, en el sentido de que aquí no se le dice al ordenador lo que debe hacer (*programación imperativa*) sino cuál es el conocimiento que debe aplicar (*programación declarativa*). Para entender mejor esta distinción, conviene examinar la diferencia entre un **comando** SI-ENTONCES (en inglés IF-THEN), propio de lenguajes como Basic o Pascal, y una **regla**. A pesar de que ambas tienen una estructura semejante, "SI *condición* ENTONCES *acción*", su significado y su uso son completamente diferentes, por las razones siguientes:

1. Los comandos tienen un *carácter secuencial*: se ejecutan en un momento determinado dependiendo del lugar que ocupan dentro de la secuencia de instrucciones. Las reglas, en cambio, tienen un *carácter modular*, por lo que —en principio— pueden ser añadidas a la base de conocimientos sin preocuparse del orden. El motor de inferencia es el que determinará en cada momento cuáles de las reglas debe examinar y ejecutar, en función de la información disponible. El concepto de *encadenamiento*, hacia adelante o hacia atrás, es exclusivo de las reglas y no nada que ver con el uso de comandos IF-THEN.

Como consecuencia, los SBR tienen mecanismos de control del razonamiento, mencionados anteriormente, tales como agendas, prioridades, activación y desactivación de conjuntos de reglas, metarreglas, refractariedad, actualidad y especificidad, etc., muy distintos del orden rígido que caracteriza la ejecución secuencial de comandos.

2. La aplicación de las reglas se basa en la *comparación de patrones*. El uso de *variables* está relacionado con el carácter modular, pues es lo que hace posible que una misma regla pueda adaptarse a distintas situaciones, dando lugar a conclusiones diferentes cuando la variable aparece en el consecuente. Este uso de las variables es específico de las reglas y es completamente diferente del uso de variables en los lenguajes de programación tradicionales (ver el último comentario de [6.2.3]).

3. La distinción entre *dependencia reversible e irreversible* discutida en la sección 6.3.3 es otro rasgo distintivo de los sistemas basados en reglas, que permite retractar una conclusión cuando deja de ser cierta la información que permitió deducirla. Claramente, un comando IF-THEN nunca podrá deshacer la acción aunque deje de cumplirse la condición que contenía.

4. El hecho de que el *conocimiento* contenido en un sistema basado en reglas sea *explícito* permite posibilidades que escapan por completo a los programas basados en comandos, tales como la capacidad de *explicar* su razonamiento, de *reorganizar* el conocimiento que posee y de *aprender* de sus errores.

5. El *tratamiento de la incertidumbre* es otro de los rasgos más típicos de los SBR. Ya hemos mencionado anteriormente cómo se realiza este proceso en el sistema MYCIN. En cambio, los comandos IF-THEN nunca admiten grados de verdad intermedios, ni en la condición ni en la conclusión.

Surge entonces la siguiente pregunta: si los sistemas basados en reglas muestran tantas ventajas frente a la programación basada en comandos, ¿por qué siguen utilizándose aún los lenguajes tradicionales, como FORTRAN, C o Pascal? Una respuesta posible se basa en la inercia histórica: estos programas se mantienen por la gran cantidad de rutinas que tienen disponibles. Sin embargo, esta explicación no es suficiente. Las razones principales son dos: la eficiencia y la economía de recursos. En efecto, siempre para un problema exista una solución analítica —un algoritmo— cualquier programa tradicional, al no necesitar realizar una búsqueda heurística, será más rápido que un sistema experto. Además, cualquier "sistema experto", por pequeño que sea, necesita un motor de inferencia completo, por lo que ocupará más memoria y consumirá más recursos que un programa tradicional.

Por otro lado, las tareas de bajo nivel, como lectura y escritura de archivos, gestión de base de datos o acceso a dispositivos son más fáciles de realizar en lenguajes tradicionales. De hecho, el motor de inferencia de un sistema basado en reglas se construirá a partir de un algoritmo y, aunque

diseñáramos un "motor de inferencia heurístico" tarde o temprano tendríamos que recurrir a la programación basada en comandos, del mismo modo que un lenguaje de alto nivel necesita un compilador para llegar al lenguaje de la máquina. Por tanto, no es posible ni aconsejable prescindir completamente de la programación algorítmica, a pesar de las numerosas ventajas que presentan los sistemas basados en conocimiento.

6.7.2 Comparación con la Lógica de Predicados

En su formulación más simple, las reglas pueden considerarse como una versión reducida de la lógica de predicados. Tal como señala Clancey [1993], el paradigma de las reglas guarda semejanza con la demostración automática de teoremas, aunque en este nuevo método las proposiciones lógicas —las reglas— no son contempladas como datos para un programa —el demostrador de teoremas— sino como el programa mismo.

Por otra parte, la principal diferencia entre ambos métodos es que las reglas reducen la expresividad y la capacidad de inferencia con el fin de lograr una mayor eficiencia. La limitación en la *expresividad* consiste en la imposibilidad, o al menos en las restricciones— a la hora de utilizar cuantificadores existenciales. La reducción en la capacidad de *inferencia* se observa al comprobar que de la regla "$p \rightarrow q$" y de la afirmación "$\neg q$" puede deducirse "$\neg p$" en lógica, mientras que un sistema basado en reglas no puede realizar ninguna otra inferencia a parte de establecer el consecuente cuando se ha afirmado el antecedente; dicho con otras palabras, las reglas aplican el *modus ponens* pero son incapaces de implementar el *modus tollens*. Por este motivo, el encadenamiento de reglas, a pesar de estar basado en el principio de resolución, está lejos de extraer todas las conclusiones que dicta la lógica, aunque las que realiza las obtiene de modo mucho más eficiente.

Una ventaja adicional de las reglas, además de la eficiencia, es la capacidad de tratar la incertidumbre que aparece en prácticamente todos los problemas de la vida real. De este modo, las reglas superan una de las limitaciones de la lógica clásica y se acercan a las lógicas modales (sec. 5.5.1), al razonamiento no monótono (sec. 5.5.3) y al razonamiento aproximado (sec. 6.6). Sin embargo, insistimos en que el tratamiento de la incertidumbre mediante reglas es objeto de un intenso debate [Díez, 1994, cap. 2].

6.7.3 Crítica de los Sistemas Basados en Reglas

La controversia principal en cuanto al fundamento teórico de las reglas se refiere a su carácter declarativo frente al imperativo. En esta sección y en la anterior hemos señalado la semejanza de las reglas con la lógica (indudablemente declarativa) y las diferencias frente a la programación basada en comandos (claramente imperativa). Sin embargo, la cuestión no es tan simple. De hecho, si observamos con detenimiento cualquier sistema experto *real*, encontraremos muchos rasgos que apartan a las reglas del carácter puramente declarativo. Por ejemplo, es frecuente encontrar cláusulas que no contienen información relativa al dominio, sino que se han introducido para controlar la inferencia. El tener que ordenar cuidadosamente las cláusulas dentro de cada reglas es ya un signo que contradice el carácter puramente declarativo de la inferencia basada en reglas.

Se observa así que el diseñador de un sistema basado en reglas debe tener muy en cuenta cuál va a ser el flujo de información en la ejecución del programa, por los tres motivos señalados anteriormente: por eficiencia, por el diálogo con el usuario e incluso por el propio contenido de la inferencia, es decir, por las conclusiones a las que puede dar lugar. En general es mucho más difícil predecir cómo va a comportarse un sistema basado en reglas que un método algorítmico, por lo que la labor de programación es mucho más compleja [Jackson, 1990, sec. 12.1 y apéndice B.4]. En consecuencia, cuando un sistema experto crece por encima de cierto límite se hace muy costoso o prácticamente imposible comprobar la consistencia del conocimiento que posee. En particular, la adición de una nueva regla puede modificar el comportamiento del programa de forma imprevisible.

En el fondo, estas desventajas son consecuencia de que las reglas no son completamente declarativas y de que la *modularidad*, pregonada como una de sus principales cualidades es sólo sintáctica, no semántica. Dicho de otro modo, la base de conocimientos puede crecer indefinidamente mediante la adición de nuevas reglas, pero eso no significa que siga siendo consistente. Por tanto, se hace necesario buscar métodos para estructurar la base de conocimientos, pues sin ellos puede resultar muy difícil depurar y ampliar un sistema basado en reglas.

El otro punto por el que el uso de reglas es más criticado se refiere al tratamiento de la incertidumbre, que ya hemos mencionado. Dado que se trata de un tema algo complejo, no vamos a tratarlo aquí; el lector puede encotrar una discusión detallada en [Pearl, 1988, sec.1.2] y en [Díez, 1994, sec. 2.4].

6.8 APÉNDICE: EXPRESIVIDAD Y TRATABILIDAD

En el capítulo anterior, vimos que la lógica de proposiciones tiene una capacidad expresiva muy limitada. La lógica de predicados posee mayor expresividad: en ella podemos representar frases como "Todos los hombres son mortales" o "Juan tiene un hijo (es decir, existe alguien que es hijo de Juan)". En lógicas de orden superior podemos representar expresiones como "Todos los mamíferos tienen alguna propiedad en común", que escapan a la lógica de predicados. En principio, debemos escoger el mecanismo de representación del conocimiento que nos proporcione la mayor **expresividad** posible.

Ahora bien, para construir un programa inteligente no basta con representar el conocimiento: *es necesario realizar inferencias*. Desde este punto de vista, hay que tener en cuenta que la lógica de proposiciones es decidible, mientras que la de predicados es sólamente semidecidible y las de orden superior son indecidibles. Se observa así que una mayor expresividad puede llevar a la imposibilidad de demostrar algunos teoremas y, en consecuencia, a la imposibilidad realizar ciertas inferencias.[7]

Más aún, en la práctica no basta saber que un problema tiene solución computacional: hace falta que pueda resolverse dentro del tiempo y del espacio de memoria disponibles.[8] En IA, como en cualquier otra rama de la informática, la eficiencia es una cuestión primordial. La mayor parte de las aplicaciones reales (robótica, sistemas expertos, comprensión del lenguaje natural, etc.) exigen que la respuesta esté disponible en unos segundos o, a lo sumo, en unos minutos; si no, el programa es inútil.

En resumen, para que un problema sea **tratable** hace falta que exista un método para resolverlo (en lógica, es la cuestión de la **decidibilidad**) y que el método sea suficientemente rápido cuando se ejecuta sobre un computador (es la cuestión de la **eficiencia**). Generalmente, al aumentar la expresividad suele disminuir la tratabilidad, y vice versa. Hemos visto ya que es así en el caso de la lógica; también en el razonamiento basado en marcos y en el razonamiento

[7] Esta situación es similar a la que ocurre en teoría de autómatas: cuanto más potente es el lenguaje, más potente debe ser el autómata que lo reconoce: para lenguajes regulares basta tener un autómata finito; para lenguajes libres de contexto se necesita por lo menos un autómata de pila; para reconocer un lenguaje estructurado por frases hace falta una máquina de Turing.

[8] Un ejemplo claro es el ajedrez: resultaría sencillo construir un programa que evaluara *todas* las jugadas posibles y escogiera la mejor; sin embargo, no existe ni se podrá existir jamás un ordenador capaz de ejecutar el programa completo, aunque esté funcionando durante millones de años.

temporal se da el mismo fenómeno. (Una discusión más amplia, en la que pueden encontrarse las referencias originales, aparece en [Kramer & Mylopoulos, 1992].)

En la representación computacional del conocimiento no podemos renunciar a ninguno de estos dos factores: la *expresividad* es necesaria para poder *plantear el problema* y la *tratabilidad* es necesaria para llegar a *resolverlo*. Por eso se hace necesario buscar un compromiso entre ambos. Hablando *grosso modo,* podemos decir que los métodos que estudiamos en este libro (reglas, redes y marcos) intentan ofrecer mayor expresividad que la lógica de proposiciones, aunque sin llegar a la capacidad de la lógica de predicados.

Así, en cuanto a la capacidad de representación (expresividad), las reglas que hemos estudiado en este capítulo contienen variables y cuantificadores universales implícitos, pero en general no permiten representar cuantificadores existenciales, por lo que se quedan claramente a medio camino entre las dos lógicas mencionadas. En cuanto a la inferencia, las reglas ni siquiera cubren toda la capacidad de la lógica de proposiciones, tal como hemos comentado anteriormente.

Además, dentro de un mismo formalismo —reglas, redes o marcos— existen sistemas que priman uno de los dos factores, sacrificando en cierta medida el otro. Por ejemplo, las reglas de GoldWorks son mucho más expresivas que las de Nexpert, y esto provoca que el primero sea mucho menos eficiente que el segundo.

6.9 BIBLIOGRAFÍA RECOMENDADA

Existen al menos tres libros "clásicos" sobre este tema. El primero de ellos es el de Buchanan y Shortliffe [1984], "*Rule-based expert systems: The MYCIN experiments of the Stanford Heuristic Programming Project*", en que cada capítulo es una versión corregida y actualizada de los artículos originales aparecidos en diversas revistas sobre MYCIN y los demás sistemas expertos que derivaron de él. Su interés va mucho más allá de un proyecto particular, pues los métodos que describe siguen ejerciendo una influencia decisiva sobre todos los sistemas expertos de la actualidad.

Muchos de los libros dedicados a sistemas expertos dan por supuesto que son se trata de sistemas basados en reglas, o al menos se centran en éstos muy especialmente. Tal es el caso de los otros dos "clásicos" que vamos a

recomendar. Uno es el de Hayes-Roth, Waterman y Lenat [1983], *Building Expert Systems*, en que colaboraron los autores más importantes de este campo y, tal como su título indica, explica los pasos para construir un sistema experto (basado en reglas). En tercer lugar, el libro de Waterman [1986], *A Guide to Expert Systems*, además de sintetizar las ideas más importantes de la obra anterior y ofrecer abundantes consejos prácticos, hace una extensa revisión de numerosos sistemas expertos y herramientas desarrollados hasta esa fecha, con gran cantidad de referencias bibliográficas.

7 REDES ASOCIATIVAS

F. J. Díez Vegas

En su sentido más amplio, una red está formada por un conjunto de nodos unidos entre sí por cierto tipo de enlaces. Dentro de la informática existen muchos tipos de redes, de naturaleza muy diferente. En algunos casos se trata de sistemas físicos conectados entre sí, tales como las redes de ordenadores. *En otros casos son modelos teóricos, como las* redes asociativas *que vamos a estudiar en seguida. Los trabajos sobre* redes neuronales, *por poner otro ejemplo, suponen que cada neurona va a ser implementada por un procesador físico, aunque en la práctica la mayor parte de los estudios se realizan sobre simulaciones mediante ordenador.*

En las ***redes asociativas****, que constituyen el objeto de este capítulo, cada nodo representa un concepto (en algunos modelos puede representar también una proposición) y los enlaces corresponden generalmente a relaciones de inclusión, pertenencia, causalidad, o a categorías gramaticales, como verbo principal, sujeto, objeto, complementos, etc. Entre ellas, se conocen como* ***redes semánticas*** *las destinadas a representar o a comprender el lenguaje natural. Veremos más adelante que en la mayor parte de los modelos de redes semánticas, a diferencia de otros modelos de redes asociativas, hay varias clases diferentes de enlaces, que se identifican mediante etiquetas o —en la representación gráfica— mediante distintos tipos de flechas.*

Además de las redes semánticas, vamos a estudiar en este capítulo las ***redes de clasificación*** *y las* ***redes causales****. Conviene señalar que cada uno de estos tres grandes apartados engloba varios modelos muy diferentes entre sí. Por otra parte, existe un fuerte solapamiento entre redes semánticas y redes de clasificación, como iremos viendo en las secciones correspondientes. Por ello y*

a pesar de los diferentes criterios propuestos en la literatura, la clasificación de estos modelos sigue siendo difícil.

Antes de pasar adelante, conviene repasar las "Nociones sobre grafos" *mostradas en la sección 3.2.1. El concepto de red es casi sinónimo de grafo, aunque este último término posee un significado matemático preciso, mientras que el concepto de red es más amplio. La diferencia principal entre ambos es que en las redes puede haber diferentes tipos de enlaces, e incluso enlaces que unen más de dos nodos. (En redes suele hablarse de enlaces y en grafos de arcos.)*

7.1 GRAFOS RELACIONALES

7.1.1 Modelo de Memoria Semántica, de Quillian

En su tesis doctoral, Ross Quillian [1968] trató de construir un modelo computacional de la memoria humana, más concretamente, un programa de ordenador capaz de almacenar información sobre el significado de las palabras con el fin de que pudiera llegar a comprender el lenguaje natural. El modelo de Quillian consiste en representar el significado de los términos de la lengua inglesa de forma semejante a como aparecen en un diccionario. La representación consta de un conjunto de nodos unidos entre sí por diferentes clases de enlaces asociativos.

La Fig. 7.1 muestra un ejemplo de la red semántica en que aparecen representados dos significados diferentes de la palabra planta y el significado de alimento:

Planta-1: estructura viva que no es un animal, tiene hojas y toma su alimento del aire, del agua o de la tierra.
Planta-2: aparato que las personas utilizan en la industria.
Alimento: una cosa que un ser tiene que tomar dentro de sí para mantenerse vivo.

En la figura se observa que cada plano contiene un nodo-tipo ("*type node*"), encerrado en un óvalo, y varios nodos-réplica ("*token nodes*"). Hablando grosso modo, podríamos decir que el nodo-tipo corresponde a los encabezamientos de las definiciones, que en un diccionario suelen aparecer con negrita, y los nodos-réplica son las demás palabras que aparecen en la definición.

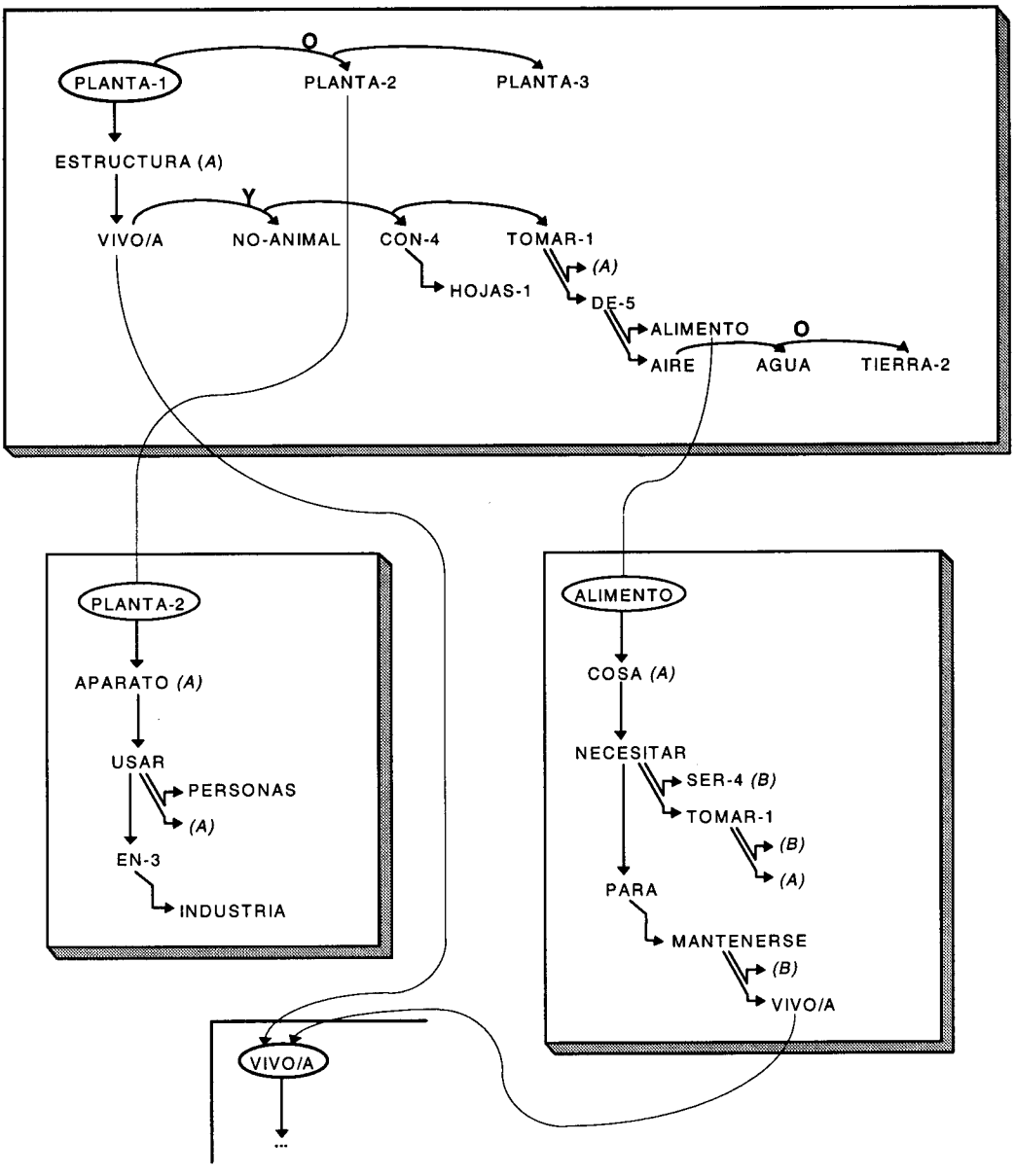


Fig. 7.1. Red semántica de Quillian

Conviene destacar el uso de variables para indicar conceptos que aparecen en el mismo plano de definición (es decir, en el mismo rectángulo). Así, en la definición de alimento aparecen dos variables: "*(A)*" indica el alimento y "*(B)*" el

ser que toma el alimento para mantenerse vivo. Una forma alternativa de solucionar el problema de los conceptos que desempeñan varios papeles en una definición consistiría en establecer enlaces entre nodos idénticos. En realidad, ambas soluciones son equivalentes, y el optar por una u otra depende muchas veces de cuál de ellas da lugar a una representación gráfica más clara y sencilla.

Los números se utilizan para distinguir los diferentes significados de un mismo término. Observar que con-4 representa "el cuarto significado de la palabra *con*" y tomar-1 "el primer significado de *tomar*". Así se evitan ambigüedades en las definiciones, pues queda claro, por ejemplo, que las hojas-1 que tiene una planta son distintas de las hojas-2 que tiene un libro.

En las redes de Quillian aparecen seis tipos de enlaces:

1. **Subclase**: une un nodo-tipo con la clase a la que pertenece. Por ejemplo, planta-1 es una estructura, planta-2 es un aparato y alimento es una cosa.
2. **Modificación**: une dos nodos-réplica, de modo que el segundo modifica el alcance del primero. Por ejemplo, en el plano superior de la Fig. 7.1, hay un enlace que modifica el concepto de estructura.
3. **Disyunción**: lleva la etiqueta "O". Une dos o más nodos. En la figura anterior hay un enlace de este tipo para unir los tres posibles significados de "planta".
4. **Conjunción**: lleva la etiqueta "Y". Al igual que el anterior, une dos o más nodos. Por ejemplo, las cuatro características de una planta en cuanto estructura (viva, no animal, con hojas y que se alimenta del aire, etc.) están unidas entre sí por un enlace de este tipo.
5. **Propiedad**: une tres nodos, *A, B* y *C*, según muestra la Fig. 7.2, donde *A* es la relación, *B* el sujeto y *C* el objeto. Por ejemplo, en el significado de planta-2, la relación es "usar", el sujeto es "personas" y el objeto es el aparato (la propia planta industrial). Hay algunos enlaces de este tipo, como el que une con-4 y hojas-1, en que el elemento *B* está ausente, y sólo aparecen *A* y *C*.

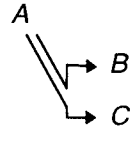


Fig. 7.2. Enlace de tipo "propiedad" entre tres nodos.

6. **Referencia al tipo**: van desde un nodo-réplica al correspondiente nodo-tipo; se dan siempre entre planos diferentes. Podemos entender mejor su finalidad con una metáfora. Supongamos que al consultar una definición en un diccionario tropezamos con una palabra que no conocemos; podemos entonces buscar la definición de esta palabra. Ahora bien, la búsqueda es más rápida si sabemos dónde se encuentra esta segunda definición. Por ejemplo, en la definición de planta-1 aparece el término alimento, que lleva un enlace del tipo que estamos tratando, dirigido hacia el nodo-tipo, contenido en el plano que define su significado. Otros dos enlaces de esta clase van desde los nodos-réplica vivo/a hasta el nodo-tipo en el plano de definición de este adjetivo.

Hemos comentado ya anteriormente (cap. 5) que el interés de representar computacionalmente el conocimiento consiste ante todo en poder realizar **inferencias** con el fin de obtener nueva información. El trabajo de Quillian se encuadra dentro de los estudios que se realizaban en aquella época sobre la comprensión del lenguaje natural. Sin embargo, esta meta planteada en términos generales resultaba demasiado ambiciosa, por lo que la tesis de este autor se limitó a una forma muy sencilla de inferencia: cómo comparar el significado de dos palabras, suponiendo que esta tarea significa un primer paso hacia la comprensión de frases y relatos.

Para conocer el resultado producido por el programa de Quillian, veamos el siguiente ejemplo:

Comparar: PLANTA, VIVO/A.
1ª intersección: VIVO/A
— Una planta-1 es una estructura viva.
2ª intersección: VIVO/A
— Una planta-1 es una estructura viva que obtiene alimento del aire, del agua o de la tierra-2. Este alimento es una cosa que un ser-4 tiene que tomar-1 para-3 mantenerse-2 vivo/a.

Las dos soluciones que ofrece el programa para la comparación de estas palabras corresponden a dos caminos diferentes. El primero de ellos consta de tres enlaces, que unen planta-1 (nodo-tipo), con estructura, con vivo/a (nodos-réplica) y con vivo/a (nodo-tipo). El otro camino, correspondiente a la segunda solución, consta de doce enlaces, de los cuales los dos primeros coinciden con el camino mencionado anteriormente.

La forma en que el programa realiza la inferencia consiste en partir de los dos nodos-tipo cuyo significado se quiere comparar y activar los nodos vecinos de cada uno de ellos, dando lugar así lugar a dos *esferas de activación* de radio 1. Cada intersección de ambas esferas corresponde a un camino que une los dos nodos, es decir, una relación entre los significados de ambos conceptos. En el paso siguiente se activan los nodos vecinos de los anteriores, obteniendo esferas de radio 2, y así sucesivamente, tantas veces como se desee, en busca de nuevas soluciones. El resultado final del programa se obtiene convirtiendo cada camino en una frase explicativa. Hemos de señalar que el trabajo de Quillian se ocupaba de la lengua inglesa; en el caso de realizar un programa similar para la lengua castellana tendríamos en este punto la dificultad añadida de generar concordancias para las palabras; por ejemplo, en vez de tener un único artículo determinado "*the*", necesitaríamos cuatro artículos (el, la, los, las) y habría que buscar la concordancia con el sustantivo correspondiente.

Pero donde más se nota la dependencia del trabajo de Quillian con respecto a un idioma particular es en el significado de las palabras. Hay muchos términos ingleses polisémicos, como "*table*", que en castellano se traducirían con palabras muy diferentes (mesa o tabla); y viceversa. Esta dependencia del idioma no es de extrañar, pues el programa se orienta hacia la representación y comprensión del lenguaje natural. Sin embargo, el propio Quillian señaló la conveniencia de pasar de la representación de palabras a la representación de conceptos, sin depender de ningún idioma particular. Posteriormente, veremos cómo Schank intentó resolver este problema.

7.1.2 Sistema SCHOLAR, de Carbonell

Inspirado en el trabajo de Quillian, Jaime Carbonell [1970] construyó SCHOLAR como sistema de enseñanza asistida por ordenador. Lo que más nos interesa ahora de este programa es que poseía una base de conocimiento estructurada en forma de red, lo cual le permitía generar dinámicamente preguntas para el alumno y responder a las que éste le planteaba. Además, tenía la ventaja de que el módulo encargado de generar y responder a las preguntas (lo que hoy en día llamaríamos el *motor de inferencia*) era prácticamente independiente del dominio representado en la red, con lo cual podría utilizarse para generar fácilmente programas similares para la enseñanza de otros temas.

Una de las aportaciones más trascendentes del trabajo de Carbonell es la distinción entre los nodos que representan conceptos, tales como `LATITUD` o `PAÍS`, y los que representan ejemplos particulares (en el cap. 8 veremos que hoy

en día suelen denominarse *instancias*), tales como SURAMÉRICA, ARGENTINA o URUGUAY.

Otra de las ideas fundamentales de SCHOLAR, que había sido sugerida anteriormente por Quillian [1969], consiste en definir cada nodo mediante dos elementos: la clase a la que pertenece y una lista de propiedades. Una *propiedad* viene dada por un atributo, un valor y, opcionalmente, otras subpropiedades. Por ejemplo, el nodo ARGENTINA está definido como una instancia de la clase PAÍS (ver Fig. 7.3); la propiedad correspondiente al atributo LOCALIZACIÓN tiene como valor SURAMÉRICA y como subpropiedades la LATITUD y los PAÍSES-LIMÍTROFES. Además, cada propiedad puede llevar asociadas unas funciones destinadas a lograr inferencias más eficientes; éstas funciones se utilizan hoy en día en la representación basada en marcos, con el nombre de *demonios* (sec. 8.2.2).

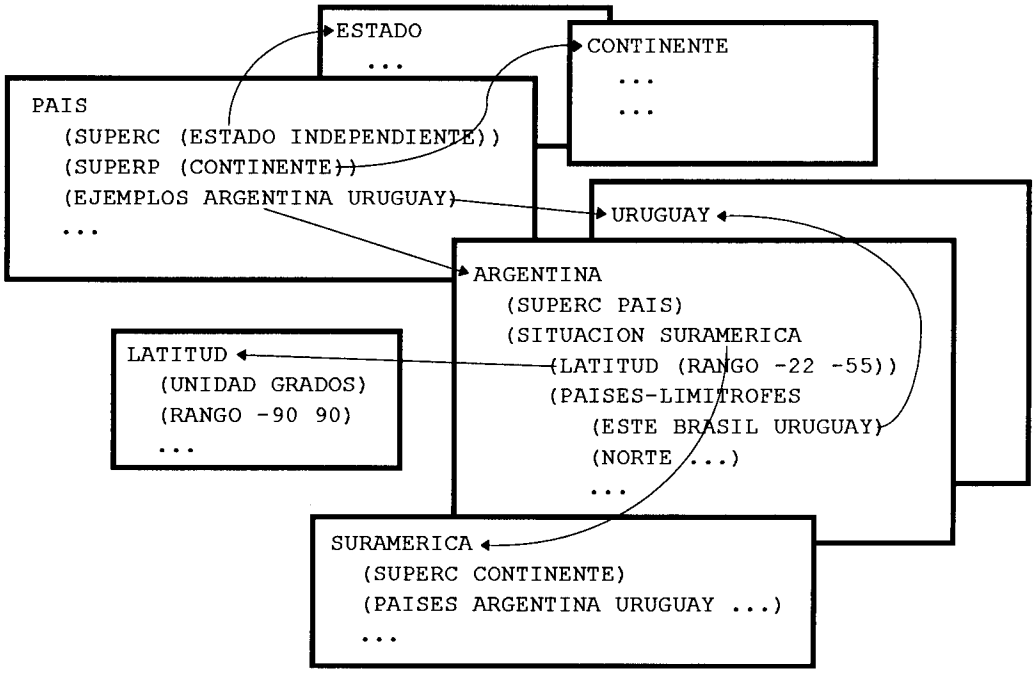


Fig. 7.3. Sistema SCHOLAR, de Carbonell

El modelo de Carbonell incluye otras características interesantes pero menos relevantes para este capítulo. Por ahora, basta señalar que la distinción entre conceptos (o clases) e instancias, y la asignación de propiedades en forma

de pares atributo-valor constituyen en la actualidad dos características esenciales de la representación basada en marcos y, de una u otra forma, de la mayor parte de los modelos de redes semánticas.

Sin embargo, el hecho de que SCHOLAR no está dedicado a la representación del significado de los conceptos ni a la comprensión del lenguaje natural, es discutible hasta qué punto se puede considerar una red semántica. De hecho, hoy en día no suele considerarse como tal (ver, por ejemplo, [Sowa, 1992]).

7.1.3 Grafos de Dependencia Conceptual, de Schank

Roger Schank [1972, 1975], al igual que Ross Quillian, estaba interesado en la comprensión del lenguaje natural mediante ordenador. Sin embargo, su perspectiva era diferente, puesto que Schank se apoyaba más en los estudios de la lingüística que en los modelos psicológicos de la memoria humana. Otra diferencia entre ambos trabajos es que Schank trató de abordar el problema del lenguaje independientemente de cualquier idioma particular, mientras que el estudio de Quillian dependía esencialmente de la lengua inglesa; dicho de otro modo, a Schank no le interesaba representar las *palabras* sino los *conceptos*.

La idea principal de este método consiste en interpretar —representar— cualquier frase mediante un número de **primitivas**, a saber: 6 categorías conceptuales, 16 reglas sintácticas y varias acciones primitivas (Schank afirma que 12 de ellas son suficientes para comprender gran parte del lenguaje natural). Las *categorías conceptuales* son: objeto físico (una cosa o un ser vivo), acción, atributo de un objeto físico, atributo de una acción, tiempo y localización. Las *reglas sintácticas* determinan los diferentes tipos de relación que pueden existir entre los elementos de una frase; en seguida mostraremos algunas de ellas. Y, por último, entre las *acciones primitivas* se encuentran las siguientes:

PTRANS	Transferir físicamente (cambiar de lugar un objeto)
ATRANS	Transferir una relación abstracta, como posesión o control
MTRANS	Transferir mentalmente (decir, contar, comunicar, etc.)
PROPEL	Empujar
MOVE	Mover un miembro de un animal
GRASP	Coger, atrapar
INGEST	Ingerir
etc.	

Explicaremos estas ideas mediante algunos ejemplos. En la Fig. 7.4 tenemos la representación de la frase "Juan bebe agua".

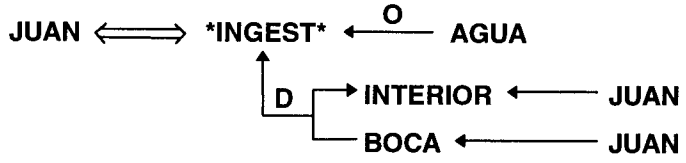


Fig. 7.4. Representación de "Juan bebe agua" mediante un grafo conceptual.

El elemento central del diagrama viene dado por una de las acciones primitivas: *INGEST* (ingerir). También podemos reconocer cuatro relaciones sintácticas: *sujeto-verbo* (en forma de doble flecha), *objeto-verbo* (una flecha con una O), *posesión* o *parte-de* (dos flechas sencillas indican que se trata de la boca y del interior de Juan) y *dirección* (una flecha marcada con una D indica la dirección en que el agua es ingerida).

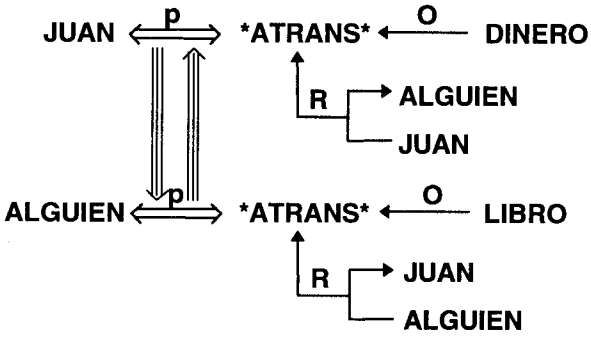


Fig. 7.5. Representación de "Juan compró un libro" mediante un grafo conceptual.

Veamos otro ejemplo. Si tratamos de analizar el significado de "Juan compró un libro", observamos que en la acción de comprar hay un doble intercambio: el libro pasa del vendedor al comprador y el dinero en sentido contrario. Cada una de estas transferencias corresponde a la acción primitiva ATRANS (si la transferencia es física, es decir, si Juan se lleva el libro en la mano y paga en efectivo, podríamos representarla mediante PTRANS). La Fig. 7.5 muestra la frase anterior en forma de grafo. En ella aparecen dos nuevas relaciones sintácticas: la de *recepción* (la primera flecha marcada con una R indica que alguien recibe el dinero de Juan) y la de *causalidad* (cada una de las

flechas triples indica que una transferencia depende de la otra). Otro elemento nuevo que nos muestra este ejemplo consiste en indicar con sendas pes que las transferencias se realizaron en el pasado.

De este modo, cada frase se descompone en elementos simples con el fin de tener una representación independiente del idioma; en efecto, los ejemplos anteriores muestran que el grafo conceptual que representa el significado de una frase sería el mismo para cualquier otra lengua, y sólo cambiaría la etiqueta que lleva cada uno de los símbolos-conceptos.

Las **ventajas de utilizar primitivas** son principalmente dos. La primera es que éstas *determinan unívocamente* la representación del conocimiento, mientras que si no existiera un conjunto limitado de primitivas, las posibilidades de representar cada frase serían numerosas, y por tanto no se podría especificar la forma de construir un intérprete. La segunda ventaja está muy relacionada con la primera, y consiste en la posibilidad de construir un *intérprete* capaz de realizar *inferencias*; sin un número limitado de elementos y relaciones esta tarea sería prácticamente imposible. (Obsérvese el contraste con las redes semánticas de Quillian, que se limitaban a comparar el significado de las palabras, sin extraer nueva información.)

Uno de los tipos de **inferencias** que pueden darse mediante estos grafos [Schank & Rieger, 1974] consiste en establecer las *condiciones*. Por ejemplo, de la frase "Juan comió un filete" se infiere que Juan existía, que el filete existía y que ambos estuvieron en contacto en algún momento. Otro de los tipos de inferencias consiste en hallar las *causas*, y entre ellas las *intenciones*. Así, de "Juan pidió el libro a María" se infiere que María tenía un libro y que Juan quería leer ese libro. También se puede inferir el *resultado* de las acciones: de "Juan comió un filete" se deduce que el filete dejó de existir, y de "Juan viajó a Barcelona" se deduce que Juan estuvo en Barcelona.

Reflexionando sobre el compromiso planteado entre la expresividad y la eficiencia de que hablábamos en la sección 6.8, recordamos que para que un sistema computacional sea eficiente resulta aconsejable que haya un número limitado de elementos y relaciones. En los grafos, este propósito se cumple utilizando un conjunto reducido de *primitivas*. Por otro lado, el sistema debe tener la potencia y flexibilidad necesarias para abordar el problema que queremos resolver; y uno de los problemas más complejos que trata de resolver la inteligencia artificial —probablemente el más complejo— es la comprensión del lenguaje natural. Los grafos conceptuales resuelven este problema mediante

la *descomposición* de cualquier frase en los elementos y acciones más simples que intervienen.

A pesar de que el programa de Schank significó un avance respecto del modelo de Quillian, la idea de utilizar un conjunto mínimo y completo de primitivas también conlleva algunos inconvenientes. Veamos algunas de las críticas que se plantean:

1. En primer lugar, se cuestiona la validez de definir unas primitivas universales. En efecto, ciertos autores consideran que el significado de una frase, salvo en los casos más sencillos, es inseparable de la lengua en que se encuentra formulada. De ahí se deduce, por un lado, que es imposible lograr traducciones fidelignas y, por otro, que no tiene sentido intentar construir grafos conceptuales independientes del idioma.

2. Otra dificultad consiste en el hecho de que los grafos conceptuales requieren una descripción demasiado detallada de las acciones. Los dos ejemplos anteriores demuestran cómo frases muy sencillas, de tres o cuatro palabras, dan lugar a grafos con muchos más elementos y una estructura relativamente compleja; además, se pierde la relación intuitiva entre la frase y el grafo que la representa. Si hubiéramos escogido otros ejemplos menos triviales, los grafos habrían resultado difíciles de interpretar a simple vista. Más aún, en algunos casos la descomposición puede resultar demasiado artificiosa y acabar ocultando la esencia de algunas acciones, en que el todo es más que la suma de sus elementos.

3. El problema más importante surge a la hora de estudiar cuántas y cuáles deben ser las primitivas. Esta cuestión está muy relacionada con la anterior, es decir, con el grado de detalle que debemos alcanzar en la descomposición. De hecho, en un trabajo posterior Schank y Carbonell [1979] admitieron la posibilidad de utilizar primitivas de nivel superior, tales como "comprar", con lo que no sería necesario descomponer este verbo en dos transferencias. Algunos autores, como Sowa [1992], afirman que no tiene sentido definir un conjunto mínimo de primitivas, sino que es más útil trabajar con distintos niveles de detalle, de modo que el sistema pueda explicitar los elementos cuando sea necesario.

Éstas son las críticas que se plantean en cuanto al uso de las primitivas. Sin embargo, el trabajo de Schank es cuestionable también por centrar la representación en torno al verbo. Como esta es una crítica aplicable a otros sistemas desarrollados en la misma época (los de Simmons, Rumelhart,

Schubert, Cercone y otros), lo vamos a tratar en una sección a parte. Concluimos esta sección recordando que, a pesar de sus limitaciones, los grafos conceptuales significaron un avance importante en el campo de las redes semánticas, y han influido notablemente en los sistemas que se están investigando hoy en día. En la sección 8.3 estudiaremos el trabajo posterior de Schank sobre la comprensión del lenguaje natural.

7.1.4 Problemas de los Grafos Relacionales

En los modelos que hemos comentado hasta ahora, cada nodo corresponde un concepto. Sin embargo, estos tipos de redes presentan serias limitaciones a la hora de abordar el lenguaje natural; de hecho, ni siquiera son capaces de representar la lógica de primer orden (cap. 5). En efecto, estos grafos son capaces de tratar un operador existencial implícitamente mediante un nodo genérico; por ejemplo, la frase "Juan tiene un amigo" puede representarse mediante un grafo. Sin embargo, estos modelos de redes son incapaces de representar el operador universal, por lo que una frase como "Todo hombre tiene algún amigo" o "Hay un hombre a quien todos admiran" no pueden ser traducidas a grafos relacionales.

Otra de las limitaciones de estos grafos es la dificultad o imposibilidad de tratar la interacción entre más de dos proposiciones. Así, una afirmación como "Cuando es de noche baja la temperatura", puede representarse trazando un enlace de simultaneidad entre los dos verbos. Sin embargo, la afirmación "Cuando es de noche y hay niebla resulta peligroso conducir" indica una implicación cuyo antecedente viene dado por la *conjunción* de dos proposiciones. Para poder traducirla a un grafo se hace necesario poder representar la conjunción de proposiciones como un nodo (o como un contexto) para poder trazar un enlace hacia el consecuente de la implicación. Algo parecido puede decirse de la afirmación "Luis piensa que si ahorra dinero podrá irse de viaje"; en este caso, el objeto de "pensar" no es una proposición sino una implicación en la que participan dos proposiciones.

7.2 REDES PROPOSICIONALES

Una vez mostradas las limitaciones de los modelos relacionales, vamos a estudiar de ciertos tipos de redes en que los nodos no representan solamente conceptos, sino que también pueden representar sintagmas, cláusulas, frases,

párrafos e incluso historias completas. Los tres trabajos más importantes en esta línea son los de Shapiro [1971, 1979], Hendrix [1979] y Sowa [1984].

7.2.1 Redes de Shapiro

Stuart Shapiro [1979] fue el primero en implementar una red asociativa que incluyera todos los operadores y cuantificadores de la lógica de primer orden. En su sistema, llamado SNePS, cada nodo podía representar un concepto, una variable (semejante a las que se utilizan en lógica), una frase, una relación (tal como la negación de una frase) o una regla de inferencia (tal como "Quien a buen árbol se arrima buena sombre le cobija").

Veamos un ejemplo. La frase "Antonio piensa que Luisa está leyendo un libro" se representaría así:

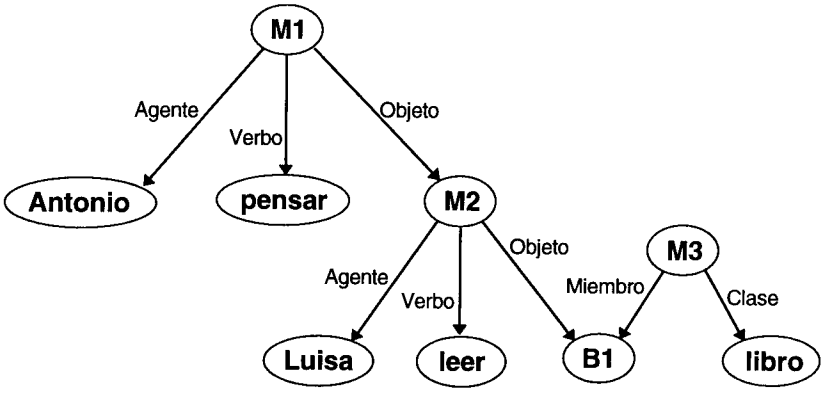


Fig. 7.6. Red proposicional de Shapiro.

En esta figura, la etiqueta "objeto" colocada sobre un arco tiene un significado gramatical. A primera vista podría parecer que el nodo "B1" es innecesario y que bastaría trazar un enlace desde el nodo "M2" hasta el nodo "libro"; sin embargo, este último representa la clase formada por todos los libros, o mejor dicho, el concepto de libro. Por eso es preciso introducir un nodo auxiliar, "B1", que represente el libro particular que Luisa está leyendo. (Recordar la distinción entre clase e instancia mencionada al hablar del trabajo de Carbonell.)

Hemos visto también en el ejemplo anterior que una proposición puede estar contenida dentro de otra proposición, teniendo así varios *niveles anidados*. También se pueden formar conjunciones, disyunciones, implicaciones, etc. en

que intervengan varias proposiciones. El contexto de cada una de ellas viene determinado por su posición en la red, y así es posible determinar el *alcance* de los cuantificadores universales y existenciales.

7.2.2 Representación mediante Grafos de Sowa

En las redes de particiones [Hendrix, 1979], y en los grafos conceptuales [Sowa, 1984] la determinación de los contextos se realiza trazando rectángulos que agrupen cierto número de nodos. Puesto que las redes de particiones son muy similares a los grafos conceptuales y éstos están mucho más desarrollados que aquéllas, nos vamos a limitar aquí a estudiar el modelo propuesto por Sowa. En esta representación, el ejemplo anterior vendría dado por el siguiente diagrama:

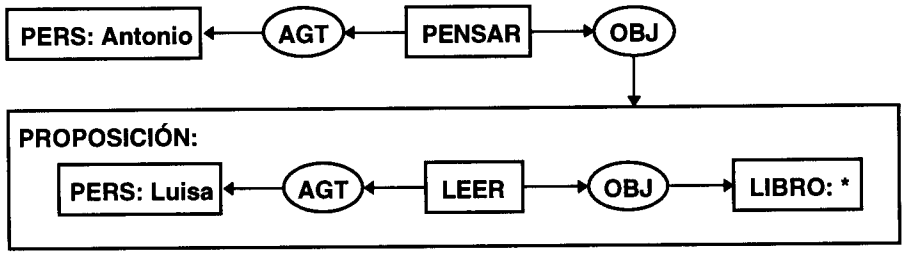


Fig. 7.7. Grafo conceptual de Sowa para el mismo ejemplo de la Fig. 7.6.

A pesar de que aparentemente la representación de Shapiro y la de Sowa son muy diferentes entre sí, si estudiamos ambos diagramas con detenimiento observamos que son más las semejanzas que las diferencias. Una semejanza entre ambos diagramas consiste en que el segundo de ellos contiene un nodo de la clase PROPOSICIÓN para representar la cláusula "Luisa lee un libro", y se corresponde exactamente con el nodo "M2" de la Fig. 7.6. Más aún, podríamos trazar un rectángulo que encerrara todo el grafo de la Fig. 7.7 y tendríamos otro nodo del tipo "PROPOSICIÓN" que correspondería al nodo "M1" de la Fig. 7.6.

En cuanto a las diferencias entre ambas representaciones, los grafos de Shapiro llevan una etiqueta junto a cada arco para indicar el tipo de relación entre los nodos; en los grafos de Sowa, en cambio, se utiliza un círculo con el mismo fin. En realidad se trata de una diferencia insignificante que afecta sólamente a la forma de dibujar los grafos y no a la propia representación.

Otra diferencia es que Sowa indica explícitamente en cada nodo la clase a la que pertenece. (Más adelante hablaremos de la organización jerárquica de estas clases.) Así, "PERS: Antonio" significa que Antonio es una persona. Ésta es una información que no aparecía en la representación de Shapiro, por lo que en la Fig. 7.6 era necesario añadir un nodo "M3" para afirmar que lo que Luisa está leyendo (denotado por "B1") es un libro. En la Fig. 7.7, en cambio, existe un nodo denominado "LIBRO: *". Observar que, en los grafos de Sowa, cuando se conoce la identidad de un nodo, se escribe su nombre junto a la clase a la que pertenece; cuando no se conoce, puede utilizarse un asterisco, que representa el operador existencial: "existe un libro". En la práctica, el asterisco suele omitirse.[1]

Tanto la representación gráfica de Shapiro como la de Sowa pueden expresarse en *notación lineal*; el grafo de la Fig. 7.7 se convertiría en:

```
[PENSAR] -
  →(AGT)→[PERS: Antonio]
  →(OBJ)→[PROPOSICIÓN: [LEER] -
            (AGT)→[PERS: Luisa]
            (OBJ)→[LIBRO]]
```

Como se puede observar, los nodos aparecen entre corchetes y las etiquetas de los arcos entre paréntesis. Esta notación tiene las ventajas de que ocupa menos espacio en una página impresa y de que es más fácil de manipular en un programa de ordenador.

Para comprobar la gran capacidad expresiva de este modo de representación, veamos otros ejemplos de cómo se representarían mediante grafos de Sowa algunas frases del lenguaje natural:

- *Un río* — `[RÍO: *] ≡ [RÍO]`
- *Un río caudaloso* — `[RÍO]→(ATR)→[CAUDALOSO]`
- *Algunos ríos* — `[RÍO: {*}]`
- *Tres ríos* — `[RÍO: {*}@3]`

[1] De hecho, el nodo "PENSAR" también podrían llevar un asterisco para indicar que son instancias particulares del concepto "pensar", y lo mismo ocurre para cualquier otro nodo que represente un verbo. También los verbos se encuentran incluidos en la jerarquía de conceptos, como veremos más adelante.

- *El río (uno específico)* [RÍO: #]
- *¿Qué río?* [RÍO: ?]
- *El río Ebro* [RÍO: Ebro]
- *Los ríos Duero y Tajo* [RÍO: {Duero, Tajo}]
- *Todo río, todos los ríos* [RÍO: ∀]
- *Cinco litros de agua*
  [AGUA]→(MEDIDA)→[CANTIDAD: 5 L.]
- *Las 10 de la mañana* [HORA: 10 a.m.]
- *Todos los seres humanos son mortales.*
  [PERS: ∀]→(ATR)→[MORTAL]
- *Cierto ser humano es mortal.*
  [PERS]→(ATR)→[MORTAL]
- *Él no ha venido.*
  (NEG)←[PROPOSICIÓN: [VENIR]-
  →(AGT)→[MASCULINO: #]
  →(PASADO)]
- *Probablemente iré mañana.*
  [PROBABLE]←(MOD)←[PROPOSICIÓN: [IR]-
  →(AGT)→[PERS: #yo]
  →(TIEMPO)→[DIA: #mañana]]
- *¿Quién es el profesor de Luis?*
  [PERS: ?]→(PROFESOR-DE)→[PERS: Luis]
- *¿Dónde está tu libro?*
  [LUGAR: ?]←(LOC)←[LIBRO: #]←(POS)←[PERS: #tú]

Obsérvese que en estos ejemplos aparecen algunos nodos marcados con el signo "#". Este signo indica que se trata de un ser o un objeto concreto, cuya identidad habrá que determinar. Cuando se dice "el libro" y se representa por "LIBRO: #" el intérprete que convierte el lenguaje natural en un grafo deberá averiguar cuál es el libro en cuestión; cuando se dice "yo" y se representa por "PERS: #yo", habrá que determinar quién ha pronunciado esa frase; y para identificar el nodo "DÍA: #mañana" hay que averiguar primero qué día es

hoy. En cambio, si se dijera "un libro" y se representara por "LIBRO: *", o simplemente por "LIBRO", no se hace necesario identificar dicho libro.

Una de las características de la mayor parte de los modelos de redes consiste en utilizar **variables** con el fin de representar un objeto o un concepto que aparece en varios lugares diferentes. Por ejemplo, en la afirmación "Quien tiene un coche, lo utiliza" aparecen dos proposiciones, ambas con el mismo sujeto y el mismo objeto directo. En la representación gráfica, nos serviremos de una línea discontinua para identificar los nodos que representan un mismo ente. (La "T" se utiliza para representar un nodo sin especificar la clase a la que pertenece):

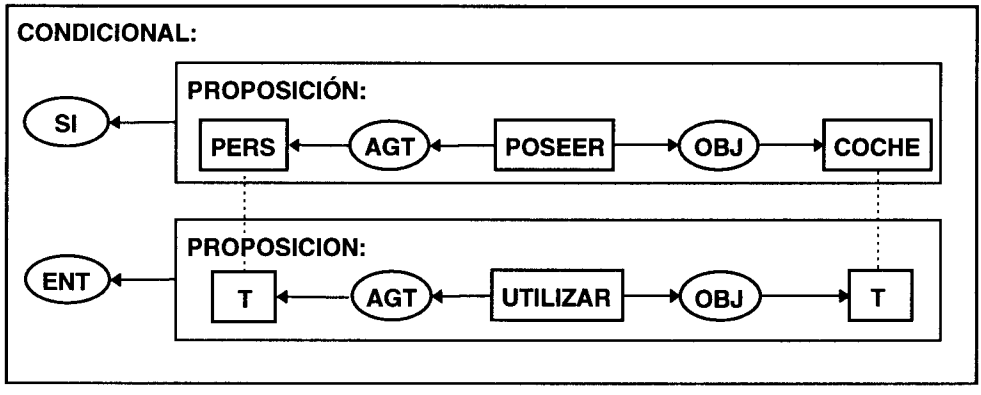


Fig. 7.8. Representación de "Quien tiene un coche, lo utiliza".

En la notación lineal, al no poder unir los nodos mediante líneas discontinuas, se utilizan variables para indicar los nodos que coinciden:

```
[CONDICIONAL:-
  (SI)→[PROPOSICIÓN: [POSEER]-
            (AGT)→[PERS: *x]
            (OBJ)→[COCHE: *y]]
  (ENT)→[PROPOSICIÓN: [UTILIZAR]-
            (AGT)→[T: *x]
            (OBJ)→[T: *y]]]
```

Obsérvese la semejanza con la representación de esta frase en la lógica de predicados:

$$\forall x, \forall y, \quad (\text{COCHE}(y) \wedge \text{POSEE}(x, y)) \rightarrow \text{UTILIZA}(x, y)$$

Las variables no sólo pueden representar un nodo-concepto, sino cualquier nodo más complejo; el siguiente ejemplo, corresponde a la frase "Dije a Ana que la clase se había suspendido y ella se lo comentó a Luis":

```
[DECIR]-
   (AGT)→[PERS: #yo]
   (RCP)→[PERS: Ana *x]
   (OBJ)→[PROPOSICIÓN:
             [CLASE: #]←(OBJ)←[SUSPENDER] *y]
   (PASADO)
[COMENTAR]-
   (AGT)→[*x]
   (RCP)→[PERS: Luis]
   (OBJ)→[*y]
   (PASADO)
```

En este caso, la variable *x* representa una persona, Ana, mientras que la variable *y* representa toda una proposición, "la clase se había suspendido", que es lo que Ana comentó a Luis.

Recapitulando lo que hemos visto hasta ahora, se observa que en un grafo de Sowa intervienen elementos de tres clases:

- **Conceptos genéricos**, tales como LIBRO, PERS, PENSAR y COMER.
- **Conceptos individuales**, tales como [PERS: Luis], que representa a una persona determinada, o [PENSAR], que representa un acto de pensamiento (ver la Fig. 7.7).
- **Relaciones conceptuales**, que pueden ser *unitarias* (PASADO, NEG, etc.), *binarias* (AGT, OBJ, ATR, LOC, etc.) e incluso *ternarias* (ENTRE, para indicar que *A* se encuentra ENTRE *B* y *C*).

El conjunto de conceptos genéricos y de relaciones conceptuales establecido inicialmente en el sistema se conoce como *conjunto canónico*, y a partir de él se pueden definir nuevos conceptos y relaciones. La definición del concepto de profesor a partir de otros conceptos (no necesariamente canónicos) podría ser la siguiente:

Concepto: PROFESOR(x) **es**

[PERS: *x] ← (AGT) ← [ENSEÑAR] → (OBJ) → [ESTUD: {*}]

Es decir, un profesor es una persona que enseña a (varios) estudiantes. Del mismo modo puede definirse relaciones conceptuales; por ejemplo, podemos definir que dos personas son hermanos/as cuando existe otra de la que ambos son hijos:

Relación: HERMANO(x,y) **es**

[PERS: *x] ← (HIJO) ← [PERS] → (HIJO) → [PERS: *y]

Por otra parte, también es posible definir *esquemas*, que corresponden a situaciones típicas. La siguiente definición

Esquema: AVE(x) **es**

[AVE: *x] ← (AGT) ← [VOLAR]

indica que una de las características más típicas de las aves es que pueden volar. Sin embargo, eso no significa que todas las aves puedan volar (de hecho, los pingüinos y las avestruces son aves que no vuelan). Por tanto, la definición de un *esquema* se diferencia de la de un *concepto* en que no implica que deba cumplirse necesariamente lo que el esquema describe. En seguida veremos su relación con el razonamiento por defecto.

7.2.3 Inferencia en Grafos de Sowa

La inferencia en este tipo de grafos se implementa mediante un conjunto de *operaciones* básicas (restricción, generalización, unión y simplificación), que vamos a describir a continuación. Supongamos que tenemos el siguiente ejemplo:

G1: [PERS] ← (AGT) ← [BEBER] → (OBJ) → [AGUA]

La **restricción** consiste, dicho *grosso modo*, en concretar más la información del grafo. Así, podríamos restringir G1 para obtener G2 ó G3:

G2: [PERS: Marta] ← (AGT) ← [BEBER] → (OBJ) → [AGUA]

G3: [NIÑA] ← (AGT) ← [BEBER] → (OBJ) → [AGUA]

En el primer caso, indicamos quién es la persona que bebe agua. En el grafo G3, la restricción consiste en concretar que la persona que bebe agua es una niña. Tanto desde G2 como desde G3 podemos obtener una nueva restricción:

G4: [NIÑA: Marta]←(AGT)←[BEBER]→(OBJ)→[AGUA]

La operación recíproca de la anterior es la **generalización**. Así, G4 puede generalizarse hacia G2 o hacia G3, y cada uno de estos dos puede generalizarse a G1. Puesto que este proceso no introduce ninguna información nueva, si el grafo original representaba una proposición verdadera, también será verdadera la proposición del grafo generalizado. La restricción, en cambio, introduce nueva información que no estaba incluida en el grafo original, y por eso a veces se pierde la veracidad del grafo resultante.

Otras dos operaciones posibles son la **unión** y la **simplificación**, que suelen aplicarse conjuntamente sobre un par de grafos. Si introducimos un nuevo grafo G5,

G5: [NIÑA: Marta]←(AGT)←[BEBER]→(INSTR)→[VASO]

tras aplicar la unión a G4 y G5 y tras simplificar el resultado podemos obtener un nuevo grafo G6:

G6: [BEBER]
(AGT)→[NIÑA: Marta]
(OBJ)→[AGUA]
(INSTR)→[VASO]

Este proceso de unión y simplificación nos recuerda la unificación que se utiliza en lógica matemática con el fin de aplicar la regla de resolución. De hecho, estas operaciones, junto con unas sencillas reglas de cálculo, permiten implementar mediante grafos conceptuales toda la lógica de primer orden e incluso algunas características de la lógica de orden superior. Es lo que podríamos denominar *razonamiento exacto* o *razonamiento deductivo* mediante grafos de Sowa.

Además, este autor trató de extender su método de representación para que fuera capaz de abordar otros problemas de inteligencia artificial que escapan a los planteamientos de la lógica clásica. Una de las extensiones posibles consiste en introducir relaciones conceptuales modales (sec. 5.5.1); hemos visto anteriormente un ejemplo al representar la frase "Probablemente iré mañana".

Otra posible extensión consiste en utilizar *esquemas*. También hemos visto un ejemplo anteriormente, en el que se afirmaba que las aves generalmente vuelan. De este modo los grafos conceptuales tratan de implementar el razonamiento por defecto (sec. 5.5.3) y el razonamiento mediante guiones (sec.

8.3), que se encuentran descritos en otros lugares de este libro. Por ahora, baste señalar que, a diferencia del razonamiento exacto, el razonamiento con incertidumbre (también llamado razonamiento aproximado) mediante grafos de Sowa aún no se encuentra completamente desarrollado; en realidad, su creador se limita a señalar y sugerir algunas posibilidades. No es de extrañar que esto sea así pues, a diferencia de la deducción en lógica clásica, que está bien consolidada desde hace varias décadas, en el razonamiento con incertidumbre existen aún muchas cuestiones sin resolver; en la actualidad se está investigando con intensidad un buen número de métodos, cada uno de ellos con sus cualidades y sus deficiencias.

7.3 REDES DE CLASIFICACIÓN

7.3.1 Extensión e Intensión

Antes de hablar de las redes de clasificación vamos a comentar una idea importante que hasta ahora sólo habíamos comentado de forma general en el primer capítulo. Se trata de la diferencia entre el significado extensional e intensional de un término o de una expresión y la forma más clara de explicarlo es mediantes unos ejemplos.

Hombre. *Significado intensional:* animal racional. *Significado extensional:* conjunto de todos los seres humanos que pueblan la tierra.

Número primo. *S. intensional:* número natural mayor que 1 y que sólo es divisible por sí mismo y por la unidad. *S. extensional:*.{2,3,5,7,11,13, ...}

Planetas solares. *S. intensional:* planetas que giran alrededor del sol. *S. extensional:* {Mercurio, Venus, Tierra, Marte, Júpiter, Saturno, Urano, Neptuno, Plutón}.

Hermanos de Juan Sánchez. *S. intensional:* personas que tienen los mismos padres que Juan Sánchez. *S. extensional:* {Luis Sánchez, Antonio Sánchez, Pedro Sánchez}.

En estos ejemplos se observa que el significado intensional se refiere a la *definición* del concepto, mientras que el extensional se refiere a los elementos a los que se les puede aplicar el concepto *dentro de un universo determinado.* El significado intensional no varía (salvo que nos pongamos de acuerdo en redefinir el concepto). Sin embargo, el alcance extensional puede variar en el tiempo: los seres humanos nacen y mueren, el número de planetas solares podría aumentar o

disminuir por un fenómeno cósmico; incluso podemos imaginar un universo en que Juan Sánchez nunca hubiera tenido hermanos.

El motivo de tratar la diferencia entre extensión e intensión dentro de esta sección dedicada a las redes de clasificación es que en la literatura se ha discutido mucho si estas redes deben representar *conceptos* (intensionales) o *clases* (extensionales). El tratamiento a fondo de esta cuestión es complejo y, por otro lado, en una primera aproximación podemos prescindir de la diferencia entre ambos significados, pues a cada concepto le corresponde una clase, y viceversa; sólo al tratar algunas cuestiones sutiles relativas a la lógica modal sería necesario tener en cuenta esta distinción. Dado el nivel introductorio de este libro nos basta saber que existen dos enfoques diferentes. Sólo conviene añadir que los sistemas más importantes que existen actualmente (los grafos de Sowa y los lenguajes derivados de KL-ONE[2]) optan por el significado intensional frente al extensional.

7.3.2 Jerarquía de Conceptos

Ya el primer sistema desarrollado por Quillian [1968] utilizaba una jerarquía de clasificación, aunque ésta desempeñaba un papel secundario. Las redes de clasificación adquirieron su mayor capacidad cuando Collins y Quillian [1969] introdujeron la **herencia** de propiedades, de la que vamos a hablar más adelante. Desde entonces, gran parte de los modelos de redes han incluido alguna forma de clasificación. En los primeros sistemas que se construyeron, solía existir un tipo de enlace llamado IS-A (en español, "es un" o "es una"). Desgraciadamente, este tipo de enlace era muy ambiguo, pues se utilizaba tanto para indicar la *inclusión* de una subclase dentro de una clase superior ("un elefante *es un* animal") como para indicar la *pertenencia* de un elemento a una clase ("Dumbo *es un* elefante"). Las críticas contra el enlace IS-A fueron numerosas en la literatura y varios autores propusieron diferentes modelos de redes con un fundamento teórico mucho más sólido. Otra de las cuestiones teóricas que se discutieron abundantemente fue si las redes de clasificación han de representar significados intensionales o extesionales, según hemos comentado anteriormente. A pesar de estas diferencias, todas las redes de clasificación

[2] KL-ONE es un lenguaje de representación del conocimiento diseñado por Brachman [1979] como implementación de algunas ideas que trataban de dar un fundamento teórico más sólido a las redes semánticas (ver la sec. 7.3.4).

coinciden en sus aspectos esenciales, que son los que vamos a mostrar a continuación.

Como ejemplo de red de clasificación, observemos la Fig. 7.9, que muestra una pequeña porción de la que utilizan los grafos de Sowa. En realidad, este esquema podría refinarse mucho más. Por ejemplo, entre el nodo animal y el nodo perro podría haber una cadena que contuviera conceptos como vertebrado, animal-de-sangre-caliente, mamífero, etc.

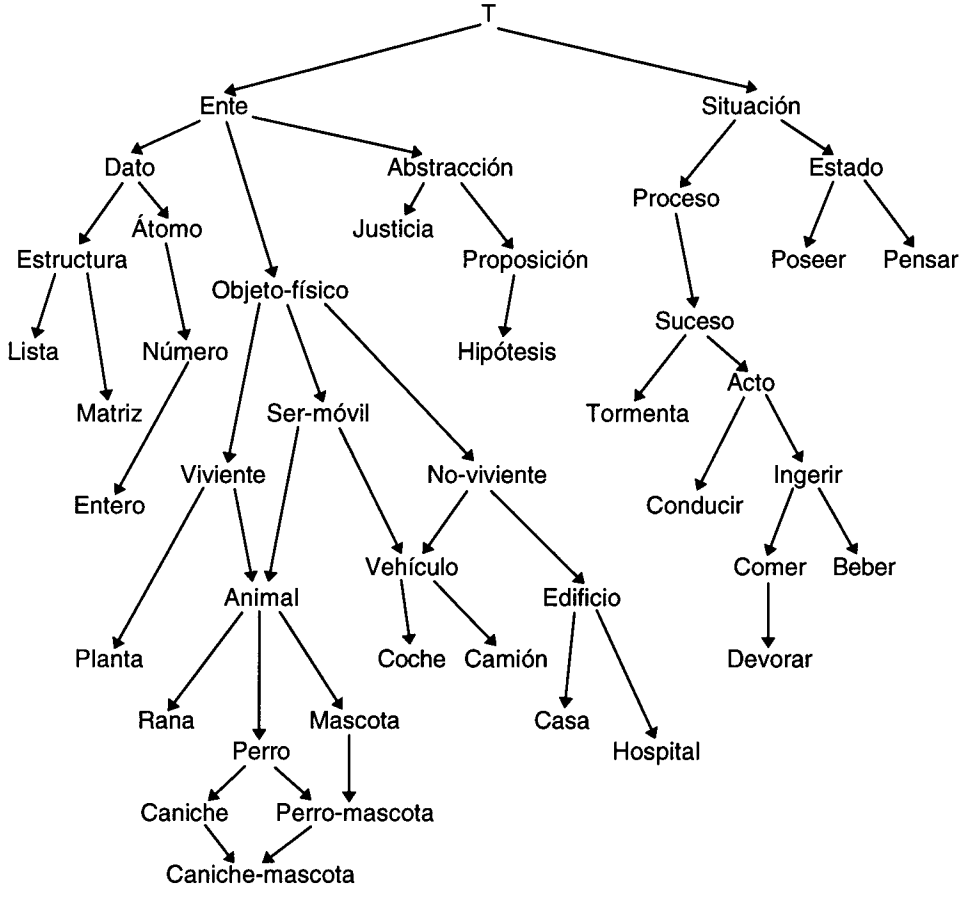


Fig. 7.9. Jerarquía de clasificación

En este esquema, el concepto superior, que engloba todos los demás, se representa por "T". Los arcos indican una relación de orden parcial, de modo que un arco trazado desde un nodo A a un nodo B indica la relación $B<A$; esta

relación se interpreta diciendo que el concepto *A* es más general que *B*. Al pasar del significado intensional al extensional, esto implica que la clase asociada a *B* está *incluida* en la clase asociada a *A*.

Podemos entender mejor ahora la relación entre los grafos dirigidos acíclicos y las relaciones de orden parcial (sec. 3.2.1). La Fig. 7.9 consiste en un GDA en que hay bucles pero no ciclos (si hubiera ciclos se violaría la propiedad transitiva típica de las relaciones de orden).

Se observa también en esta figura que la red de clasificación no sólo incluye objetos físicos, sino que cada uno de los nodos que encontramos al estudiar los grafos de Sowa pertenece a un concepto de esta red, incluidos los nodos que representan proposiciones. También los verbos están clasificados en esta jerarquía. Así, podemos encontrar la cadena devorar < comer < ingerir, la cual indica que devorar es una forma específica de comer, y comer es una forma específica de ingerir.

7.3.3 Mecanismos de Herencia

El interés principal de agrupar los conceptos en una red jerárquica consiste en poder realizar un tipo particular de inferencia, que consiste en que un concepto herede las propiedades de sus antepasados. Por ejemplo, si se dice que el concepto de viviente implica respirar, de ahí se puede inferir que el concepto de caniche posee la misma propiedad; y dado un caniche particular concluiremos que respira. En realidad, la inferencia mediante herencia de propiedades consiste en aplicar una cadena de silogismos extraídos de la lógica clásica: "Si *X* es un caniche, los caniches son perros, los perros son animales, los animales son vivientes y los vivientes respiran, entonces *X* respira". (La primera premisa indica pertenencia, las tres siguientes son relaciones de inclusión y la quinta afirma una propiedad de un concepto o clase.)

En realidad, existen dos tipos de herencia en redes jerárquicas: estricta y por defecto. La **herencia estricta** consiste en que todos los conceptos descendientes de *A* poseen necesariamente las mismas propiedades de *A*. La **herencia por defecto**, en cambio, *supone* que los descendientes de *A* poseen las mismas propiedades de *A* mientras no se indique lo contrario. Por ejemplo, podemos afirmar que las aves vuelan. Sin embargo, los pingüinos son una excepción dentro de las aves. Al construir la red de clasificación asociaremos la propiedad "vuela" al nodo ave y la propiedad "no vuela" al nodo pingüino, que es descendiente del anterior. En este caso, las propiedades de los nodos

inferiores dentro de la jerarquía tienen preferencia sobre las propiedades de sus antepasados.

Como síntesis, podemos decir que *la herencia estricta se ciñe a las leyes de la lógica clásica*, mientras que *la herencia por defecto se sitúa dentro del razonamiento no monótono*, porque permite que la llegada de nueva información invalide un resultado inferido anteriormente.

Otro problema relacionado con la herencia por defecto surge al considerar redes de clasificación con bucles. En efecto, si la clasificación pudiera representarse en forma de árbol, cada concepto tendría solamente un padre del que heredar. Sin embargo, en un GDA puede haber varios padres para un mismo nodo y esto puede dar lugar a contradicciones entre los diferentes valores por defecto heredados. De ahí surge la necesidad de establecer mecanismos para resolver estos conflictos, lo cual vuelve a plantear un problema de razonamiento no monótono. El tema es complejo y no vamos a entrar en él; el lector puede encontrar un tratamiento detallado de este problema en [Touretzky, 1986, 1992].

7.3.4 Sistemas Taxonómicos / Sistemas Asertivos

El conocimiento contenido en una red semántica puede ser de dos clases, *asertivo* y *taxonómico*, y esto da lugar a dos grupos de redes, en función del tipo de conocimiento al que dan más importancia.

El *conocimiento asertivo* consiste en realizar afirmaciones particulares, tales como "Luis compró un libro" o "Hay un hombre a quien todos admiran". Los grafos de Schank constituyen el ejemplo más claro de *redes asertivas*: en ellas no existen definiciones de conceptos ni clasificaciones jerárquicas, sino solamente afirmaciones concretas.

El *conocimiento taxonómico*, del que ya hemos hablado en esta sección, describe los conceptos. La red de clasificación que aparece en la Fig. 7.9, completada con la descripción de propiedades para cada concepto y con los correspondientes mecanismos de herencia, puede ser un ejemplo de *red taxonómica*. El sistema KL-ONE [Brachman, 1979] está diseñado especialmente para clasificar los conceptos y manejar sus propiedades. Los sistemas basados en marcos que existen en la actualidad (sec. 8.1.4) también incluyen redes de clasificación, de modo que en muchos casos, resulta difícil trazar la frontera entre ambos campos, y la inclusión en uno u otro grupo depende esencialmente de la perspectiva histórica de quien lo estudia.

En cualquier caso, lo que ha quedado claro en la evolución de las redes semánticas es la distinción entre ambos tipos de conocimiento, hasta el punto de que algunos modelos incluyen por un lado una red de clasificación, que en general es estática (salvo que el usuario introduzca nuevos conceptos), y por otro lado un sistema para el tratamiento de proposiciones. En las redes de Sowa existe esta distinción, que es aún más clara en el sistema KRYPTON [Brachman *et al.*, 1983].

En efecto, KRYPTON consta de dos componentes bien diferenciados. El primero de ellos, denominado *T-box*, contiene el conocimiento taxonómico en forma de red de clasificación de conceptos (es la misma red que Brachman había construido para KL-ONE). El segundo componente, denominado *A-box*, contiene el conocimiento asertivo, expresado mediante predicados, que coinciden con los conceptos y los roles del *T-box*; la inferencia se realiza mediante un cierto demostrador automático de teoremas, semejante a los mencionados en la sec. 5.5.

7.4 REDES CAUSALES

La característica esencial de una red causal es que representa un modelo en que los nodos corresponden a variables (edad, estenosis mitral, hipertensión arterial, fiebre...) y los enlaces a relaciones de **influencia** (un enlace $X \rightarrow Y$ indica que el valor que toma X *influye* en el valor de Y); esta influencia suele ser una relación de **causalidad** (*X produce Y*). A diferencia de los modelos anteriores, que estaban orientados sobre todo a la comprensión y representación del lenguaje natural, los modelos causales se orientan sobre todo a problemas de diagnóstico. Muchos de ellos están destinados a la medicina, aunque también hay otros que se ocupan del diagnóstico de averías en aparatos mecánicos, eléctricos y electrónicos.

Vamos a estudiar en primer lugar CASNET, el primer sistema experto basado en una red causal y luego nos detendremos en el estudio de las redes bayesianas.

7.4.1 El Sistema Experto CASNET

El sistema experto CASNET (el nombre procede de *Causal Associational NETwork*) fue desarrollado en la Universidad de Rutgers en los años 70 [Weiss *et al.*, 1978; Kulikowski & Weiss, 1982]. Su objetivo era ayudar a los médicos

en el diagnóstico y el tratamiento del glaucoma, una enfermedad ocular. La elección de este campo se debe a que el ojo humano es un sistema de tamaño reducido y relativamente aislado (tiene pocas interacciones con los demás órganos del cuerpo) y a que en oftalmología hay pocos diagnósticos, pocas pruebas clínicas y pocos tratamientos posibles.[3]

A pesar de ser uno de los primeros sistemas expertos, CASNET posee características avanzadas de las que carecen muchos sistemas desarrollados posteriormente, incluso hoy en día. Los programas de diagnóstico por ordenador anteriores estaban basados principalmente en métodos estadísticos, en árboles de decisión y en algoritmos de reconocimiento de patrones. Estos programas tenían serias deficiencias en cuanto al *razonamiento temporal* (eran incapaces de seguir la evolución de la enfermedad), en cuanto al *diagnóstico múltiple* (la mayor parte de ellos suponía que existía un único diagnóstico que excluía los demás) y en cuanto a la *explicación* de sus conclusiones (no podían justificar cómo y por qué habían obtenido los resultados). Por eso se hizo necesario utilizar nuevas técnicas de representación y de inferencia. Kulikowski y Weiss [1982, p. 29] afirman:

> Al desarrollar el formalismo de CASNET reunimos ideas procedentes de dos campos de las ciencias de la computación: el reconocimiento estadístico de patrones y la inteligencia artificial (IA). Del reconocimiento de patrones extrajimos la noción de redes de inferencia y la de ordenación probabilista de hipótesis. La IA nos sugirió la utilización de una estructura conceptual [una red de conceptos] para representar los procesos de la enfermedad.

Vamos a describir en primer lugar los elementos de CASNET que intervienen en el diagnóstico; después hablaremos de los que intervienen en las recomendaciones terapéuticas. Uno de los rasgos más característicos de CASNET es la estructuración de los nodos en tres niveles:

Observaciones: Incluye los síntomas (lo que el enfermo siente), los signos (lo que el médico observa) y los resultados de las pruebas de laboratorio.

Estados patofisiológicos: Son las alteraciones que se producen en el funcionamiento normal de un órgano; en este caso, el ojo.

[3] Estamos mencionando aquí de pasada uno de los problemas claves de la inteligencia artificial aplicada. En realidad debería buscarse el método en función del problema. Sin embargo, cuando se está tratando de poner a prueba un nuevo método, suele escogerse el problema en función de aquél. En este caso hay que buscar un problema suficientemente sencillo para que sea tratable y suficientemente complejo para demostrar que el método es aplicable también a otros problemas.

Estados de enfermedad: Dentro de este nivel, las enfermedades se encuentran clasificadas en un árbol taxonómico; los nodos inferiores corresponden a especificaciones de los nodos superiores.

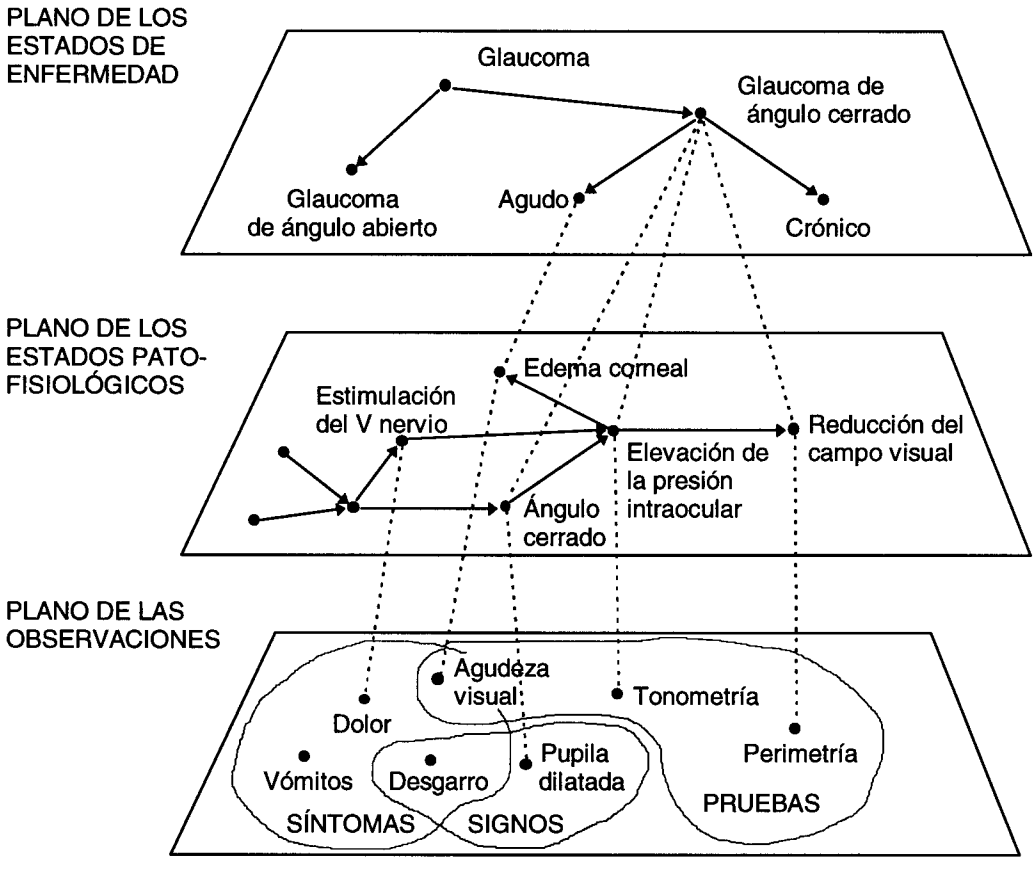


Fig. 7.10. Red de CASNET para el diagnóstico del glaucoma.

En la red de CASNET existen varios **tipos de enlaces**, tal como muestra la Fig. 7.10. *(1)* Dentro del plano de las enfermedades, los enlaces indican relaciones de inclusión. *(2)* Dentro del plano de los estados patofisiológicos, los enlaces corresponden a las relaciones de causalidad que determinan la evolución de la enfermedad; por ejemplo, la elevación de la presión ocular produce edema corneal y reducción del campo visual. *(3)* Los enlaces entre el plano superior y el

mediano indican los estados patofisiológicos producidos por una determinada enfermedad. *(4)* Por último, los enlaces entre el plano mediano y el inferior indican las observaciones a que da lugar cada estado patofisiológico. Pueden verse ejemplos de estos tipos de enlaces en la. Cada enlace entre un nodo s_i y otro s_j lleva un factor asociado, a_{ij}, para indicar el "grado de conexión causal" entre ambos estados.

Los **diagnósticos** pueden consistir en estados patofisiológicos (hipótesis simples) o en enfermedades (que son hipótesis complejas, pues cada enfermedad viene dada por un conjunto de estados patofisiológicos). El proceso consiste en interpretar los hallazgos en función de los estados patofisiológicos que han podido provocarlos. Las a_{ij} asociadas a estos enlaces permiten asignar a cada estado-hipótesis un valor entre tres posibles: confirmado, descartado o indeterminado. Cada hipótesis puede llevar modificadores para indicar su intensidad, duración, evolución... y su grado de confirmación (seguro, casi seguro, probable, etc.).

Hemos descrito hasta ahora los elementos de CASNET que intervienen en el diagnóstico. Sin embargo, este sistema experto es capaz de ofrecer también **recomendaciones terapéuticas**. Para ello cuenta con un cuarto grupo de nodos, uno por cada tratamiento, que también pueden dibujarse dentro de un plano, aunque éste no sería ya paralelo a los anteriores. Existen dos tipos de enlaces, no mencionados hasta ahora. Uno de ellos se destina a unir los estados patofisiológicos y las enfermedades con los tratamientos oportunos, o para unir un tratamiento con las complicaciones a que puede dar lugar. El otro tipo de enlace se utiliza para unir los tratamientos entre sí con el fin de representar las interacciones, la toxicidad y las dependencias temporales (es decir, el orden en que deben aplicarse los tratamientos). La elección de una terapia concreta depende del diagnóstico, de la historia pasada del enfermo y de los resultados predecibles para cada tratamiento.

7.4.2 Redes Bayesianas

7.4.2.1 Antecedentes

En los problemas de diagnóstico nos interesa hallar cuál es la hipótesis más probable de acuerdo con la evidencia disponible, la cual viene dada por un conjunto de hallazgos. Sin embargo, en las aplicaciones prácticas las probabilidades directas (es decir, la probabilidad a priori de las hipótesis y las probabilidades de que las alteraciones den lugar a ciertos hallazgos) suelen ser

más fáciles de obtener que las inversas (las probabilidades de las hipótesis dados los hallazgos). El teorema de Bayes nos permite pasar de unas a otras mediante la siguiente expresión, en la cual las d_i representan los diagnósticos y las e_j los hallazgos:

$$P(d_1,\ldots,d_n \mid e_1,\ldots,e_m) = \frac{P(e_1,\ldots,e_m \mid d_1,\ldots,d_n)\cdot P(d_1,\ldots,d_n)}{\sum_{d'_1,\ldots,d'_n} P(e_1,\ldots,e_m \mid d'_1,\ldots,d'_n)\cdot P(d'_1,\ldots,d'_n)}$$

Sin embargo, esta expresión es imposible de aplicar por la enorme cantidad de información que requiere. Por eso se introduce la hipótesis de que los diagnósticos son *exclusivos* (dos de ellos no pueden ser ciertos a la vez) y *exhaustivos* (no hay otro diagnóstico posible fuera de ellos). Hay por tanto una variable D, que puede tomar n valores (los n diagnósticos d_i posibles) y m variables —binarias en general— correspondientes a los posibles hallazgos, tal como muestra la Fig. 7.11.

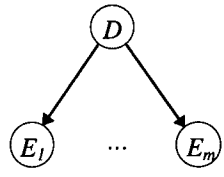


Fig. 7.11. Método clásico de diagnóstico probabilista.

A pesar de estas simplificaciones, el modelo sigue siendo inaplicable, incluso para un número limitado de diagnósticos y de hallazgos, por la necesidad de conocer todas las probabilidades que intervienen. Por ello se introduce una nueva hipótesis, la *independencia condicional*, que consiste en afirmar que, para cada diagnóstico, la probabilidad de encontrar un hallazgo es independiente de que se hayan encontrado otros. Matemáticamente se expresa así:

$$P(e_1,\ldots,e_m \mid d_i) = P(e_1 \mid d_i)\cdot \ldots \cdot P(e_m \mid d_i), \quad \forall d_i$$

y en consecuencia, la probabilidad de un diagnóstico viene dada por

$$P(d_i \mid e_1,\ldots,e_m) = \frac{P(e_1 \mid d_i)\cdot \ldots \cdot P(e_m \mid d_i)\cdot P(d_i)}{\sum_j P(e_1 \mid d_j)\cdot \ldots \cdot P(e_m \mid d_j)\cdot P(d_j)}$$

Así el problema resulta tratable, y con este método se construyeron algunos de los primeros sistemas de diagnóstico por ordenador en los años 60. Sin embargo, las dos hipótesis introducidas, la de diagnósticos exclusivos-exhaustivos y la de independencia condicional, en general se ajustan a la realidad. Por eso, durante largos años surgieron fuertes críticas contra el empleo de técnicas bayesianas en inteligencia artificial y los investigadores prefirieron desarrollar otras técnicas, como los factores de certeza del sistema MYCIN y la lógica difusa (sec. 6.6).

La situación cambió notablemente con la aparición de las redes bayesianas (RB), que mantenían las ventajas del método probabilista clásico, principalmente la de estar basadas en una teoría matemática bien establecida, y a la vez permitían un cálculo de eficiencia razonable sin tener que introducir hipótesis extrañas. En efecto, en las redes bayesianas no hace falta suponer que los diagnósticos son exclusivos y exhaustivos (las RB pueden diagnosticar de forma natural varias alteraciones presentes simultáneamente) y, en cuanto a la hipótesis de independencia condicional, no se aplica de forma global al conjunto de diagnósticos y hallazgos, sino sólo para los *efectos inmediatos* de una alteración. De esta forma las RB consiguen cierto grado de modularidad, la cual permite computar la probabilidad de forma local, como veremos más adelante.

7.4.2.2 El Sistema Experto DIAVAL

DIAVAL, el primer sistema experto bayesiano español, ha sido desarrollado en la UNED por F.J. Díez, con el fin de ayudar a los médicos en el diagnóstico de enfermedades de corazón, especialmente de valvulopatías. Para ello, el programa considera los datos personales del enfermo, sus antecedentes, los síntomas y signos y los hallazgos de distintas pruebas clínicas, sobre todo de la ecocardiografía.

La Fig. 7.12 muestra una porción de la red bayesiana de DIAVAL. Ya hemos mencionado anteriormente que en todas las redes causales los enlaces indican relaciones de *influencia* o de *causalidad* entre los nodos. La red, que es esencialmente un modelo patofisiológico de las enfermedades cardíacas, consta en total de unos 300 nodos. Aquí aparecen solamente algunos de los que están más directamente relacionados con la insuficiencia mitral; los puntos suspensivos indican que la red se extiende hacia otras causas y otros efectos.

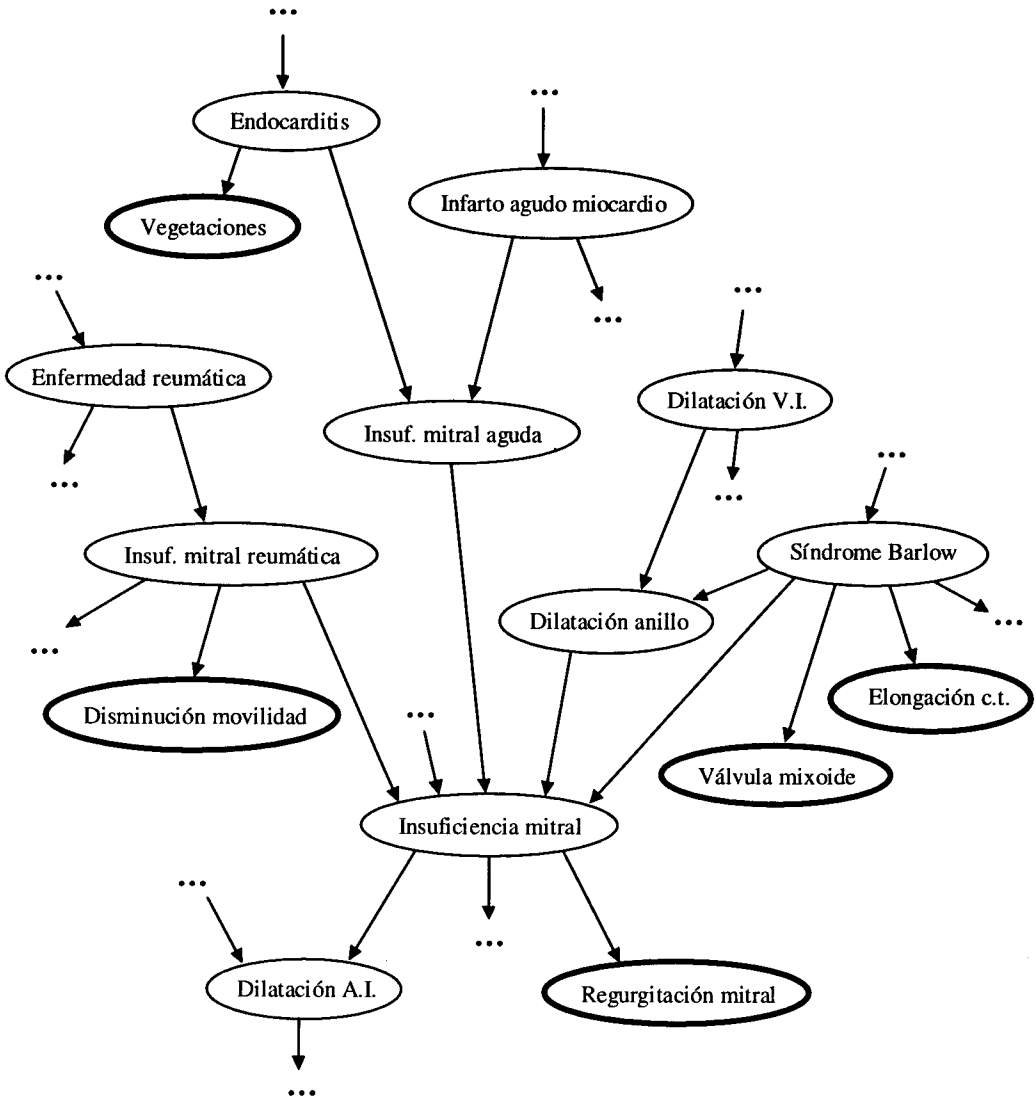


Fig. 7.12. Porción de la red bayesiana del sistema experto DIAVAL.

En realidad, esta figura representa solamente los nombres de los nodos y la estructura de la red. Para completarla habría que indicar los valores que toma cada variable y la información numérica, es decir, las probabilidades a priori y condicionales de la red.

En DIAVAL, los nodos que no tienen descendientes corresponden a las observaciones. En la figura estos nodos terminales están representados por óvalos de trazo más grueso; concretamente, los cinco hallazgos que muestra este ejemplo corresponden a observaciones del ecocardiograma: vegetaciones, elongación de las cuerdas tendíneas, etc. La inferencia consiste en asignar los valores de las variables que se conocen y, a partir de esta información, realizar la propagación de la evidencia mediante el paso de mensajes entre nodos, con el fin de hallar la probabilidad a posteriori de los demás nodos; en particular, puede interesarnos conocer cuál es el valor más probable de la variable "insuficiencia mitral" (si es "ausente", "leve" "moderada" o "severa") y la causa que ha podido producirla, pues de estos dos datos depende la necesidad de realizar un tratamiento y cuál de ellos es el más adecuado: prótesis, cirugía reconstructiva, etc. En la próxima sección explicaremos mediante un ejemplo sencillo cómo se realiza esta propagación de evidencia, y a la vez presentaremos las propiedades fundamentales de las redes bayesianas.

7.4.2.3 Definición de Red Bayesiana

Vamos a imaginar una red bayesiana muy simple para el diagnóstico del paludismo. La variable A puede tomar dos valores, $+a$, el cual significa "el paciente tiene paludismo" y $\neg a$, que significa "el paciente no tiene paludismo".

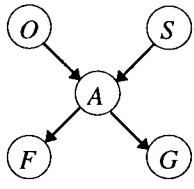


Fig. 7.13. Red bayesiana para el paludismo.

Los dos factores que influyen en la probabilidad a priori de que un enfermo tenga paludismo son el país de origen (O) y el grupo sanguíneo (S). A su vez el paludismo puede manifestarse en forma de fiebre (F) o puede detectarse mediante la prueba de la gota gruesa (G). La variable O puede tomar tres valores, o_1, o_2 y o_3, correspondientes a las zonas de alto, medio y bajo riesgo, respectivamente; las demás variables son binarias.

Una vez que tenemos la estructura de la red, hay que introducir los valores numéricos. Concretando más, para cada nodo debemos indicar la probabilidad

condicionada a los valores de sus padres; para los nodos sin padres daremos simplemente su probabilidad a priori, es decir, sin condicionamiento. En nuestro ejemplo serán $P(o)$, $P(s)$, $P(a|o,s)$, $P(f|a)$ y $P(g|a)$.

Las redes bayesianas poseen por definición una propiedad matemática conocida como *separación direccional*, la cual significa que la probabilidad de una variable X, una vez determinados los valores de los padres de X, es independiente de los demás nodos-variables que no son descendientes de X. Por ejemplo, para el nodo F, la independencia condicional puede expresarse como

$$P(f,o,s,g|a) = P(f|a)$$

o bien como

$$P(f|a,o,s,g) = P(f|a)$$

Aplicando repetidamente la separación direccional podemos *factorizar* la probabilidad conjunta de la red, con el fin de expresarla en función de las probabilidades condicionales que definen la red:

$$P(o,s,a,f,g) = P(o)\cdot P(s)\cdot P(a|o,s)\cdot (f|a)\cdot (g|a)$$

Como síntesis de lo anterior, tenemos la siguiente definición:

Red bayesiana: Es un grafo dirigido acíclico conexo más una distribución de probabilidad sobre sus variables, la cual cumple la propiedad de separación direccional.

Hemos definido la separación condicional desde un punto de vista formal. Ahora bien, el lector se preguntará: ¿por qué se introduce esta propiedad?, ¿qué conexión tiene con el mundo real? La respuesta a esta cuestión, que en el fondo consiste en explicar la *semántica* de las redes bayesianas, se halla en la relación entre la noción intuitiva de *causalidad* y las propiedades matemáticas de la *independencia* probabilísitica.

Lo entenderemos mejor volviendo al ejemplo anterior (Fig. 7.13). Supongamos que un enfermo tiene paludismo. El modelo que hemos planteado nos dice que la probabilidad de que este enfermo presente fiebre depende solamente del hecho de que tiene paludismo, y no influyen su país de origen, ni su grupo sanguíneo, ni el resultado que se haya obtenido en las pruebas de laboratorio. Del mismo modo, cuando el enfermo *no* tiene paludismo la probabilidad de que presente fiebre —según este modelo— es independiente de los tres factores

mencionados anteriormente. Esto es, ni más ni menos, lo que está expresando la ecuación de la independencia condicional para F.

Por tanto, en las redes bayesianas la presencia de un enlace indica influencia causal, y la *ausencia* de un enlace indicar implícitamente que no existen otras interacciones. Las relaciones de dependencia e independencia causales se corresponden exactamente con las relaciones de dependencia e independencia probabilistas.

Ahora bien, supongamos que un médico experto nos dice que el país de origen sí influye en la probabilidad de que el enfermo padezca fiebre, pues hay otras enfermedades distintas del paludismo que también producen este mismo síntoma. En este caso, deberíamos añadir nuevos nodos al modelo con el fin de representar esas enfermedades, y trazar luego los enlaces correspondientes, con sus respectivas probabilidades condicionales. Este proceso, en el que se añade más información al modelo, es una de las formas de *refinamiento* del conocimiento que posee el sistema.

7.4.2.4 Inferencia en Redes Bayesianas

Supongamos ahora que tenemos un modelo causal expresado mediante una red bayesiana; en el caso de un sistema experto real, la red puede tener cientos de nodos, algunos de los cuales corresponden necesariamente a las observaciones disponibles. La inferencia consiste en fijar el valor de las variables conocidas con certeza y realizar un cálculo con el fin de hallar la probabilidad de las variables cuyo valor no se conoce con certeza.

En el ejemplo anterior, la evidencia disponible podría ser $\mathbf{e} = \{O = o_2, F = +f, G = \neg g\}$, lo cual significa que el paciente procede de una zona de riesgo medio (o_2) y presenta fiebre ($+f$) pero la prueba de la gota gruesa ha dado un resultado negativo ($\neg g$). Nuestro objetivo es determinar si el enfermo padece paludismo y con qué certeza podemos dar el diagnóstico; es decir, buscamos la probabilidad a posteriori de A, expresada como $P(a|\mathbf{e})$.

En este ejemplo tan reducido podríamos realizar un cálculo muy sencillo. Sin embargo, vamos a utilizar un método que, aunque parezca a primera vista más aparatoso, tiene la ventaja de que resulta mucho más eficiente y elegante en redes de mayor tamaño. El algoritmo que vamos a describir es aplicable solamente en *poliárboles* (sec. 3.2.1), aunque existen diferentes formas de adaptarlo al caso de redes con bucles.

En primer lugar, si el nodo que nos interesa es A, dividimos la evidencia en dos subconjuntos; $\mathbf{e}_A^+$ contiene la evidencia que está "por encima" de A en el poliárbol, y $\mathbf{e}_A^-$ la que está "por debajo". En el ejemplo anterior, $\mathbf{e}_A^+ = \{o_2\}$ y $\mathbf{e}_A^- = \{+f, \neg g\}$. Aplicando la definición de probabilidad condicional y la separación direccional, tenemos

$$P(a|\mathbf{e}) = \frac{P(a,\mathbf{e})}{P(\mathbf{e})} = \frac{P(a,\mathbf{e}_A^+,\mathbf{e}_A^-)}{P(\mathbf{e})} = \frac{P(a,\mathbf{e}_A^+)\cdot P(\mathbf{e}_A^-|a)}{P(\mathbf{e})} = \alpha\cdot\pi(a)\cdot\lambda(a)$$

donde hemos introducido tres definiciones:

$$\pi(a) \equiv P(a,\mathbf{e}_A^+)$$
$$\lambda(a) \equiv P(a|\mathbf{e}_A^-)$$
$$\alpha \equiv P(\mathbf{e})$$

Su significado es el siguiente. $\pi(a)$ indica la probabilidad de A dadas sus causas, o dicho de otro modo, cuál de los valores a es más probable teniendo en cuenta toda la información referente a las causas de A. De modo semejante, $\lambda(a)$ indica la probabilidad de $\mathbf{e}_A^-$ para cada valor a, es decir, cuál de los valores de A explica mejor la información que se ha encontrado referente a sus efectos.

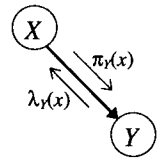


Fig. 7.14. Paso de mensajes para la propagación de la evidencia.

Los valores de π y λ correspondientes a cada nodo se pueden calcular mediante el paso de mensajes entre nodos vecinos, tal como expresa la Fig. 7.14; el valor de α se determinará posteriormente mediante normalización, al imponer la condición de que la suma de las probabilidades sea la unidad. Dado que éste es un libro de texto de carácter general, no vamos a entrar en la definición de las expresiones que intervienen. Baste decir que el cálculo de la probabilidad puede realizarse de forma distribuida, es decir, teniendo una implementación en que

cada nodo sea en un procesador físico; los enlaces han de ser conexiones, a través de las cuales se transmitirán los mensajes.

7.4.3 Ventajas y Limitaciones de las Redes Causales

Para concluir esta sección vamos a resumir algunas de las ventajas y limitaciones de las redes causales en general y de las redes bayesianas en particular.

La principal **ventaja** de razonar en base a un modelo del mundo real consiste en que el sistema posee un conocimiento profundo de los procesos que intervienen, en vez de limitarse a una mera asociación de datos e hipótesis. Como consecuencia de ello, los programas resultantes son capaces de *explicar* la cadena causal de anomalías que va desde la enfermedad diagnosticada hasta los efectos observados.[4]

Otra **ventaja** de los modelos causales es que pueden realizar tres tipos de razonamiento:

Razonamiento abductivo: es el más típico de los problemas de diagnóstico. Consiste en buscar cuál es la causa que mejor explica los efectos observados. Diríamos que se trata de un razonamiento "hacia arriba".

Razonamiento deductivo: es el recíproco del anterior, pues va desde las causas hacia los efectos, es decir, "hacia abajo". Es el tipo de razonamiento que se aplica, por ejemplo, en el caso de enfermedades hereditarias, y por este motivo a los hijos de quienes padecen algunas de estas enfermedades se les recomienda que se sometan de estudios periódicos, con el fin de prevenir su aparición. En caso de que los efectos previstos se sitúen en el futuro, hablaremos de razonamiento predictivo, que es el utilizado en medicina al elaborar el pronóstico de un enfermo.

Razonamiento intercausal: es un razonamiento "en horizontal". Cuando un síntoma, una alteración o una enfermedad tiene un número limitado de causas, la confirmación de una de ellas reduce la sospecha de otras e, inversamente, la evidencia disponible puede llevar a descartar todas las hipótesis menos una, que pasaría a ser el diagnóstico más verosímil.

En la práctica clínica, los tres tipos de razonamiento se realizan simultáneamente. Por ejemplo, si a un hospital llega un paciente con dolor

[4] En el capítulo 9 veremos cómo la capacidad de explicación es uno de los rasgos más característicos de los sistemas expertos.

torácico, el médico piensa que quizá se deba a un infarto de miocardio (razonamiento abductivo). Si se trata de un paciente de 60 años, obeso, fumador e hipertenso, la sospecha será más fuerte que si el paciente tuviera 18 años (razonamiento deductivo). Si el electrocardiograma confirma este resultado, probablemente el médico no se detendrá a examinar otras hipótesis; en cambio, si en el ECG no se observan los signos esperados, el médico empezará a considerar otros diagnósticos, como pudieran ser una hernia de hiato o una pericarditis (razonamiento intercausal).

La distinción entre estos tres tipos de razonamiento permite que los modelos causales tengan presente la correlación que existe entre los hallazgos. Ya dijimos al hablar de los primeros programas de diagnóstico bayesiano que su problema principal es que normalmente no se cumple la hipótesis de independencia condicional tal como se planteaba en estos sistemas. Tampoco los sistemas basados en reglas son capaces de distinguir los tres tipos de razonamiento, y por eso pueden llegar a conclusiones erróneas [Díez, 1994, sec. 2.4].

En general, el diagnóstico —independientemente de que lo haga un especialista o un sistema experto— no se realiza en una sola fase, sino que la información disponible lleva a establecer una hipótesis de trabajo, la cual orienta la búsqueda de nueva información con el fin de confirmar o rechazar tal hipótesis; en medicina este proceso se observa con toda claridad en la forma en que los doctores seleccionan las preguntas y solicitan estudios complementarios.

El principal **inconveniente** de las redes causales es la limitación en su rango de aplicaciones. En efecto, aunque su rendimiento es excelente en problemas de diagnóstico, para otros tipos de problemas como la planificación, el control o el diseño, se hace necesario recurrir a mecanismos de representación y de inferencia, tales como puedan ser las reglas (cap. 6). Por otra parte, existen problemas de diagnóstico en que las redes causales no son aplicables debido a que no se conocen aún los mecanismos que intervienen y, en consecuencia, es imposible desarrollar modelos causales.

7.4.3.1 Ventajas e Inconvenientes de las Redes Bayesianas

Las redes bayesianas, además de las ventajas comunes a otros métodos de razonamiento causal, poseen una sólida teoría probabilista que les permite dar una interpretación objetiva de los factores numéricos que intervienen y dicta de

forma unívoca la forma de realizar la inferencia.[5] De este modo combinan las ventajas de los métodos probabilistas y de los modelos causales.

Sin embargo, tienen dos inconvenientes principales, además de la limitación en cuanto al rango de aplicaciones mencionada anteriormente. Por un lado, las RB necesitan gran cantidad de probabilidades numéricas. Normalmente no se dispone de toda esta información mediante estudios objetivos, y por ello se hace necesario recurrir a estimaciones subjetivas de expertos humanos.

Por otro lado, la presencia de bucles en las redes bayesianas complica extraordinariamente los cálculos. Cuando la red contiene numerosos nodos y bucles, los métodos de simulación estocástica suelen resultar más eficientes que los métodos exactos, aunque puede resultar costoso —en términos de tiempos de computación— lograr el grado de aproximación deseado. Otra solución consiste en buscar simplificaciones sobre el modelo original con el fin de poder realizar un cálculo exacto. Son métodos que están aún en vías de desarrollo y de año en año aparecen nuevos resultados. Las redes bayesianas constituyen una metodología muy joven (surgieron a principios de los 80 y su expansión comenzó en 1988) y aún quedan muchos puntos por resolver.

7.5 COMENTARIOS FINALES

Después de haber presentado en este capítulo los modelos más importantes de redes asociativas, vamos a hacer algunas reflexiones sobre la *terminología* que suele emplearse en este campo.

Los primeros sistemas desarrollados en IA que utilizaban la representación en forma de red [Masterman, 1961; Quillian, 1968] estaban destinados al tratamiento del lenguaje natural. Quillian llamó "red semántica"[6] a su sistema porque su primer objetivo era representar el *significado* de los vocablos del idioma inglés. Sin embargo, el término se aplicó a otros sistemas, cada vez más diferentes de los primeros. Dice Brachman [1979, p. 4]:

> Mientras que el propósito original de Quillian fue representar la semántica de las palabras inglesas en sus redes, otras representaciones de apariencia muy similar

[5] El sistema experto CASNET utilizaba un mecanismo de propagación de evidencia basado en la combinación de factores numéricos, pero sin darles una interpretación probabilística.

[6] Generalmente se considera a Quillian como creador de las redes semánticas. Sin embargo, Sowa [1992] afirma que fue Margaret Masterman [1961] la primera en construir una *red semántica* (incluso le aplicó este nombre); Ross Quillian conoció este trabajo en el mismo año 1961.

se utilizaron pronto para modelar toda clase de elementos no semánticos (por ejemplo, proposiciones, estructura de objetos físicos, acoplamiento de puertas electrónicas). Más aún, prácticamente todo formalismo con forma de red que ha aparecido en la literatura desde 1966 ha recibido tarde o temprano la etiqueta de red semántica.

Con el paso del tiempo, el concepto ha ido evolucionando. Así, el programa SCHOLAR (sec. 7.1.2), que fue desarrollado como una aplicación de las redes semánticas, ya no suele considerarse hoy en día como tal. En cambio, los grafos de dependencia conceptual estudiados anteriormente sí se suelen considerar dentro de las redes semánticas, a pesar de que su propio autor [Schank, 1975], rechazó que debieran recibir este nombre.

En cuanto a las redes de clasificación, si se utilizan para representar conceptos mediante una jerarquía de clases y propiedades, no cabe duda de que pueden considerarse redes semánticas. Sin embargo, cuando un sistema experto basado en marcos define clases e instancias, tal como mostraremos en el capítulo 8, es dudoso hasta qué punto puede hablarse de red semántica.

Por eso hemos escogido el título "*Redes asociativas*" para este capítulo, con el fin de incluir también las redes causales y todas las redes de clasificación, y reservamos el nombre de *redes semánticas* para aquéllas que se dedican a la representación del lenguaje natural, siguiendo así la denominación que es más habitual en la literatura reciente sobre este tema. En cualquier caso, recuerde el lector que las clasificaciones son muchas veces subjetivas y, en el fondo, importa más conocer el contenido que ponerle una etiqueta.

7.6 BIBLIOGRAFÍA RECOMENDADA

En el libro "*Associative Networks: Representation and Use of Knowlegde by Computers*", editado por Findler [1979], aparecieron los artículos de Brachman [1979], Hendrix [1979], Shapiro [1979], Schank y Carbonell [1979], y otros trabajos relevantes que no hemos citado.

Entre a los libros que más importantes destacamos el de Schank [1975], "*Conceptual Information Processing*", y el de Sowa [1984], "*Conceptual Structures: Information Processing in Mind and Machine*". Algunos trabajos recientes sobre este campo se encuentran recogidos en [Sowa, 1991].

Muchos de los artículos citados pueden ser difíciles de encontrar en sus fuentes originales. Afortunadamente, la obra de Brachman y Levesque [1985],

Readings in Knowledge Representation, reproduce algunos de los más importantes relativos a la representación del conocimiento; allí pueden encontrarse las siguientes referencias citadas en este capítulo: Quillian [1968], Schank y Rieger [1974], Brachman [1979], Brachman, Fikes y Levesque [1983]. También contiene algunos artículos sobre herencia y razonamiento por defecto, y sobre la relación entre lógica y redes semánticas.

En cuanto a las redes bayesianas, la obra "clásica" es el libro de Pearl [1988], "*Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*". También es recomendable conocer el libro de Neapolitan [1990] y la tesis doctoral de Fco. Javier Díez [1994].

8 MARCOS Y GUIONES

F. J. Díez Vegas

Los marcos *fueron propuestos por Marvin Minsky en 1975 como método de representación del conocimiento y de razonamiento, intentando superar las limitaciones de la lógica a la hora de abordar problemas como la visión artificial, la comprensión del lenguaje o el razonamiento de sentido común. En su propuesta, más que describir con detalle un método completo, Minsky trató de dar ideas y sugerencias, y por eso no es extraño que del planteamiento original hayan surgido tantos modelos y tan diferentes entre sí. De hecho, si examinamos con detenimiento cada uno de ellos, veremos que las diferencias son mucho mayores que las semejanzas, y que a pesar de utilizar una terminología común responden a conceptos muy distintos. Nuestro propósito consiste en mostrar al lector la idea original de marco para estudiar después algunos de las aplicaciones concretas que se han desarrollado a partir de ella. Hablaremos de PIP, el primer sistema experto basado en marcos, del lenguaje KRL y de las herramientas actuales que utilizan marcos; estudiamos éstas últimas por su importancia actual en el campo de la IA, especialmente en cuanto a la construcción de aplicaciones prácticas, aunque desde un punto de vista teórico tienen poco en común con los marcos de Minsky y con el sistema experto PIP.*

Por otro lado, los guiones *surgieron en el grupo de trabajo de Roger Schank también a mediados de los años 70, con el fin de extender las capacidades de los grafos de dependencia conceptual, pues éstos resultaban muy insuficientes a la hora de generar las inferencias adecuadas en la comprensión del lenguaje natural. En cuanto extensión de estos grafos, deberíamos haber estudiado los guiones dentro del capítulo 7. Sin embargo, nos ha parecido más oportuno estudiarlos después de los marcos, pues la*

inferencia mediante guiones es prácticamente un caso particular del modelo propuesto por Minsky. Tal como hemos hecho hasta ahora con los demás métodos, también para los marcos y los guiones intentaremos mostrar la estrecha relación que existe entre representación del conocimiento y razonamiento (inferencia).

8.1 CONCEPTO DE MARCO

8.1.1 La Propuesta de Minsky

El concepto de marco como método de representación del conocimiento fue introducido por Marvin Minsky [1975]. De su artículo (versión de 1981) extraemos la siguiente cita:

> Un marco es una estructura de datos para representar una situación estereotipada, como encontrarse en un cierto tipo de sala de estar o asistir a un cumpleaños infantil. Cada marco lleva asociadas varias clases de información. Parte de esta información indica cómo usar el marco. Otra parte se refiere a lo que uno espera que suceda a continuación. Otra parte indica qué hacer cuando no se confirman tales expectativas.

Cada marco se caracteriza por un conjunto de *campos* (en inglés, "*slots*"). Las distintas formas de representación basadas en este formalismo se distinguen entre sí especialmente por la forma en que definen y utilizan los campos.

En la propuesta original de Minsky, los campos sirven para identificar los marcos. Por ejemplo, podemos representar el marco `casa` con varios campos, algunos de los cuales serán las habitaciones. Cada habitación es a su vez un marco con varios campos. El marco `dormitorio` puede tener una cama y una mesa de noche como campos; el marco `cuarto-de-baño` puede tener una bañera, un lavabo, etc. El hecho de que unos marcos pueden ocupar los campos de otros marcos da lugar a una red, aunque debemos aclarar que en la propuesta original no se trata de una red clasificación, ni siquiera de una red jerárquica.

Los marcos están especialmente concebidos para tareas de reconocimiento. (Minsky pensaba sobre todo en problemas de comprensión del lenguaje natural y de visión artificial.) Siguiendo con el ejemplo anterior, si llegamos a una casa en que no habíamos estado anteriormente y vemos una cama, la coincidencia con un campo del marco `dormitorio` puede llevarnos identificar la habitación de que se trata, y los demás campos de este marco nos

hacen pensar que en la misma habitación puede haber una mesa de noche, un armario ropero, etc. En cambio, el observar un lavabo nos lleva a pensar que se trata de un `cuarto-de-baño`, y eso nos hace suponer que en la misma habitación hay una bañera, un espejo, etc. Por tanto, la información contenida en los marcos sirve tanto para reconocer un marco como para predicir la existencia de otros elementos.

De este modo, la información recibida hace que se activen unos marcos y esto a su vez provoca la activación de otros marcos conectados con los primeros, dando lugar así a una *red de activación*, cuyo objetivo es predecir y explicar la información que se va a encontrar en esa situación. Este reconocimiento basado en expectativas se denomina a veces *reconocimiento descendente*.

Otra de las ideas novedosas de Minsky es la posibilidad de tener distintos marcos para definir una misma entidad desde distintos *puntos de vista*. Por ejemplo, un mismo hombre puede ser profesor, padre de familia, cliente de un banco, víctima en un accidente, etc. Para cada uno de estos aspectos existirán marcos diferentes. El marco `profesor` puede tener campos como área de docencia, centro de trabajo o años de experiencia. El marco `padre-de-familia` tendrá campos como esposa, hijos o fecha de la boda. En cuanto cliente de un banco tendrá un número de cuenta, un saldo, quizá alguna tarjeta de crédito...

En realidad, el propósito de Minsky consistía en dar sugerencias e ideas más que en concretar los detalles de un cierto tipo de representación. De hecho, el artículo anteriormente citado, más que especificar con detalle una forma de representar el conocimiento, ofrece numerosas ideas y sugerencias en cuanto al razonamiento por analogía, valores por defecto, expectativas, conocimiento parcial sobre una entidad, ajuste parcial ("*partial matching*"), etc., aunque en muchos casos no explica cómo podrían llevarse a la práctica estas ideas. Por este motivo no es de extrañar que los marcos hayan sido utilizados de formas muy diferentes, como veremos en este capítulo. Vamos a mostrar a continuación algunas de las aplicaciones concretas que surgieron a partir de la propuesta de Minsky y al final del capítulo hablaremos del debate entre los partidarios y los detractores de los marcos como método de representar el conocimiento.

Comentario

Para entender mejor el nombre de "marco" conviene recordar que el término original "*frame*" tiene muchos significados. Entre otros se encuentran,

además de "marco", los siguientes: "condición o estado", "armazón, estructura básica alrededor de la cual se construye algo", "conjunto de circunstancias que rodean un suceso", "imagen en un rollo de película, y de forma más general, los objetos en que se centra la atención de una cámara de cine". Minsky pensaba sobre todo en "imagen en un rollo de película", pero no debemos olvidar que "*frame*" encierra también todos los significados anteriores. Por eso, al traducirlo por "marco" estamos perdiendo casi todos los matices del término original.

También conviene señalar que, según Minsky, cada marco posee varios "*slots*" o "*terminals*", que en castellano suelen denominarse "campos". La palabra "*slot*" significa "ranura", es decir, un hueco destinado a contener algo que encaja allí. Aunque algunos traductores emplean este término castellano, a nosotros nos parece más correcto hablar de "campos", aunque se pierda así la distinción que existe en inglés entre el campo de un marco (*slot*) y el campo de un registro en una base de datos (*field*).

8.1.2 El Sistema Experto PIP

Una de las primeras aplicaciones de la propuesta de Minsky fue el sistema experto PIP (*Present Illness Program*) [Pauker *et al.*, 1976; Szolovits & Pauker, 1978]. En este sistema experto, cada enfermedad viene representada por un marco. La Fig. 8.1 muestra el correspondiente al síndrome nefrótico; aunque el lector no comprenda todos los términos médicos que aparecen, lo importante es observar los nombres de los campos, que se encuentran resaltados en negrita.

El campo "`nombre`" identifica el marco y el campo "`es-un-tipo-de`" lo clasifica, indicando si es estado fisiológico, enfermedad, etc. Según los creadores de PIP, «cada marco suele contener de cinco a diez hallazgos [datos observados], tres o cuatro reglas de exclusión [campo "`no-debe-haber`"], de diez a veinte parámetros de ponderación [criterios mayores y menores] y de cinco a diez enlaces con otros marcos de la red [campos "`puede-estar-producido-por`", "`puede-complicarse-por`", "`puede-producir`", "`diagnóstico-difencial`"...]». En un momento dado cada marco se encuentra en uno solo de cuatro estados posibles: durmiente, semiactivo, activo o aceptado. Inicialmente todos los marcos están *durmientes*.

El proceso de diagnóstico empieza con la recogida de información. Cuando se encuentra en el paciente un síntoma o un signo, todos los marcos caracterizados por ese `hallazgo` *se activan*, con lo cual pasan a considerarse

Nombre: Síndrome nefrótico
Es-un-tipo-de: Estado clínico
Hallazgo: Concentración de albúmina baja
Hallazgo: Proteinuria > 25g / 24 horas
Hallazgo: Edema simétrico masivo
Hallazgo: Lípidos en la orina presentes
No-debe-haber: Proteinuria ausente
Es-suficiente:
Edema masivo junto con proteinuria > 25 g / 24 horas
Criterios mayores:

Concentración de sero-albúmina	
Baja:	1.0
Alta:	-1.0
Proteinuria	
> 25 g / 24 horas:	1.0
Intensa:	0.5
Ausente o leve:	-1.0

Criterios menores:

Concentración de colesterol	
Elevada:	1.0
No elevada:	-1.0
Lípidos en la orina	
Presentes:	1.0
Ausentes:	-0.5

Puede-estar-producido-por:
Glomerulonefritis
Fármacos nefrotóxicos
Síndrome nefrótico idiopático
Puede-complicarse-por:
Hipovolemia
Celulitis
Puede-producir:
Retención de sodio
Diagnóstico-diferencial:
Si presión venosa yugular elevada,
considerar: pericarditis constrictiva.
Si ascitis presente,
considerar: cirrosis

Fig. 8.1. Marco "síndrome nefrótico" del sistema experto PIP (abreviado).

como candidatos para el diagnóstico. Los marcos vecinos de un marco activado —es decir, los que se encuentran unidos a él a través de alguno de los enlaces mencionados— pasan a estar *semiactivos*, por lo que posteriormente podrán ser evaluados cuando se reciba nueva información. De este modo se genera un espacio de búsqueda limitado, evitando los problemas de una explosión incontrolada del conjunto de hipótesis.

Una vez que se ha realizado la primera fase de recogida de información, el sistema procede a evaluar las hipótesis generadas, con el objetivo de encontrar el marco-diagnóstico que explique mejor toda la evidencia disponible. (Observar que éste es un proceso de *reconocimiento* semejante a los que se realizan en visión artificial o en la comprensión del lenguaje natural.)

El campo "`es-suficiente`" puede llevar a considerar un marco como *aceptado*, mientras que el campo "`no-debe-haber`" puede excluirlo del conjunto de hipótesis. Sin embargo, en muchos casos la evidencia disponible no permite aceptar ni rechazar un diagnóstico con certeza absoluta, por lo que es necesario ponderar la evidencia a favor y en contra, de acuerdo con los "`criterios-mayores`" y "`criterios-menores`". Cuando la evidencia acumulada supera cierto umbral, el marco correspondiente pasa a considerarse *aceptado*. En caso de que no se haya llegado todavía a un diagnóstico, puede procederse a una segunda fase de recogida de información, orientada por el conjunto de hipótesis disponible (los marcos activos), dando lugar así a un proceso cíclico de generación de preguntas y de ponderación de la evidencia, hasta llegar a un diagnóstico satisfactorio.

La ventaja principal de PIP frente a los sistemas de diagnóstico basados en *árboles de decisión*[1] era la flexibilidad en el diálogo (el usuario podía introducir la información en el orden deseado), la generación de preguntas en función de la evidencia disponible, y el diagnóstico de múltiples enfermedades (un árbol de decisión se basa en la hipótesis de que los diagnósticos son exclusivos entre sí, es decir, que no puede haber dos enfermedades presentes al mismo tiempo).

[1] En un árbol de decisión, cada nodo representa una pregunta con varias respuestas alternativas, una por cada rama que emerge del nodo; los terminales corresponden a los diagnósticos posibles. Los árboles de decisión se utilizaron antes del surgimiento de la IA en numerosos programas de ayuda en la toma de decisiones ("computer-aided decision making"), especialmente para el diagnóstico médico.

8.1.3 El Lenguaje KRL

Al año siguiente de la publicación del trabajo de Minsky, un grupo de investigadores dirigido por Bobrow y Winograd [1977] decidió llevar a la práctica su propuesta, diseñando e implementando el lenguaje de programación KRL (*Knowledge Representation Language*). Los autores pretendían que este lenguaje pudiera servir de base en la resolución de gran variedad de problemas de IA, tales como el reconocimiento del habla o los sistemas expertos, que hasta entonces se habían basado en lenguajes de propósito general: Lisp, FORTRAN, etc. El proyecto era muy ambicioso y quedó lejos de alcanzar sus objetivos, pero su influencia ha sido notable en muchos de los sistemas y herramientas de la actualidad.

El propósito fundamental de KRL consistía en ofrecer los mecanismos necesarios para una **representación estructurada** del conocimiento (en contraposición a la lógica de predicados y sus variantes, que carecen de estructura). Por ello la información se agrupa en torno a los objetos o unidades ("*units*"), que consituyen el eje fundamental de la representación del conocimiento. Veamos un ejemplo:

```
[Viaje UNIDAD Abstracción
    <IDENTIDAD (un Suceso)>
    <modo (O Avión Automóvil Autobús)>
    <destino (una Ciudad)>]

[Visita UNIDAD Especialización
    <IDENTIDAD (una Interacción-Social)>
    <visitante (una Persona)>
    <visitado (Conjunto-De (una Persona)>]

[Suceso137 UNIDAD Individuo
    <IDENTIDAD {(una Visita con
                    visitante = Pedro
                    visitados =
                        (Elementos Antonio Luisa))
               {(un Viaje con
                    destino = Salamanca
                    modo = Autobús)}>]
```

Las dos primeras unidades definidas en este ejemplo corresponden a lo que hoy llamaríamos *clases* o *marcos*, mientras que la última es una *instancia*, la cual aparece descrita desde dos perspectivas: en cuanto viaje y en cuanto visita (acto social).

Los campos de una unidad pueden tener *valores por defecto*, que provienen de la *herencia de propiedades* (en el ejemplo anterior se observan algunos aspectos de la organización jerárquica (sec. 7.3)) o bien de la existencia de *prototipos*. Un *prototipo* es una unidad que en vez de corresponder a un objeto concreto representa un elemento típico de una clase. La sección 5.5.3 trata con más detalle el razonamiento por defecto.

Otro mecanismo de razonamiento muy importante es la *comparación* o *ajuste* (en inglés "*matching*"), similar a la comparación de patrones de que hablamos en el capítulo de las reglas. Cuando los campos de un objeto conocido se ajustan a las especificaciones de un marco, aumenta la evidencia de que tal objeto pertenece a la clase definida por el marco. KRL permite ponderar la distinta importancia que cada atributo desempeña en la identificación de un objeto. Este mecanismo es en esencia el mismo que utiliza el sistema experto PIP del que hemos hablado anteriormente. Observar que la comparación de marcos es más flexible que la comparación de reglas, pues en el primer caso puede bastar un ajuste parcial, mientras que una regla sólo se ejecuta cuando se han satisfecho una por una todas sus cláusulas.

Entre los objetivos básicos de KRL destaca la preocupación por la eficiencia, aun a costa de perder elegancia desde un punto de vista teórico. Una muestra de ello es la *redundancia*, que consiste en incluir explícitamente cierta información para no tener que deducirla. Además, ofrece numerosas posibilidades de controlar el razonamiento. Una de ellas es la *asignación de procedimientos* a los marcos o sus campos ("*procedural attachment*", en inglés), los cuales pueden ser de dos tipos: los *sirvientes*, que son procedimientos activados por el marco para alcanzar cierto objetivo, y los *demonios*, que se activan automáticamente en ciertas circunstancias. Los sirvientes son similares a los métodos o gestores de mensajes ("*message handlers*") de la programación orientada a objetos. De los demonios hablaremos más adelante.

Otros mecanismos para el control del razonamiento son la profundidad de procesamiento variable (el usuario puede determinar así cuántos marcos van a ser activados), la gestión de las agendas, el establecimiento de niveles de prioridad, y algunos métodos de naturaleza heurística o estratégica semejantes a los que mencionamos al hablar de la representación mediante reglas (sec. 6.4). Por último, KRL incluye varias posibilidades, unas casi automáticas, otras programables por el usuario, para detectar errores, tales como la asignación de valores incompatibles o absurdos en los campos de un marco.

Muchas de las ideas desarrolladas por los creadores de KRL han sido integradas en herramientas desarrolladas posteriormente. Como suele ocurrir, el proyecto original era mucho más general que los derivados comerciales que han surgido de él, aunque éstos últimos tienen la ventaja de ser más eficientes y más sencillos de utilizar. A ellos está dedicada la próxima sección.

8.1.4 Herramientas Basadas en Marcos

En la actualidad existen numerosas herramientas comerciales basadas en marcos. Aunque hay algunas, como FrameKit, que utilizan solamente este tipo de representación, la mayor parte de ellas permite combinar marcos y reglas. Entre ellas podemos citar ART, KEE, GoldWorks, Nexpert y otras de las que hablamos en los capítulos 6 (reglas) y 9 (sistemas expertos).

Aunque estas herramientas están inspiradas en la propuesta de Minsky, en la práctica carecen de muchas de las propiedades que él sugirió. Una de estas propiedades esenciales era la organización de los marcos mediante una red (no necesariamente jerárquica) en la que unos marcos podían ocupar los campos de otros marcos, dando lugar así a una *red de activación*. Sin embargo, en las herramientas actuales existen mecanismos de inferencia que no suelen diferenciar entre marcos activos e inactivos.

Por otra parte, muchas de las características que conforman los marcos en la actualidad están inspiradas en los sistemas basados en redes de clasificación (ver la sección 7.3). Por ejemplo, tanto la distinción entre clase o concepto e instancia como el concepto de herencia son hallazgos de las redes semánticas que no aparecen en la propuesta original de Minsky. En cambio, la *organización jerárquica* de los marcos desempeña un papel esencial en las herramientas actuales, de modo que la herencia de campos, de valores por defecto, de demonios, etc., constituye en ellas el principal modo de inferencia.

La diferencia entre los marcos originales y los actuales se debe principalmente a que aquéllos estaban orientados sobre todo al reconocimiento de imágenes y del lenguaje natural, mientras que en la actualidad suelen utilizarse en la construcción de sistemas expertos, donde el diagnóstico mediante comparación de patrones desempeña un lugar menos importante.

8.2 INFERENCIA MEDIANTE MARCOS

Hemos mencionado ya que los sistemas actuales basados en marcos se basan sobre todo en la herencia a partir de una red jerárquica. Algunos de los mecanismos que vamos a mostrar a continuación surgieron —a veces con nombres diferentes— en redes semánticas, como el sistema SCHOLAR de Carbonell (sec. 7.1.2) o las redes jerárquicas (sec. 7.3), mientras que otros imitan la forma en que el lenguaje KLR implementó las sugerencias de Minsky.

El punto de partida consiste en la creación de una red de marcos e instancias. Cada **marco** representa un concepto o una clase, y cada **instancia** representa un elemento dentro de la clase. En algunas herramientas —como GoldWorks— cada instancia pertenece a un solo marco, mientras que en otras —como Nexpert— cada instancia puede pertenecer a varios marcos. Cada marco hereda los campos de todos sus antepasados en la red, y cada instancia hereda campos y valores de todos los marcos a los que pertenece. Algunas herramientas sólo permiten herencia descendente, mientras que en otras admiten herencia bidireccional, de modo que el valor asignado a una instancia puede convertirse en valor por defecto del marco al que pertenece.

8.2.1 Facetas

Las facetas definen las propiedades de un campo. Entre las que suelen darse en las herramientas actuales destacamos las siguientes:

Valor por defecto: Es el valor que toma el campo, salvo que posteriormente se indique lo contrario. En general, el valor por defecto se asigna al marco y es heredado por todas las instancias creadas posteriormente dentro de él.

Multivaluado: Es una faceta que indica si el campo es univaluado o multivaluado. En el primer caso, la introducción de un nuevo valor desplaza el anterior, mientras que si el campo es multivaluado pueden coexistir varios valores simultáneamente. (Esta propiedad aparecía también en algunos sistemas basado en reglas; ver la sec. 6.2.3.)

Restricciones: Limitan el conjunto de valores que puede recibir el campo; en caso de una asignación incorrecta, el sistema reconoce que ha habido un error y toma las medidas oportunas. Por ejemplo, para la `edad` de una persona puede haber una restricción que admita solamente *valores numéricos*, o se puede restringir más aún el *rango* de valores para que la edad esté entre 0 y 130 años. También se puede exigir, por poner otro

ejemplo, que la `ocupación` de una persona sea una *instancia* del marco `profesión`.

Certeza: Indica la credibilidad del valor asignado al campo. Es semejante al factor de certeza asociado a cada proposición dentro de un sistema basado en reglas (sec. 6.6.1).

Facetas de interfaz: contienen texto, preguntas y mensajes destinados a la interacción con el usuario. Por ejemplo, para el campo edad puede haber una faceta que contenga la cadena de caracteres "¿Cuál es la edad del paciente?", la cual se utiliza para solicitar información; o puede haber mensajes como "La edad del paciente se expresa en años" o "La edad debe estar entre 0 y 130 años", que aparecerán en el interfaz cuando sea necesario.

Existe otro tipo de facetas, los demonios, que por su especial importancia vamos a tratar a parte con más detenimiento, y otras de menor importancia, tales como las que se utilizan para incluir comentarios y documentar las bases de conocimientos.

Algunas herramientas ofrecen la posibilidad de que el usuario defina sus propias facetas cuando lo necesite. Por ejemplo, en el sistema experto DIAVAL, implementado sobre GoldWorks, cada paciente es una instancia del marco `paciente`, y posee campos como peso, estatura, superficie corporal, etc. El diseñador del programa definió nuevas facetas para indicar cuál es la *unidad* en que se mide cada parámetro (Kg, cm, m^2, etc.), el rango de *valores admisibles* y el rango de *valores típicos* [Díez, 1994].

8.2.2 Demonios

Los demonios son funciones o procedimientos que se invocan *automáticamente* al realizar ciertas operaciones sobre el valor de un campo. Se utilizan sobre todo para actualizar los campos que dependen de otros, manteniendo así la consistencia del sistema.

Supongamos que una empresa desea construir un sistema experto para gestionar, entre otros asuntos, las nóminas de sus empleados. Podemos definir un marco `empleado` que tenga como campos, entre otros, la `categoría` y el `sueldo`. Naturalmente, el segundo depende del primero, y deseamos que la consistencia entre ambos se mantenga de forma automática. Para ello tenemos dos posibles soluciones:

1.- Añadimos un demonio en el campo `categoría`, de modo que *cuando se modifique* la categoría de un empleado, este demonio se encargará de actualizar el campo `sueldo`.

2.- Añadimos un demonio en el campo `sueldo`, de modo que *cuando se solicite* el valor de ese campo, el demonio lo recalculará teniendo en cuenta la `categoría` del empleado en ese momento.

La primera solución es más eficiente y elegante que la segunda, pero eso nos importa poco por ahora. Lo que deseamos mostrar con este ejemplo es que un campo puede tener asociados distintos tipos de demonios, y cada uno de ellos se activa ante un suceso concreto. Entre ellos se encuentran los siguientes:

Demonio de necesidad: se ejecuta cada vez que se necesita conocer el valor de ese campo y no hay ningún valor asignado.

Demonio de acceso: se ejecuta cada vez que se solicita el valor de un campo, aunque ya hubiera un valor asignado.

Demonio de asignación: se ejecuta cada vez que se añade un valor al campo.

Demonio de modificación: se ejecuta cuando varía el valor de un campo.

Demonio de borrado: se ejecuta al eliminar el valor de un campo.

Los demonios no solamente se utilizan para relacionar entre sí los campos de una misma instancia; también sirven para actualizar la presentación gráfica del interfaz, para buscar en una base de datos la información requerida, para controlar dispositivos externos, etc.

8.2.3 Puntos de Vista

Hemos dicho que, en general, cada campo de una instancia puede tener asignado un valor. Sin embargo, algunas herramientas avanzadas permiten que un campo de una instancia tenga varios valores correspondientes a distintos puntos de vista (el término original es "*views*").

Veamos un ejemplo adaptado del sistema FrameKit:

```
(definir-instancia carne-de-vaca
  (comestible (valor (general sí)
                     (hindú no))))
```

Esto significa que estamos representando la carne de vaca como una instancia que posee un campo, `comestible`, cuyo valor depende del punto de

vista que adoptemos. A la hora de realizar inferencias en el sistema con el fin de determinar si la carne de vaca puede formar parte de un menú, el resultado dependerá del punto de vista.

Este mecanismo de control del razonamiento, propio de la representación basada en marcos, está inspirado en la propuesta de Minsky, y su objetivo es similar al que pretenden las *perspectivas* que aparecen en el lenguaje KRL, aunque la forma de implementación es completamente diferente.

8.3 GUIONES

8.3.1 Planteamiento del Problema

Aunque los incluimos en este capítulo por su semejanza con los marcos, los *guiones* fueron creados por Schank y sus colaboradores como continuación o amplicación de su trabajo sobre representación de dependencia conceptual que fue mostrado en la sección 7.1.3. En efecto, Schank había investigado la representación de frases y las inferencias que podían realizarse en el proceso de comprensión. Sin embargo, muchas veces el significado de una frase no puede estudiarse de forma aislada, sino que ésta toma su sentido dentro de un contexto. Incluso el problema de escoger el significado concreto de una palabra polisémica muchas veces sólo puede resolverse dentro de un contexto. (Éste problema, que se denomina en inglés "*disambiguation*", ya había sido abordado por Quillian en sus redes semánticas.)

El término *guión* es la traducción de "*script*" y está tomado en sentido metafórico del cine, como "argumento de una obra, expuesto con todos los pormenores necesarios para su cabal realización". En IA, un guión es básicamente una *estructura de conocimiento que contiene una secuencia estereotipada de acciones*. Aquí hay que entender "acción" en sentido amplio, es decir, incluyendo también la conversación, el pensamiento, los actos de voluntad y los sentimientos.

Observar también que, en cuanto estructura de conocimiento, los guiones están más cerca de los marcos que de las redes proposicionales, y por este motivo los hemos incluido en este capítulo. Nos referimos, naturalmente, a los marcos propuestos por Minsky (sec. 8.1.1), mucho más flexibles que los "marcos de las herramientas actuales" (sec. 8.1.4). Al igual que un marco tiene campos que sirven para reconocer los objetos, un guión tiene escenas que sirven para reconocer situaciones. Por tanto, un guión puede considerarse como un

tipo particular de marco, con las peculiaridades de que cada campo corresponde a un *suceso* y de que los campos-sucesos forman una *secuencia.*

8.3.2 Representación del Conocimiento

Tal como hemos afirmado, un guión representa una secuencia de acciones, unidas entre sí por una relación de causalidad, en el sentido de que la realización de una de ellas permite que ocurra la siguiente.

El ejemplo más conocido es el guión denominado $RESTAURANTE, que describe los sucesos típicos de una comida o cena en un restaurante. En lenguaje natural, este guión (adaptado de [Schank, 1977]) podría expresarse así:

> El cliente entra en el restaurante y se sienta. El camarero le entrega el menú. El cliente selecciona unos platos. El cocinero prepara la comida y el camarero la sirve. El cliente come la comida que le han servido; después paga y se va del restaurante.

Los programas que utilizan guiones almacenan la información mediante los grafos de dependencia conceptual que hemos estudiado en la sección 7.1.3. En nuestro ejemplo, el hecho de entrar en el restaurante viene dado por a la acción primitiva PTRANS. El "comer" se representa mediante un grafo semejante al de la Fig. 7.4. (era la representación de "Juan bebe agua") y el grafo correspondiente a "pagar la comida" es semejante al de la Fig. 7.5. ("Juan compró un libro").

Sin embargo, la representación de guiones va más allá de la representación de frases aisladas. A parte de que los guiones *enlazan causalmente* secuencias de acciones, la diferencia principal consiste en que cada guión posee *roles* correspondientes a las personas que intervienen. En el ejemplo anterior, los roles son cliente, cocinero y camarero, y cada uno de ellos suele aparecer en varias ocasiones a lo largo de la historia. Del mismo modo, existen también *objetos* propios de cada guión.

Los *roles* y algunos de los *objetos* se representan (en el nivel simbólico) mediante variables y esto permite que puedan ser asignados a diferentes personas o cosas; por ejemplo, el rol "cliente" puede ser desempeñado por Luis, por María, o incluso por un grupo de amigos que van juntos a cenar; del mismo modo, prácticamente cualquier alimento puede constituir la comida solicitada. El imponer *restricciones* sobre el tipo de personas que pueden desempeñar ciertos roles y sobre el tipo de objetos que cumplen cierta función será de gran utilidad

en el proceso de interpretación de textos, tal como veremos más adelante al hablar de la inferencia.

Además de los componenetes mencionados anteriormente, cada guión posee *cabeceras*, que pueden ser de varios tipos. Uno de ellos indica las *precondiciones* que pueden llevar a que ocurra la situación descrita; por ejemplo, para el guión $RESTAURANTE, una precondición puede ser "tener hambre". Si en un texto escrito aparece una frase como "Luis tenía hambre", ese guión es uno de los candidatos para explicar la información que aparezca a continuación; la inferencia se basa en que un restaurante es uno de los lugares donde se puede conseguir comida con el fin de saciar el hambre. Otra de las cabeceras puede indicar un *instrumento* para las acciones contenidas en otro guión; así, una frase como "Luis viajó a Barcelona" puede activar los guiones $AUTOBÚS, $TREN o $AVIÓN, pues cada uno de ellos sirve como "instrumento" para el viaje. Por último, entre las cabeceras que pueden llevar a la activación de un guión, mencionamos la relativa al *lugar* donde ciertos acontecimientos suelen ocurrir: la palabra "comida" puede invocar $RESTAURANTE, "natación" puede invocar $PISCINA o $PLAYA, "película" puede invocar $CINE, etc.

En resumen, un guión completo se compone principalmente de los siguientes elementos:

- **escenas:** los sucesos descritos en el guión, enlazados causalmente en forma de secuencia;
- **roles y objetos:** corresponden las personas y las cosas que intervienen; incluyen restricciones para indicar qué personas u objetos pueden ser asignados a las variables;
- **cabeceras:** además de la que da nombre al guión, hay otras que representan condiciones, instrumentos y lugares; su misión es activar el guión en el momento oportuno.

8.3.3 Inferencia mediante Guiones

El objetivo de utilizar guiones no se limita a la representación del conocimiento, sino que está orientado ante todo a la comprensión del lenguaje natural, más concretamente, a la interpretación de textos escritos. En la práctica, para comprobar que la interpretación ha sido correcta, o dicho de otro modo, para comprobar que el sistema ha "comprendido" el texto, suele seguirse uno de estos dos métodos: *(1)* formularle *preguntas* concretas y ver si responde

adecuadamente, o *(2)* pedirle que repita el mismo texto con otras palabras (*paráfrasis*)..

En el proceso de inferencia, el primer paso consiste en *seleccionar el guión* que mejor explica la historia que se desea analizar. La aparición explícita de una palabra como "restaurante" puede invocar directamente el guión que lleva ese nombre. También las cabeceras que aparecen en un guión (precondiciones, instrumentos, lugares...) pueden provocar la activación de un guión, tal como hemos explicado anteriormente.

Posteriormente, hay que *asignar las variables*, es decir, identificar los roles, objetos y lugares que intervienen. A partir de ahí, el guión "instanciado" permite extraer la información que no aparecía explícitamente en la historia escrita; en esto consiste precisamente la *inferencia*.

Podemos explicar este proceso con más detalle estudiando la siguiente historia:

> Luis y Rosa fueron a cenar a "Torre Blanca" el jueves pasado. En cuanto llegaron, dos camareros les atendieron con mucha amabilidad. Pidieron sopa de marisco, cordero asado y tarta de manzana. Todo estaba delicioso. Dejaron una buena propina y decidieron volver allí el fin de semana siguiente.

En primer lugar, sabemos que el lugar más habitual para "ir a cenar" es un restaurante. Éste puede ser un motivo suficiente para *invocar* el guión $RESTAURANTE, a pesar de que la palabra que lo identifica no aparece explícitamente en el texto. En general, la interpretación de una historia no activa un solo guión sino varios de ellos, por lo que se hace necesario un proceso de *selección* posterior con el fin de determinar cuál o cuáles de ellos son válidos y cuáles deben ser rechazados.

Como hemos indicado, el segundo paso consiste en *asignar las variables*. El rol de cliente lo desempeñan Luis y Rosa. El lugar donde se desarrolló la comida se llama "Torre Blanca", y suponemos que es un restaurante. El nombre de los camareros nos es desconocido, pero sabemos que eran dos en vez de uno solo. Observar que la interpretación implica un proceso de reconocimiento, que en la práctica se implementa mediante comparación de patrones: la representación del guión debe especificar que el cliente es una persona o un grupo de personas; Luis y Rosa son dos nombres de hombre y de mujer, y por tanto se ajustan a la condición exigida. Igualmente la sopa, el cordero y la tarta cumplen

la condición de ser alimentos, por lo que pueden ser asignados como valor de la variable &COMIDA, que aparece en el guión.[2]

Una de las cuestiones básicas de la interpretación de un texto es averiguar los *referentes de los pronombres* y el *sujeto* de la frase cuando éste se ha omitido. (La omisión del sujeto es poco frecuente en inglés pero muy frecuente en castellano.) La concordancia de género y número es una condición que limita el espacio de búsqueda, pero en muchos casos los criterios sintácticos son insuficientes. Por ejemplo, en la historia anterior, el sujeto de "pidieron sopa..." podría ser "los camareros", que es además el referente anterior más reciente. Sin embargo, el guión nos dice que quien pide la comida en un restaurante no son los camareros sino el cliente; en este caso, Luis y Rosa. Sin la información semántica sería imposible resolver este tipo de ambigüedades que surgen en el proceso de interpretación.

Otro tipo de inferencia que hemos mencionado anteriormente consiste en dar por suspuesto que han ocurrido *los demás sucesos* que se encuentran en el guión, aunque no aparezcan explícitamente en la historia escrita. Así, podemos suponer que Luis y Rosa se sentaron, que los camareros les sirvieron los platos pedidos y ellos los comieron, que al acabar de cenar pagaron la cuenta, etc.[3]

Por último, señalemos que existen dos métodos básicos de afrontar el problema de la comprensión de textos mediante guiones: ascendente y descendente. El *ascendente* consiste en analizar palabra por palabra, activando los marcos que mejor explican toda la información que va apareciendo. El método *descendente*, en cambio, selecciona un marco y trata de "encajar" en él la información de forma predictiva: la que se ajusta a las expectativas es asimilada y el resto se ignora. Este segundo método se utiliza cuando solamente se quiere extraer la información más relevante, pues tiene la ventaja de ser más eficiente y más robusto (es decir, menos sensible a detalles que puedan perturbar la interpretación). La contrapartida a esta "robustez" es que en muchos casos este método desprecia información relevante simplemente porque no se ajusta a

[2] El reconocimiento de patrones de que estamos hablando guarda cierta semejanza con el que utilizan los programas de deducción lógica (sec. 5.4.2) y el encadenamiento de reglas (sec. 6.3.1), pero es mucho más flexible y potente, porque se apoya más en la información semántica.

[3] Estas observaciones pueden parecernos triviales, pues nosotros, lectores humanos, realizamos estas inferencias automática e inconscientemente, sin ningún esfuerzo. No obstante, debemos caer en la cuenta de que lo que es trivial para un ser humano no tiene por qué serlo para un ordenador, y este comentario se aplica a todos las ramas de la inteligencia artificial, no solamente al procesamiento del lenguaje natural.

las expectativas del guión. Por este motivo se han construido sistemas que combinan el reconocimiento ascendente y el descendente, con el fin de aprovechar las ventajas de cada uno de ellos; algunos de estos sistemas se encuentran descritos brevemente en [Dyer *et al.*, 1992].

8.3.4 Ventajas, Inconvenientes y Extensiones

A pesar de que el tratamiento del lenguaje natural es un problema muy complejo —probablemente el más complejo de la IA— y todavía queda mucho camino por recorrer, los guiones han signicado un paso hacia adelante muy importante. Su éxito principal consiste en poder "leer" textos relativamente breves (tales como artículos de periódico) sobre temas concretos, extrayendo información. Al igual que en otros campos de la IA, a veces sorprende observar la precisión y "sensatez" con que alguno de estos programas responde a ciertas preguntas, mientras que en otras ocasiones las respuestas son tan absurdas que provocan la sonrisa o la decepción.

Concretando más la crítica de los guiones en comparación con otros mecanismos de representación del conocimiento, conviene recordar ante todo que fueron desarrollados por Schank y sus colaboradores como extensión de los grafos de dependencia conceptual. En consecuencia, una de las cualidades de los guiones es que representan el conocimiento sin depender de ningún idioma particular. Por ejemplo, el guión $RESTAURANTE constará de las mismas escenas cualquiera que sea la lengua a la que se vaya a aplicar el sistema; también las inferencias a que da lugar vendrán representadas en forma de grafos conceptuales, independientes del idioma. Este hecho es importante de cara a los programas de *traducción automática*, pues hoy en día se considera útil disponer de una representación "interlingua" del texto original que permita después expresar el contenido en uno o en varios idiomas diferentes.

El principal avance que han aportado los guiones frente a los grafos de dependencia conceptual es el *control de inferencias*. Por ejemplo, en la sección 7.1.3 comentábamos que de la frase "Juan comió un filete" se podía inferir que Juan existía, que el filete existía y que ambos estuvieron en contacto en algún momento. Sin embargo, estas inferencias son de poca utilidad a la hora de comprender los hechos esenciales. Los guiones, en cambio, permiten integrar la información con el fin de formar una historia coherente, y realizan solamente las inferencias relativas a la cadena causal de sucesos.

En cuanto a los inconvenientes, el principal de ellos es la rigidez del mecanismo de representación: cada guión representa una secuencia fija de acciones. Aunque algunos esquemas permiten que haya líneas alternativas dentro de un mismo guión (por ejemplo, en un restaurante puede ocurrir que el cliente pague la cuenta o que se marche sin pagar cuando no ha quedado satisfecho), la flexibilidad sigue siendo bastante reducida.

La limitación anterior está muy relacionada con la incapacidad de los guiones para compartir información entre ellos. Sería deseable que en la comprensión de una historia pudieran existir diferentes guiones que se combinaran entre sí de forma dinámica. Por ejemplo, el conocimiento relativo a una comida (es decir, las escenas que pueden ocurrir mientras un grupo de personas está comiendo) debe estar incluido en el guión $RESTAURANTE, pero también en el guión $AUTOSERVICIO, $HAMBURGUESERÍA e incluso en $COMIDA-EN-CASA; sería deseable que estos guiones pudieran compartir la información común sin tener que repetirla en cada uno de ellos. Por otro lado, una comida en un restaurante puede entenderse como una situación en la que el cliente paga por recibir un servicio. Una de las escenas que pueden aparecer en este tipo de situaciones es que el cliente no quede satisfecho y, en vez de pagar, decida presentar una reclamación. En ese sentido, el guión $RESTAURANTE comparte información con $HOSPITAL, $SASTRERÍA, $TALLER-DE-AUTOMÓVILES, $ACADEMIA, etc. Se observa por tanto la necesidad de describir un mismo acontecimiento desde distintos puntos de vista (en cuanto "comida" y en cuanto "servicio", por ejemplo), combinando varios guiones para comprender una cierta historia.

Otra limitación de los guiones es su incapacidad para expresar las *motivaciones e intenciones*. En un ejemplo anterior se afirmaba que "Luis y Rosa dejaron una propina en el restaurante y decicieron volver allí otro día"; del contexto de la historia se deduce que el motivo era su satisfacción por la amabilidad de los camareros y por la calidad de la comida. Pero un sistema artificial que razone simplemente basándose en la secuencia de acciones del guión $RESTAURANTE (aparecen descritas en la sección anterior) nunca podrá llegar a establecer tal relación. De nuevo se observa la necesidad de comprender una historia desde distintos puntos de vista; en este ejemplo, hace falta conocer que la calidad de un servicio predispone al cliente a dejar propina y a volver a solicitarlo.

Una solución aportada por Schank para superar la rigidez de los marcos consiste en representar el conocimiento en forma de **paquetes de organización**

de memoria o MOPs (del inglés, "memory organization packets"). Al igual que los guiones, cada MOP viene descrito por una secuencia de escenas; la diferencia con respecto a los guiones consiste en los enlaces que existen entre los distintos MOPs. Por ejemplo, M-COMIDA se compone de las siguientes escenas: `preparar-comida`, `preparar-mesa`, `servir-comida`, `sentarse` y `comer`; del mismo modo, M-SERVICIO se compone de `solicitar-servicio`, `realizar-servicio` y `pagar-servicio`. Combinando elementos de estos dos, podemos construir M-RESTAURANTE con varias escenas, cada una de las cuales corresponde a las escenas de los MOPs anteriores: `pedir-la-comida` es `solicitar-servicio`, `traer-la-comida` es `realizar-servicio`, el `sentarse` de M-RESTAURANTE es `sentarse` de M-COMIDA, etc. Igualmente, los tres roles de M-RESTAURANTE (`cliente`, `camarero` y `cocinero`) son los participantes en las acciones de M-COMIDA y M-SERVICIO: el `cliente` pide la comida, el `cocinero` la prepara, el `camarero` la sirve, el `cliente` paga el servicio, etc. Estos enlaces permiten utilizar simultáneamente los tres MOPs anteriores, aplicando en cada momento el que mejor explica los acontecimientos que aparecen en la historia.

Concluimos esta sección señalando la estrecha relación de los MOPs con dos campos de la IA: el *aprendizaje* y el *razonamiento basado en casos*. En efecto, a la hora de interpretar una historia es poco probable que exista un guión o un MOP que se ajuste completamente al texto, por lo que en la práctica será necesario adaptar alguno de los que existen. Si estas modificaciones se almacenan de forma ordenada, el sistema podrá aprovechar la experiencia adquirida y aplicarla en la comprensión de nuevos casos, teniendo así un mecanismo de aprendizaje mediante MOPs (sec. 10.3.4).

Por otro lado, los MOPs sirvieron de base para el surgimiento de una nueva rama de la IA, el *razonamiento basado en casos* (RBC), que consiste básicamente en recordar alguna de las soluciones aplicadas a problemas del pasado y adaptarla a un nuevo problema. Ésta es una técnica usada con frecuencia por los seres humanos en la vida cotidiana; en Derecho —por citar un ejemplo de un campo especializado— la jurisprudencia desempeña un papel fundamental a la hora de dictar una sentencia. El algoritmo del RBC consta de los siguientes pasos: *(1)* seleccionar entre los problemas del pasado el que más se asemeja al problema actual; *(2)* hallar las semejanzas y diferencias, con el fin de determinar en qué medida es necesario adaptar la solución que se aplicó en el pasado; *(3)* almacenar tanto el problema actual como su solución, para que

puedan ser aprovechados en el futuro (en esto consiste el aprendizaje). En general es poco útil almacenar el caso como un bloque, pues es poco probable que en el futuro se encuentre un problema que coincida con él punto por punto; por eso conviene extraer los diferentes aspectos que intervienen (los diferentes puntos de vista) y es en este proceso donde la representación en forma de *paquetes de organización de memoria* muestra su utilidad.[4]

8.4 COMENTARIOS

Los guiones pueden considerarse, en cierto sentido, como una síntesis entre los grafos de dependencia conceptual (sec. 7.1.3) y los marcos de Minsky (sec. 8.1.1). Por tanto, las reflexiones que ofrecemos a continuación son aplicables en cierta medida también a los guiones.

8.4.1 Paradigmas en la Utilización de los Marcos

En nuestra opinión, existen básicamente dos modos en que los marcos han sido y son utilizados:

Patrones para la comparación (en inglés, "pattern matching"). De acuerdo con la propuesta de Minsky, cuando la información disponible indica que el problema planteado (diagnosticar una enfermedad o comprender un texto) se ajusta a un marco, el resto de la información contenida en dicho marco nos permite inferir la presencia de otros elementos. El proceso de razonamiento podría resumirse en los siguientes pasos: activación, reconocimiento y predicción. El sistema experto PIP y los guiones corresponden a este modo de utilizar los marcos.

Almacenes de información. En este caso los marcos significan ante todo una forma estructurada de almacenar información. La distinción entre clases e instancias y su organización en forma de red jerárquica permite utilizar la herencia de propiedades como mecanismo de razonamiento por defecto.

La capacidad de procesamiento asociada a cada marco varía notablemente de unos sistemas a otros. En un extremo, los marcos constituyen una especie de armarios en cuyos cajones (los campos) se almacenan los datos, separados del mecanismo de inferencia, que viene dado principalmente por

[4] El RBC es un campo de gran actividad dentro de la IA, y resulta imposible sintetizar en un solo párrafo las cuestiones relevantes. El lector interesado puede consultar las referencias que aparecen al final de este capítulo y en la sección 10.3.4.

un conjunto de reglas; la representación sería entonces semejante a una base de datos relacional. En el otro extremo, son las propias instancias quienes realizan la inferencia, tal como ocurre en la programación orientada a objetos.

Existen además sistemas híbridos, como los *prototipos* de Janice Aikins [1983], en que las reglas se almacenan en los campos de las instancias, de donde pasan al motor de inferencias.

El lenguaje KRL (sec. 8.1.3) participa de ambos paradigmas. Sin embargo, la mayor parte de las herramientas basadas en marcos que existen en la actualidad se ajustan al segundo paradigma, tal como se desprende de la descripción que aparece en la sección 8.1.4. De hecho, estas herramientas se encuentran bastante alejadas del concepto original de marco; la diferencia más notable es que no utilizan el proceso de activación-reconocimiento-predicción que constituía uno de los rasgos más significativos de la idea de Minsky.

8.4.2 Valoración

En los últimos capítulos hemos estudiado diferentes mecanismos de representación del conocimiento utilizados en IA. El primero de ellos (tanto histórica como conceptualmente) es la lógica y por eso es importante comparar con ella todos los demás métodos. Vimos en el capítulo 6 que las reglas surgieron como aplicación de la lógica, reduciendo la expresividad con el fin de mejorar la eficiencia. De los esquemas basados en redes (cap. 7), unos estaban inspirados en la psicología (las redes semánticas de Quillian), otros en la lingüística (los grafos de dependencia conceptual de Schank); otros trataban de implementar las características de la lógica de primer orden (las redes de Hendrix, Shapiro y Sowa). Los marcos estudiados en este capítulo se inspiran también en la psicología y, en gran medida, significaron un enfrentamiento con quienes defendían en los años 70 la lógica como fundamento de cualquier mecanismo de razonamiento en IA.

La principal ventaja de los marcos frente a la lógica es que representan el conocimiento de forma estructurada: asignar un valor V en el campo C de un marco M es equivalente a establecer la proposición $C(M,V)$. Por ejemplo, M puede ser "Juan-Pérez", C= "edad" y V= "38 años". De este modo, toda la información relativa a un objeto o a una persona queda reunida en un marco en vez de estar dispersa en un conjunto de proposiciones sin ninguna estructura, y esto significa mayor eficiencia de cara al razonamiento (más rapidez a la hora de buscar una inferencia concreta) y mayor facilidad para mantener la base de

conocimientos. Igualmente puede decirse que los marcos están más estructurados que las reglas.

Ahora bien, la cuestión que se debate es si los marcos aportan nuevas posibilidades de razonamiento frente a la lógica. Existen dos argumentos que apoyan una respuesta afirmativa. Uno de ellos consiste en el razonamiento por defecto, que queda fuera del alcance de la lógica clásica. El otro es la capacidad de los marcos (en la propuesta original) para reconocer entidades y situaciones; por ejemplo, cuando "algo" tiene cocina, cuarto de baño, salón y dormitorios, es una casa. Podríamos intentar expresar esta inferencia mediante una regla cuyo antecedente contuviera una conjunción de premisas. Sin embargo, los marcos son capaces de establecer que ese "algo" es una casa incluso cuando la información es incompleta. Este tipo de razonamiento aproximado es otra característica propia de los marcos que escapa a la lógica (al menos a la lógica clásica).

Más aún, Minsky [1975] afirma que ni siquiera es deseable que la IA se apoye en la lógica a la hora de abordar problemas del mundo real. Podríamos sintetizar sus argumentos en dos puntos. Por un lado, la lógica realiza en general muchas más inferencias de las deseadas, porque es incapaz de "conocer" cuáles son relevantes para el problema actual, y en consecuencia la explosión combinatoria puede hacer que el proceso de inferencia sea inútil en la práctica. Por otro lado, la lógica realiza solamente aquellas deducciones que pueden obtenerse con toda certeza a partir de los axiomas; esta propiedad, denomina consistencia (cap. 5), es considerada por los partidarios de la lógica como una de sus virtudes más importantes. Minsky, por el contrario, piensa que en la práctica es un defecto, pues en la vida real sólo disponemos de información incierta e incompleta, y necesitamos tomar decisiones aun corriendo el riesgo de equivocarnos. Limitar la inferencia a aquellas proposiciones que pueden conocerse con absoluta certeza significa que en la mayor parte de las situaciones del mundo real no podríamos llegar a ninguna conclusión.

Parece ser que en los últimos años se ha enfriado el debate entre los partidarios y los detractores de los distintos métodos. Hoy en día se reconoce más fácilmente que cada uno de ellos tiene sus cualidades y sus deficiencias. En cualquier caso, determinar hasta qué punto un mecanismo de representación de conocimiento debe apoyarse en los principios de la lógica, sigue siendo una cuestión abierta en la que es necesario seguir investigando.

8.5 BIBLIOGRAFÍA RECOMENDADA

El artículo de Minsky [1975], donde propone los marcos, el de Bobrow y Winograd [1977], sobre el lenguaje KRL, y el de Hayes [1979], en que critica los marcos y defiende la lógica, se encuentran recogidos en [Brachman & Levesque, 1985]. Todos los tratados generales sobre IA dedican algún capítulo a este tema, al igual que el libro sobre representación del conocimiento de Ringland y Duce [1986].

La referencia clásica sobre guiones es [Schank & Abelson, 1977]. Los *paquetes de organización de memoria* (MOPs) aparecen descritos en [Schank, 1980, 1982a, 1982b]. También puede ser interesante consultar en la "Encyclopedia of Artificial Intelligence" [Shapiro, 1992] los artículos "Scripts", "Dynamic memory", "Memory organization packets", "Argument comprehension" y "Reasoning, case-based", donde se discuten más ampliamente los temas tratados en este capítulo y se ofrecen numerosas referencias.