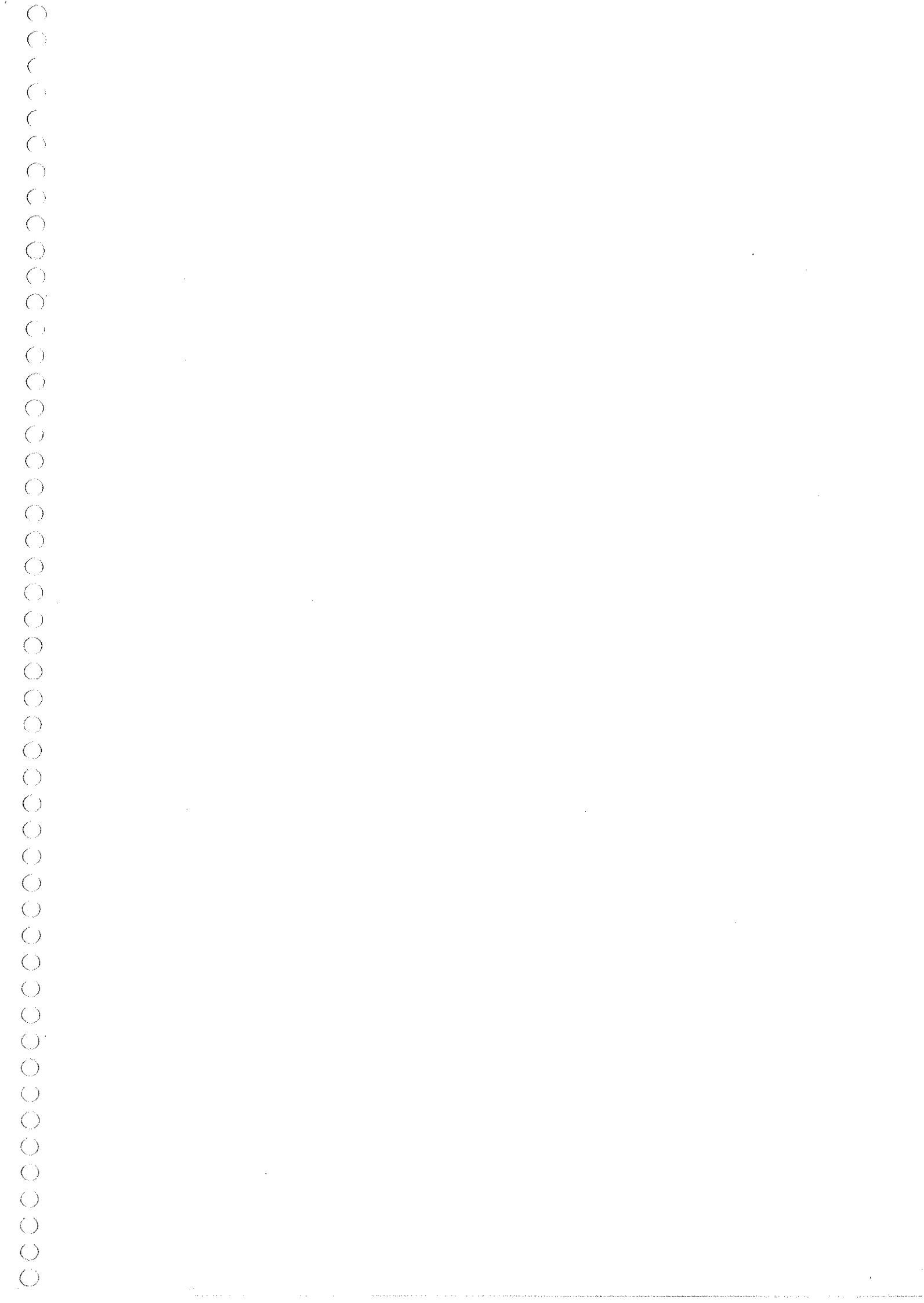




**INTELIGENCIA ARTIFICIAL**  
**Ing. Eduardo Amar**







## **UNIDAD DIDACTICA 1**

**Introducción a la  
Inteligencia Artificial**



**UNIDAD DIDACTICA 1****EJE CONCEPTUAL**

Introducción a la Inteligencia Artificial

**TEMAS**

<b>1.1. ¿Qué es la Inteligencia Artificial? .....</b>	<b>4</b>
1.1.1. Pueden .....	4
1.1.2. Máquinas .....	4
1.1.3. Pensar .....	6
<b>1.2. Aproximaciones a la IA .....</b>	<b>7</b>
1.2.1. Basadas en procesamiento de símbolos .....	7
1.2.2. Aproximaciones subsimbólicas .....	8
<b>1.3. Breve historia de la IA.....</b>	<b>8</b>
<b>1.4. Mundo especial para agentes con IA .....</b>	<b>11</b>
<b>1.5. Sistemas que resuelven problemas de la IA .....</b>	<b>13</b>
1.5.1. Acciones que debe llevar a cabo el sistema .....	19
1.5.2. Definición mediante búsqueda en espacio de estados .....	19
1.5.3. Sistemas de producción .....	22
1.5.4. Análisis del problema.....	23
1.5.5. Características de los Sistemas de producción .....	30

## 1.1. ¿Qué es la Inteligencia Artificial?

La **Inteligencia Artificial (IA)**, en una definición amplia y un tanto circular, tiene por objeto el estudio del comportamiento inteligente en las máquinas. A su vez, el comportamiento inteligente supone percibir, razonar, aprender, comunicarse y actuar en entornos complejos.

- Una de las metas a largo plazo de la IA es el desarrollo de máquinas que puedan hacer todas estas cosas igual, o quizás incluso mejor, que los humanos.
- Otra meta de la IA es llegar a comprender este tipo de comportamiento, sea en las máquinas, en los humanos o en otros animales.

Por tanto, la IA persigue al mismo tiempo metas científicas y metas de ingeniería.

La IA ha estado siempre rodeada de controversia. La cuestión básica de la IA **¿Pueden pensar las máquinas?** ha interesado tanto a filósofos como a científicos e ingenieros. En un famoso artículo, Alan Turing, uno de los fundadores de la informática, expresó esta misma cuestión, pero formulada en términos más adecuados para su comprobación empírica. Es lo que se ha dado en llamar el *Test de Turing* (Turing, 1950). Describiremos este test un poco más adelante, en esta misma sección, pero primero es importante destacar lo que ya observara Turing: que la respuesta a la pregunta **¿Pueden pensar las máquinas?** depende de cómo definamos las palabras *máquinas* y *pensar*. Turing podría haber añadido que la respuesta depende también de cómo se defina la palabra *pueden*.

### 1.1.1. Pueden

Consideremos primero la palabra *pueden*, ¿Queremos decir que las máquinas pueden pensar ya ahora, o que algún día podrán pensar? ¿Queremos decir que las máquinas podrían ser capaces de pensar, en principio (incluso aunque nunca lleguemos a construir ninguna que lo haga), o lo que perseguimos es una implementación real de una máquina pensante? Estas cuestiones son realmente importantes puesto que todavía no disponemos de ninguna máquina que posea amplias habilidades pensantes.

Algunas personas creen que las máquinas pensantes tendrían que ser tan complejas, y disponer de una experiencia tan compleja (por ejemplo, interaccionando con el entorno o con otras máquinas pensantes), que nunca seremos capaces de diseñarlas o construirlas. Una buena analogía nos la proporcionan *los procesos que regulan el clima global del planeta*: incluso aunque conociésemos todo lo que es importante acerca de estos procesos, este conocimiento no nos capacitaría necesariamente para duplicar el clima en toda su riqueza. Ningún sistema menos complejo que el formado por la superficie de la tierra, la atmósfera y los océanos – embebido en un espacio interplanetario, calentado por el sol e influenciado por las mareas – sería capaz de duplicar los fenómenos climáticos con todo detalle.

De forma similar, la *inteligencia de nivel humano*, a escala real, podría ser *demasiado compleja*, o al menos demasiado dependiente de la fisiología humana, para existir fuera de su encarnación en seres humanos inmersos en su entorno. La cuestión de si alguna vez seremos capaces, o no, de construir máquinas pensantes de nivel humano no admite aún una respuesta definitiva. El progreso de la IA hacia esta meta ha sido constante, aunque más lento de lo que algunos pioneros del tema habían predicho. Personalmente, soy optimista sobre nuestro eventual éxito en esta empresa.

### 1.1.2. Máquinas

Consideremos ahora la palabra *máquina*. Para mucha gente, una máquina es todavía un artefacto más bien estúpido. La palabra evoca imágenes de engranajes rechinando, de chorros de vapor siseando y de piezas de acero martilleando. ¿Cómo podría llegar a pensar una cosa como esa? Sin embargo, hoy en día, los ordenadores han ampliado en gran medida nuestra noción de lo que una máquina puede ser, y nuestra creciente comprensión de los mecanismos biológicos está expandiéndose incluso más aún.

Consideremos, por ejemplo, un *virus simple*, como el denominado *Bacteriófago E6*, mostrado esquemáticamente en la Figura 1.1. Su cabeza contiene ADN vírico. Este virus es capaz de adherirse a la pared celular de una bacteria mediante las fibras de su cola, pinchar la pared e inyectar su ADN en ella. Este ADN hace que la bacteria fabrique millares de copias de cada una de las piezas del virus. Después, las piezas se ensamblan automáticamente ellas mismas,

formando nuevos virus que salen de la bacteria para repetir el proceso. El ensamblaje completo se parece mucho al de una máquina, por lo que podríamos, con toda propiedad, decir que se trata de una máquina – *una máquina hecha de proteínas*.

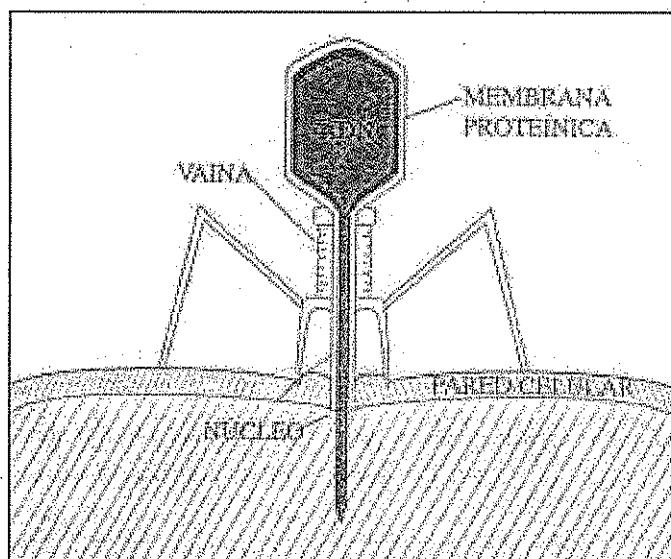


Figura 1.1

Pero ¿qué ocurre con otros procesos y organismos biológicos? El genoma completo de la bacteria *Haemophilus influenzae Rd* ha sido secuenciado recientemente (Fleischmann, 1995). Este genoma consta de 1.830.137 pares de bases (identificadas con las letras A, G, C y T). Esto equivale, aproximadamente, a  $3,6 \times 10^6$  bits, es decir, casi medio megabyte. Aunque todavía no se conoce la función de todos y cada uno de sus 1.743 genes, los científicos están comenzando a explicar el desarrollo y el funcionamiento de este organismo en los mismos términos en los que explicarían una máquina, – una máquina muy compleja, desde luego. De hecho, hay técnicas que son muy familiares a los informáticos, tales como el uso de cronogramas para los circuitos lógicos, que están demostrando ser útiles para entender como los genes regulan los complejos procesos bioquímicos de un virus que infecta a bacterias (McAdams y Shapiro, 1995).

La secuenciación de los genomas completos de otros organismos, incluyendo el *genoma humano*, está en curso o completada. Una vez que conozcamos estas "marcas", ¿pensaríamos en estos organismos – bacterias, gusanos, moscas, ratones, delfines, humanos – como máquinas? Si los humanos fuesen máquinas, entonces es que las máquinas pueden pensar! Tendríamos una demostración de su existencia. Lo que ocurre, "simplemente", es que *no conocemos aún como funciona la máquina humana*.

Sin embargo, aunque estuviésemos de acuerdo acerca de lo que es una máquina, este último argumento es rebatible. Aunque una máquina hecha de proteínas puede pensar, quizás una hecha de silicio no sería capaz de hacerlo. Un conocido filósofo, John Searle, cree que la materia de la que estamos hechos es fundamental para la inteligencia (Searle, 1980; Searle, 1992). Para él, el pensamiento solo puede ocurrir en máquinas muy especiales – *las máquinas vivientes hechas de proteínas*.

La *hipótesis del sistema físico de símbolos* de Newell y Simon (Newell y Simon, 1976) está en oposición directa a las creencias de Searle (y a la noción de encarnación mencionada anteriormente). Esta hipótesis establece que *un sistema físico de símbolos dispone de los medios necesarios y suficientes para desarrollar una actividad general inteligente*. De acuerdo con Newell y Simon, un sistema físico de símbolos es una máquina, tal como un ordenador digital, que es capaz de manipular datos simbólicos – sumar números, reordenar listas de símbolos (por ejemplo, ordenar alfabéticamente una lista de nombres), reemplazar algunos símbolos por otros, etc. Un aspecto importante de esta hipótesis es que no importa de que esté hecho el sistema físico de símbolos. Decimos que la hipótesis de Newell y Simon es "neutral respecto al sustrato".

*Una entidad inteligente podría estar hecha de proteínas, de relés mecánicos, de transistores, o de cualquier otra cosa, con tal de que sea capaz de procesar símbolos.*

Otros pensadores creen que no es realmente importante que las máquinas estén hechas de silicio o de proteínas; piensan que el comportamiento inteligente es, en su mayor parte, el resultado de lo que ellos llaman *procesamiento subsimbólico*, es decir, *procesamiento de señales*, no de símbolos. Consideremos, por ejemplo, el reconocimiento de rostros familiares. Los humanos hacemos esto sin esfuerzo alguno, y aunque no se sabe exactamente como lo hacemos, se sospecha que la mejor explicación para el proceso es la que se basa en el tratamiento de imágenes, o de partes de ellas, como señales multidimensionales, no como símbolos.

Podríamos citar muchos otros puntos de vista sobre el tipo de máquinas que podrían ser capaces de desarrollar habilidades pensantes como las de los humanos. Algunas de las hipótesis que se oyen a menudo son las siguientes:

- El cerebro *procesa la información en paralelo*, mientras que los ordenadores convencionales lo hacen *secuencialmente*. Por tanto, tenemos que construir nuevas variedades de ordenadores paralelos para realizar progresos en la IA.
- La maquinaria computacional convencional está basada en la *lógica binaria* (verdadero o falso). Por tanto, los sistemas realmente inteligentes tendrán que usar algún tipo de *lógica borrosa*.
- Las neuronas animales son mucho más complejas que simples conmutadores – los bloques constructivos básicos de los ordenadores actuales -. Por tanto, necesitaremos *neuronas artificiales verdaderamente realistas* para construir máquinas inteligentes.

Quizá es aún demasiado pronto para que se alcance un consenso en el campo de la IA sobre el tipo de maquinaria requerida, aunque muchos investigadores aceptan ya la hipótesis del sistema físico de símbolos.

### 1.1.3. Pensar

Finalmente, llegamos a la palabra más difícil: *pensar*. En lugar de intentar definir esta palabra, Turing propuso un test, el llamado *Test de Turing*, mediante el cual pudiera decidirse si una máquina particular es o no inteligente. El test fue descripto originalmente como un juego. Citemos del artículo de Turing (Turing, 1950):

En el juego participan tres personas, un hombre (A), una mujer (B) y un interrogador (C), que puede ser de cualquier sexo. El interrogador permanece en una sala, separado de los otros dos, pero pudiendo comunicarse con ellos mediante un teletipo. El objetivo del juego para el interrogador es determinar cual de los otros dos es el hombre y cual es la mujer. El interrogador los designa mediante las etiquetas X e Y, y al final del juego debe decir "X es A e Y es B" o "X es B e Y es A". Para ello, el interrogador puede plantear preguntas a A y a B, tales como: C: ¿Podría decirme X cual es la longitud de su pelo?

Supongamos que X es realmente A; entonces es A quien debe responder. El objetivo de A en el juego es intentar que C haga una identificación errónea.

El objetivo del juego para el tercer participante (B) es ayudar al interrogador.

Ahora podemos plantearnos la siguiente cuestión: "¿Qué sucedería si una máquina interpretase el papel de A en el juego?" ¿El interrogador hará tantas identificaciones erróneas como cuando el juego es interpretado por un hombre y una mujer? Estas cuestiones reemplazan a nuestra cuestión original: "¿Pueden pensar las máquinas?"

A menudo, el Test de Turing se simplifica, planteándolo como un juego en el que *una máquina intenta convencer a un interrogador humano de que ella es también humana*. Esta versión simplificada del Test de Turing no se considera usualmente como un test de inteligencia muy útil, debido a que es posible, incluso para programas muy simples, engañar al interrogador humano durante un buen rato. Por ejemplo, el programa ELIZA de Joseph Weizenbaum (Prog.

DOCTOR.PRO) usa algunos trucos muy simples para ello, pero es capaz de desarrollar un diálogo que resulta aparentemente realista para el interrogador poco avezado, aunque es un diálogo completamente vano (Weizenbaum, 1965). El programa JULIA de Mauldin es también un programa de diálogo de este tipo, pero más reciente y sofisticado (Mauldin, 1994).

Aparte del Test de Turing, una cuestión que merece la pena intentar responder es la de cuales deben ser las habilidades requeridas en una máquina para que podamos calificarla como inteligente. Actualmente, existen muchos programas de ordenador capaces de realizar cosas realmente prodigiosas, incluyendo la planificación óptima de rutas aéreas para economizar combustible, la simulación global de condiciones meteorológicas, la planificación del uso de recursos en una factoría, etc. ¿Son inteligentes estos programas? ¿Deben ser considerados como objeto de estudio de la IA? Habíamos comenzado este capítulo describiendo máquinas que difícilmente podrían ser etiquetadas como inteligentes. ¿Son gradualmente más y más inteligentes conforme se incrementa su complejidad? Personalmente, lo creo así, pero existen indudablemente opiniones para todos los gustos.

## 1.2. Aproximaciones a la IA

Aún aceptando que la IA ya ha sido capaz de producir algunos sistemas prácticos muy útiles, la creencia general es que el objetivo último de alcanzar una inteligencia de nivel humano está aún muy distante. Siendo esto así, todavía hay un gran debate sobre cuales son las *mejores aproximaciones hacia la IA* – mejores en el sentido de sentar los fundamentos centrales para conseguir las metas planteadas a largo plazo, así como mejores en el sentido de producir resultados prácticos a corto plazo – . En consecuencia, durante los últimos 40 años ha emergido un cierto número de paradigmas diferentes. Cada uno de ellos tiene abogados apasionados, y algunos han producido resultados suficientemente interesantes como para no ser descartados sin más. Quizá lo que se requiere es una combinación apropiada de estas aproximaciones diferentes. En cualquier caso, cada investigador cree, a menudo, que la suya es la metodología avanzada que merece especial atención. Los principales paradigmas pueden ser clasificados en dos grupos.

### 1.2.1. Basadas en procesamiento de símbolos

El primer grupo incluye lo que llamaríamos las **aproximaciones basadas en procesamiento de símbolos**. Se sustentan sobre la hipótesis del **sistema físico de símbolos de Newell y Simon**, y aunque esta hipótesis no puede considerarse aún como universalmente aceptada, en ella se basa mucho de lo que podríamos llamar IA "clásica" (lo que el filósofo John Haugeland llama "IA al viejo estilo"). Un miembro destacado de esta familia de aproximaciones es el que se basa en la aplicación de operaciones lógicas sobre bases de conocimiento declarativo. Inspirado originalmente en los informes de John McCarthy sobre su "sistema consejero" (McCarthy, 1958), este estilo de IA representa el "conocimiento" sobre un problema del dominio mediante sentencias declarativas, a menudo basadas en sentencias de la lógica de predicados o sustancialmente equivalentes a ellas. Para deducir consecuencias a partir de este conocimiento se aplican técnicas de inferencia lógica. Este método admite numerosas variantes, incluyendo aquellas cuyo énfasis está en la axiomatización formal del dominio en un lenguaje lógico. Cuando se aplica a problemas "reales", este método requiere la representación de una cantidad sustancial de conocimiento del dominio, por lo que se suele hablar de aproximaciones basadas en el conocimiento. Se han construido muchos sistemas basados en estos métodos y nos referiremos a algunos de ellos más adelante.

En muchas de las aproximaciones basadas en procesamiento de símbolos, el análisis de los comportamientos deseados, o la síntesis de máquinas para conseguirlos, se extienden a través de varios niveles.

- El nivel superior corresponde al *nivel del conocimiento* (Newell, 1982), en el cual se especifica el conocimiento necesario para que la máquina alcance sus objetivos.
- A continuación viene el *nivel simbólico*, donde se representa este conocimiento mediante estructuras simbólicas, como, por ejemplo, listas escritas en el lenguaje de programación LISP, y se especifican operaciones sobre estas estructuras.
- Después están los niveles inferiores, en los cuales, realmente se implementan las *operaciones de procesamiento de símbolos*. Muchas aproximaciones basadas en

procesamiento de símbolos utilizan una metodología de diseño "descendente"; se comienza en el nivel de conocimiento y se procede hacia abajo a través de los niveles simbólico y de implementación.

### 1.2.2. Aproximaciones subsimbólicas

El segundo grupo de aproximaciones hacia la IA incluye lo que se denominan **aproximaciones subsimbólicas**. Estas siguen usualmente un estilo de *diseño "ascendente"*, comenzando en el nivel más bajo y procediendo hacia los niveles superiores.

En los niveles más bajos, el concepto de símbolo no es tan apropiado como el concepto de señal. Entre las aproximaciones subsimbólicas, una aproximación muy prominente es la que algunos han llamado "*vida artificial*". Los defensores de este estilo (Wilson, 1991 y Brooks, 1990) señalan que la inteligencia humana se desarrolló solo después de más de mil millones de años de vida sobre la tierra. Según ellos, para conseguir máquinas inteligentes tendremos que seguir muchos de estos pasos evolutivos. Primero, debemos concentrarnos en la duplicación de las capacidades de procesamiento de señal y control de las que disponen los animales más simples – los insectos, por ejemplo – y subir por la escalera evolutiva en pasos sucesivos. Esta estrategia no solo conducirá a la obtención a corto plazo de máquinas útiles, sino que desarrollará el substrato sobre el cual deben construirse necesariamente los niveles superiores de inteligencia.

Este segundo grupo de aproximaciones también pone énfasis en los fundamentos simbólicos. Brooks introdujo la hipótesis de los fundamentos físicos, en contraste con la hipótesis de los sistemas físicos de símbolos (Brooks, 1990). Según su hipótesis, se puede obtener un comportamiento complejo sin usar modelos centralizados; para ello, bastaría con dejar que los diversos módulos de comportamiento de un agente interactúen independientemente con el entorno. Sin embargo, Brooks acepta que para conseguir IA de nivel humano puede ser necesaria la integración de las dos aproximaciones.

La interacción entre una máquina y su entorno conduce a lo que se denomina comportamiento emergente. En palabras de una investigadora (Maes, 1990):

La funcionalidad de un agente debe verse como una propiedad emergente de la interacción intensiva del sistema con su entorno dinámico. La especificación del comportamiento del agente aislado no explica la funcionalidad que exhibe cuando el agente está operando. Por el contrario, su funcionalidad se basa en gran medida en las propiedades del entorno. No solo hay que tener en cuenta las características dinámicas del entorno sino que éstas deben ser explotadas para servir al funcionamiento del sistema.

Las *redes neuronales* son un ejemplo bien conocido de máquinas que provienen de la *escuela subsimbólica*. Estos sistemas, inspirados en modelos biológicos, son interesantes, principalmente por su capacidad de aprendizaje.

También se han conseguido resultados interesantes mediante *procesos que simulan ciertos aspectos de la evolución biológica*: cruzamiento, mutación y reproducción de los organismos mejor adaptados.

Otras aproximaciones ascendentes, cercanas a las aproximaciones del tipo "*vida artificial*", se basan en la *teoría del control* y en el *análisis de sistemas dinámicos*.

A medio camino entre las **aproximaciones** descendentes y ascendentes está la **basada en autómatas situados** (Kaelbling y Rosenschein, 1990). Kaelbling y Rosenschein proponen un lenguaje de programación para especificar, en un alto nivel de abstracción, el comportamiento deseado en un agente, así como un compilador para crear la circuitería necesaria a partir de los programas escritos en este lenguaje.

### 1.3. Breve historia de la IA

Cuando empezaron a desarrollarse los primeros ordenadores durante las décadas de los años 40 y 50, algunos investigadores escribieron programas que podían realizar tareas elementales

de razonamiento. Entre los resultados más prominentes de esta época podemos citar los primeros programas de ordenador capaces de jugar al ajedrez (Shannon, 1950; Newell, Shaw y Simon, 1958), los programas capaces de jugar a las damas (Samuel, 1959; Samuel, 1967) y los programas para demostrar teoremas de geometría plana (Gelernter, 1959). En 1956, John McCarthy y Claud Shannon coeditaron un volumen titulado Automata Studies (Shannon y McCarthy, 1956). La mayoría de los artículos del volumen trataban sobre los fundamentos matemáticos de la teoría de autómatas, por lo que McCarthy, decepcionado, decidió acuñar el término **Inteligencia Artificial** y usarlo como título de una conferencia celebrada en Darmouth en 1956. En esta conferencia se presentaron algunos trabajos muy relevantes, incluyendo uno de Allen Newell, Cliff Shaw y Herbert Simon sobre un programa llamado Teorizador Lógico (Newell, Shaw y Simon, 1957), que podía demostrar teoremas en lógica proposicional. Aunque se propusieron muchos otros términos para el campo, tales como procesamiento de información compleja, inteligencia de máquinas, programación heurística o cognología, solo el nombre de Inteligencia Artificial ha perdurado, sin duda a causa de la creciente progresión de libros de texto, cursos, congresos y revistas que usaban este término.

El primer paso hacia la inteligencia artificial fue dado mucho tiempo atrás por Aristóteles (384-322 a. C.), cuando comenzó a explicar y a codificar ciertos estilos de razonamiento deductivo que él llamó *silogismos*. Algunos de los esfuerzos tempranos para automatizar la inteligencia nos parecerían quijotescos hoy en día. Ramón LLull (1235-1316), un místico y poeta catalán, construyó una *máquina de engranajes*, llamada *Ars Magna*, que supuestamente era capaz de responder a todas las preguntas. Pero hubieron también científicos y matemáticos que perseguían la automatización del razonamiento. Martin Gardner (Gardner, 1982) atribuye a Gottfried Leibniz (1646-1716) el sueño de un "álgebra universal mediante la cual todo el conocimiento, incluyendo las verdades morales y metafísicas, pueda ser algún día representado en un único sistema deductivo". Leibniz llamó a su sistema *cálculo filosófico o raciocinador*, fue, claro está, un sueño que no pudo ser realizado con el aparataje tecnológico de la época. No comenzó a haber un progreso sustancial hasta que George Boole (Boole, 1854) desarrolló los fundamentos de la *lógica proposicional*. El propósito de Boole, entre otras cosas, era "recoger... algunos fundamentos probables relativos a la naturaleza y a la constitución de la mente humana". Hacia el final del siglo XIX, Gottlieb Frege propuso un sistema de notación para el razonamiento mecánico, con lo que inventó mucho de lo que hoy conocemos con el nombre de *cálculo de predicados* (Frege, 1879). Llamó a su lenguaje *Begriffsschrift*, lo que puede ser traducido como "escritura de conceptos".

En 1958, John McCarthy propuso la utilización del cálculo de predicados como un lenguaje para representar y usar conocimiento en un sistema al que llamó "*sistema consejero*" (McCarthy, 1958). A este sistema, en lugar de programarlo, había que decirle lo que necesitaba saber para resolver un problema. Una modesta, pero influyente, implementación de estas ideas fue abordada por Cordell Green en su sistema QA3 (Green, 1969). Como resultado de muchas controversias entre los investigadores de la IA, el cálculo de predicados y sus variantes han sobrevivido como el fundamento básico para la representación del conocimiento.

Los lógicos del siglo XX, incluyendo a Kurt Gödel, Stephen Kleene, Emil Post, Alonzo Church y Alan Turing, formalizaron y clarificaron lo que puede ser hecho y lo que no puede ser hecho mediante sistemas lógicos y computacionales. Posteriormente, informáticos como Stephen Cook y Richard Karp identificaron las clases de cómputos que, siendo posibles en principio, requerirían cantidades de tiempo y de memoria completamente impracticables.

Muchos de estos resultados de la lógica y de la informática se referían a "verdades que no pueden ser deducidas" y a "cálculos que no pueden ser realizados". Seguramente animados por estos hallazgos negativos, algunos filósofos y científicos (Lucas, 1961; Penrose, 1989) los interpretaron como confirmaciones de que la inteligencia humana no podría ser nunca mecanizada. Estos pensadores creían que los humanos son, de alguna forma, inmunes a las limitaciones computacionales inherentes a las máquinas. Sin embargo, la mayoría de los lógicos y de los informáticos creen que estos resultados negativos de ningún modo implican que las máquinas tengan límites que no sean aplicables también a los humanos.

El primer artículo moderno que trataba sobre la *posibilidad de mecanizar la inteligencia al estilo humano* fue el de Alan Turing que ya hemos citado anteriormente (Turing, 1950). Durante el mismo período, Warren McCulloch y Walter Pitts teorizaban sobre las *relaciones entre elementos computacionales simples y neuronas biológicas* (McCulloch y Pitts, 1943).

Demostraron que es posible calcular cualquier función computable mediante redes de puertas lógicas. Otro trabajo, de Frank Rosenblatt (Rosenblatt, 1962), exploraba el uso de *redes de tipo neuronal, denominadas perceptrones*, para el aprendizaje y el reconocimiento de patrones. Algunas otras corrientes de trabajo, entre ellas la cibernetica (Wiener, 1948), la psicología cognitiva, la lingüística computacional (Chomsky, 1965) y la teoría del control adaptativo (Widrow y Of., 1960), han contribuido también a esta matriz intelectual dentro de la cual se ha desarrollado la IA.

Una gran parte del trabajo inicial en la IA (durante la década de los años 60 y la primera parte de la década de los 70) se dedicaba a explorar diversas representaciones de problemas, técnicas de búsqueda y heurísticas generales que se aplicaban en programas de ordenador capaces de resolver puzzles sencillos, de jugar contra el usuario o de recuperar información. Uno de los programas más influyentes fue el *Solucionador General de Problemas* (GPS, o General Problem Solver) de Allen Newell, Cliff Shaw y Herbert Simon. Entre los problemas de muestra resuelto por estos sistemas pioneros se incluían la *integración simbólica* (Slagle, 1963), los problemas de álgebra (Bobrow, 1968), los puzzles análogicos (Evans, 1968) y el control de robots móviles (Nilsson, 1984). Muchos de estos sistemas constituyen el tema central de los artículos reunidos en el histórico volumen *Computer and Thought* (Feigenbaum y Feldman, 1963).

Los intentos de "escalar" estos programas y sus técnicas para enfrentarlos a aplicaciones de importancia práctica revelaron que solo valían para resolver "problemas de juguete". La construcción de sistemas más potentes requería la inclusión de mucho más conocimiento sobre el dominio de aplicación. Los últimos años de la década de los 70 y los primeros de la década de los 80 vieron el desarrollo de programas más realistas, que contenían el conocimiento necesario para mimetizar el comportamiento de los expertos humanos en tareas tales como el diagnóstico, el diseño y el análisis. Fueron explorados y desarrollados varios métodos para la representación de conocimiento específico del problema. El programa al que se atribuye el mérito de ser el primero que demostró la importancia de recoger grandes cantidades de conocimiento específico del dominio fue *DENDRAL*, un sistema para predecir la estructura de moléculas orgánicas a partir de su fórmula química y de su espectrograma de masas. (Feigenbaum, Buchanan y Lederberg, 1971) (Lindsay, 1980). Despues se desarrollaron otros "sistemas expertos", incluyendo sistemas para diagnóstico médico (Shortliffe, 1976 y Millar, Pople y Myers, 1982), sistemas para configurar ordenadores (McDermott, 1982) y sistemas para valorar posibles yacimientos de minerales (Campbell, 1982 y Duda, Gaschnig y Hart, 1979). Un buen resumen de la historia de la IA a lo largo de este período fue escrito por (McCorduck, 1979).

Una de las áreas en la que se han realizado progresos sustanciales al escalar el tamaño del problema es el área de los juegos. El 11 de mayo de 1997, un programa de ordenador de IBM, denominado *DEEP BLUE*, consiguió vencer al entonces campeón del mundo de ajedrez, Garry Kasparov, por 3.5 a 2.5 en un encuentro a seis partidas. Este alto nivel de juego se ha conseguido gracias a la sinergia de sofisticados algoritmos de búsqueda, ordenadores de alta velocidad y hardware específico para el juego del ajedrez.

La inteligencia humana abarca muchas habilidades, incluyendo la habilidad para percibir y analizar escenas visuales y la habilidad para entender o generar el lenguaje. Estos son temas específicos que han recibido mucha atención. Larry Roberts desarrolló uno de los primeros programas de análisis de escenas (Roberts, 1963). Este trabajo preliminar fue seguido de una extensa cantidad de investigación en visión artificial (Nalga, 1993) es un buen libro de texto general, guiada mediante estudios científicos de los sistemas de visión animal (Letvin, 1959; Hubel, 1988 y Marr, 1982).

Uno de los sistemas pioneros en comprensión del lenguaje natural fue el desarrollado por Ferry Winograd (Winograd, 1972). Durante la década de los años 70 se llevó a cabo un proyecto coordinado multicentro que desarrolló prototipos de sistemas para la comprensión fluida del habla; el sistema LUNAR (Woods, 1973), desarrollado por William Woods, era capaz de responder a preguntas orales en inglés sobre las muestras de rocas recogidas por las misiones lunares de la NASA.

Aunque actualmente existen diversos sistemas de comprensión del lenguaje natural, su competencia se restringe a áreas temáticas específicas y a vocabularios especializados. El

desarrollo de sistemas con un alcance más amplio requerirá de nuevos avances en la representación de grandes cantidades de conocimiento general de sentido común. El proyecto CYC (Guha y Lenat, 1990; Guha y Lenat, 1995) tiene como uno de sus objetivos la recogida y representación de conocimiento de este tipo.

Aunque el interés en las redes neuronales decayó un poco tras el trabajo pionero realizado en los últimos años de la década de los 50 por Frank Rosenblatt, resurgió con energía en los años 80. Las redes de elementos no lineales con interconexiones de pesos variables se consideran actualmente como una clase muy importante de herramientas para el modelado no lineal. Hoy en día existen diversas aplicaciones importantes de las redes neuronales. El trabajo sobre redes neuronales, junto con el trabajo en temas de vida artificial, ha ayudado a focalizar la investigación actual en IA sobre los problemas relacionados con la conexión entre procesos simbólicos y los sensores y efectores de los robots inmersos en un entorno físico.

Si proyectamos las tendencias actuales hacia el futuro, es razonable esperar un nuevo énfasis en el desarrollo de sistemas autónomos integrados, robots y "softbots". Los softbots (Etzioni y Weld, 1994) son agentes software que recorren Internet buscando la información que ellos creen que puede ser de interés para sus usuarios. La constante presión ejercida para mejorar las capacidades de los robots y de los agentes software motivará y guiará la investigación en inteligencia artificial durante muchos años.

#### 1.4. Mundo especial para agentes con IA

La investigación en IA ha conducido al desarrollo de numerosas ideas y técnicas relevantes dirigidas al objetivo de automatizar la inteligencia. Estas ideas y técnicas serán descriptas en el contexto de una serie de "**agentes**" cada vez más capaces y complejos. Hay muchos tipos de agentes y entornos que podríamos considerar.

Por ejemplo, podemos imaginar robots que trabajen en las condiciones de gravedad cero del espacio exterior, en las aguas oscuras de los fondos oceánicos, en edificios de oficinas o en factorías, o en el mundo simbólico de Internet. Pero estos agentes prácticos del "*mundo real*" son a veces demasiados complejos para ilustrar de forma transparente los conceptos de la IA en los que se basa su inteligencia.

En lugar, lo que haremos será introducir una serie de *agentes "de juguete"* en un entorno ficticio al que llamaremos "*mundo espacial cuadriculado*". Aunque este mundo simple es fácil de describir, podemos añadirle cierta complejidad para hacerlo suficientemente rico como para que requiera "inteligencia" en sus agentes.

El *mundo espacial cuadriculado* es un espacio tridimensional demarcado por una matriz bidimensional de celdas a la que llamaremos "*suelo*". Cada celda puede contener objetos que tienen diversas propiedades, y pueden existir "*paredes*" delimitando conjuntos de celdas. Los agentes están confinados en el suelo y pueden moverse de celda a celda. Los objetos pueden estar en el suelo o colocados sobre una pila de otros objetos apilados sobre el suelo. A veces, usaremos solo el subespacio bidimensional definido por el suelo. En la Figura 1.2 se muestra un ejemplo típico de mundo espacial cuadriculado, en el que hay dos robots. Uno de ellos es un robot bidimensional sencillo, con sensores para percibir si las celdas adyacentes están o no libres para moverse a ellas; el otro, más complejo, dispone de un brazo con el que puede manipular objetos.

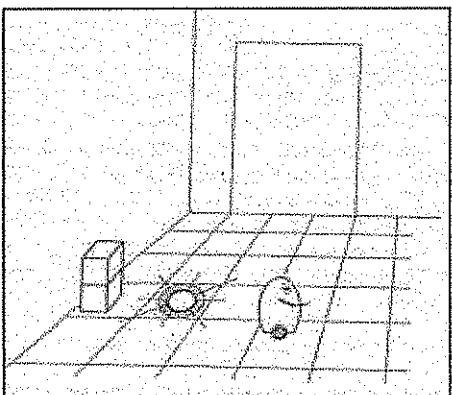


Figura 1.2

Los lectores ya familiarizados con la bibliografía sobre IA reconocerán fácilmente que el mundo espacial cuadriculado puede ser particularizado a algunos de los muchos "*mundos*" utilizados en la investigación sobre IA, incluyendo el mundo de bloques, el mundo del mosaico (Pollack y Ringuette, 1990), el mundo de los wumpus (Russell y Norvig, 1995) y el mundo de las hormigas (Koza, 1992). Todos ellos son mundos discretos, en el sentido de que existe un número finito de posiciones, de agentes, de objetos y de instantes de tiempo. Muchas de las técnicas de la IA que describiremos se aplican a mundos discretos y requerirían un cierto procesamiento subsimbólico para operar en mundos continuos.

## 1.5. Sistemas que resuelven problemas de la IA

### ¿Qué es la Inteligencia Artificial?

La Inteligencia Artificial (IA) estudia como lograr que las máquinas realicen tareas que, por el momento, son realizadas mejor por los seres humanos. Esta definición es, por supuesto, bastante efímera ya que hace referencia al estado actual de la informática. Además falla al no incluir algunas áreas que potencialmente tienen un gran impacto, tales como aquellos problemas que no pueden ser resueltos adecuadamente ni por las máquinas ni por los hombres.

### Los problemas de la Inteligencia Artificial

¿Cuáles son entonces los problemas de los que se ocupa la IA? La mayoría de los primeros trabajos en este campo hicieron gran hincapié en las tareas formales, como juegos y demostración de teoremas. Samuel escribió un programa de juego de damas que no solo jugaba partidas contra un oponente, sino que además utilizaba la experiencia adquirida en las partidas para mejorar su rendimiento. El ajedrez también suscitó un gran interés. La lógica teórica fue el primer intento de demostrar los teoremas matemáticos; con ella se pudo demostrar algunos teoremas que aparecen en el primer capítulo de los Principia Mathematica de Whitehead y Russell. El demostrador de teoremas de Gelernter exploró otra área de las matemáticas: la geometría. Los juegos y la demostración de teoremas comparten la propiedad de que son tareas en las que se considera que es necesaria la inteligencia para desarrollarlas.

Otra primera incursión dentro de la IA se centró en la clase de problemas que aparecen a diario – como cuando decidimos como llegar al trabajo por la mañana – con frecuencia denominados de sentido común. Estos problemas incluyen el razonamiento sobre objetos físicos y sus relaciones (por ejemplo, un objeto solo puede estar en un lugar a la vez), como también razonamiento sobre acciones y sus consecuencias (por ejemplo, si se deja caer algo, chocará contra el suelo y posiblemente se romperá). Para estudiar este tipo de razonamientos, Newell, Shaw y Simon construyeron el Resolutor General de Problemas, el cual se aplicó tanto a variadas tareas de sentido común como al problema de realizar manipulaciones simbólicas en expresiones lógicas.

Conforme las investigaciones en IA progresaron y fueron desarrollándose técnicas de manipulación de grandes cantidades de conocimiento sobre el mundo, se realizaron algunos avances en las tareas descriptas y aparecieron nuevas áreas de investigación. Estas áreas incluyen la percepción (visión y habla), comprensión del lenguaje natural y resolución de problemas en campos especializados como diagnósticos médicos y análisis químico.

Las tareas de percepción son difíciles ya que incluyen señales analógicas (previas a las digitales); estas señales suelen contener bastante ruido aunque normalmente se percibe a la vez una gran cantidad de objetos (algunos de los cuales pueden estar parcialmente tapados por otros).

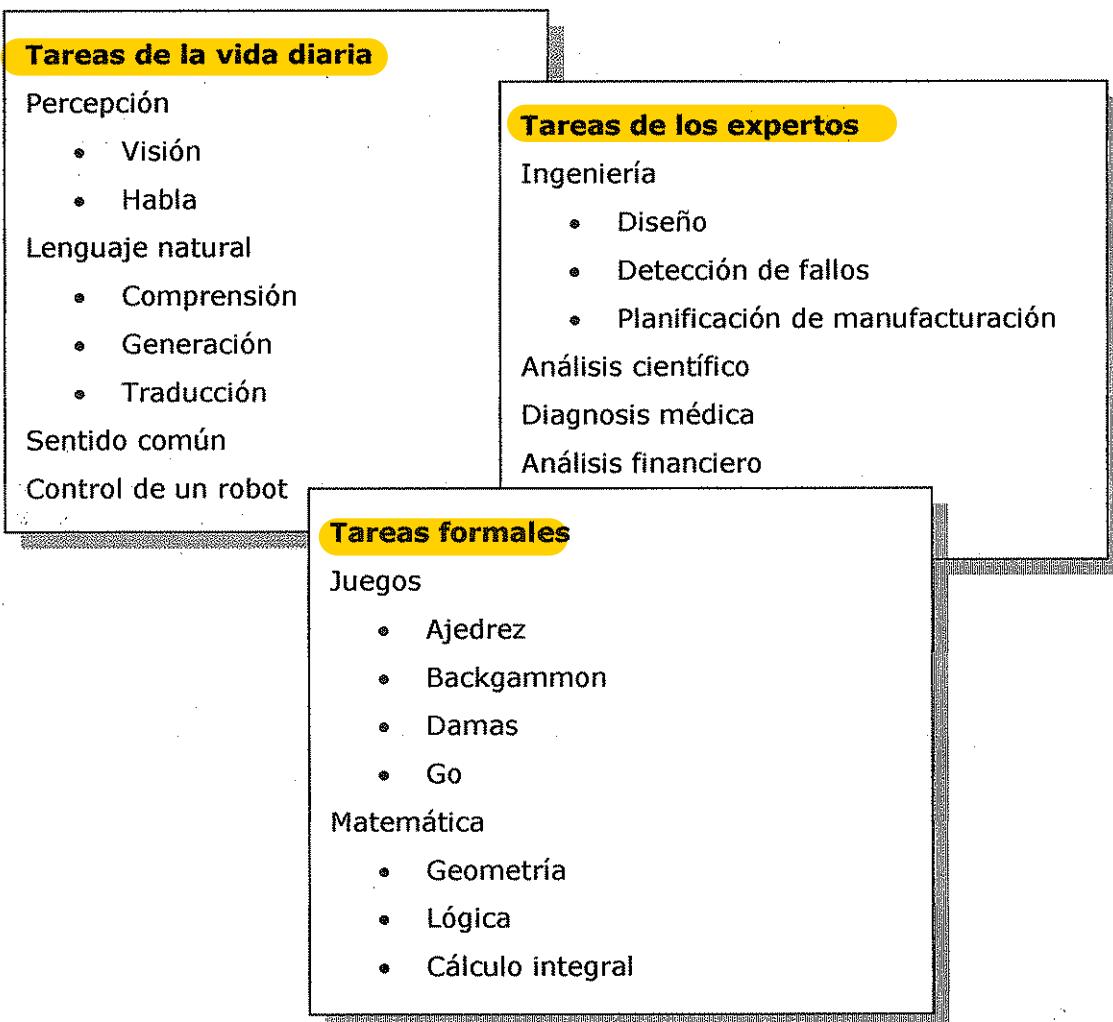
La habilidad de utilizar un lenguaje para comunicar gran variedad de ideas es quizá el aspecto más importante que separa a los humanos del resto de los animales. La comprensión del lenguaje hablado es un problema de percepción difícil de resolver por las razones ya explicadas. Es posible, sin embargo, restringir el problema al lenguaje escrito. Este problema, normalmente denominado comprensión del lenguaje natural, es aún extremadamente difícil. Para poder comprender frases sobre un cierto tema, es necesario no solo poseer un conocimiento amplio sobre el propio lenguaje (vocabulario y gramática), sino también manejar el suficiente conocimiento sobre dicho tema para reconocer las suposiciones no expresadas en el texto.

Además de estas tareas de la vida diaria, mucha gente puede también realizar tareas más especializadas en las cuales es necesaria una cuidada adquisición de experiencia. Ejemplos de lo anterior son tareas como el diseño en ingeniería, los descubrimientos científicos, los diagnósticos médicos y la planificación financiera. Los programas que pueden resolver problemas sobre estos dominios también están bajo la tutela de la inteligencia artificial.

Si bien las habilidades de un experto necesitan un conocimiento que la mayoría no poseemos, con frecuencia es mucho menor que el conocimiento necesario en las tareas más comunes, y con frecuencia más fácil de representar y tratar en los programas.

Como consecuencia, las áreas donde la IA está prosperando como una disciplina práctica (en oposición a una puramente de investigación) es precisamente en los dominios donde es necesario únicamente un conocimiento experto sin la ayuda de sentido común. Existen en la actualidad miles de programas llamados sistemas expertos utilizados diariamente en cualquier área de la industria y la administración. Cada uno de estos sistemas intenta resolver parte, o quizás todos los problemas de cierta entidad que antes necesitaban gran cantidad de conocimiento técnico humano.

### Algunas aplicaciones de la Inteligencia Artificial



¿Cuáles son nuestras suposiciones fundamentales sobre la inteligencia?

¿Qué tipo de técnicas son las más adecuadas para resolver los problemas de IA?

¿A qué nivel de detalle, si es que no es por completo, se puede intentar modelar la inteligencia humana?

¿Cómo se puede saber cuándo se ha tenido éxito en la construcción de un programa inteligente?

## Las suposiciones subyacentes

En el centro de las investigaciones sobre inteligencia artificial aparece lo que Newell y Simon (1976) llaman la hipótesis del sistema de símbolos físicos. Ellos definen un sistema de símbolos físicos como sigue:

Un sistema de símbolos físicos consiste en un conjunto de entidades, llamadas símbolos, que son patrones físicos que pueden funcionar como componentes de otro tipo de entidad llamada expresión (o estructura de símbolos). De esta forma, una estructura de símbolos está compuesta por un número de instancias (señales o tokens) de símbolos relacionados de alguna forma física (como que una señal debe seguir a otra). En algún instante, el sistema contendrá una colección de estas estructuras de símbolos. Además de estas estructuras, el sistema contiene también una colección de procesos que operan sobre expresiones para producir otras expresiones: procesos de creación, modificación, reproducción y destrucción. Un sistema de símbolos físicos es una máquina que produce a lo largo del tiempo una colección evolutiva de estructuras de símbolos. Este sistema existe en un mundo de objetos tan extenso como sus propias expresiones simbólicas.

Ellos entonces enunciaron la hipótesis así:

**La hipótesis del sistema de símbolos físicos. Un sistema de símbolos físicos posee los medios necesarios y suficientes para realizar una acción inteligente genérica.**

Esta hipótesis es solo una hipótesis. Esto significa que no hay manera de probarla o refutarla hablando en términos lógicos. De esta forma, debe estar sujeta a una validación empírica. Puede encontrarse que es falsa. Puede encontrarse que el peso de la evidencia dice que es cierta. Pero el único camino para determinar su certeza es mediante la experimentación.

Las computadoras proporcionan el medio perfecto para esta experimentación ya que pueden ser programadas para simular cualquier sistema de símbolos físicos.

Como la facilidad de construcción de máquinas para computar se ha ido incrementando, también se ha incrementado la posibilidad de dirigir las investigaciones empíricas hacia la hipótesis del sistema de símbolos físicos. En cada investigación, se selecciona una tarea concreta en la que pueda considerarse que es necesario el uso de la inteligencia. Se propone un programa que realice esa tarea y a continuación se comprueba.

Sin embargo, modelos subsimbólicos (como por ejemplo, las redes neuronales) están comenzando a cuestionar los simbólicos como tareas de bajo nivel. Aún hoy se debaten ciertos conflictos entre modelos simbólicos y la hipótesis del sistema de símbolos físicos. Y es importante hacer notar que el éxito de los sistemas subsimbólicos no es necesariamente una evidencia en contra de la hipótesis. Con frecuencia es posible llevar a cabo una tarea de más de una forma.

La importancia de la hipótesis del sistema de símbolos físicos es doble. Es una teoría significativa de la naturaleza de la inteligencia humana y también es de gran interés para los psicólogos. Esto también forma la base de la creencia de que es posible construir programas que lleven a cabo las tareas inteligentes que son ahora realizadas por la gente.

## ¿Qué es una técnica de IA?

Los problemas abordados por la inteligencia artificial configuran un amplio espectro. Tienen muy poco en común excepto que todos ellos son complicados. ¿Existen entonces técnicas apropiadas para solucionar algunos de estos problemas? La respuesta es afirmativa, los hay. ¿Qué puede decirse de estas técnicas, si es que se puede, aparte del hecho de que manipulan símbolos?

Uno de los más rápidos y sólidos resultados que surgieron en las primeras tres décadas de las investigaciones en IA fue que la inteligencia necesita conocimiento. Para compensar este arrollador logro, imprescindiblemente, el conocimiento posee algunas propiedades poco deseables, tales como:

- Es voluminoso.
- Es difícil caracterizarlo con exactitud.
- Cambia constantemente.
- Se distingue de los datos en que se organiza de tal forma que se corresponde con la forma en que va a ser usado.

Entonces, ¿en qué punto nos quedamos en la definición de una técnica de IA?

Concluimos en que una técnica de IA es un método que utiliza conocimiento representado de tal forma que:

- El conocimiento representa las generalizaciones. En otras palabras no es necesario representar de forma separada cada situación individual. En lugar de esto, se agrupan las situaciones que comparten propiedades importantes. Si el conocimiento no posee esta propiedad, puede necesitarse demasiada memoria. Si no se cumple esta propiedad es mejor hablar de "datos" que de conocimiento.
- Debe ser comprendido por las personas que lo proporcionan. Aunque en muchos programas, los datos pueden adquirirse automáticamente (por ejemplo, mediante lectura de instrumentos), en muchos dominios de la IA, la mayor parte del conocimiento que se suministra a los programas lo proporcionan personas, haciéndolo siempre en términos que ellos comprenden.
- Puede modificarse fácilmente para corregir errores y reflejar los cambios en el mundo y en nuestra visión del mundo.
- Puede usarse en gran cantidad de situaciones aun cuando no sea totalmente preciso o completo.
- Puede usarse para ayudar a superar su propio volumen, ayudando a acotar el rango de posibilidades que normalmente deben ser consideradas.

Como conclusión se ponen de manifiesto tres importantes técnicas de IA:

**Búsqueda:** proporciona una forma de resolver los problemas en los que no se dispone de un método más directo tan bueno como una estructura en la que empotrar algunas técnicas directas existentes.

**Uso del conocimiento:** proporciona una forma de resolver problemas complejos explotando las estructuras de los objetos involucrados.

**Abstracción:** proporciona una forma de separar aspectos y variaciones importantes de aquellos otros sin importancia y que en caso contrario podrían colapsar un proceso.

Para solucionar problemas complicados los programas que utilizan estas técnicas presentan numerosas ventajas frente a los que no lo hacen. Los primeros son mucho menos frágiles, no se despistarán totalmente debido a una pequeña perturbación en la entrada. El conocimiento del programa es comprendido fácilmente por la gente. Y estas técnicas pueden trabajar con facilidad en grandes problemas en donde los métodos directos fallan.

### El nivel del modelo

Antes de ponerse en tarea, es una buena idea decidir qué es exactamente lo que se intenta lograr. Podemos preguntarnos ¿Qué pretendemos al construir programas que realicen las tareas inteligentes que los humanos hacen? ¿Estamos intentando construir programas que realicen las tareas de la misma forma en que lo hace el hombre? O ¿Estamos intentando construir programas que simplemente realicen las tareas de la forma que parezca más sencilla? Han existido proyectos de IA centrados en cada uno de estos objetivos.

Los esfuerzos dedicados a construir programas que lleven a cabo tareas de la misma forma que el hombre, se dividen en dos clases.

- Los programas de la primera clase se encargan de problemas que no se adecuan mucho con nuestra definición de tarea perteneciente a la IA; son aquellos problemas que una computadora puede resolver fácilmente, pero cuya resolución implica el uso de mecanismos de los que no dispone el hombre. Un ejemplo clásico de esta clase de programas lo constituye el Observador y Memorizador Elemental (EPAM) (Elementary Perceiver and Memorizer) (Feigenbaum, 1963), el cual memoriza parejas asociadas de sílabas sin sentido. Memorizar parejas de sílabas sin sentido es fácil para una computadora: simplemente se almacenan. Para recuperar una sílaba correspondiente dado su estímulo, la computadora examina la sílaba estimulada y responde con la que está almacenada a continuación.
- La segunda clase de programas que intentan modelar lo humano, son aquellas que realizan tareas que se adecuan claramente con nuestra definición de tareas de IA. Hacen cosas que no son triviales para una computadora.

Hay varias razones para querer modelar la forma de trabajar humana para llevar a cabo estas tareas:

- Verificar las teorías psicológicas de la actuación humana. PARRY (Colby, 1975) es un ejemplo de programa escrito por esta razón; utiliza un modelo de comportamiento paranoico humano para simular el comportamiento conversacional de una persona paranoica. El programa resultó lo suficientemente bueno como para que algunos psicólogos que tuvieron la oportunidad de conversar con el programa mediante un terminal, diagnosticaran que su comportamiento era paranoico.
- Capacitar a las computadoras para comprender el razonamiento humano. Por ejemplo, que una computadora pudiera leer un artículo en el periódico y al responder a preguntas tales como ¿Por qué los terroristas mataron a los rehenes? El programa pueda simular los procesos de razonamiento de los seres humanos.
- Capacitar a la gente para comprender a las computadoras. En muchas circunstancias, la gente no está dispuesta a fiarse del resultado de una computadora a no ser que entienda como ha llegado la máquina a esa conclusión. Si el proceso de razonamiento de una computadora es similar al humano, producir una explicación adecuada resultaría mucho más fácil.
- Explotar el conocimiento que se puede buscar en el hombre. Debido a que el hombre es el sistema del que mejor se conoce cómo lleva a cabo las tareas con las que estamos familiarizados, tiene sentido fijarse en él para buscar pistas de cómo actuar.

Esta última motivación es probablemente la más penetrante de las cuatro. Motivó desde el principio la aparición de sistemas que intentaban obtener un comportamiento inteligente imitando al hombre a nivel de las neuronas individuales. Ejemplos de esto son los primeros trabajos teóricos de McCulloch y Pitts (1943), el trabajo sobre perceptrones, desarrollado en principio por Frank Rosenblatt, pero descrito mejor en Perceptrons (Minsky y papera, 1969) y Design for a Brain (Sabih, 1952). Se demostró, sin embargo, que era imposible producir un comportamiento mínimamente inteligente con estos sencillos dispositivos. Una de las razones era que había severas limitaciones teóricas en la arquitectura de la red neuronal que se estaba utilizando. Más recientemente han surgido nuevas arquitecturas de redes neuronales. Estas estructuras no se hallan sujetas a las mismas limitaciones teóricas que los perceptrones. Estas nuevas arquitecturas se denominan imprecisamente "conexiónadas" (connectionist), y se usan como base para programas de resolución de problemas y aprendizaje.

### Criterios de determinación del éxito

Una de las preguntas más importantes a responder en toda investigación científica o de ingeniería es ¿Cómo sabremos si hemos tenido éxito?. La Inteligencia Artificial no es una excepción. ¿Cómo podemos saber si hemos construido una máquina inteligente?

En 1950, Alan Turing propuso el siguiente método para determinar si una máquina es capaz de pensar. Este método es conocido como el test de Turing. Para realizarlo se necesitan dos personas y la máquina que se desea evaluar. Una de las personas actúa de entrevistador y se encuentra en una habitación, separado de la computadora y de la otra persona. El entrevistador hace preguntas tanto a la persona como a la computadora mecanografiando las cuestiones y recibe las respuestas de igual forma. El entrevistador solo los conoce por A y B, y

debe intentar determinar quien es la persona y quien la máquina. El objetivo de la máquina es hacer creer al entrevistador que es una persona, si lo consigue, se concluye que la máquina piensa. Se permite que ésta haga cualquier cosa para engañar al entrevistador. Así, por ejemplo, si la pregunta es ¿Cuánto es 123324 por 73981? La máquina podría esperar unos minutos y responder erróneamente (Turing, 1963).

El aspecto más serio, sin embargo, es la cantidad de conocimiento que necesitaría la máquina para pasar el test de Turing.

Aún tendrá que pasar mucho tiempo para que una máquina supere el test de Turing. Algunos piensan que nunca lo harán. Pero supongamos que estamos dispuestos a aceptar menos que una imitación completa de la persona. ¿Podemos calibrar el éxito de la IA en dominios más restringidos?

Con frecuencia la respuesta es afirmativa. Algunas veces se puede dar una medida bastante buena del logro obtenido por un programa. Por ejemplo, un programa puede jugar al ajedrez al mismo nivel que los jugadores humanos; su nivel vendrá dado por el de los jugadores a los que pueda derrotar. Actualmente, los programas logran clasificaciones más altas que la inmensa mayoría de los jugadores de ajedrez. Para otros dominios, es posible dar una medida menos precisa del éxito de un programa. Por ejemplo, DENDRAL, es un programa que analiza componentes orgánicos para determinar su estructura. Es cruel dar una medida precisa del nivel de éxito de DENDRAL, comparándolo con el de los químicos; sin embargo, se han logrado análisis que han sido publicados como resultados de investigación original. Desde luego es al menos una ayuda competente.

En otros dominios técnicos, es posible hacer una comparación del tiempo que tarda un programa en llevar a cabo una determinada tarea con el que tarda una persona en hacer lo mismo. Por ejemplo, existen bastantes programas usados por empresas de informática para configurar sistemas particulares a las necesidades del cliente (en donde el programa R1 es pionero). Estos programas normalmente necesitan minutos para realizar tareas que antes requerían horas para un habilidoso ingeniero. Estos programas se evalúan normalmente atendiendo a si se ahorra o se gana dinero.

Si lo que se quiere es escribir programas que simulen el comportamiento humano ante una tarea, la forma de medir el éxito está en que el comportamiento del programa se corresponda con el humano, así como mediante distintas clases de experimentos y análisis de protocolos. En este sentido no se busca un programa que simplemente sea tan bueno como sea posible, se busca un programa que falle donde la gente lo hace.

### Ejercicio

Desarrollar un programa que enseñe a un niño a restar.

### 1.5.1. Acciones que debe llevar a cabo el sistema

Para construir un sistema que resuelva un problema específico, es necesario realizar estas cuatro acciones:

- Definir el problema con precisión. La definición debe incluir especificaciones precisas tanto sobre la o las situaciones iniciales como sobre las situaciones finales que se aceptarían como soluciones al problema.
- Analizar el problema. Algunas características de gran importancia pueden tener un gran efecto sobre la conveniencia o no de utilizar las diversas técnicas que resuelven el problema.
- Aislar y representar el conocimiento necesario para resolver el problema.
- Elegir la mejor técnica que resuelva el problema y aplicarla al problema particular.

### 1.5.2. Definición del problema mediante una búsqueda en espacio de estados

Suponga que comenzamos por el problema de "Jugar al ajedrez". A pesar de que es previsible que la mayoría de la gente únicamente con esta sentencia actúe correctamente, la definición del problema tal y como está, es incompleta. Para construir un programa que "juegue al ajedrez", primero se debe especificar la posición inicial del tablero, las reglas que definen los movimientos legales y las posiciones del tablero que representan una victoria tanto para un lado como para el otro. Además, se debe explicitar previamente el objetivo implícito de no solo realizar un movimiento legal de ajedrez, sino también ganar la partida, si es posible.

Para el problema de "jugar al ajedrez" es bastante fácil dar una descripción formal y completa del problema. La posición inicial se puede describir como un array de 8 por 8 posiciones donde cada una contiene un símbolo, de acuerdo con las piezas situadas en posición de comienzo de una partida oficial de ajedrez. El objetivo se define como cualquier posición del tablero en la que el contrario no pueda realizar ningún movimiento legal y su rey esté amenazado.

Los movimientos legales representan la forma de llegar a algún estado objetivo partiendo del estado inicial. Se pueden describir fácilmente como un conjunto de reglas compuestas por dos partes: una parte izquierda que se usa a modo de patrón para ser contrastado con la situación actual del tablero, y una parte derecha que describe el cambio que debe producirse en el tablero para que refleje el movimiento.

Existen diferentes formas de expresar estas reglas; una de ellas es la que se muestra en la Figura 1.3.

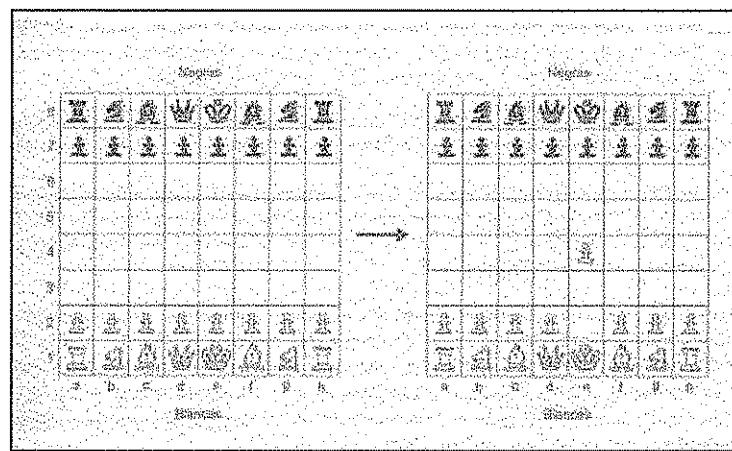


Figura 1.3

Sin embargo, si se utilizan reglas como la anterior, es necesario escribir un número muy grande de ellas, ya que existiría una regla para cada una de las  $10^{120}$  posibles posiciones del tablero. Al usar tal cantidad de reglas aparecen dos serias dificultades prácticas:

- Nadie puede suministrar un conjunto completo de tales reglas. Serían demasiadas y no se podría evitar la aparición de errores.
- Ningún programa puede manipular todas estas reglas. Podría usarse un sistema de hashing para poder encontrar más rápidamente las reglas más relevantes pero aún así, almacenar tantas reglas crea muchas dificultades.

Con el fin de minimizar estos problemas, se debe buscar la forma de escribir las reglas que describen los movimientos legales de la forma más general posible. Para lograrlo, es adecuado introducir una notación conveniente para describir los patrones y las sustituciones. Por ejemplo, la regla descripta en la Figura 1.3, al igual que muchas como ella, puede escribirse como se muestra en la Figura 1.4. En general, cuanto más sucintamente se describan las reglas, menos trabajo se empleará para introducirlas y el programa las usará con más eficiencia.

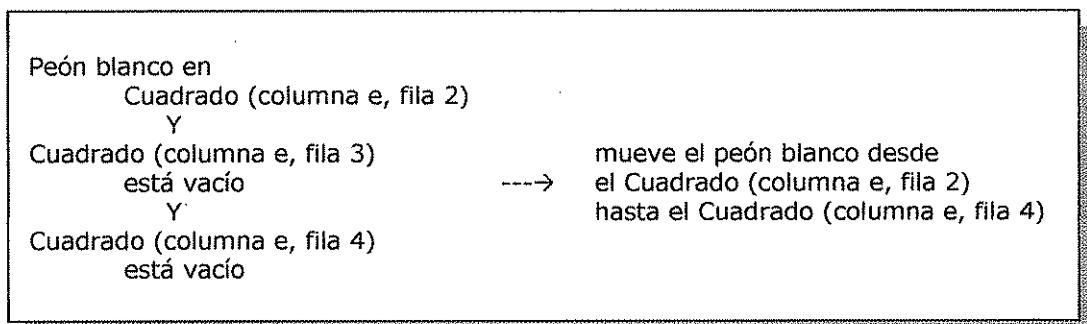


Figura 1.4

Se ha definido el problema de jugar al ajedrez como un problema de movimientos a través de un espacio de estados, donde cada estado se corresponde con una posición legal del tablero. Se puede entonces jugar al ajedrez, comenzando a partir de un estado inicial, mediante el uso de un conjunto de reglas de movimientos que trasladan de un estado a otro para intentar finalizar en alguno de los estados finales.

La representación como espacio de estados surge de forma natural en el ajedrez, ya que el conjunto de estados, que se corresponde con el conjunto de posiciones del tablero, es artificial y bien estructurado. El mismo tipo de representación es también adecuada para problemas menos estructurados, aunque sea necesario utilizar estructuras más complejas que una matriz para describir un estado individual. La representación como espacio de estados forma la base de la mayoría de los métodos de IA que se discuten aquí. **Su estructura se corresponde con la estructura de la resolución de problemas por dos importantes razones:**

- Permite definir formalmente el problema, mediante la necesidad de convertir alguna situación dada en la situación deseada usando un conjunto de operaciones permitidas.
- Permite definir el proceso de resolución de un problema como una combinación de técnicas conocidas (representadas por una regla que define un movimiento en el espacio) y búsqueda, la técnica general de exploración en el espacio intenta encontrar alguna ruta desde el estado actual hasta un estado objetivo. La búsqueda es un proceso de gran importancia en la resolución de problemas difíciles para los que no se dispone de técnicas más directas.

Con el fin de percibir la generalidad de la representación en espacio de estados, se muestra su uso para describir un problema muy diferente al de jugar al ajedrez.

**El problema de las jarras de agua:** Se tienen dos jarras, una de cuatro litros de capacidad y otra de tres. Ninguna de ellas tiene marcas de medición. Se tiene una bomba que permite llenar las jarras de agua. ¿Cómo se puede lograr tener exactamente dos litros de agua en la jarra de cuatro litros de capacidad?

Para este problema el espacio de estados se puede representar como un conjunto de pares ordenados de enteros  $(x,y)$ , de forma que  $x = 0, 1, 2, 3$  o  $4$  e  $y = 0, 1, 2$  o  $3$ ;  $x$  representa el número de litros de agua que almacena la jarra de cuatro litros, e  $y$  representa la cantidad de agua de la jarra de tres. El estado inicial es  $(0,0)$ . El estado objetivo es  $(2,n)$  siendo  $n$  cualquier valor (el problema no especifica cuantos litros debe haber en la jarra de tres litros).

Los operadores usados para resolver el problema se describen en la Figura 1.5. Al igual que en el problema del ajedrez, están representados en forma de reglas en las que la parte izquierda se compara con el estado actual, y en las que la parte derecha describe el nuevo estado al que se llega mediante la aplicación de la regla.

1	$(x,y)$ si $x < 4$	$\rightarrow (4,y)$	Llenar la jarra de 4 litros
2	$(x,y)$ si $y < 3$	$\rightarrow (x,3)$	Llenar la jarra de 3 litros
3	$(x,y)$ si $x > 0$	$\rightarrow (x-d,y)$	Vaciar un poco la jarra de 4 litros
4	$(x,y)$ si $y > 0$	$\rightarrow (x,y-d)$	Vaciar un poco la jarra de 3 litros
5	$(x,y)$ si $x > 0$	$\rightarrow (0,y)$	Vaciar la jarra de 4 litros en el suelo
6	$(x,y)$ si $y > 0$	$\rightarrow (x,0)$	Vaciar la jarra de 3 litros en el suelo
7	$(x,y)$ si $x+y \geq 4$ e $y > 0$	$\rightarrow (4,y-(4-x))$	Verter agua desde la jarra de 3 litros a la jarra de 4 litros hasta que la jarra de 4 litros esté llena
8	$(x,y)$ si $x+y \geq 3$ y $x > 0$	$\rightarrow (x-(3-y),3)$	Verter agua desde la jarra de 4 litros a la jarra de 3 litros hasta que la jarra de 3 litros esté llena
9	$(x,y)$ si $x+y \leq 4$ e $y > 0$	$\rightarrow (x+y,0)$	Verter todo el agua de la jarra de 3 litros en la jarra de 4 litros
10	$(x,y)$ si $x+y \leq 3$ y $x > 0$	$\rightarrow (0,x+y)$	Verter todo el agua de la jarra de 4 litros en la jarra de 3 litros
11	$(0,2)$	$\rightarrow (2,0)$	Verter 2 litros de la jarra de 3 litros en la jarra de 4 litros
12	$(x,2)$	$\rightarrow (0,2)$	Vaciar la jarra de 4 litros en el suelo

Figura 1.5

Litros en la Jarra de 4 litros	Litros en la jarra de 3 litros	Regla aplicada
0	0	
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 o 12
2	0	9 u 11

Una solución al problema de las jarras de agua

Resumiendo lo dicho anteriormente, para poder producir una descripción formal de un problema debe hacerse lo siguiente:

- Definir un espacio de estados que contenga todas las configuraciones posibles de los objetos más relevantes (y quizás algunos imposibles). Es, por supuesto, posible definir este espacio sin tener que hacer una enumeración de todos y cada uno de los estados que contienen.
- Identificar uno o más estados que describan situaciones en las que comience el proceso de resolución del problema. Estos se denominan estados iniciales.
- Especificar uno o más estados que pudieran ser soluciones aceptables del problema. Estos estados se denominan estados objetivo.
- Especificar un conjunto de reglas que describan las acciones (operadores) disponibles.

De esta forma, el problema puede resolverse con el uso de las reglas en combinación con una estrategia apropiada de control para trasladarse a través del espacio problema hasta encontrar una ruta desde un estado inicial hasta un estado objetivo.

### 1.5.3. Sistemas de producción

Debido a que la búsqueda es el núcleo de muchos procesos inteligentes, es adecuado estructurar los programas de IA de forma que se facilite describir y desarrollar el proceso de búsqueda. Los sistemas de producción proporcionan tales estructuras. Más abajo se introduce una definición de sistema de producción. Conviene no confundirse con otros usos de la palabra producción, tales como lo que se produce en una fábrica.

Un sistema de producción consiste en:

- Un conjunto de reglas compuestas por una parte izquierda (un patrón) que determina la aplicabilidad de la regla y una parte derecha, que describe la operación que se lleva a cabo si se aplica la regla.
- Una o más bases de datos/conocimiento que contengan cualquier tipo de información apropiada para la tarea en particular. Partes de la base de datos pueden ser permanentes, mientras que otras pueden hacer referencia solo a la solución del problema actual. La información almacenada en estas bases de datos debe estructurarse de forma adecuada.
- Una estrategia de control que especifique el orden en el que las reglas se comparan con la base de datos, y la forma de resolver los conflictos que surjan cuando varias reglas puedan ser aplicadas a la vez.
- Un aplicador de reglas.

(Prog. GRANJERO.PRO)

Hasta aquí, la definición de un sistema de producción es muy general. Abarca una gran cantidad de sistemas, incluyendo nuestras descripciones del jugador de ajedrez y de la solución al problema de las jarras de agua. También abarca una familia de intérpretes generales de sistemas de producción que incluye:

- Lenguajes básicos de sistemas de producción, tales como OPS5 (Brownston, 1985) y ACT\* (Anderson, 1983).
- Sistemas más complejos, con frecuencia híbridos, denominados armazones de sistemas expertos (expert systems shell), los cuales poseen entornos completos (hablando relativamente) para la construcción de sistemas expertos basados en conocimiento.
- Arquitecturas generales de resolución de problemas tales como SOAR (Laird, 1987), sistema basado en un conjunto específico de hipótesis motivadas cognoscitivamente por la naturaleza del problema a resolver.

Todos estos sistemas mantienen la arquitectura de un sistema de producción en su totalidad y permiten que el programador escriba reglas para definir los problemas particulares que tienen que resolverse.

#### 1.5.4. Análisis del problema

La búsqueda heurística es un método muy general que se puede aplicar a una gran clase de problemas. Incluye gran variedad de técnicas específicas, cada una de las cuales es particularmente efectiva para una pequeña clase de problemas. A fin de poder elegir el método más apropiado (o una combinación de métodos) para un problema en particular, es necesario analizarlo con arreglo a varias dimensiones clave:

- ¿Puede el problema descomponerse en un conjunto de subproblemas independientes (o casi) más pequeños o sencillos?
- ¿Pueden ignorarse pasos dados o al menos deshacerse si se comprueba que no eran adecuados?
- ¿Es predecible el universo del problema?
- ¿Una solución es buena de manera evidente, sin necesidad de compararla con todas las demás posibles soluciones?
- La solución deseada, ¿es un estado del mundo o una ruta hacia algún estado?
- ¿Es necesaria una gran cantidad de conocimiento para resolver el problema o solo es necesario para restringir la búsqueda?
- La computadora a la que simplemente se le da el problema ¿Puede emitir una solución, o es necesario que ésta interactúe con una persona?

#### ¿Puede descomponerse el problema?

Suponga que se desea resolver el problema de calcular la expresión:

$$\int (x^2 + 3x + \sin^2 x - \cos^2 x) dx$$

Se puede resolver este problema descomponiéndolo en tres problemas más pequeños, cada uno de los cuales puede resolverse usando una pequeña colección de reglas específicas. La Figura 1.6 muestra el árbol del problema que se genera mediante el proceso de descomposición. El árbol puede generarse con un sencillo programa recursivo de integración de esta forma: en cada paso, se verifica si el problema en el que se trabaja se puede resolver directamente. Si lo es, se devuelve inmediatamente la respuesta. Si el problema no se puede resolver fácilmente, el integrador verifica si puede descomponer el problema en otros más simples. Si puede, crea estos problemas y se llama recursivamente a sí mismo con ellos. Mediante el uso de esta técnica de descomposición del problema, se pueden resolver fácilmente problemas muy grandes.

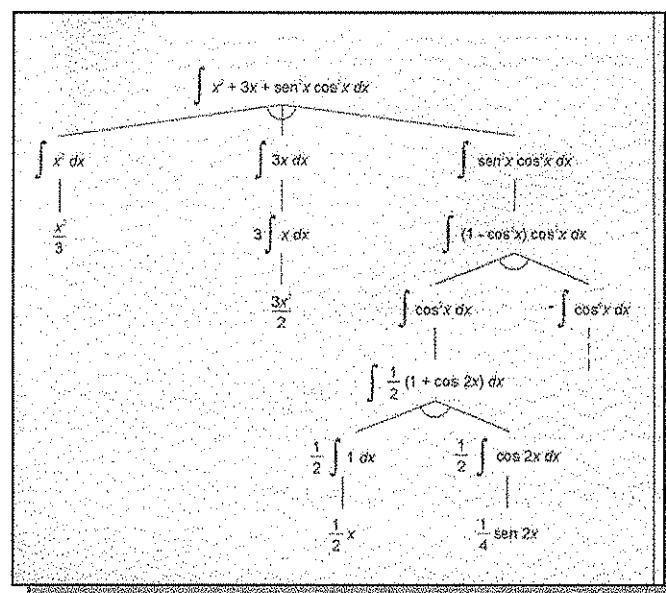


Figura 1.6

Consideré ahora el problema ilustrado en la Figura 1.7. Este problema está extraído de un dominio que con frecuencia está referenciado en la literatura de IA como el mundo de los bloques. Se permiten los siguientes operadores:

DESPEJADO ( $x$ ) [el bloque  $x$  no tiene nada sobre él]  $\rightarrow$  SOBRE ( $X$ , Mesa) [coge  $x$  y ponlo sobre la mesa]

DESPEJADO ( $x$ ) y DESPEJADO ( $y$ )  $\rightarrow$  SOBRE ( $X, Y$ ) [poner  $x$  sobre  $y$ ]

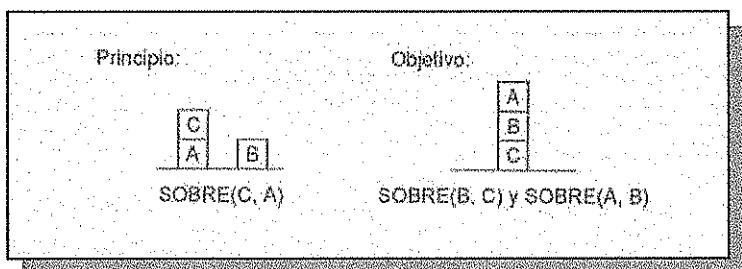


Figura 1.7

La aplicación de la técnica de descomposición del problema a este sencillo ejemplo de mundo de bloques se lleva al árbol solución mostrado en la Figura 1.8. En la figura, los objetivos están subrayados; los estados alcanzados no lo están. La idea de esta solución es la de reducir el problema de conseguir B sobre C y A sobre B a dos problemas separados. El primero de estos nuevos problemas, conseguir B sobre C, es sencillo, dado el estado inicial. Simplemente se coloca B sobre C. El segundo subobjetivo no es tan sencillo, puesto que los únicos operadores permitidos toman un solo bloque a la vez, se tiene que eliminar A quitando C antes de coger A y situarlo sobre B. Esto puede hacerse fácilmente. Sin embargo, al intentar combinar las dos subsoluciones en una sola, se falla. Independientemente de cual se haga primero, no se podrá realizar el segundo tal y como se ha planeado. En este problema los dos subproblemas no son independientes. Interactúan, y tales interacciones deben ser consideradas a fin de llegar a una solución para la totalidad del problema.

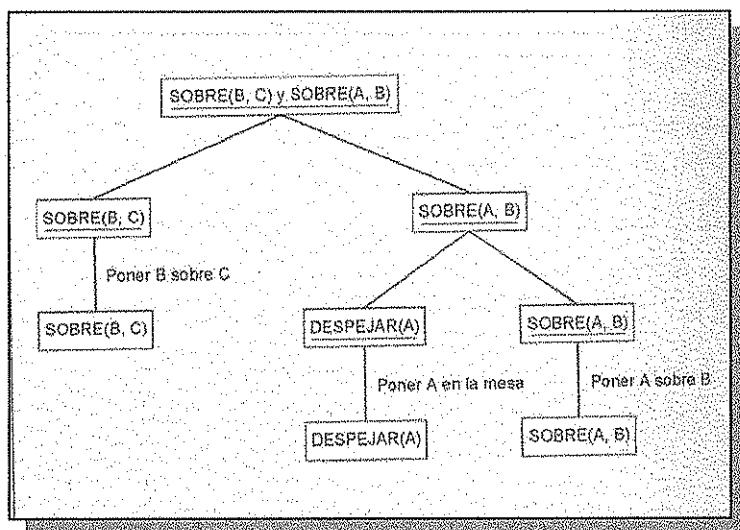


Figura 1.8

Estos dos ejemplos, el de la integración simbólica y el del mundo de los bloques, ilustran la diferencia entre problemas que se pueden descomponer y los que no se pueden descomponer.

### ¿Pueden deshacerse o ignorarse pasos hacia una solución?

Suponga que se intenta probar un teorema matemático. En primer lugar se prueba un lema que se piensa que será útil. En cierto instante, se comprueba que el lema no supone una ayuda para nada. ¿Se está en un apuro?

No. Todo lo que se necesita saber para probar el teorema es todavía cierto y está memorizado. Algunas reglas que pudieron aplicarse al principio aún pueden hacerlo. Basta con continuar

como si se comenzara de nuevo. Todo el esfuerzo realizado se ha perdido en explorar un callejón sin salida.

**El 8-puzzle:** El 8-puzzle es un cajón cuadrado en el que hay situados ocho bloques cuadrados. El cuadrado restante está sin llenar. Cada bloque tiene un número. Un bloque adyacente al hueco puede deslizarse hacia él. El juego consiste en partir de una posición de salida para llegar a una posición especificada como objetivo. El objetivo es transformar la posición inicial en la posición objetivo mediante el deslizamiento de los bloques.

En la Figura 1.9 se ve un juego de muestra con el 8-puzzle. Al intentar resolver el 8-puzzle, se pueden realizar movimientos estúpidos. Por ejemplo, en el juego que se muestra arriba, se podría empezar deslizando el bloque 5 al espacio vacío. Al hacer esto, no se podrá deslizar el bloque 6 al hueco porque éste se ha movido, pero podemos volver atrás y deshacer el primer movimiento, deslizando el bloque 5 adonde estaba. Entonces ya podemos mover el bloque 6. Los errores pueden recuperarse también, pero no de una forma tan sencilla como en el problema de la demostración del teorema. Se debe realizar un paso adicional para deshacer cada movimiento incorrecto, mientras que no se necesita realizar ninguna acción para "deshacer" un lema inútil. Además, el mecanismo de control de resolución de un 8-puzzle no debe perder de vista el orden en que se realizan las operaciones para que éstas puedan deshacerse si es necesario. La estructura de control del demostrador de teoremas no necesita almacenar toda esta información.

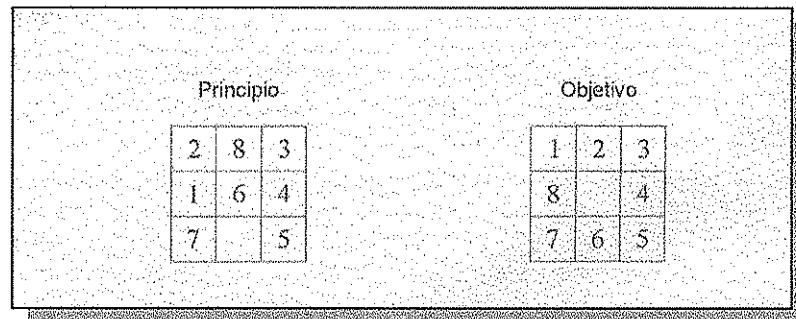


Figura 1.9

Considere otra vez el problema del juego del ajedrez. Suponga que un programa que juegue al ajedrez realiza un movimiento estúpido y cae en la cuenta un par de movimientos después; el programa no puede jugar como si nunca hubiera hecho el movimiento estúpido ni puede volver atrás y empezar el juego desde ese punto. Todo lo que puede hacer es intentar hacerlo mejor en la situación actual y partir desde ésta.

Estos tres problemas – la demostración de teoremas, el 8-puzzle y el ajedrez – ilustran las diferencias existentes entre tres importantes clases de problemas:

- **Ignorables** (por ej., la demostración de teoremas), en la que pueden ignorarse pasos dados.
- **Recuperables** (por ej., el 8-puzzle), en el que pueden deshacerse pasos dados.
- **No recuperables** (por ej., el ajedrez), en el que no pueden deshacerse pasos dados.

Estas tres definiciones hacen referencia a los pasos de la solución de un problema y, por lo tanto, pueden surgir para caracterizar sistemas de producción específicos para resolver problemas más que el problema en sí mismo. Quizá una formulación diferente de un mismo problema haría que el problema fuera caracterizado de forma diferente. Estrictamente hablando, esto es cierto. Sin embargo, debido a muchos grandes problemas, existe solo una formulación (o un pequeño número de ellas esencialmente equivalentes) que de forma natural describe el problema. Esto es así para cada uno de los problemas expuestos anteriormente. Cuando éste sea el caso, tiene sentido considerar la recuperabilidad de un problema de forma equivalente a la recuperabilidad de una formulación natural para él.

La recuperabilidad de un problema juega un papel importante en la determinación de la complejidad de la estructura de control necesaria para resolver el problema. Los problemas ignorables se resuelven utilizando una sencilla estructura de control que nunca vuelve hacia

atrás. Estas estructuras de control son fáciles de implementar. Los problemas recuperables se resuelven con estrategias un poco más complicadas que a veces cometan errores. Para recuperarse de tales errores será necesaria una vuelta atrás, de forma que la estructura de control debe implementarse con una pila "push-down", en la que las decisiones se conservan en caso de que necesiten ser deshechas más tarde. Los problemas no recuperables, por otro lado, se resuelven mediante un sistema que debe aplicar muchísimo esfuerzo en la toma de decisiones ya que éstas son irrevocables. Algunos problemas irrecuperables se resuelven con métodos de estilo recuperable usando un proceso de planificación, en el que se analiza por adelantado una secuencia entera de pasos para descubrir adonde conducirá antes de dar el primer paso. Más adelante se explican las clases de problemas en los que es posible hacer esto.

### ¿Es predecible el universo?

Suponga nuevamente que estamos jugando al 8-puzzle. Cada vez que se hace un movimiento, se sabe exactamente qué ocurrirá. Esto significa que es posible planificar una secuencia entera de movimientos y estar seguros de que se conoce cual será el resultado. Es posible utilizar una planificación para evitar tener que deshacer movimientos, si bien todavía se deben hacer comprobaciones de movimientos en tiempo de planificación. De esta forma, es necesaria una estructura de control que permita la comprobación.

Sin embargo, en otros juegos diferentes al 8-puzzle, no es posible un proceso de planificación. Suponga que queremos jugar al bridge. Una de las decisiones que hay que tomar es qué carta jugar en la primera baza. Sería deseable planificar la mano completa antes de realizar esta primera jugada. Pero ahora no es posible hacer tal planificación con certeza debido a que no se sabe con exactitud donde están las cartas y qué harán los otros jugadores en sus turnos. Lo mejor que se puede hacer es investigar distintos planes y utilizar las probabilidades de las consecuencias que se derivan de su elección para resaltar el que tenga la más alta probabilidad estimada de llegar a una buena puntuación en el juego.

Estos dos juegos ilustran la diferencia entre problemas de consecuencia-cierta y de consecuencia-incierta. Una forma de describir una planificación es que es un problema resuelto sin realimentación del entorno. Para resolver los problemas de consecuencia-cierta, un sistema en lazo abierto es adecuado ya que el resultado de una acción se puede predecir perfectamente. Así, la planificación puede utilizarse para generar una secuencia de operadores que garantizan llegar a una solución. Para los problemas de consecuencia-incierta, sin embargo, la planificación puede al menos generar una secuencia de operadores que tiene una buena probabilidad de conducir a una solución. Para resolver estos problemas, es necesario tener en cuenta que un proceso de revisión de planes se lleve a cabo cuando el plan se realice y proporcione la necesaria realimentación. Además de no garantizar una solución, la planificación aplicada a los problemas de consecuencia-incierta tiene el inconveniente de que suele ser muy cara ya que el número de rutas de solución que necesita explorar crece exponencialmente con el número de puntos en los cuales la consecuencia no puede predecirse.

Las dos últimas características explicadas, ignorar versus recuperable y consecuencia-cierta versus consecuencia-incierta, interactúan de una forma interesante. Tal y como se ha mencionado siempre, una forma de resolver los problemas irrecuperables es planificando una solución completa antes de embarcarse en la implementación del plan. Sin embargo, este proceso de planificación solo es útil en los problemas de consecuencia-cierta. Así, uno de los tipos de problemas más difíciles de resolver son los irrecuperables de consecuencia-incierta. Ejemplos de tales problemas son los siguientes:

Bridge. Sin embargo, puede mejorarse un poco ya que existen estimaciones exactas de las probabilidades de cada una de las posibles consecuencias.

Control de un brazo de robot. La consecuencia es incierta debido a varias razones. Alguien podría poner algo en la ruta del brazo; los mecanismos del brazo podrían atascarse; un leve error podría causar que el brazo choque con una pila de objetos.

Ayudar a un abogado a decidir cómo defender a su cliente contra un cargo de asesinato. En este caso no se puede dar probablemente una lista de posibles consecuencias, y mucho menos dar sus probabilidades.

### Una solución adecuada ¿es absoluta o relativa?

Consideré el problema de responder a preguntas basadas en una base de datos de hechos simples, tal como ésta:

1. Marco fue un hombre.
2. Marco era pompeyano.
3. Marco nació en el año 40 d.c.
4. Todos los hombres son mortales.
5. Todos los pompeyanos murieron con la erupción del volcán en el año 79 d.c.
6. Ningún mortal vive más de 150 años.
7. Estamos en el año 1994 d.c.

Suponga que se hace la siguiente pregunta "¿Está Marco vivo?". Al representar cada uno de estos hechos en un lenguaje formal, tal como la lógica de predicados, y al utilizar métodos formales de inferencia, puede derivarse fácilmente una respuesta a la pregunta. De hecho, cualquiera de las dos formas de razonamiento conducirá a la respuesta, tal y como se muestra en la Figura 1.10. Nuestro interés se centra en responder a esta pregunta sin importar qué camino se ha seguido para hacerlo. Si se sigue un camino que lleva a la respuesta con éxito, no hay razón para volver atrás y ver si existen otros caminos que también lleguen a la solución.

	Justificación
1. Marco fue un hombre.	axioma 1
4. Todos los hombres son mortales.	axioma 4
8. Marco es mortal.	1,4
3. Marco nació en el 40 d.c.	axioma 3
7. Estamos en el año 1994 d.c.	axioma 7
9. Marco tiene 1954 años.	3,7
6. Ningún mortal vive más de 150 años.	axioma 6
10. Marco está muerto.	8,6,9
 o	
7. Estamos en el año 1994 d.c.	axioma 7
5. Todos los pompeyanos murieron con la erupción del volcán en el año 79 d.c.	axioma 5
11. Todos los pompeyanos están muertos.	7,5
2. Marco era pompeyano.	axioma 2
12. Marco está muerto.	11,2

Figura 1.10

Pero ahora considere de nuevo el problema del viajante de comercio. El objetivo es encontrar la ruta más corta que lleve a cada ciudad exactamente una vez. Suponga que las ciudades a visitar y las distancias entre ellas son las que aparecen en la Figura 1.11.

	Boston	Nueva York	Miami	Dallas	San Francisco
Boston		250	1450	1700	3000
Nueva York	250		1200	1500	2900
Miami	1450	1200		1600	3300
Dallas	1700	1500	1600		1700
San Francisco	3000	2900	3300	1700	

Figura 1.11

El vendedor podría comenzar desde Boston. En ese caso, una ruta que podría seguir es la que tiene 8850 millas de longitud. Pero, ¿es ésta la solución al problema? La respuesta es que no se puede asegurar hasta que no se intenten todas las demás rutas y se esté seguro de que ninguna de ellas es más corta. En este caso, como se ve en la Figura 1.12, la primera ruta no es definitivamente la solución al problema del viajante.

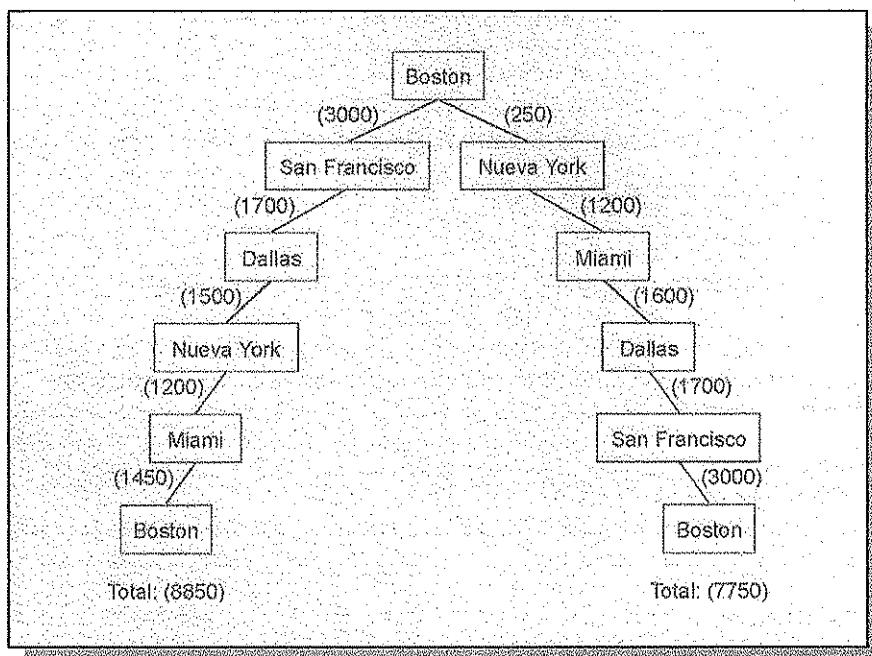


Figura 1.12

Estos dos ejemplos ilustran la diferencia que existe entre los problemas de algún-camino y los problemas de el-mejor-camino. Los problemas de el-mejor-camino son, por lo general, más complicados de computar que los problemas de algún-camino. Los problemas de algún-camino se resuelven frecuentemente en un tiempo razonable mediante heurísticas que sugieran rutas adecuadas para explorar. Si las heurísticas no son perfectas, la búsqueda de una solución puede no ser tan directa como sea posible, pero esto no tiene importancia. Para ciertos problemas de el-mejor-camino, sin embargo, no pueden usarse heurísticas que puedan omitir la mejor solución. Se requiere una búsqueda mucho más exhaustiva.

### ¿La solución es un estado o una ruta?

Considere el problema de encontrar una interpretación consistente a la frase:

El presidente del banco comió un plato de ensalada de pasta con el tenedor.

Existen varios componentes de esta frase que, si se aíslan, pueden tener más de una interpretación. Sin embargo, los componentes tienen que formar un todo, y de esta forma se restringen las otras posibles interpretaciones. Algunas causas de ambigüedad en la frase son las siguientes:

La palabra "banco" puede referirse a una institución financiera o a un objeto para sentarse. Sin embargo, solo uno de ellos puede tener un presidente.

La palabra "plato" es el objeto directo del verbo "comer". Es posible comerse un plato, pero es más usual comerse la ensalada de pasta que el plato.

Una ensalada de pasta es una ensalada que contiene pasta. Sin embargo, existen otras formas semánticas de agrupar dos nombres. Por ejemplo, la comida de perros normalmente no contiene perros.

La frase "con el tenedor" puede modificar distintas partes de la oración. En este caso, modifica al verbo "comer". Pero, si la frase hubiera sido "con vegetales", la estructura de la modificación sería diferente. Y si la frase hubiera sido "con sus amigos", la estructura sería también diferente.

Debido a las interacciones existentes entre los distintos constituyentes de esta oración, puede ser necesaria una búsqueda para encontrar una interpretación completa de la misma. Sin embargo, para resolver el problema de encontrar esta interpretación se necesita generar solo la interpretación misma. No es necesario ningún registro del proceso seguido para encontrar la interpretación.

Compare esto con el problema de las jarras de agua. En este caso no basta con indicar que se ha resuelto el problema y que la solución es (2,0). Para este tipo de problemas, lo que realmente se pide no es el estado final, sino el camino que se ha seguido para encontrar ese estado. Así, la solución a este problema debe ser una secuencia de operaciones (a veces denominada "plan") que produce el estado final.

Estos dos ejemplos, comprensión del lenguaje natural y el problema de las jarras de agua, ilustran la diferencia que existe entre problemas en los que la solución es un estado y problemas en los que la solución es una ruta hacia un estado. A este nivel, esta diferencia se puede ignorar para formular todos los problemas de IA de forma que la solución sea un estado. Si se intenta esto para problemas como el de las jarras de agua, se deben volver a describir los estados para que cada uno de ellos represente un camino parcial hacia la solución en lugar de un simple estado del mundo. Así, esta cuestión no es formalmente relevante. Sin embargo, debido al dilema ignorabilidad versus recuperabilidad, existe con frecuencia una formulación natural (y económica) del problema en donde los estados se corresponden con situaciones del mundo, y no secuencias de operaciones. En este caso, la respuesta a esta cuestión nos indica si va a ser necesario almacenar el camino del proceso de resolución del problema.

### ¿Cuál es el papel del conocimiento?

Considere de nuevo el problema del ajedrez. Suponga que se dispone de una potencia de computación ilimitada. ¿Cuánto conocimiento sería necesario para realizar un programa perfecto? La respuesta es muy poco –únicamente las reglas que determinen los movimientos legales y algún sencillo mecanismo de control que implemente el procedimiento de búsqueda apropiado. Algun conocimiento adicional sobre cosas tales como buenas estrategias y tácticas ayudaría, por supuesto, considerablemente a restringir la búsqueda para aumentar la velocidad de ejecución del programa.

Considere ahora, sin embargo, el problema de leer los periódicos del día para decidir cuales de ellos apoyan a los republicanos y cuales a los demócratas en unas elecciones. Asuma de nuevo una potencia de computación ilimitada, ¿cuánto conocimiento sería necesario para que la computadora intente resolver este problema? Esta vez la respuesta es que se necesita una gran cantidad. Sería necesario conocer cosas como éstas:

El nombre de los candidatos de cada partido.

El hecho de que si la mayor exigencia que se tiene es bajar los impuestos, probablemente se está apoyando a los republicanos.

El hecho de que si la mayor exigencia que se tiene es que mejore la educación, probablemente se está apoyando a los demócratas.

El hecho de que si uno se opone a un gran aparato del Estado, probablemente se está apoyando a los Republicanos.

Y muchos más.....

Estos dos problemas, el ajedrez y la comprensión de artículos periodísticos, ilustran la diferencia existente entre aquellos problemas en los que mucho conocimiento solo es necesario para acotar la búsqueda y aquellos en los que el conocimiento se emplea para poder reconocer una solución.

### ¿Necesita la tarea interaccionar con una persona?

Algunas veces resulta provechoso programar las computadoras para resolver problemas de una manera que la mayoría de la gente no sería capaz de entender. Esto es así si el nivel de interacción entre el hombre y la computadora es del tipo Entrada-problema Salida-solución. Sin embargo, se están desarrollando cada vez más programas que necesitan interacción

intermedia con el hombre, tanto para proporcionar información adicional de entrada al programa como para proporcionar noticias adicionales al usuario.

Considérese, por ejemplo, el problema de la demostración de teoremas matemáticos. Si:

Todo lo que se quiere saber es si existe una demostración.

El programa es capaz de hallar la demostración por sí mismo.

Entonces no importa la estrategia que use el programa para hallar la demostración. Puede utilizar, por ejemplo, el procedimiento de resolución, el cual puede ser muy eficiente, aunque no sea muy natural para el hombre. Pero si alguna de estas condiciones no se cumple, la forma de hallar la demostración tiene mucha importancia. Suponga que se intenta probar algún nuevo y difícil teorema. Se puede pedir una demostración que siguiera los patrones tradicionales de forma que un matemático esté seguro de que la demostración es correcta solo con leerla. Por otra parte, dar con una prueba del teorema puede llegar a ser tan complejo que el programa no sepa por donde empezar. Hasta ahora, las personas son superiores al realizar las estrategias de alto nivel que se necesitan para demostrar un teorema, de forma que una computadora necesita pedir ciertos consejos. Por ejemplo, normalmente en geometría resulta más fácil encontrar una prueba si alguien indica como representarla gráficamente. Para poder utilizar estos consejos, el razonamiento seguido por una computadora debe ser análogo al realizado por el consejero humano, al menos en ciertos aspectos. Debido a que las computadoras trabajan en áreas muy significativas de nuestras vidas tales como diagnósticos médicos, la gente no podrá aceptar el veredicto que genere un programa si no puede comprender el razonamiento que ha seguido para darlo.

Así, se puede hacer una distinción entre dos tipos de problemas:

Los "solitarios", en los que a la computadora se le da una descripción del problema y ésta proporciona una respuesta sin ningún tipo de comunicación ni necesidad de explicaciones del proceso de razonamiento.

Los "conversacionales", en los que existe una comunicación intermedia entre el hombre y la computadora, bien para proporcionar una ayuda adicional a la máquina o para que la computadora proporcione información al usuario.

Por supuesto, esta distinción no es totalmente estricta al describir los dominios particulares del problema. Tal y como se ha dicho, la demostración de teoremas matemáticos podría ser tanto de un tipo como de otro. Sin embargo, para una aplicación en particular, se necesitan sistemas de uno u otro tipo, y esta decisión tiene gran importancia a la hora de elegir el método de resolución del problema.

### 1.5.5. Características de los Sistemas de producción

Se ha examinado una serie de características que distinguen varios tipos de problemas. También se ha argumentado que los sistemas de producción representan la forma más adecuada de describir las operaciones que se llevan a cabo en la búsqueda de una solución a un problema. En este punto pueden surgir dos razonables preguntas:

¿Pueden los sistemas de producción, al igual que los problemas, ser descriptos por un conjunto de características que arrojen alguna luz sobre como implementarlos fácilmente?

Si es así, ¿qué relaciones existen entre los tipos de problemas y los tipos de sistemas de producción que son adecuados para resolver estos problemas?

La respuesta a la primera pregunta es afirmativa. Considere las siguientes definiciones de sistemas de producción. Un sistema de producción monótono es aquel en el que la aplicación de una regla nunca prevé la posterior aplicación de otra regla que podría haberse aplicado cuando se seleccionó la primera. Un sistema de producción no monótono es aquél en el que lo anterior no es cierto. Un sistema de producción parcialmente conmutativo es aquel que tiene la propiedad de que si una determinada aplicación de una secuencia de reglas transforma el estado  $x$  en el estado  $y$ , entonces alguna permutación permitida (por ejemplo, deben satisfacerse las precondiciones de una regla para que pueda ser aplicada) de estas reglas, también transforma el estado  $x$  en el estado  $y$ . Un sistema de producción conmutativo es aquel que es a la vez monótono y parcialmente conmutativo.

La relevancia de esta clasificación de los sistemas de producción estriba en la relación que existe entre las categorías de los mismos y las estrategias apropiadas de implementación. Sin embargo antes de explicar estas relaciones, puede resultar provechoso clarificar las definiciones viendo como se relacionan con problemas específicos.

De esta forma, se llega a la segunda de las cuestiones expresadas anteriormente la cual indaga sobre si existe una relación interesante entre las clases de sistemas de producción y las clases de problemas. Dado un problema resoluble, existe un número infinito de sistemas de producción que proporcionan formas de encontrar soluciones. Algunos de ellos serán más eficientes y naturales que otros. Un problema que puede ser resuelto con un sistema de producción, puede ser resuelto con uno conmutativo (la clase más restrictiva), sin embargo, el sistema conmutativo puede ser tan inmanejable que sea prácticamente inútil. Puede utilizar estados individuales para describir secuencias enteras de reglas aplicadas por un sistema no conmutativo más simple. Así, desde un punto de vista formal, no existe relación alguna entre tipos de problemas y tipos de sistemas de producción debido a que todos los problemas pueden resolverse utilizando todos los tipos de sistemas de producción. Pero desde un punto de vista práctico, definitivamente existe relación entre tipos de problemas y los tipos de sistemas de producción que se prestan de forma natural a representar estos problemas. Para ver esto, veamos algunos ejemplos. La Figura 1.13 muestra las cuatro categorías en que se dividen los sistemas de producción de acuerdo con las dos dicotomías, sistemas monótonos versus no monótonos y sistemas parcialmente conmutativos versus no parcialmente conmutativos, de forma que algunos problemas pueden resolverse más naturalmente por un tipo de sistema. La esquina superior izquierda la forman los sistemas conmutativos.

Los sistemas parcialmente conmutativos y los monótonos son adecuados para resolver problemas ignorables. Esto no es sorprendente desde el momento en que las definiciones de ambos son esencialmente las mismas. Sin embargo, se advierte que los problemas ignorables son aquellos en los que una formulación natural será un sistema parcialmente conmutativo y monótono. Los problemas que implican la creación de nuevos objetos más que el cambio de los viejos suelen ser ignorables. La demostración de teoremas, tal y como se ha descrito, es un ejemplo de proceso creativo. Realizar deducciones a partir de hechos conocidos es también un proceso creativo. Ambos procesos se pueden implementar fácilmente con un sistema parcialmente conmutativo y monótono.

Los sistemas parcialmente conmutativos y monótonos son importantes desde el punto de vista de la implementación porque no contemplan la característica de volver hacia estados pasados cuando se descubre que se ha seguido un camino incorrecto. Si bien con frecuencia es adecuado implementar estos sistemas con vuelta atrás (backtracking) para garantizar una búsqueda sistemática, la base de datos actual que representa el problema, no necesita regenerarse. El resultado de esto es, con frecuencia, un notable incremento de la eficiencia debido a que la base de datos no tiene que ser regenerada y no es necesario estar al tanto de cada cambio que se produjo en el proceso de búsqueda.

Hasta ahora se han explicado aquellos sistemas de producción que son a la vez monótonos y parcialmente conmutativos. Estos sistemas son adecuados para problemas en los que las cosas no cambian; *se crean*. Por otro lado, los sistemas no monótonos y parcialmente conmutativos, se adecuan a aquellos problemas en los que se realizan cambios pero éstos son reversibles y en los que el orden de las operaciones no es crítico. Este es normalmente el caso de los problemas de manipulación física, como la navegación de robots en una superficie plana. Suponga que el robot dispone de los siguientes operadores: ir hacia el norte (N), ir hacia el este (E), ir hacia el sur (S) e ir hacia el oeste (O). Para llevar a cabo su objetivo, no importa si el robot ejecuta N-N-E o N-E-N. Dependiendo de cómo se eligen los operadores, el problema del 8-puzzle y el del mundo de bloques pueden considerarse también como parcialmente conmutativos.

Estos dos tipos de sistemas parcialmente conmutativos son importantes desde el punto de vista de la implementación porque tienden a alcanzar muchos estados individuales duplicados durante el proceso de búsqueda.

	Monotono	No monotono
Parcialmente conmutativo	Demostración de teoremas	Navegación de robots
No parcialmente conmutativo	Síntesis química	Bridge

Figura 1.13

Los sistemas de producción que no son parcialmente conmutativos son adecuados para muchos problemas en los que se producen cambios irreversibles. Por ejemplo, considere el problema de realizar un proceso que produzca un compuesto químico. Los operadores disponibles incluyen cosas como "Añadir el agente químico x al recipiente" o "Cambiar la temperatura hasta t grados". Estos operadores pueden causar cambios irreversibles en el compuesto. El orden en que se realizan las operaciones es importante para determinar el resultado final. Es posible que si x se añade a y, se forme un compuesto estable, de forma que la posterior adición de z no produzca ningún efecto; sin embargo, si z se añade a y, puede formarse un compuesto estable diferente, de forma que no tenga efecto alguno la posterior adición de x. En los sistemas de producción que no son parcialmente conmutativos es menos probable que se llegue al mismo nodo varias veces durante el proceso de búsqueda. Cuando se trate con sistemas que describan procesos irreversibles, es particularmente importante tomar las decisiones correctas la primera vez, aunque si el universo es predecible, puede que una buena planificación las haga menos determinantes.

### Ejercicio

Encontrar una buena representación del espacio de estados para el problema del misionero y los caníbales.

Tres misioneros y tres caníbales se encuentran en una orilla de un río. A todos ellos les gustaría pasar a la otra orilla. Los misioneros no se fían de los caníbales. Por eso, los misioneros han planificado el viaje de forma que el número de misioneros en cada orilla del río nunca sea menor que el número de caníbales en esa misma orilla. Solo disponen de una lancha de dos plazas. ¿Cómo podrían atravesar el río sin que los misioneros corran peligro de ser devorados por los caníbales?

## **UNIDAD DIDACTICA 2**

**Búsqueda y  
Planificación**



**UNIDAD DIDACTICA 2****EJE CONCEPTUAL**

Búsqueda y Planificación

**TEMAS**

<b>2.1. Técnicas de búsqueda a ciegas.....</b>	<b>4</b>
2.1.1. Explosión Combinatoria.....	4
2.1.2. Ramificación y acotación .....	5
2.1.3. Búsqueda primero en anchura .....	6
2.1.4. Búsqueda primero en profundidad.....	7
<b>2.2. Técnicas de búsqueda heurística .....</b>	<b>10</b>
2.2.1. Generación y Prueba .....	12
2.2.2. Escalada o Remonte de colinas .....	13
2.2.3. Búsqueda El primero mejor .....	17
2.2.4. Reducción de problemas .....	24
2.2.5. Verificación de restricciones.....	28
2.2.6. Análisis de medios y fines .....	31
<b>2.3. Búsqueda en Problemas de juegos.....</b>	<b>34</b>
2.3.1. Juegos de dos jugadores .....	34
2.3.2. El procedimiento minimax .....	35
2.3.3. El procedimiento alfa-beta.....	37
<b>2.4. Búsqueda con Sistemas evolutivos.....</b>	<b>39</b>
2.4.1. Conceptos de Genética .....	39
2.4.2. Algoritmos Genéticos.....	43
2.4.3. Poblaciones .....	45
2.4.4. Operadores genéticos .....	46
2.4.5. Función de evaluación .....	47
<b>2.5. Planificación.....</b>	<b>52</b>
2.5.1. Introducción .....	52
2.5.2. Componentes de un sistema de planificación .....	53
2.5.3. Planificación mediante una pila de objetivos .....	58
2.5.4. Planificación no lineal mediante fijación de restricciones.....	63
2.5.5. Planificación jerárquica .....	64
2.5.6. Sistemas reactivos .....	65

## 2.1. Técnicas de búsqueda a ciegas

Se ha visto que para poder resolver un problema, primero hay que reducirlo a una forma en la que pueda darse una definición precisa. Esto puede lograrse mediante la *definición de un espacio de estados del problema* (incluyendo los estados iniciales y finales), y de un *conjunto de operadores* para trasladarse a través del espacio. *El problema se reduce entonces a buscar una ruta a través del espacio que une un estado inicial con un estado objetivo.* El proceso de resolución del problema puede modelarse como un sistema de producción y entonces se debe elegir la estructura de control apropiada para el sistema de producción con el fin de que el proceso de búsqueda sea lo más eficiente posible.

### Estrategia de control

Hasta ahora se ha ignorado por completo la cuestión de cómo se decide que regla hay que aplicar durante el proceso de búsqueda de la solución de un problema. Esta cuestión surge debido a que con frecuencia es posible aplicar más de una regla cuando sus partes izquierdas casan con el estado actual. Sin necesidad de pensar mucho, parece claro que estas decisiones tienen un impacto crucial en la rapidez, y tal vez en la posibilidad, de resolución del problema.

**El primer requisito** que debe cumplir una buena estrategia de control es *que cause algún cambio.* Considere el problema de las jarras de agua del apartado anterior. Suponga que se implementa una sencilla estrategia de control que cada vez empieza por la primera regla de la lista y elija la primera que encuentre que es aplicable. Si se hace esto, nunca se encontrará la solución al problema. Se estará llenando la jarra de cuatro litros de agua indefinidamente. **Las estrategias de control que no causan cambio de estado nunca alcanzan la solución.**

**El segundo requisito** que debe cumplir una buena estrategia de control es *que sea sistemática.* He aquí otra simple estrategia de control para el problema de las jarras de agua: en cada ciclo, elegir aleatoriamente una de entre todas las reglas aplicables. Esta estrategia es mejor que la primera: produce cambios y puede encontrar la solución eventualmente. Sin embargo, puede volver al mismo estado varias veces durante el proceso y suele utilizar muchos más pasos de los necesarios. Debido a que la estrategia de control no es sistemática, es posible utilizar secuencias de operadores no apropiadas varias veces hasta encontrar finalmente la solución. **El requisito de que una estrategia de control sea sistemática se corresponde con una necesidad de cambio global (en el curso de varios pasos) tanto como de cambio local (en el curso de un paso sencillo).**

### 2.1.1. Explosión Combinatoria

Se puede pensar que buscar una solución es tan simple como empezar en el principio y recorrer su camino hasta la conclusión. Sin embargo, en la mayoría de los problemas en los que querría usar una computadora para resolverlos, la situación es muy diferente. Generalmente, usará la computadora para resolver problemas en los que el número de nodos en el espacio de búsqueda es muy grande, y según crece el espacio de búsqueda, crece el número de diferentes caminos posibles hacia el objetivo.

El problema es que cada nodo añadido al espacio de búsqueda añadirá más de un camino: es decir, el número de caminos hacia el objetivo se incrementará más rápido con cada nuevo nodo.

Para comprender este incremento, considere el número de formas en que puede colocar tres objetos – A, B y C – sobre una mesa. Las seis diferentes ordenaciones son como sigue:

A B C
A C B
B C A
B A C
C B A
C A B

Aunque puede probarse a sí mismo rápidamente que estas son todas las formas en que A, B y C pueden ser colocadas, se puede obtener el mismo número con un teorema de la rama de las matemáticas llamada *combinatoria*, que es el estudio de la manera en que los casos pueden combinarse. El teorema establece que el número de formas en que N objetos pueden combinarse (o colocarse) es igual a  $N!$  ( $N$  factorial). El factorial de un número es el producto de todos los números del conjunto formado por el mismo número y todos los menores que él hasta el 1. En consecuencia,  $3!$  es  $3 \times 2 \times 1$  o 6.

Dada esta información, puede ver que, si tuviera 4 objetos para colocar, entonces habría  $4!$ , o 24 combinaciones. Con 5 objetos, el número es 120, con 6 objetos, es 720. Sin embargo, con, digamos, 1.000 objetos, el número de posibles combinaciones es enorme. Cuando hay más que un manojo de posibilidades, rápidamente se hace imposible examinar- y de hecho, incluso enumerar- todas las combinaciones.

Cuando relacione el concepto de explosión combinatoria con la resolución de problemas, podrá ver que cada nodo adicional añadido al espacio de búsqueda incrementa el número de posibles soluciones en una cifra bastante mayor que uno. Por consiguiente, en algún momento, hay demasiadas posibilidades para trabajar con ellas. Debido a que el número de posibilidades crece tan rápidamente, sólo los problemas más simples se dirigen a búsquedas exhaustivas. Una búsqueda exhaustiva, o "fuerza bruta", teóricamente funcionará siempre, no es práctica porque consume demasiado tiempo, demasiados recursos de computadora, o ambas cosas. Por esta razón, se han desarrollado otras técnicas de búsqueda.

### Ejercicio

Un arqueólogo debe recorrer 5 sitios clave para llevar a cabo una investigación sobre restos fósiles de Dinosaurios en Argentina. Estos sitios son Trelew, San Martín de los Andes, Purmamarca, Valle de la Luna y Lago Escondido. Al no decidirse sobre cuál es el orden que le convendría seguir en el recorrido, piensa que es una buena medida describir todas las posibles combinaciones diferentes y finalmente elegir una de ellas con los ojos cerrados.

Desarrollar un programa que permita ingresar un número variable de sitios a visitar (entre 2 y 8) y muestre en pantalla todos los posibles circuitos que podría definir.

### 2.1.2. Ramificación y acotación

Considere el siguiente problema:

El problema del viajante de comercio: Un vendedor tiene una lista de ciudades, cada una de las cuales debe visitar exactamente una vez. Existen carreteras directas entre cada pareja de ciudades de la lista. Encontrar la ruta más corta posible que debe seguir el vendedor que empieze y termine en alguna de estas ciudades.

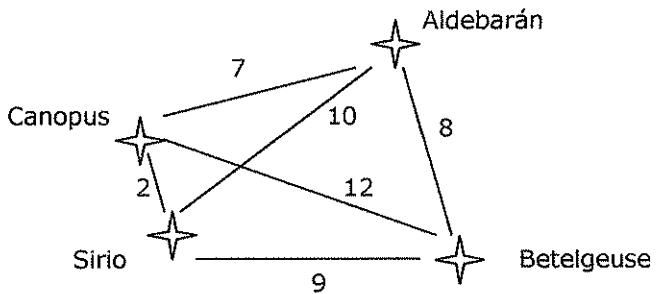
En principio, se puede resolver el problema con una sencilla estructura de control que cause movimiento y sea sistemática. Se podría simplemente explorar todas las posibles rutas en el árbol y devolver la que tenga menor longitud. Esta estrategia puede funcionar en la práctica para listas con muy pocas ciudades, pero se colapsa rápidamente conforme el número de ciudades aumenta. Si existen  $N$  ciudades, el número de rutas diferentes entre ellas es de:  $1 \times 2 \times \dots \times (N-1)$ , o  $(N-1)!$ . El tiempo empleado para examinar una ruta es proporcional a  $N$ . Así, el tiempo total empleado para completar la búsqueda es proporcional a  $N!$ ; si se asume que hay solo 10 ciudades, es de 3.628.800, un número bastante grande. El viajante podría perfectamente tener que visitar 25 ciudades. Encontrar la solución a este problema necesitaría más tiempo que el que podríamos gastar. Este fenómeno se denomina explosión combinatoria, para combatirla, es necesaria una *nueva estrategia de control*.

Se puede superar la sencilla estrategia perfilada anteriormente usando una técnica denominada *ramificación y acotación* (*branch and bound*). Comienza generando rutas completas, manteniéndose la ruta más corta encontrada hasta el momento. Deja de explorar una ruta tan pronto como su distancia total, hasta ese momento, sea mayor que la que se ha marcado como la más corta. Usar esta técnica garantiza hallar la ruta más corta. Desgraciadamente, aunque este algoritmo es más eficiente que el anterior, todavía necesita un tiempo exponencial. La cantidad exacta de tiempo utilizado en un problema en particular,

depende del orden en que se exploren las rutas. Sin embargo, todavía es inadecuada para problemas grandes.

### Ejercicio

Un viajante de comercio interestelar debe recorrer 4 estrellas, cada una de las cuales debe visitar exactamente una vez. Existen rutas directas entre cada pareja de estrellas a visitar y sus distancias se expresan en años luz. Se desea encontrar la ruta más corta posible que debe seguir el vendedor que empieza y termine en alguna de las siguientes estrellas.



Desarrollar un programa que resuelva este problema, teniendo en cuenta que se debe dejar de explorar una ruta tan pronto como su distancia total, hasta ese momento, sea mayor que la que se ha marcado como la más corta. El programa debe mostrar además por pantalla, el seguimiento que se va haciendo de cada camino, detallando para cada uno, las estrellas visitadas y la distancia total recorrida y en caso de abandonar un camino, por no ser conveniente, mostrar el mensaje correspondiente.

#### 2.1.3. Búsqueda primero en anchura

La búsqueda primero en anchura es la opuesta a la búsqueda primero en profundidad. En este método se evalúa cada nodo del mismo nivel antes de proceder al siguiente nivel más profundo. He aquí este método de recorrido con C como objetivo:

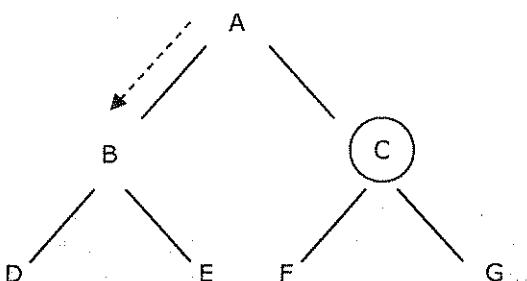


Figura 2.1

Esta ilustración muestra que se visitan los nodos ABC. Como la búsqueda primero en profundidad, la búsqueda primero en anchura garantiza que encontrará una solución, si existe, porque eventualmente degenerará en una búsqueda exhaustiva.

Una estrategia de control sistemática para el problema de las jarras de agua podría ser la siguiente: se construye un árbol cuya raíz sea el estado inicial; todas las ramificaciones de la raíz se generan al aplicar cada una de las reglas aplicables al estado inicial. La Figura 2.2 muestra la apariencia del árbol en este punto. Ahora, para cada nodo, se generan todas las posibles situaciones resultantes de la aplicación de todas las reglas adecuadas. En la Figura 2.3 se muestra el estado actual del árbol. Se continúa con este proceso hasta que alguna regla produce un estado objetivo.

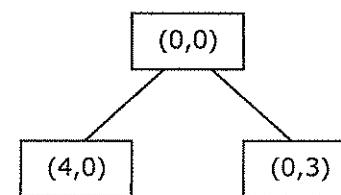


Figura 2.2

Este proceso, denominado búsqueda primero en anchura (breadth-first search), se describe con precisión de la siguiente forma.

#### **Algoritmo: Búsqueda primero en anchura**

1. Crear una variable llamada LISTA-NODOS y asignarle el estado inicial.
2. Hasta que se encuentre un estado objetivo o LISTA-NODOS esté vacía, hacer:
  - a) Eliminar el primer elemento de LISTA-NODOS y llamarlo E. Si LISTA-NODOS está vacía, terminar.
  - b) Para que cada regla se empareje con el estado descrito en E hacer:
    - i. Aplicar la regla para generar un nuevo estado.
    - ii. Si el nuevo estado es un estado objetivo, terminar y devolver este estado.
    - iii. En caso contrario, añadir el nuevo estado al final de LISTA-NODOS.

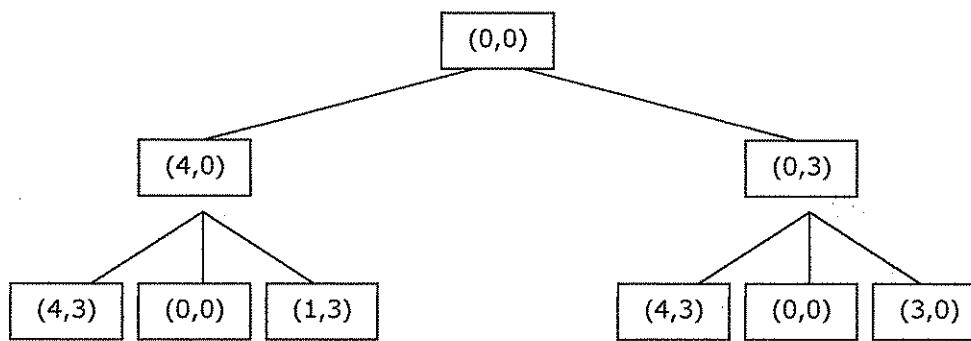


Figura 2.3

#### **2.1.4. Búsqueda primero en profundidad**

Una búsqueda primero en profundidad significa que se explora cada camino posible hacia el objetivo hasta su conclusión antes de intentar otro camino. Para comprender exactamente como funciona esta búsqueda, considere este árbol en el que F representa el objetivo:

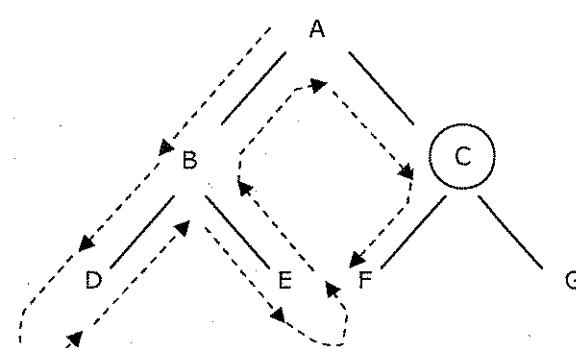


Figura 2.4

En este tipo de recorrido, *va por la izquierda hasta que o bien se alcanza un nodo terminal o bien encuentra el objetivo*. Si alcanza un nodo terminal, retrocede un nivel, va a la derecha y luego a la izquierda hasta encontrar el objetivo o nodo terminal. Repetirá este procedimiento hasta haber encontrado el objetivo o haber examinado el último nodo en el espacio de búsqueda.

Esta estrategia de control sistemática, se basa en continuar por una sola rama del árbol hasta encontrar una solución o hasta que se tome la decisión de terminar la búsqueda por esa dirección. Terminar la búsqueda por una ruta tiene sentido cuando se llega a un callejón sin salida, se produce un estado ya alcanzado o la ruta se alarga más de lo especificado en algún límite de "inutilidad". Si esto ocurre, se produce una *vuelta-atrás (backtracking)*. Se revisita el estado más recientemente creado desde el que sea posible algún movimiento alternativo más y se crea así un nuevo estado. Esta forma de vuelta-atrás se denomina *vuelta-atrás cronológica (chronological backtracking)* debido a que el orden en el que se deshacen los pasos depende únicamente de la secuencia temporal en que se hicieron originalmente esos pasos. En definitiva, el paso más reciente es siempre el primero que se deshace. Esta es la forma de vuelta-atrás a la que se hace referencia cuando se utiliza simplemente el término "vuelta-atrás". Sin embargo, existen otras formas de replegamiento de los pasos dados al computar. El procedimiento de búsqueda descrito se denomina también búsqueda primero en profundidad (depth-first search). El siguiente algoritmo lo define con precisión.

#### **Algoritmo: Búsqueda primero en profundidad**

1. Si el estado inicial es un estado objetivo, terminar y devolver un éxito.
2. En caso contrario, hacer lo siguiente hasta que se marque un éxito o un fracaso.
  - a) Generar un sucesor, E, del estado inicial. Si no existen más sucesores, marcar un fracaso.
  - b) Llamar a la Búsqueda en profundidad con E como estado inicial.
  - c) Si se devuelve un éxito, marcar un éxito. En caso contrario, continuar con el ciclo.

La Figura 2.5 muestra una instantánea de una búsqueda primero en profundidad para el problema de las jarras de agua.

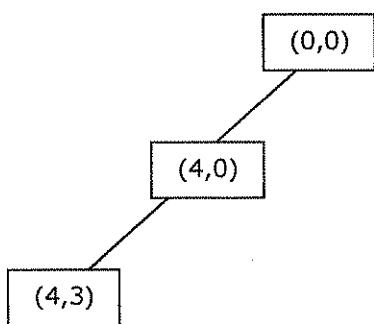


Figura 2.5

Al comparar estos dos sencillos métodos aparecen las siguientes observaciones:

#### **Ventajas de la Búsqueda primero en profundidad**

- La búsqueda primero en profundidad necesita menos memoria ya que solo se almacenan los nodos del camino que se sigue en ese instante. Esto contrasta con la búsqueda primero en anchura en la que debe almacenarse todo el árbol que haya sido generado hasta ese momento.
- Si se tiene suerte (o si se tiene cuidado en ordenar los estados alternativos sucesores), la búsqueda primero en profundidad puede encontrar una solución sin tener que examinar gran parte del espacio de estados. En el caso de la búsqueda primero en

anchura deben examinarse todas las partes del árbol de nivel  $n$  antes de comenzar con los nodos de nivel  $n+1$ . Esto es particularmente relevante en el caso de que existan varias soluciones aceptables. La búsqueda primero en profundidad acaba al encontrar una de ellas.

### Ventajas de la Búsqueda primero en anchura

- La búsqueda primero en anchura no queda atrapada explorando callejones sin salida. Esto se contrapone con la búsqueda primero en profundidad en la que se puede seguir una ruta infructuosa durante mucho tiempo, y quizás para siempre, antes de acabar en un estado sin sucesores. Esto es particularmente un problema en la búsqueda primero en profundidad si hay ciclos (por ejemplo, un estado tiene como sucesor un estado que es también uno de sus antecesores), a no ser que se tenga un cuidado especial en verificar tales situaciones.
- Si existe una solución, la búsqueda primero en anchura garantiza que se logre encontrarla. Además, si existen múltiples soluciones, se encuentra la solución mínima (es decir, tal que requiere el mínimo número de pasos). Esto está garantizado por el hecho de que no se explora una ruta larga hasta que se hayan examinado todas las rutas más cortas que ella. En cambio, en la búsqueda primero en profundidad es posible encontrar una solución larga en alguna parte del árbol, cuando puede existir otra mucho más corta en alguna parte inexplorada del mismo.

Lo deseable sería que pudieran combinarse las ventajas de estos dos métodos.

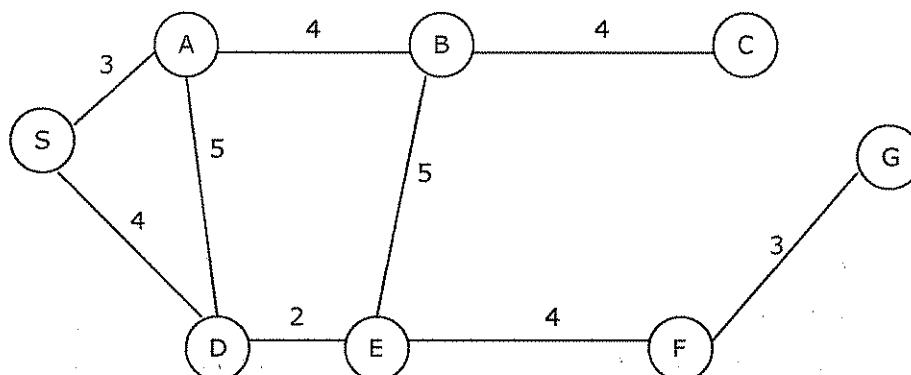
Para el problema de las jarras de agua, la mayoría de las estrategias de control que provoquen movimiento y sean sistemáticas logran encontrar la respuesta: el problema es sencillo. Sin embargo, éste no es siempre el caso. Para poder llegar a la solución de algunos problemas antes de morir, es necesario también demandar una estructura de control eficiente.

### Ejercicio

Desarrollar dos programas que resuelvan el problema de las jarras de agua, utilizando las estrategias de búsqueda primero en anchura y primero en profundidad.

### Ejercicio

Desarrollar dos programas que busquen trayectorias desde el nodo inicial S, al nodo meta, G, utilizando las técnicas de búsqueda primero en anchura y primero en profundidad, de acuerdo al siguiente esquema:



Los programas deben ir mostrando en pantalla, para cada recorrido efectuado, el detalle de los nodos visitados y la distancia recorrida.

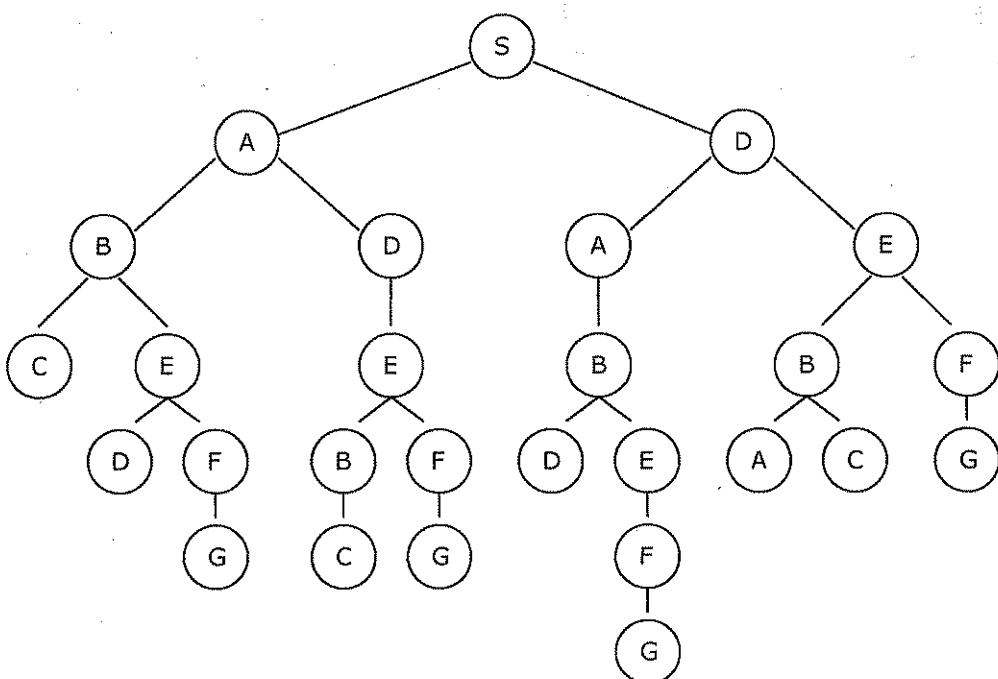


Figura 2.6

Árbol de búsqueda hecho a partir de una red

## 2.2. Técnicas de búsqueda heurística

Con el fin de resolver problemas complicados con eficiencia, con frecuencia es necesario comprometer los requisitos de movilidad y sistematicidad, y construir una estructura de control que no garantice encontrar la mejor respuesta pero que casi siempre encuentre una buena solución. De esta forma, surge la idea de **heurística**. Una heurística es una técnica que aumenta la eficiencia de un proceso de búsqueda, posiblemente sacrificando demandas de completitud. Las heurísticas son como los guías de turismo: resultan adecuados en el sentido de que generalmente suelen indicar las rutas interesantes; son malos en el sentido de que pueden olvidar puntos de interés para ciertas personas. Algunas heurísticas ayudan a guiar el proceso de búsqueda sin sacrificar ninguna demanda de completitud que el proceso haya podido tener previamente. Otras (en realidad, muchas de las mejores) pueden ocasionalmente causar que una buena ruta sea pasada por alto. Pero, en promedio, mejoran la calidad de las rutas que exploran. Al usar buenas heurísticas se pueden esperar buenas (aunque posiblemente no óptimas) soluciones para problemas difíciles, tales como el del viajante de comercio, en un tiempo menor al exponencial. Existen algunas heurísticas de propósito general que son adecuadas para una amplia variedad de dominios de problemas. Además, es posible construir heurísticas de propósito especial que exploten el conocimiento específico del dominio para resolver problemas particulares.

La heurística del vecino más próximo es un ejemplo de una buena heurística de propósito general válida para varios problemas combinatorios. Consiste en seleccionar en cada paso la alternativa localmente superior. Al aplicarla al problema del viajante de comercio, surge el siguiente proceso:

1. Seleccionar arbitrariamente una ciudad de comienzo.
2. Para seleccionar la siguiente ciudad, fijarse en las ciudades que todavía no se han visitado y seleccionar aquella que sea más cercana. Ir a esa ciudad.
3. Repetir el paso 2 hasta que todas las ciudades hayan sido visitadas.

Cuando se aplican a problemas específicos, la eficacia de las técnicas de búsqueda heurísticas depende en gran medida de la forma en que exploten el conocimiento del dominio particular, ya que, por sí solas, no son capaces de salvar la explosión combinatoria a la que son tan vulnerables los procesos de búsqueda. Por esta razón, a estas técnicas se las denomina con

frecuencia métodos débiles (weak methods). A pesar de que comprender la limitada efectividad de estos métodos débiles para resolver problemas difíciles ha sido un importante resultado surgido en las últimas tres décadas de investigación en IA, estas técnicas proporcionan un marco donde situar el conocimiento del dominio específico, ya sea manualmente o como resultado de un aprendizaje automático. Es por ello que siguen formando el núcleo de la mayoría de los sistemas de IA.

### Problema de la Mochila

Consiste en elegir, de entre un conjunto de  $n$  elementos de un negocio, (cada uno con un valor  $v_i$ , y un peso  $p_i$ ), aquellos que puedan ser cargados en la mochila de un individuo, que decide hacer una visita nocturna al negocio. La mochila resiste un peso máximo  $P$  y se debe tener en cuenta que el visitante pretende acumular el mayor valor posible, entre todos los objetos que recoge.

Este es un claro ejemplo de la presentación de un problema, en el que hay dificultad para hallar una solución óptima exacta, principalmente por el tiempo que llevaría recorrer y combinar todas las posibilidades en forma exhaustiva.

- Para 20 elementos → se definen  $2^{20}=1.048.580$  subconjuntos o soluciones
- Para 60 elementos → se necesitan 365 siglos para resolver el problema, a 1 millón de soluciones por segundo

Existen entonces, **métodos heurísticos** que proporcionan soluciones factibles (que satisfacen las restricciones del problema), que aunque no optimicen la función objetivo, se acercan al valor óptimo en un tiempo de cálculo razonable.

Una clase de algoritmos heurísticos son los **métodos constructivos**, que consisten en ir agregando componentes individuales a la solución hasta que se obtiene una solución factible.

Un representante de éstos son los **algoritmos greedy** (golosos o devoradores). Estos algoritmos van construyendo paso a paso la solución, buscando el máximo beneficio en cada paso.

En el problema de la mochila, debemos ir escogiendo los elementos que aporten el mayor valor en proporción a su peso ( $v_i / p_i$ ).

### Ejercicios

1. Desarrollar un algoritmo goloso que brinde una solución para el siguiente conjunto de datos:

1	150	20
2	325	40
3	600	50
4	805	36
5	430	25
6	1200	64
7	770	54
8	60	18
9	930	46
10	353	28

Peso máximo soportado  
por la mochila:  
4200 grs.

2. Dados 3 elementos, cuyos pesos son: 1800 grs., 600 grs. Y 1200 grs. y <sup>valores</sup> cuyos valores son: \$72, \$36 y \$60 respectivamente, y dado que la mochila puede soportar hasta 3000 grs. se pide:
- Hallar una solución utilizando un algoritmo goloso.
  - Analizar dicha solución respecto a su grado de optimización y elaborar las conclusiones que considere adecuadas.

### 2.2.1. Generación y Prueba

La estrategia de generación y prueba es la más simple de todas las que se van a explicar.

Consiste en realizar los siguientes pasos:

#### Algoritmo: Generación y prueba

- Generar una posible solución. Para algunos problemas, esto significa generar un objetivo particular en el espacio problema. Para otros, supone más bien generar un camino a partir de un estado inicial.
- Verificar si realmente el objetivo elegido es una solución comparándolo con el objetivo final o comparando el camino elegido con el conjunto de estados objetivo aceptables.
- Si se ha encontrado la solución, terminar. Si no, volver al paso 1.

Si se generan las posibles soluciones de forma sistemática, si la solución existe, este procedimiento es capaz de encontrarla en algún momento. Desafortunadamente, *si el espacio problema es muy grande, "en algún momento" puede ser demasiado tiempo.*

El algoritmo de generación y prueba es un procedimiento de búsqueda *primero en profundidad ya que las soluciones completas deben generarse antes de que se comprueben*. De una forma más sistemática, es simplemente una **búsqueda exhaustiva por el espacio problema**. El método de generación y prueba puede, por supuesto, funcionar de forma que genere las soluciones de forma aleatoria, pero esto no garantiza que se pueda encontrar alguna vez la solución. Esta forma de trabajar se conoce también como el **algoritmo del Museo Británico**, en referencia a un método empleado para encontrar objetos en el museo, haciendo que éste se recorriera aleatoriamente. Entre estos dos extremos **existe un punto medio** en donde el proceso de búsqueda actúa de forma sistemática, a pesar de que algunos caminos no se consideren porque dan la impresión de que por ellos no se llega a la solución. Esta evaluación se lleva a cabo mediante una función heurística.

La forma más sencilla de implementar una generación y prueba sistemática es mediante un árbol de búsqueda primero en profundidad con vuelta-atrás. Sin embargo, si algunos estados intermedios aparecen con frecuencia en el árbol, puede resultar mejor modificar el procedimiento descrito antes, para que recorra un grafo en lugar de un árbol.

Para problemas sencillos, una generación y prueba exhaustiva es normalmente una técnica razonable. Por ejemplo, considere el problema de acomodar cuatro cubos de seis caras, cada una de las cuales se encuentra pintada con un color distinto, de manera tal que una solución a este problema consiste en disponer los cubos en una fila de forma que el bloque muestre una cara de cada color. Este problema puede resolverse una persona – que es un procesador mucho más lento para este tipo de tareas que cualquier computadora barata – en pocos minutos intentando todas las posibilidades de forma sistemática y exhaustiva. Se puede resolver con más rapidez usando un procedimiento de generación y prueba heurístico. Al dar un rápido vistazo a los cuatro cubos se puede descubrir, por ejemplo, que existen más caras rojas que de cualquier otro color. De esta forma, sería una buena idea utilizarlas tan poco como fuera posible como cara exterior, modificando la posición de aquellos cubos. Al usar esta heurística, muchas configuraciones nunca se exploran y la solución se encuentra más rápidamente.



Figura 2.7

Desafortunadamente, para problemas mucho más complicados que este, una técnica de generación y prueba heurística no es muy eficiente por sí misma. Pero cuando se combina con otras técnicas que restrinjan el espacio de búsqueda, la técnica puede llegar a ser muy eficaz.

### Ejercicio

Desarrollar un programa que considere el problema de acomodar cuatro cubos de seis caras, cada una de las cuales se encuentra pintada con un color distinto, de manera tal que una solución a este problema consiste en disponer los cubos en una fila de forma que el bloque muestre una cara de cada color. Utilizar una estrategia de generación y prueba que evalúe la cantidad de caras de un mismo color que hay a la vista y trate de reducirlas, modificando un cubo cada vez.

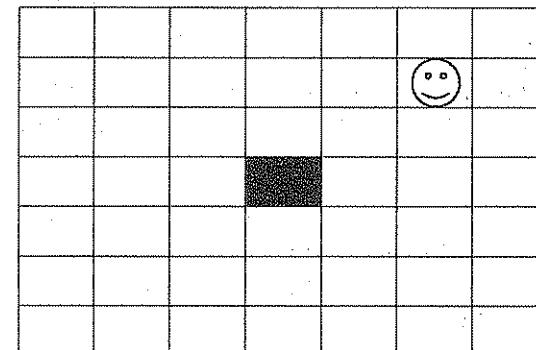
### 2.2.2. Escalada o Remonte de colinas

#### Escalada

El método de la escalada es una variante del de generación y prueba; en él existe realimentación a partir del procedimiento de prueba que se usa para ayudar al generador a decidirse por cual dirección debe moverse en el espacio de búsqueda. En un procedimiento de generación y prueba puro, la función de prueba responde solo un sí o un no. Pero si la función de prueba se amplía mediante una función heurística que proporcione una estimación de lo cercano que se encuentra un estado al estado objetivo, el procedimiento de generación puede usar esta información tal y como se muestra en un ejemplo posterior. Además esto es particularmente apropiado porque normalmente el cálculo de la función heurística puede hacerse, casi sin coste alguno, al mismo tiempo en que se está llevando a cabo la verificación de una solución. La escalada se utiliza frecuentemente cuando se dispone de una buena función heurística para evaluar los estados, pero cuando no se dispone de otro tipo de conocimiento provechoso.

#### Ejercicio

Suponga que se encuentra en una ciudad desconocida sin ningún mapa, y que quiere llegar al centro. Usted simplemente iría hacia los rascacielos. La función heurística sería en este caso la distancia existente entre su posición y los rascacielos y los estados deseables son aquellos en que esa distancia se minimiza.



Desde una posición al objetivo, avanzar de a un paso y valorar. Si no es mejor, volver atrás y buscar otro.

## Escalada simple

La forma más sencilla de implementar el método de la escalada es la siguiente:

### Algoritmo: Escalada simple

1. Evaluar el estado inicial. Si también es el estado objetivo, devolverlo y terminar. En caso contrario, continuar con el estado inicial como estado actual.
2. Repetir hasta que se encuentre una solución o hasta que no queden nuevos operadores que aplicar al estado actual:
  - a) Seleccionar un operador que no haya sido aplicado con anterioridad al estado actual y aplicarlo para generar un nuevo estado.
  - b) Evaluar el nuevo estado.
    - i. Si es un estado objetivo, devolverlo y terminar.
    - ii. Si no es un estado objetivo, pero es mejor que el estado actual, convertirlo en el estado actual.
    - iii. Si no es mejor que el estado actual, continuar con el bucle.

La principal diferencia que existe entre este algoritmo y el que se ha dado para la técnica de generación y prueba, consiste en *el uso de una función de evaluación como una forma de introducir conocimiento específico de la tarea realizada en el proceso de control*. La utilización de este conocimiento es lo que hace a éste y a otros métodos que se explican a lo largo de este capítulo, métodos de búsqueda heurística, y es este mismo conocimiento lo que da a estos métodos la capacidad de resolver algunos problemas que de otra forma serían inabordables.

Nótese que en este algoritmo se ha formulado la relativamente vaga pregunta, "¿Es un estado mejor que otro?". Para que el algoritmo pueda funcionar, es necesario proporcionar una definición precisa del término mejor. En algunos casos, significa un valor más alto de una función heurística; en otros, significa un valor más bajo. No importa lo que signifique siempre que a lo largo de una escalada específica se sea consistente con su interpretación.

## Escalada por la máxima pendiente

Una variación útil del método de escalada simple consiste en considerar todos los posibles movimientos a partir del estado actual y elegir el mejor de ellos como nuevo estado. Este método se denomina *método de escalada por la máxima pendiente (steepest-ascent hill climbing)* o *búsqueda del gradiente (gradient search)*. Nótese el contraste con el método básico, en el que el primer estado que parezca que sea mejor que el actual se selecciona como el estado actual. El algoritmo funciona así.

### Algoritmo: Escalada por la máxima pendiente

1. Evaluar el estado inicial. Si también es el estado objetivo, devolverlo y terminar. En caso contrario, continuar con el estado inicial como estado actual.
2. Repetir hasta que se encuentre una solución o hasta que una iteración completa no produzca un cambio en el estado actual:
  - a) Sea SUCC un estado tal que algún posible sucesor del estado actual sea mejor que este SUCC.
  - b) Para cada operador aplicado al estado actual hacer lo siguiente:
    - i. Aplicar el operador y generar un nuevo estado.
    - ii. Evaluar el nuevo estado. Si es un estado objetivo, devolverlo y terminar. Si no, compararlo con SUCC. Si es mejor, asignar a SUCC este nuevo estado. Si no es mejor, dejar SUCC como está.

- c) Si SUCC es mejor que el estado actual, hacer que el estado actual sea SUCC.

### Ejercicio

Resolver el ejercicio anterior utilizando una estrategia de Escalada por la máxima pendiente, o sea, considerando todos los posibles movimientos a partir del estado actual y elegir el mejor de ellos como nuevo estado.

Tanto la escalada básica como la de máxima pendiente pueden no encontrar una solución. Cualquiera de los dos algoritmos puede acabar sin encontrar un estado objetivo, y en cambio encontrar un estado del que no sea posible generar nuevos estados mejores que él. Esto ocurre si el programa se topa con un **máximo local**, una **meseta** o una **cresta**.

- Un **máximo local** es un estado que es mejor que todos sus vecinos, pero que no es mejor que otros estados de otros lugares. En un máximo local, todos los movimientos producen estados peores. Los máximos locales son particularmente frustrantes porque frecuentemente aparecen en las cercanías de una solución. En este caso se denominan estribaciones (foothills).
- Una **meseta** (plateau) es un área plana del espacio de búsqueda en la que un conjunto de estados vecinos posee el mismo valor. En una meseta no es posible determinar la mejor dirección a la que moverse haciendo comparaciones locales.
- Una **cresta** (ridge) es un tipo especial de máximo local. Es un área del espacio de búsqueda más alta que las áreas circundantes y que además posee en ella misma una inclinación (la cual se podría escalar). Pero la orientación de esta región alta, comparada con el conjunto de movimientos disponibles y direcciones en la que moverse, hace que sea imposible atravesar la cresta mediante movimientos simples.

Existen algunas formas de evitar estos problemas, si bien estos métodos no dan garantías:

- **Volver atrás hacia algún modo anterior e intentar seguir un camino diferente.** Es especialmente razonable si el nodo posee otra dirección que de la impresión de ser tan prometedora, o casi tan prometedora, como la que se eligió. Para implementar esta estrategia, se debe mantener una lista de caminos que casi se han seguido y volver a uno de ellos, si el camino que se ha seguido da la impresión de ser un callejón sin salida. Este método es especialmente adecuado para superar máximos locales.
- **Realizar un gran salto en alguna dirección** para intentar buscar en una nueva parte del espacio de búsqueda. Este método está especialmente indicado para superar mesetas. Si la única regla aplicable describe pequeños pasos, aplicarla varias veces en la misma dirección.
- **Aplicar dos o más reglas antes de realizar la evaluación.** Esto se corresponde con movimientos en varias direcciones a la vez. Este método es especialmente bueno para superar las crestas.

*Incluso con estas tres medidas de primeros auxilios, la escalada no es siempre muy eficaz.*

Considere el problema del mundo de los bloques que se muestra en la Figura 2.8. Asuma la existencia de los siguientes operadores:

- Tomar un bloque y situarlo sobre la mesa.
- Tomar un bloque y situarlo sobre otro.

Suponga que se utiliza la siguiente función heurística:

- Añadir un punto por cada bloque que esté sobre aquello en que se supone que debe estar.
- Restar un punto por cada bloque que esté situado en un lugar incorrecto.

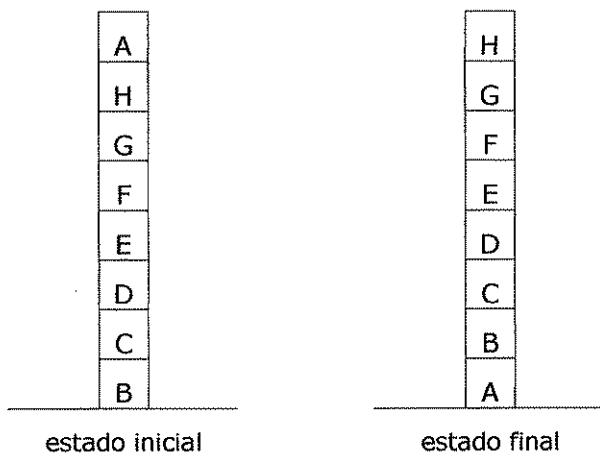


Figura 2.8

Al usar esta función, el estado objetivo tiene un valor de 8. El estado inicial tiene un valor de 4 (ya que tiene los puntos positivos de los bloques C, D, E, F, G y H, y los negativos de los bloques A y B). Solo es posible realizar un movimiento a partir del estado inicial, mover el bloque A a la mesa. Este movimiento produce un estado con valor 6 (ya que la posición de A es correcta y añade un punto en lugar de restarla). El procedimiento de escalada acepta este movimiento. En este nuevo estado, existen tres posibles movimientos que dan lugar a los tres estados que aparecen en la Figura 2.9. Estos estados tienen las siguientes puntuaciones: (a) 4, (b) 4 y (c) 4. La escalada se detiene ya que estos tres estados tienen puntuaciones más bajas que el estado actual. El proceso ha encontrado un máximo que no es el máximo global. En este punto, el problema consiste en que mediante un examen puramente local de las estructuras de apoyo, el estado actual parece ser mejor que cualquiera de sus sucesores porque tiene más bloques situados correctamente. Para resolver este problema, es necesario desmontar la estructura local adecuada (de B hasta H) porque está situada en un contexto global inadecuado.

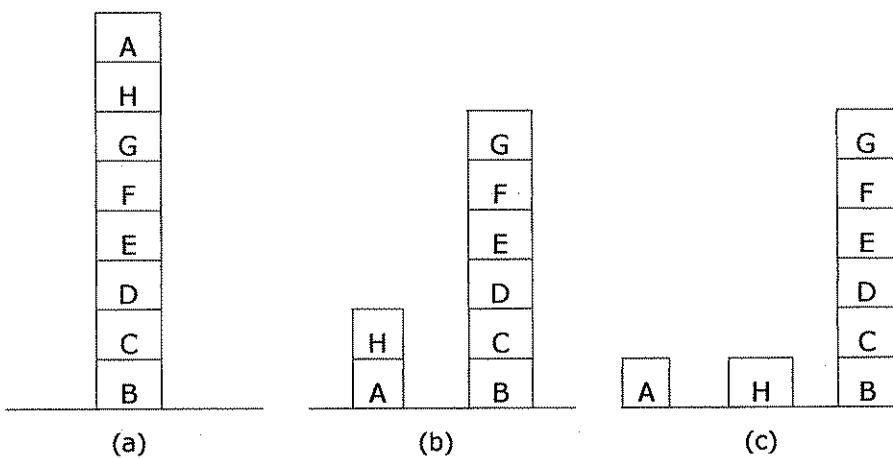
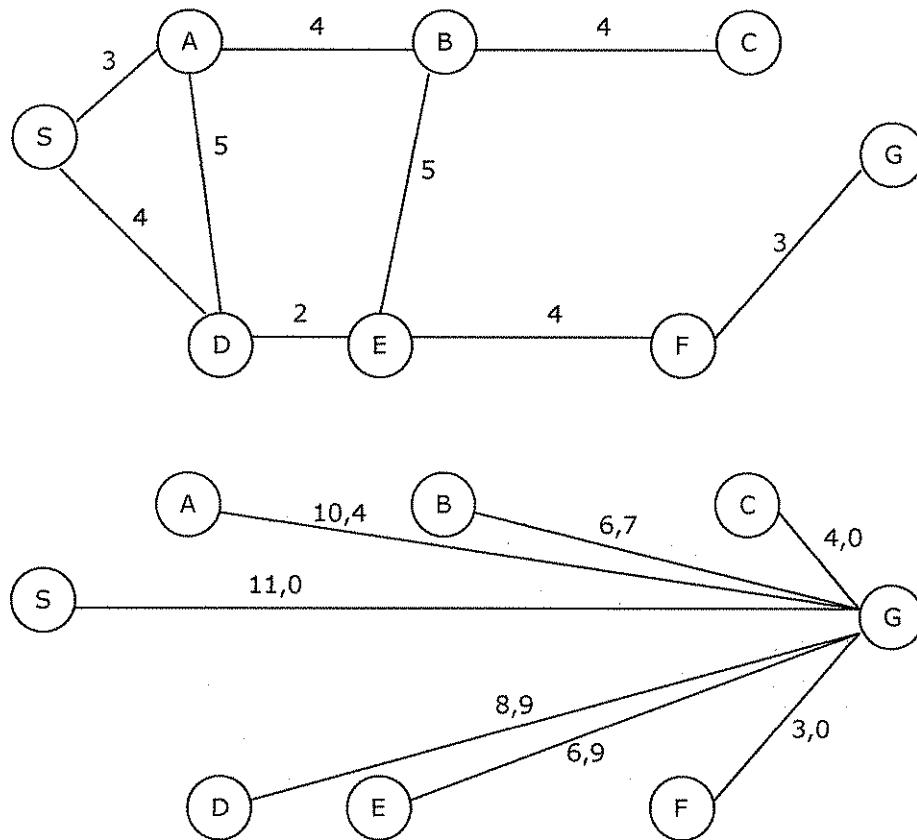


Figura 2.9

La culpa de este fallo podría recaer en el método de la escalada en sí mismo al no ser capaz de “mirar más lejos” para encontrar una solución, pero también se podría culpar a la función heurística y tratar de modificarla. La escalada puede resultar muy ineficiente con espacios problema grandes y escabrosos. Sin embargo, suele ser adecuado si se combina con otros métodos que consigan que se mueva correctamente por la vecindad en general.

### Ejercicio

Desarrollar un programa que resuelva el siguiente problema de trayectoria de nodos, aplicando la heurística de ir alcanzando nodos con la intención de minimizar la distancia hacia el objetivo, según los valores expuestos en el segundo esquema



#### 2.2.3. Búsqueda El primero mejor

Hasta este momento, solo se han explicado realmente dos estrategias de control sistemáticas, la búsqueda primero en anchura y la búsqueda primero en profundidad. Ahora se explica un nuevo método, la búsqueda de *el primero mejor* (*best-first search*), que representa una forma de combinar las ventajas que presentan tanto la búsqueda primero en anchura como la primera en profundidad.

La búsqueda primero en profundidad tiene la ventaja de que permite encontrar una solución sin tener que expandirse completamente por todas las ramas. La búsqueda primero en anchura presenta la ventaja de que no queda atrapada en callejones sin salida. Una forma de combinar ambas ventajas puede consistir en seguir un único camino cada vez, y cambiarlo cuando alguna ruta parezca más prometedora que la que se está siguiendo en ese momento.

En cada paso del proceso de búsqueda el primero mejor, se selecciona el nodo más prometedor que se haya generado hasta ese momento. Esto se puede conseguir con una función heurística apropiada. A continuación se expande el nodo elegido aplicando las reglas para generar a sus sucesores. Si alguno de ellos es una solución, el proceso termina. Si no es así, estos nuevos nodos se añaden a la lista de nodos que se han generado hasta ese momento. De nuevo, se selecciona el más prometedor de ellos y el proceso continúa de la misma forma. Lo normal es que la forma de funcionar se parezca un poco a la búsqueda primero en profundidad al explorar las ramas. Sin embargo, si no se encuentra una solución, la rama empezará a parecer menos prometedora que otras por encima de ella y que se habían ignorado. En este caso, una rama que previamente se había ignorado aparece ahora como la más prometedora y, por lo tanto, comienza su exploración. Sin embargo, la vieja rama no se

olvida. Su último nodo se almacena en el conjunto de nodos generados pero aún sin expandir. La búsqueda puede volver a él en el momento en que los otros sean lo suficientemente malos como para que éste sea de nuevo el camino más prometedor de todos.

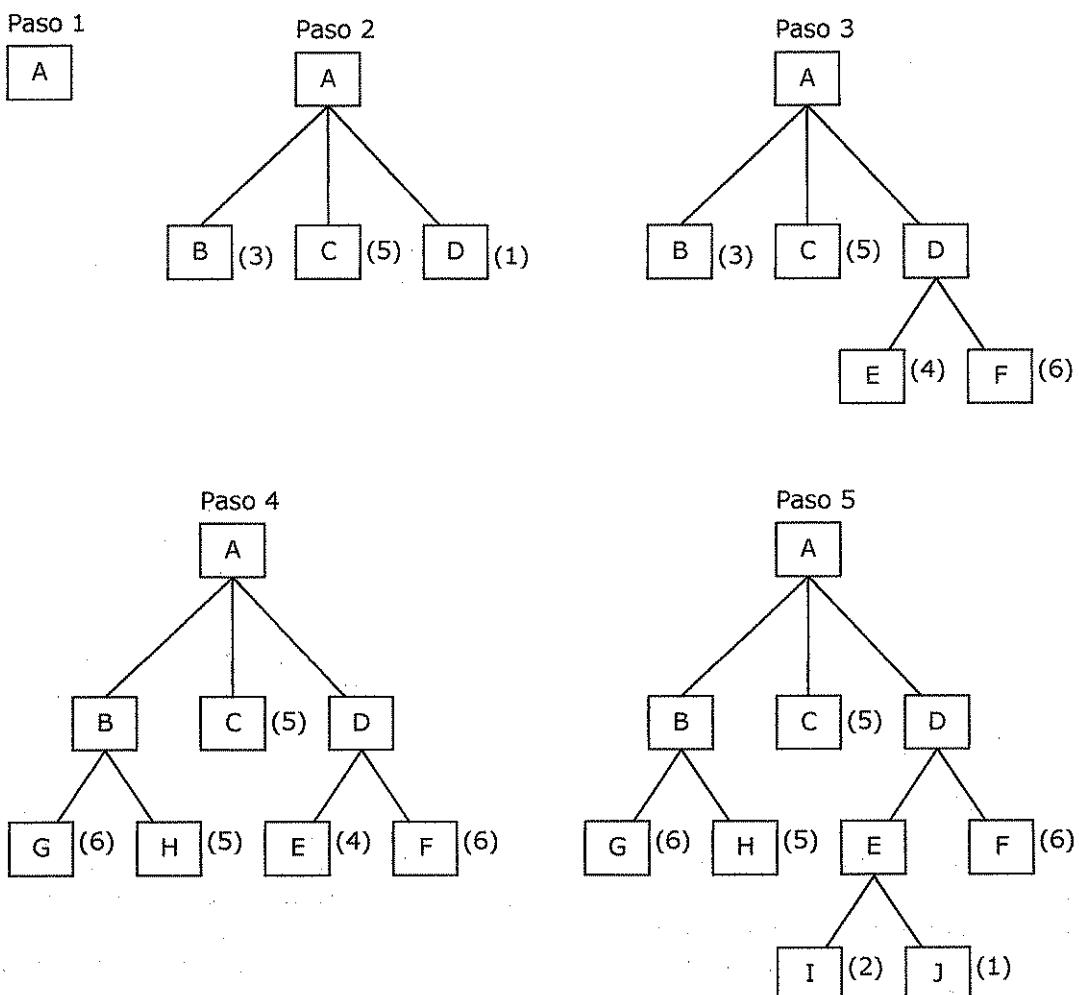


Figura 2.10

La Figura 2.10 muestra el comienzo de un proceso de búsqueda el primero mejor. Inicialmente, sólo existe un nodo, de forma que éste se expande. Al hacerlo, se generan tres nodos nuevos. En este ejemplo la función heurística es una estimación del coste necesario para llegar a una solución a partir del nodo dado, y se aplica a cada nodo. Como el nodo D es el más prometedor de todos, es el siguiente que se expande, lo que produce los nuevos nodos, E y F. La función heurística se aplica, de nuevo, a estos dos nodos. En este punto, el camino que nace del nodo B parece más prometedor, por lo que se selecciona y se generan los nodos G y H. De nuevo, al evaluar estos nuevos nodos se observa que son peores que algún otro, de forma que se vuelve la atención sobre el camino que va de D a E. Ahora se expande E, y aparecen los nodos I y J. En el siguiente paso, el nodo que se expande es J, ya que es el más prometedor de todos. El proceso continuaría así hasta que se encontrara alguna solución al problema.

Este procedimiento es *muy similar al de escalada por la máxima pendiente*, excepto en dos aspectos. En el método de la escalada, al seleccionar un movimiento todos los demás se abandonan y nunca pueden volver a ser considerados. Esta es la causa del característico comportamiento del método de escalada. *En el método de búsqueda de el primero mejor, se sigue seleccionando un movimiento, pero todos los demás se mantienen de forma que pueden visitarse si el camino que se ha seleccionado llega a ser menos prometedor.* Además de esto, *en la búsqueda de el primero mejor se selecciona el mejor estado disponible, aún si este estado tiene un valor menor que el del que se estaba explorando.* Esto contrasta con el método de la escalada, en donde el proceso detiene si no se encuentra un estado sucesor mejor que el estado actual.

A pesar de que el ejemplo anterior emplea un árbol para ilustrar el proceso de búsqueda el primero mejor, en ocasiones es importante realizar la búsqueda sobre un grafo de forma que los caminos duplicados no se exploren. Tal algoritmo debe realizar una búsqueda en un grafo dirigido donde cada nodo representa un punto en el espacio de estados. Cada nodo contiene además de una descripción de lo que representa en el espacio de estados, una indicación de lo prometedor que es, un enlace paterno que apunta al mejor nodo desde el que se ha generado el actual, y una lista de los nodos que se generan a partir de él. El enlace paterno nos posibilita restablecer el camino hacia el objetivo una vez que se ha encontrado dicho objetivo. La lista de sucesores hará posible, si se ha encontrado un camino mejor a un nodo ya existente, propagar la mejora a sus sucesores. A los grafos de este tipo se les denomina grafos O, ya que cada una de sus ramas representan caminos alternativos para la resolución del problema.

Para poder implementar un procedimiento de búsqueda sobre un grafo se necesitan dos listas de nodos:

- **ABIERTOS:** nodos que se han generado y a los que se les ha aplicado la función heurística, pero que aún no haya sido examinados (es decir, no se han generados sus sucesores). La lista ABIERTOS es, en realidad, una cola con prioridad en la que los elementos con mayor prioridad son aquellos que tienen un valor más prometedor de la función heurística. Para manipular la lista pueden utilizarse las técnicas usuales de manipulación de colas de prioridad.
- **CERRADOS:** nodos que ya se han examinado. Es necesario mantener estos nodos en memoria si lo que se desea es hacer una búsqueda sobre un grafo y no sobre un árbol, debido a que cuando se genera un nuevo nodo, se debe verificar si ese nodo se había generado con anterioridad.

También se necesita una función heurística que haga una estimación de los méritos de cada uno de los nodos que se van generando. Esto permite que el algoritmo examine primero los caminos más prometedores. Llámemos a esta función  $f'$  (para indicar que se trata de una aproximación a la función  $f$ , que es la que proporciona la verdadera evaluación de cada nodo). Para muchas aplicaciones, es adecuado definir esta función como la suma de dos componentes que denominamos  $g$  y  $h'$ . La función  $g$  es una medida del coste para ir desde el estado inicial hasta el nodo actual. Nótese que  $g$  no es una estimación de nada; su valor es exactamente la suma de los costes de aplicación de las reglas que se han ido eligiendo a través del mejor camino hasta el nodo. La función  $h'$  es una estimación del coste adicional necesario para alcanzar un nodo objetivo a partir del nodo actual. Este es el lugar en donde se emplea el conocimiento acerca del dominio del problema. La función combinada  $f'$ , representa entonces una estimación del coste necesario para alcanzar un estado objetivo por el camino que se ha seguido para generar el nodo actual. Si un nodo puede generarse por más de un camino, el algoritmo se queda solo con el mejor de ellos. Nótese que, ya que  $g$  y  $h'$  deben sumarse, es importante que  $h'$  represente una medida del coste de ir desde un nodo dado a una solución (es decir, los nodos buenos poseen valores bajos; los nodos malos tienen valores altos) en lugar de una medida de la bondad de un nodo (es decir, los buenos nodos tienen valores altos). Sin embargo, eso es fácil de arreglar gracias a la buena colaboración de los signos negativos. También es importante que  $g$  no sea negativa. Si esto no se cumple, los caminos que atraviesan ciclos a lo largo del grafo parecerán mejores conforme sean más largos.

El funcionamiento del algoritmo es muy simple. Funciona por pasos, expandiendo un nodo en cada paso hasta que se genere un nodo que se corresponde con un nodo objetivo. En cada paso se toma el nodo más prometedor que se tenga en ese momento y que no se haya

expandido. Se generan los sucesores del nodo elegido, se les aplica la función heurística y se les añade a la lista de nodos abiertos. Después de verificar si alguno de ellos se había generado con anterioridad. Al realizar esta verificación se puede garantizar que los nodos solo aparecen una vez en el grafo, aunque varios nodos puedan apuntar hacia él como su sucesor. Una vez hecho esto, se empieza con el siguiente paso.

El proceso puede resumirse como sigue.

### Algoritmo: Búsqueda el primero mejor

1. Comenzar con ABIERTOS conteniendo solo el estado inicial.
2. Hasta que se llegue a un objetivo o no queden nodos en ABIERTOS hacer:
  - a) Tomar el mejor nodo de ABIERTOS.
  - b) Generar sus sucesores.
  - c) Para cada sucesor hacer:
    - i. Si no se ha generado con anterioridad, evaluarlo, añadirlo a ABIERTOS y almacenar a su padre.
    - ii. Si ya se ha generado antes, cambiar al padre si el nuevo camino es mejor que el anterior. En este caso, se actualiza el coste empleado para alcanzar el nodo y a los sucesores que pudiera tener.

La idea básica de este algoritmo es sencilla. Desafortunadamente, es raro el caso en que los algoritmos sobre grafos resultan fáciles de escribir correctamente. Y es aún más raro que sea sencillo garantizar la corrección de tales algoritmos.

### Transformación en el espacio de configuraciones

La planificación de trayectoria de robot ilustra la búsqueda

Para ver como se puede llevar a la práctica el procedimiento de búsqueda el primero mejor, considere el problema de evitar choques que enfrenta un robot. Antes de que un robot empiece a moverse en un ambiente desordenado, debe *calcular una trayectoria entre el lugar donde se encuentra y aquel donde desea estar*. Este requisito es válido para la locomoción de todo robot a través de un medio desordenado y para el movimiento de la mano del robot a través de un espacio de trabajo lleno de componentes.

En la Figura 2.11 se muestra este problema de planificación de movimiento en un medio simple, habitado por un robot triangular. El robot desea moverse, sin hacer viraje, desde su posición inicial a la nueva posición indicada por el triángulo punteado. La pregunta que surge es ¿puede el robot pasar por el espacio que hay entre el pentágono y el octágono?

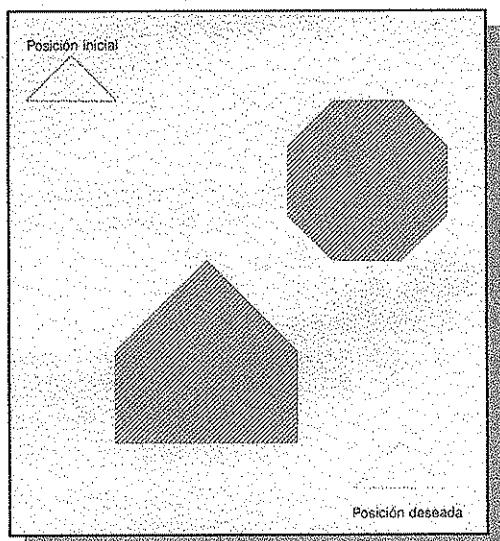


Figura 2.11

Problema de evasión de obstáculo. El problema consiste en mover el pequeño robot triangular a una nueva posición, que se muestra punteada, sin chocar con el pentágono o el octágono.

*En dos dimensiones, un truco inteligente simplifica el problema.* La idea general consiste en redescribir el problema en términos de otra representación más simple, resolver el problema en dicha representación y redescribir la solución en términos de la representación original. En conjunto, tomar este planteamiento es como hacer una multiplicación pasando de los números a sus logaritmos y viceversa.

Para evitar obstáculos, la representación original implica un objeto en movimiento y obstáculos estacionarios, y la nueva representación implica un punto en movimiento y obstáculos virtuales más grandes llamados **obstáculos del espacio de configuraciones**.

En la Figura 2.12 se muestra como se puede transformar un obstáculo ordinario en un obstáculo del espacio de configuraciones utilizando el objeto que se va a mover y el obstáculo que se va a evitar. Básicamente, se desliza el objeto alrededor del obstáculo, manteniendo el contacto entre ellos todo el tiempo, y llevando el registro de un punto de rastreo arbitrario en el objeto en movimiento conforme avanza. Mientras el punto de rastreo se mueve alrededor del obstáculo, se traza una cerca de ocho lados. Es evidente que no puede haber colisión entre el objeto y el obstáculo mientras el punto de rastreo se mantenga fuera de la cerca que rodea al obstáculo del espacio de configuraciones asociado al obstáculo original.

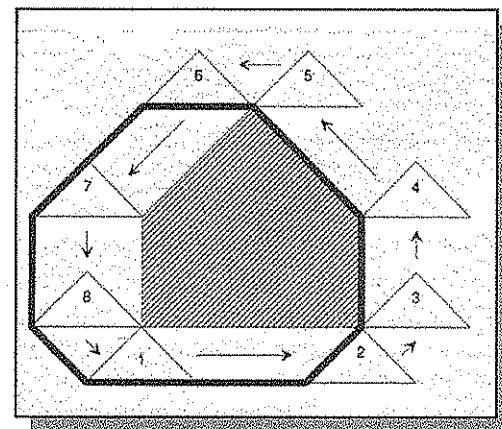


Figura 2.12

Transformación en el espacio de configuraciones. La línea gruesa muestra la posición del vértice inferior izquierdo del triángulo pequeño conforme éste se mueve alrededor del grande. Las posiciones numeradas son los puntos iniciales de cada desplazamiento en línea recta. Si se mantiene el vértice inferior izquierdo fuera de la línea gruesa se mantendrá el triángulo pequeño alejado del pentágono.

En la Figura 2.13 se muestran los dos obstáculos del espacio de configuraciones formados a partir del triángulo original, el pentágono y el octágono de la Figura 2.11. Se utilizó el vértice inferior izquierdo del triángulo. Es evidente que el robot puede pasar por la abertura, ya que los obstáculos del espacio de configuraciones no son lo suficientemente grandes como para cerrar el espacio.

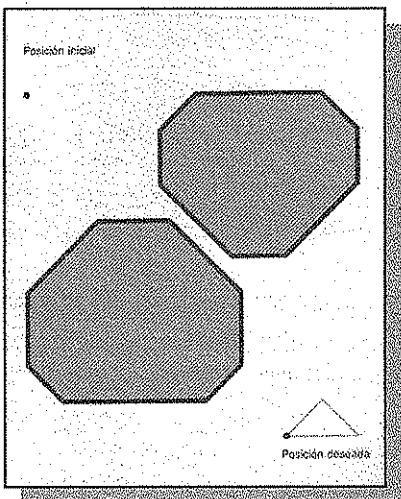


Figura 2.13

Espacio de configuraciones para el problema que se muestra en la Figura 2.11. Si el punto se mantiene fuera del área sombreada no habrá colisión.

*Sin embargo, no queda del todo claro si la trayectoria más corta es a través de la abertura. Para asegurarse que de veras lo es, usted tiene que efectuar una búsqueda.*

Hasta ahora no existe una red en donde buscar. Sin embargo, su construcción resulta fácil para problemas de espacio de configuraciones de dos dimensiones. Para ver por qué es fácil, suponga que usted se encuentra en cualquier punto de un espacio de configuraciones. Desde donde usted está, puede ver la posición deseada o bien no verla. Si puede, entonces no requerirá hacer más, ya que la trayectoria más corta es la línea recta entre usted y la posición deseada.

Si no puede ver la posición deseada desde donde está, entonces el único movimiento que tiene sentido es hacia uno de los vértices que puede haber. En consecuencia, todo movimiento queda restringido de vértice a vértice, excepto al principio, cuando el movimiento es de la posición inicial a un vértice, y al final, cuando el movimiento es de un vértice a la posición deseada. Por lo tanto, la posición inicial, la deseada, y los vértices son como los nodos de una red. Debido a que los enlaces entre los nodos se colocan solo cuando existe una línea de visión sin obstáculos entre los nodos, la red se conoce como grafo de visibilidad.

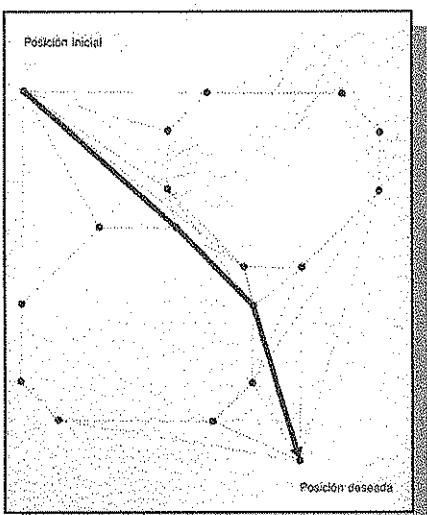


Figura 2.14

En un espacio de configuraciones de dos dimensiones, el punto robot se mueve a lo largo de las líneas rectas de un grafo de visibilidad. Una búsqueda A\* a través del grafo de visibilidad

produce la trayectoria más corta desde la posición inicial a la posición deseada. La flecha gruesa muestra la trayectoria más corta.

En la Figura 2.14 se ilustra el grafo de visibilidad para el ejemplo del movimiento del robot, junto con el resultado de efectuar una búsqueda A\* para establecer la trayectoria más corta para el robot del espacio de configuraciones -un punto- a través del espacio de obstáculos del espacio de configuraciones, todos ellos de forma extraña. En la Figura 2.15 se muestra el movimiento del robot real, junto con los obstáculos reales que rodea, todos superpuestos a la solución en el espacio de configuraciones.

Si usted permite que el objeto en movimiento gire, puede hacer que varios espacios de configuraciones correspondan a los diferentes grados de rotación del objeto en movimiento. Entonces, la búsqueda implicará movimiento no solo a través del espacio de configuración individual, sino de espacio a espacio.

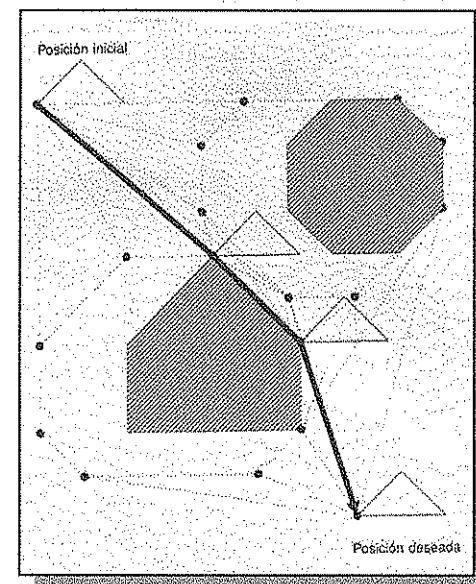


Figura 2.15

*El movimiento del robot está regido por la trayectoria más corta hallada en el grafo de visibilidad. El vértice inferior izquierdo del robot triangular, que se utilizó para producir obstáculos del espacio de configuraciones, se mueve a lo largo de la trayectoria más corta. Observe que el robot triangular nunca choca con el pentágono ni con el octágono.*

De forma más general aun, cuando usted necesita mover un brazo que sostiene un objeto, pero en tres dimensiones en lugar de dos, la construcción de un espacio de configuraciones adecuado resulta extremadamente complicada desde el punto de vista matemático. Los matemáticos amantes de la complicación han producido una copiosa bibliografía sobre la materia.

## 2.2.4. Reducción de problemas

Algunas veces es posible convertir metas difíciles en una o más submetas más fáciles de lograr. Cada submeta, a su vez, puede dividirse aun más finamente en una o más submetas de nivel inferior.

Cuando se usa el método de reducción de problemas, por lo general se reconocen las metas y se las convierte en submetas apropiadas. Cuando se usa de ese modo, la reducción del problema se conoce a menudo y de manera equivalente como *reducción de metas*.

### Los cubos en movimiento ilustran la reducción de problemas.

El procedimiento MOVER resuelve problemas de manipulación de cubos y contesta preguntas acerca de su propio comportamiento. MOVER trabaja con cubos como el que se muestra en la Figura 2.16, obedeciendo mandatos como el siguiente:

Coloca <nombre del cubo> sobre <otro nombre de cubo>.

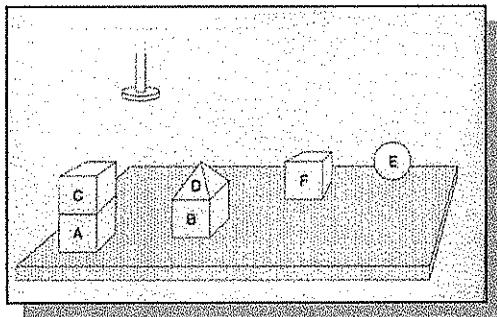


Figura 2.16

Para obedecer, MOVER planea una sucesión de movimientos para un robot de una sola mano que toma solo un cubo a la vez. MOVER consiste en *procedimientos que reducen los problemas dados a otros más simples, aventurándose así en lo que se conoce como reducción del problema*. De manera conveniente, los nombres de estos procedimientos son elementos mnemotécnicos para los problemas que el procedimiento reduce. En la Figura 2.17 se muestra como se ajustan los procedimientos.

- **COLOCA** hace que se ponga un cubo encima de otro. Funciona mediante la activación de otros procedimientos que hallan un lugar específico en la cima del cubo destino, toman al cubo que se va a desplazar, lo mueven y lo sueltan en el lugar especificado.
- **CONSIGUE ESPACIO** encuentra lugar en la cima de un cubo destino para el cubo en movimiento.
- **HAZ ESPACIO** ayuda a CONSIGUE ESPACIO, cuando es necesario, moviendo obstáculos hasta que hay suficiente espacio para el cubo en movimiento.
- **TOMA** agarra cubos. Si la mano robot está asiendo un cubo cuando se llama a TOMA, éste debe hacer que el robot se deshaga de ese cubo. También, TOMA debe hacer que se despeje la cima del objeto que se va a agarrar.
- **DESPEJA CIMA** limpia la cima. Opera deshaciéndose de todo lo que halla en la cima del objeto que se va a tomar.
- **DESHAZTE DE** aparta obstáculos poniéndolos sobre la mesa.
- **SUELTA** hace que el robot suelte lo que está asiendo.
- **MUEVE** traslada objetos, una vez que han sido agarrados, mediante el movimiento de la mano robot.

Ahora imagínese que una petición es colocar el cubo A sobre el B, dada la situación que se muestra en la Figura 2.16. Evidentemente bastaría con la secuencia siguiente.

- Toma D.
- Mueve a D a algún lugar sobre la mesa.
- Suelta a D.
- Toma a C.
- Mueve a C a algún lugar sobre la mesa.
- Suelta a C.
- Toma a A.
- Mueve a A a algún lugar sobre B.
- Suelta a A.

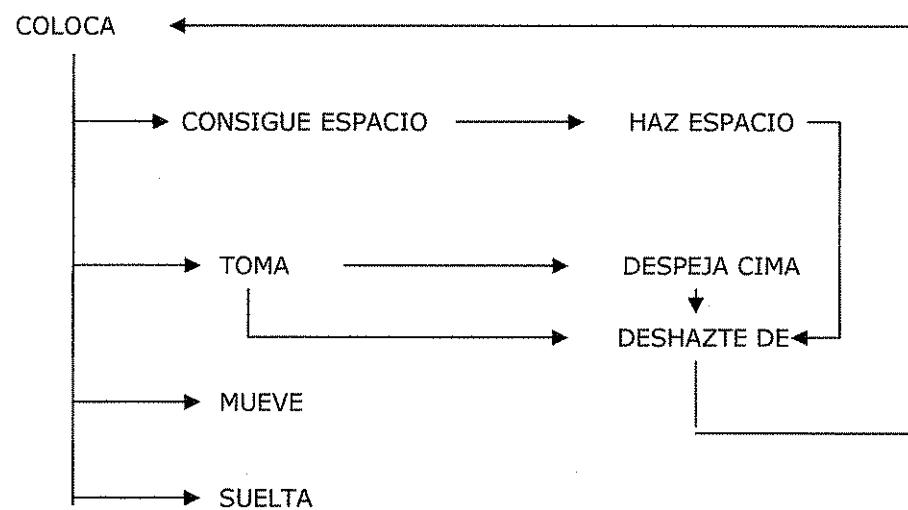


Figura 2.17

La pregunta es: *¿cómo encuentran los procedimientos de MOVER la secuencia apropiada? He aquí la respuesta:*

Primero COLOCA pide a CONSIGUE ESPACIO que identifique un lugar para el cubo A encima de B. CONSIGUE ESPACIO se dirige a HAZ ESPACIO porque el cubo D está estorbando.

HAZ ESPACIO le pide a DESHAZTE DE que le ayude a desembarazarse del cubo D. DESHAZTE DE accede y encuentra sitio para el cubo D sobre la mesa y traslada a D a dicho lugar utilizando a COLOCA.

Note que COLOCA, que se encuentra colocando el cubo A sobre B, finalmente produce una nueva tarea para él mismo, en esta ocasión para colocar el bloque C sobre la mesa. Cuando un procedimiento se utiliza a sí mismo, se dice que recurre. Los sistemas en que los procedimientos se usan a sí mismos se conocen como recursivos.

Una vez retirado el cubo D, HAZ ESPACIO puede encontrar lugar para el cubo A encima del cubo B. Recuerde que se pidió a HAZ ESPACIO que hiciera esto mediante CONSIGUE ESPACIO porque COLOCA tiene la tarea de situar el cubo A sobre B. COLOCA puede proseguir ahora, pidiéndole a TOMA que agarre el cubo A. Pero TOMA se da cuenta que no puede hacerlo porque el cubo C está en el camino. TOMA llama a DESPEJA CIMA para que le ayude. DESPEJA CIMA, a su vez, solicita la ayuda de DESHAZTE DE, con lo que DESHAZTE DE hace que el cubo C se traslade a la mesa mediante COLOCA.

Una vez que se ha despejado el cubo A, DESPEJA CIMA termina su labor. Pero si hubiera muchos cubos encima de A, y no uno solo, DESPEJA CIMA se dirigiría a DESHAZTE DE muchas veces, en lugar de una.

Ahora TOMA puede hacer su trabajo y COLOCA puede pedir a MUEVE que traslade el bloque A al lugar encontrado previamente encima de B. Finalmente COLOCA pide a SUELTA que libere el bloque A.

**La idea clave del método de Reducción de Problemas es explorar un árbol de metas.**

Un árbol de metas, como el que se muestra en la Figura 2.18, es un árbol semántico en el que los nodos representan metas y las ramas indican la forma en que usted puede lograr metas, mediante la solución de una o más submetas. Los hijos de cada nodo corresponden a submetas inmediatas; cada padre de nodo corresponde a la supermeta inmediata. El nodo superior, que no tiene padre, es la meta raíz.

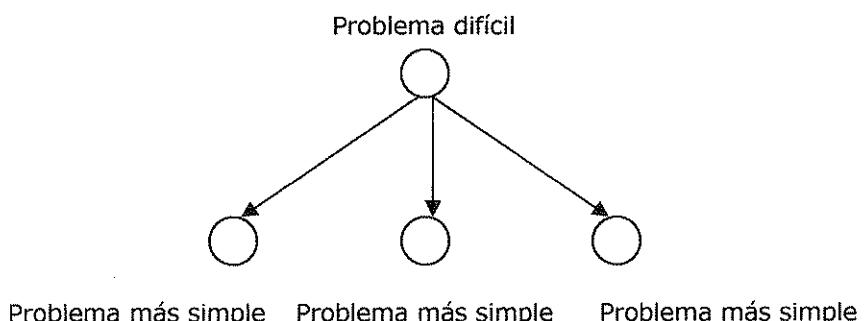


Figura 2.18

Un *árbol de metas*, como el de la Figura 2.19, hace transparentes los complicados argumentos de MOVER. La acción de despejar la cima del cubo A se muestra como una submeta inmediata de tomar el cubo A. Despejar la cima del cubo A es también una submeta de colocar al cubo A en algún lugar encima del cubo B, pero no se trata de una submeta inmediata.

*Todas las metas que se muestran en el ejemplo se satisfacen solo cuando todas las submetas inmediatas quedan satisfechas. Las metas que se satisfacen solo cuando todas sus submetas inmediatas quedan satisfechas se conocen como metas Y. Los nodos correspondientes son los nodos Y, y se les señala colocando arcos en sus ramas.*

*La mayoría de los árboles de metas contienen también metas O; estas metas se satisfacen cuando cualesquiera de sus submetas inmediatas quedan satisfechas. Los nodos correspondientes, que permanecen sin señalar, se conocen como nodos O.*

*Finalmente, algunas metas se satisfacen directamente, sin hacer referencia a ninguna submeta. Estas metas se conocen como metas hoja, y los nodos correspondientes se denominan nodos hoja.*

Debido a que los árboles de metas siempre implican nodos Y, o nodos O, o ambos, a menudo se les conoce como **árboles Y-O**.

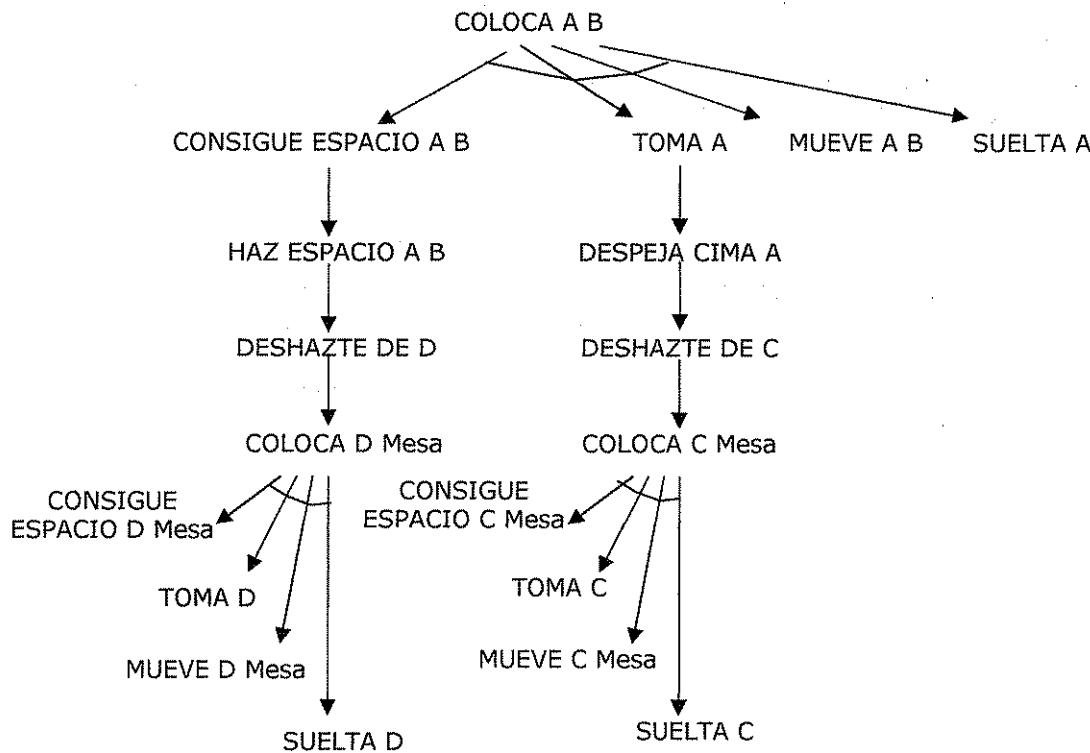


Figura 2.19

**El árbol de metas permite responder a preguntas ¿Cómo? y ¿Por qué?**

El programa MOVER es capaz de construir árboles Y-O ya que los especialistas mantienen una estrecha correspondencia con metas identificables. De hecho, los árboles Y-O de MOVER, son tan ilustrativos que pueden utilizarse para responder preguntas sobre cómo y por qué se han tomado las acciones, otorgando a MOVER cierto talento para realizar una introspección en su propio comportamiento.

Suponga, por ejemplo, que MOVER coloca el cubo A sobre el B, produciendo el árbol de metas que se muestra en la Figura 2.19.

Además, suponga que alguien pregunta: *¿Cómo limpiaste la cima de A?* Evidentemente, una respuesta razonable sería: deshaciéndome del cubo C. Por otro lado, suponga que la pregunta es: *¿por qué despejaste la cima de A?* Entonces una respuesta razonable sería: para tomar el cubo A.

Estos ejemplos ilustran estrategias generales. Para tratar con preguntas de "**¿cómo?**", usted identifica en el árbol Y-O la meta implicada. Si la meta es una Y, usted da a conocer todas las submetas inmediatas. Si la meta es una O, menciona la submeta inmediata que se logró. Para tratar con preguntas de "**¿Por qué?**", usted identifica la meta y notifica la supermeta inmediata.

### 2.2.5. Verificación de restricciones

Muchos de los problemas de IA pueden contemplarse como problemas de *verificación de restricciones* (*constraint satisfaction*) donde el objetivo consiste en **descubrir algún estado del problema que satisfaga un conjunto dado de restricciones**. Ejemplos de este tipo de problemas incluyen *rompecabezas criptoaritméticos*. El diseño de tareas puede contemplarse también como problemas de verificación de restricciones en los que el diseño debe realizarse dentro de unos límites fijos de tiempo, coste y materiales.

Al contemplar un problema como una verificación de restricciones, es frecuentemente posible una reducción sustancial en la cantidad de búsquedas que se necesitan si se compara con un método que intente formar directamente soluciones parciales mediante la elección de valores específicos para los componentes de una eventual solución. Por ejemplo, un procedimiento directo de búsqueda para resolver un problema criptoaritmético podría trabajar en un espacio de estados de soluciones parciales en las que a las letras se les asignan números, que serían sus valores. Entonces, un esquema de control primero en profundidad podría seguir un camino de asignaciones hasta descubrir una solución o una inconsistencia. En contraste con esto, un enfoque del tipo de verificación de restricciones para la resolución de este problema evita *hacer suposiciones o asignaciones de números a letras hasta que sea necesario*. En lugar de esto, el conjunto inicial de restricciones, las cuales indican que a cada número puede corresponderle solo una letra y que la suma de los dígitos debe ser la que se da en el problema, se aumenta primero para incluir las restricciones que pudieran deducirse de las reglas de la aritmética. Entonces, aunque se necesiten aún las suposiciones, el número de las que son permitidas se va reduciendo conforme la búsqueda se va restringiendo.

La verificación de restricciones es un procedimiento de búsqueda que funciona en un espacio de conjuntos de restricciones. El estado inicial contiene las restricciones que se dan originalmente en la descripción del problema. Un estado objetivo es aquel que ha satisfecho las restricciones "suficientemente", donde "suficientemente" debe definirse para cada problema en particular. Por ejemplo, para la criptoaritmética suficientemente significa que a cada letra se le ha asignado un único valor.

*El proceso de verificación de restricciones consta de dos pasos. Primero, se descubren las restricciones y se propagan tan lejos como sea posible a través del sistema. Entonces, si todavía no hay una solución, la búsqueda comienza. Se hace una suposición sobre algo y reañade como una nueva restricción. Entonces, la propagación continúa con esta nueva restricción, y así sucesivamente.*

El primer paso, la **propagación**, se hace necesario por el hecho de que normalmente existen dependencias entre las restricciones. Estas dependencias aparecen porque muchas restricciones hacen referencia a más de un objeto, y muchos objetos participan en más de una restricción. Así, por ejemplo, supongamos que la primera restricción es  $N = E + 1$ . Entonces, si se añade la restricción  $N = 3$ , podría conseguirse una restricción más fuerte para  $E$ , que sería  $E = 2$ . La propagación de restricciones también surge debido a la presencia de reglas de inferencia que permiten que se infieran restricciones adicionales a partir de las que se tenían.

*La propagación de restricciones termina por una razón de entre dos posibles.*

- La primera, porque se detecte una contradicción. Si esto ocurre, entonces no existe una solución consistente con todas las restricciones conocidas. Si la contradicción se refiere únicamente a aquellas restricciones que se dan como parte de la especificación del problema (en oposición a aquellas que se creaban como suposiciones a lo largo de la resolución del problema), entonces no existe solución.
- La segunda razón posible para que termine el proceso es que la propagación se realice de tal forma que no puedan hacerse más cambios basándose en el conocimiento actual que se posea. Si ocurre así, y todavía no se ha especificado adecuadamente una solución, entonces será necesario que se realice algún proceso de búsqueda para desbloquear el proceso.

En este punto, comienza el **segundo paso**. Para ver la forma de fortalecer las restricciones, pueden realizarse algunas **hipótesis**. En el caso del problema criptoaritmético, por ejemplo, significa hacer suposiciones sobre un valor para una cierta letra. Una vez que se ha hecho,

debe comenzar de nuevo la propagación de las restricciones a partir de este nuevo estado. Si se encuentra una solución, se muestra.

Si aún se necesitan más restricciones, se hacen. Si se detecta alguna contradicción, puede usarse una *vuelta-atrás* para intentarlo con una suposición diferente y comenzar con ella.

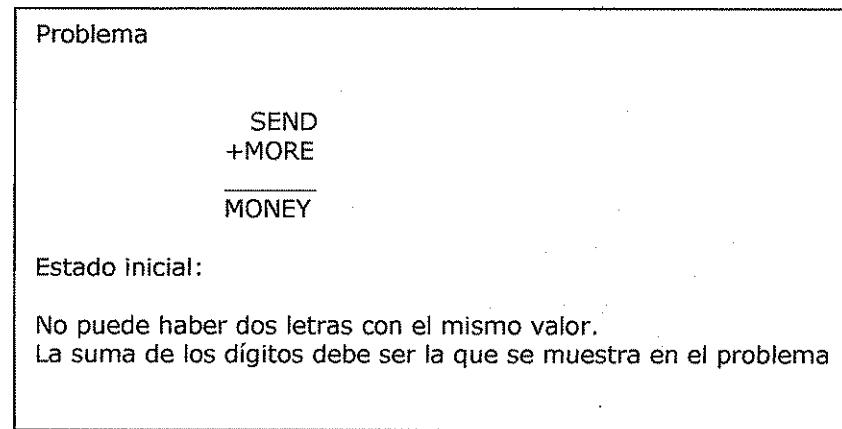


Figura 2.20

Considere el problema criptaritmético que se muestra en la Figura 2.20. El estado objetivo lo forma un estado problema en el que todas las letras tengan asignado un número, de forma que se verifiquen todas las restricciones iniciales.

El proceso de resolución del problema funciona a base de ciclos. En cada ciclo se realizan dos acciones significativas:

1. Se propagan las restricciones mediante el uso de reglas correspondientes a propiedades aritméticas.
2. Se le da un valor a alguna letra cuyo valor no haya sido aún determinado.

Inicialmente, las reglas de propagación de restricciones generan las siguientes restricciones adicionales:

- $M = 1$ , puesto que dos números de un solo dígito cada uno de ellos más un acarreo no pueden totalizar más de 19.
- $S = 8 \text{ o } 9$ , ya que  $S+M+C_3 > 9$  (para que se genere el acarreo) y  $M = 1$ ,  $S+1+C_3 > 9$ , es decir,  $S+C_3 > 8$  y  $C_3$  es al menos 1.
- $O = 0$ , ya que  $S+M(1)+C_3 (<=1)$  debe ser al menos 10 para poder generar un acarreo y puede ser como mucho 11. Pero como  $M$  es 1, entonces  $O$  debe ser 0.
- $N = E \text{ o } E+1$ , dependiendo del valor de  $C_2$ . Pero  $N$  no puede tener el mismo valor que  $E$ . Así,  $N = E+1$  y  $C_2$  es 1.
- Para que  $C_2$  sea 1, la suma de  $N+R+C_1$  debe ser mayor de 9, por lo que  $N+R$  debe ser mayor de 8.
- $N+R$  no puede ser mayor de 18, con su acarreo, y por lo tanto,  $E$  no puede ser 9.

En este momento, se asume que no pueden generarse más restricciones. Para poder progresar a partir de aquí, se deben hacer suposiciones. Suponga que a  $E$  se le asigna el valor 2. (Se elige  $E$  porque aparece tres veces y, por lo tanto, interacciona mucho con las otras letras). Ahora comienza el segundo ciclo.

El propagador de restricciones observa lo siguiente:

- $N = 3$ , ya que  $N = E+1$ .
- $R = 8 \text{ o } 9$ , ya que  $R+N(3)+C_1(1 \text{ o } 0) = 2 \text{ o } 12$ . Pero como  $N$  ya es 3, la suma de estos números no negativos no puede ser menor de 3. De esta forma,  $R+3+(0 \text{ o } 1) = 12$  y  $R = 8 \text{ o } 9$ .

- $2+D = Y$  o  $2+D = 10+Y$ , de la suma de la columna más a la derecha.

De nuevo se asume que no pueden generarse más restricciones, y es necesaria una suposición. Suponga que se elige  $C_1$  para darle un valor. Si se le da el valor 1, se llega a un callejón sin salida, tal y como se muestra en la figura. Cuando esto ocurra, el proceso debe volver atrás e intentar  $C_1=0$ .

Este proceso merece que se hagan un par de observaciones. Obsérvese que todo lo que se necesita de la propagación de restricciones es que no produzca falsas restricciones. No es necesario que produzca todas las legales. Por ejemplo, se podría haber llegado a la conclusión de que  $C_1$  era 0. Para llegar a esta conclusión se podría haber observado que si  $C_1$  era 1, aparecería lo siguiente:  $2+D=10+Y$ . Pero si este es el caso, D tendría que ser 8 o 9. Pero tanto S como R deben ser 8 o 9 y las tres letras no pueden compartir dos valores. De esta forma,  $C_1$  no puede ser 1. Si se hubiera realizado esto inicialmente, se habría evitado trabajo de búsqueda. Pero como se ha hecho necesario acudir a la búsqueda, el hecho de que la parte de búsqueda consuma más o menos tiempo que la propagación de restricciones, depende de lo que se gaste en llevar a cabo el razonamiento necesario para la propagación de restricciones.

Una segunda observación a tener en cuenta consiste en que normalmente existen dos tipos de restricciones. Las primeras son sencillas: son una lista de posibles valores para un objeto. Las del segundo tipo son más complejas: describen relaciones entre o en medio de objetos. Los dos tipos de restricciones desempeñan el mismo papel en el proceso de verificación de restricciones, y en el ejemplo criptoaritmético se han tratado de forma idéntica. En algunos problemas, sin embargo, puede resultar útil la representación diferenciada de los dos tipos de restricciones. En las sencillas, las listas de valores de las restricciones son siempre dinámicas, por lo que deben representarse explícitamente en cada estado del problema. En las más complicadas, las restricciones que expresan relaciones son dinámicas en el dominio criptoaritmético, ya que son diferentes para cada problema criptoaritmético. Pero en otros muchos dominios éstas son estáticas.

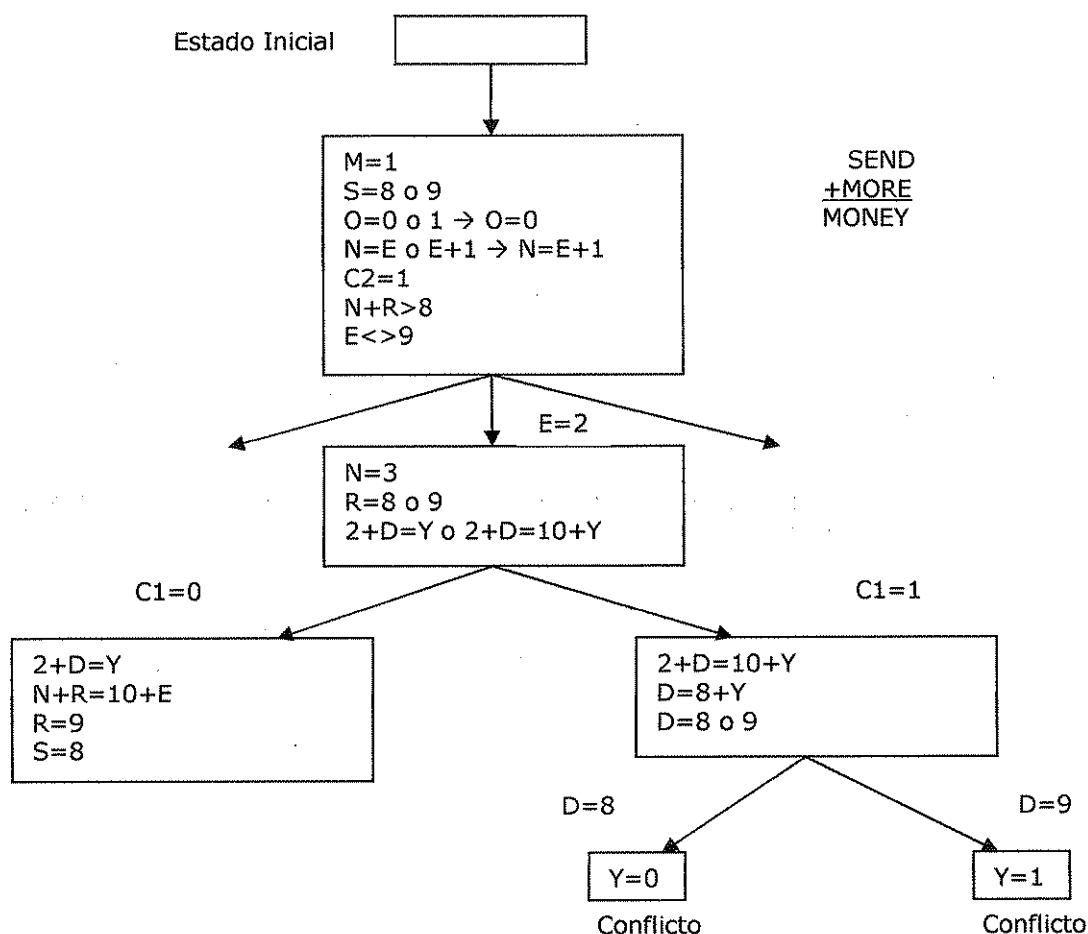


Figura 2.21

## Ejercicio

Desarrollar un programa que resuelva el problema criptoaritmético planteado.

### 2.2.6. Análisis de medios y fines

Hasta ahora, se ha presentado una colección de estrategias de búsqueda que pueden razonar tanto hacia delante como hacia atrás, pero para un problema dado, se debe elegir una dirección u otra. No obstante, a menudo es apropiado una mezcla de ambas direcciones. Esta estrategia mixta haría posible la resolución, en primer lugar, de las principales partes de un problema y, después, volver atrás y resolver los pequeños problemas que surgen al "pegar" juntos los trozos grandes. Una técnica conocida como *análisis de medios y fines (means-ends analysis)* supone una ayuda para lograrlo.

El proceso de *análisis de medios y fines* se centra en la *detección de diferencias entre el estado actual y el estado objetivo*.

Una vez que se ha aislado una diferencia, debe *encontrarse un operador que pueda reducirla*. Es posible que tal operador no pueda aplicarse en el estado actual por lo tanto, se crea el subproblema que consiste en *alcanzar un estado en que pueda aplicarse dicho operador*. Este tipo de encadenamiento hacia atrás, en donde se seleccionan los operadores y se producen subobjetivos para establecer las precondiciones del operador, recibe el nombre de *realización de subobjetivos para un operador*.

Sin embargo, es posible que el operador no produzca exactamente el estado objetivo que se desea. En este caso, se tiene un *segundo subproblema* que consiste en *llegar desde ese estado hasta un objetivo*. Pero si se ha elegido correctamente la diferencia y el operador es realmente eficaz al reducir la diferencia, estos dos subproblemas serán más fáciles de resolver que el problema original.

El proceso de análisis de medios y fines puede entonces aplicarse recursivamente. Para centrar la atención del sistema en los problemas grandes, a las diferencias se les asignan niveles de prioridad. Las diferencias de prioridad mayor deben considerarse antes que las de menor prioridad.

El primer programa de IA que utilizó un análisis de medios y fines fue el Resolutor General de Problemas (GPS) (Newell y Simon, 1963). El diseño del sistema estuvo motivado por la observación de las técnicas que usa la gente cuando resuelve un problema. Sin embargo, GPS es un claro ejemplo de lo difuso que resulta el límite entre la construcción de programas que simulan la forma de trabajar humana y la construcción de programas que simplemente resuelven problemas como pueden.

Al igual que en otras técnicas de resolución de problemas que se han explicado, el análisis de medios y fines cuenta con *un conjunto de reglas que pueden transformar un estado problema en otro*. Estas reglas normalmente no se representan con las descripciones completas de los estados en cada uno de sus lados. En lugar de esto, se representan con *un lado izquierdo* que describe las condiciones que deben cumplirse para que pueda aplicarse la regla (a estas condiciones se les denomina *precondiciones de la regla*), y *un lado derecho* que describe aquellos *aspectos del estado problema que cambiarán al aplicar la regla*. Existe una estructura de datos separada denominada *tabla de diferencias* que ordena las reglas atendiendo a las diferencias que pueden reducir.

Operador	Precondiciones	Resultados
EMPUJAR(obj,lug)	en(robot,obj) ^ grande(obj) ^ despejado(obj) ^ brazo_vacio	en(obj,lug) ^ en(robot,lug)
LLEVAR(obj,lug)	en(robot,obj) ^ pequeño(obj)	en(obj,lug) ^ en(robot,lug)
ANDAR(lug)	ninguna	en(robot,lug)
COGER(obj)	en(robot,obj)	sostiene(obj)
DEJAR(obj)	sostiene(obj)	¬sostiene(obj)
COLOCAR(obj1,obj2)	en(robot,obj2) ^ sostiene(obj1)	sobre(obj1,obj2)

Figura 2.22

### Operadores del robot

Consideré el caso del dominio de un sencillo robot doméstico. Los operadores de que se dispone se muestran en la Figura 2.22, así como sus precondiciones y resultados. La Figura 2.23 muestra la tabla de diferencias que describe cuando es apropiado cada operador.

Obsérvese que algunas veces existe más de un operador capaz de reducir una diferencia dada, y que un operador dado puede ser capaz de reducir más de una diferencia.

	Empujar	Llevar	Andar	Coger	Dejar	Colocar
Mueve objeto	*	*				
Mueve robot			*			
Despeja objeto				*		
Pone objeto en objeto						*
Vacía brazo					*	*
Sujeta objeto				*		

Figura 2.23

Tabla de diferencias

Suponga que dado su dominio, se le proporciona al robot el problema de mover un escritorio de una habitación a otra, con dos objetos encima de él. Los objetos situados sobre el escritorio deben moverse también. La principal diferencia entre el estado actual y el estado objetivo debería ser la situación del escritorio. Para reducir esta diferencia podría escogerse o bien EMPUJAR o bien LLEVAR. Si se escoge en primer lugar LLEVAR, deben encontrarse sus precondiciones. Esto proporciona dos diferencias más que deben reducirse: la localización del robot y el tamaño del escritorio. La situación del robot puede manipularse con la aplicación de ANDAR, pero no existen operadores que cambien el tamaño de un objeto (puesto que no se incluye TROCEAR-SEPARAR). Por lo tanto, este camino conduce a un callejón sin salida. Si se sigue por la otra rama, se intenta aplicar EMPUJAR.

La Figura 2.24 muestra el progreso realizado por el resolutor del problema en este punto. Se ha encontrado la forma de hacer algo útil. Pero aún no está en la posición correcta para realizarlo. Además tampoco se llega al estado objetivo. Por lo tanto, ahora hay que reducir las diferencias entre A y B y entre C y D.

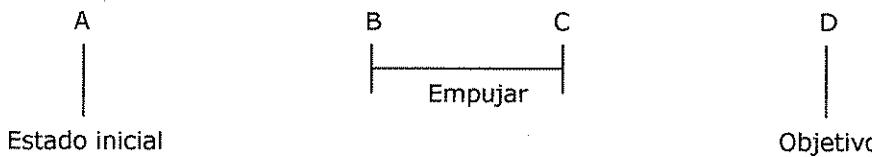


Figura 2.24  
Progreso del método de análisis de medios y fines

EMPUJAR tiene cuatro precondiciones, dos de las cuales producen diferencias entre los estados inicial y objetivo: el robot debe situarse en el escritorio y el escritorio debe estar despejado. Puesto que el escritorio ya es grande y el brazo del robot está vacío, estas dos precondiciones se pueden ignorar. Se puede llevar al robot a la posición correcta utilizando CAMINAR, y con dos aplicaciones de COGER se puede despejar la superficie del escritorio. Pero después de ejecutar el primer COGER, el intento de realizar un segundo produce otra diferencia- el brazo debe estar vacío. Para reducir esta diferencia puede usarse DEJAR. Una vez que se ha realizado EMPUJAR, el estado problema está más cerca del estado objetivo, pero no mucho. Los objetos deben volver a ponerse sobre el escritorio. COLOCAR los pondrá allí de nuevo, pero no puede aplicarse inmediatamente. Debe eliminarse otra diferencia ya que el robot debe estar sosteniendo los objetos. En la Figura 2.25 se muestra el progreso realizado por el resolutor de problemas. La diferencia final entre C y E puede reducirse aplicando ANDAR para que el robot vuelva hacia los objetos, seguido de COGER y LLEVAR.

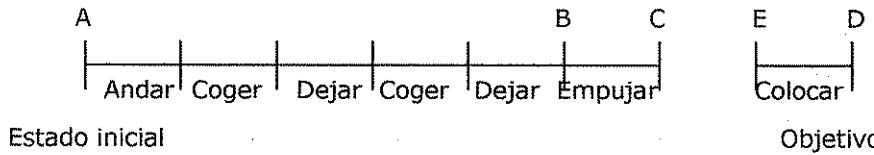


Figura 2.25  
Más progresos del método de medios y fines

## 2.3. Búsqueda en Problemas de juegos

### 2.3.1. Juegos de dos jugadores

#### Visión de conjunto

Para mucha gente los juegos provocan una inexplicable fascinación, y la idea de que las computadoras pudieran jugar ha existido al menos desde que existen las computadoras. Charles Babbage, el famoso arquitecto de computadoras del siglo XIX, pensó en programar su máquina analítica para que jugara al ajedrez, y más tarde pensó en construir una máquina para jugar a las tres en raya (Bowden, 1953). Dos de los pioneros en las ciencias de la información y de la computación contribuyeron a la incipiente literatura sobre juegos por computadora. Claude Shannon (1950) escribió un artículo en el que se describían los mecanismos que podían usarse en un programa que jugara al ajedrez. Pocos años después, Alan Turing describió un programa para jugar al ajedrez, a pesar de que nunca lo construyó. Al principio de los sesenta, Arthur Samuel tuvo éxito al construir el primer programa de juegos importante y operativo. Su programa jugaba a las damas y además de jugar, podía aprender de sus errores para mejorar su comportamiento (Samuel, 1963).

Existían dos razones para que los juegos pareciesen un buen dominio de exploración de la inteligencia de una máquina:

- Proporcionan una tarea estructurada en la que es muy fácil medir el éxito o el fracaso.
- Obviamente no necesitan grandes cantidades de conocimiento. Se pensó que se podrían resolver por búsqueda directa a partir del estado inicial hasta la posición ganadora.

La primera de estas razones aún sigue siendo válida y explica el continuado interés en el área de los juegos por computadora. Desafortunadamente, la segunda no es cierta para aquellos juegos que no sean muy simples. Por ejemplo, considere el juego del ajedrez:

- El factor de ramificación en una partida media es más o menos 35.
- En una partida media, cada jugador realiza 50 movimientos.
- Por tanto, para examinar el árbol del juego completamente, se tendrían que examinar  $35^{100}$  posiciones.

Así resulta evidente que un programa que realice una simple búsqueda exhaustiva en el árbol del juego, no podrá seleccionar ni siquiera su primer movimiento durante el tiempo de vida de su oponente. Es necesario algún tipo de procedimiento de búsqueda heurística.

Una forma de contemplar todos los procedimientos de búsqueda que se han explicado es que esencialmente se trata de **procedimientos de generación y prueba**, en donde la comprobación se realiza después de cantidades distintas de trabajo realizadas por el generador. En un extremo, el generador proporciona propuestas de soluciones completas, que el comprobador evalúa. En el otro extremo, el generador genera movimientos individuales en el espacio de búsqueda, cada uno de los cuales se evalúa a continuación mediante el comprobador para pasar a elegir el más prometedor de ellos. Mirándolo así, resulta claro que para mejorar la efectividad de un programa resolutor de problemas es necesario hacer dos cosas:

- **Mejorar el procedimiento de generación** de forma que solo se generen movimientos (o caminos) buenos.
- **Mejorar el procedimiento de prueba** para que solo se reconozcan y exploren en primer lugar los mejores movimientos (o caminos).

En los programas para jugar es particularmente importante que se hagan las dos cosas. Si se usa un simple *generador de movimientos legales*, el procedimiento de prueba (que probablemente utiliza una combinación de búsqueda y una función de evaluación heurística) deberá procesar cada uno de ellos. Como el procedimiento de búsqueda debe tener en cuenta muchas posibilidades, debe ser rápido. Por lo tanto, *probablemente no pueda realizar su trabajo con precisión*. Suponga, por otra parte que en lugar de un generador de movimientos legales, se usa un *generador de movimientos plausibles* en el que solo se genera un pequeño número de movimientos prometedores. Conforme se incrementa el número de movimientos

legales posibles, la aplicación de técnicas heurísticas para seleccionar solo aquellos que sean algo prometedores, se vuelve más importante. Con un generador de movimientos más selectivo, el procedimiento de prueba puede permitirse el lujo de emplear más tiempo en la evaluación de cada uno de los movimientos que se le proporcionan, por lo que puede producir un resultado más fiable. De este modo, con la incorporación de conocimiento heurístico tanto en el generador como en el comprobador, se mejora el rendimiento del sistema total.

Por supuesto, en los juegos, al igual que en otros dominios de problemas, la búsqueda no es la única técnica disponible. En algunos juegos existen al menos algunas ocasiones en las que son apropiadas otras técnicas más directas. Por ejemplo, en el ajedrez, tanto las aperturas como los finales están ampliamente estudiados, por lo que es mejor jugar consultando con una tabla de una base de datos que almacene patrones.

#### **Entonces para jugar una partida completa, deben combinarse las técnicas orientadas a la búsqueda con las que no lo son.**

La forma ideal de usar un procedimiento de búsqueda para encontrar una solución a un problema, es generar movimientos a través del espacio del problema hasta que se encuentra un estado objetivo. En el contexto de programas de juegos, un estado objetivo es aquel en el cual ganamos. Desafortunadamente, para juegos interesantes como el ajedrez, normalmente no es posible buscar hasta encontrar un estado objetivo, ni siquiera disponiendo de un buen generador de movimientos plausibles. La profundidad del árbol (o grafo) resultante y su factor de ramificación son demasiado grandes.

Con la cantidad de tiempo disponible, solo es posible buscar menos de diez movimientos en el árbol (llamados capas en la literatura de juegos). Así pues, para elegir el mejor movimiento, deben compararse las posiciones del tablero resultantes para determinar cual es la más ventajosa. Esto se lleva a cabo con una función de evaluación estática, que usa toda la información disponible para evaluar posiciones individuales del tablero estimando la probabilidad de que conduzcan eventualmente a la victoria. Su función es similar a la función heurística  $h'$  en el algoritmo A\*: en ausencia de una información completa, elige la posición más prometedora. Naturalmente, la función de evaluación estática puede aplicarse simplemente a las posiciones generadas por los movimientos propuestos. Pero, puesto que es difícil producir una función como ésta que sea realmente muy precisa, es mejor aplicarla tantos niveles hacia abajo en el árbol de juego como lo permita el tiempo disponible.

#### **2.3.2. El procedimiento minimax**

*El procedimiento de búsqueda minimax es un procedimiento de búsqueda en profundidad limitada.*

La idea consiste en comenzar en la posición actual y usar el generador de movimientos plausibles para generar un conjunto de posiciones sucesivas posibles. Entonces se puede aplicar la función de evaluación estática a esas posiciones y escoger simplemente la mejor. Después de hacer esto, puede llevarse hacia atrás ese valor hasta la posición de partida para representar nuestra evaluación de la misma. La posición de partida es exactamente tan buena para nosotros como la posición generada por el mejor movimiento que podamos hacer a continuación. Aquí se va a suponer que la función de evaluación estática devuelve valores elevados para indicar buenas situaciones para nosotros, de manera que nuestra meta es maximizar el valor de la función de evaluación estática de la siguiente posición del tablero.

En la Figura 2.26 se muestra un ejemplo de esta operación. Supone una función de evaluación estática que devuelve valores entre -10 y 10, donde 10 indica una victoria para nosotros, -10 una victoria para el oponente y 0 un empate. Puesto que nuestra meta es **maximizar el valor de la función heurística**, escogemos movernos a B. Al llevar hacia atrás el valor de B hasta A, podemos concluir que el valor de A es 8, puesto que sabemos que se puede llegar a una posición con un valor de 8.

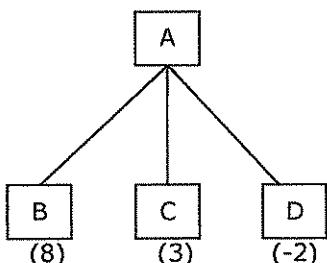


Figura 2.26

Búsqueda de una sola capa

Puesto que sabemos que *la función de evaluación estática no es completamente precisa*, nos gustaría llevar la búsqueda más allá de una sola capa. Esto podría ser muy importante, por ejemplo, en un juego de ajedrez donde estemos en medio de un intercambio de piezas.

Después de nuestro movimiento, podría parecer que nuestra situación es muy buena, pero, si mirásemos un movimiento más allá, veríamos que una de nuestras piezas también es capturada, por lo que la situación no era tan favorable como en principio parecía. Por tanto, nos gustaría prever qué es lo que sucederá en cada una de las nuevas posiciones del juego del siguiente movimiento realizado por el oponente. En lugar de aplicar la función de evaluación estática a cada una de las posiciones que acabamos de generar, aplicamos a cada una de ellas el generador de movimientos plausibles, generando un conjunto de posiciones posteriores para cada una de ellas. Si queremos pararnos ahí, en una previsión de dos capas, podríamos aplicar la función de evaluación estática a cada una de dichas posiciones, tal y como muestra la Figura 2.27.

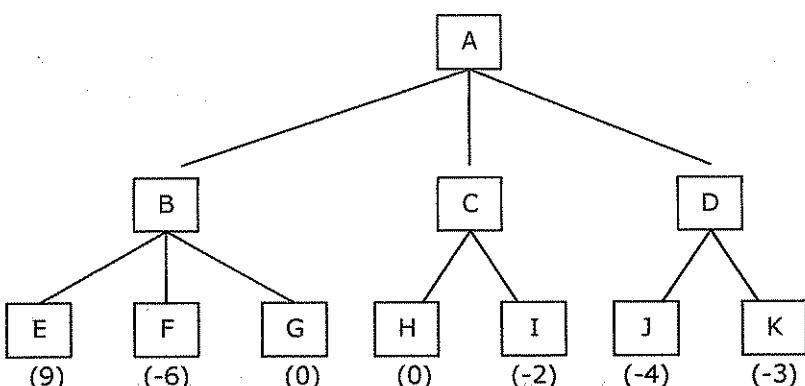


Figura 2.27

Búsqueda de dos capas

Pero ahora debemos tener en cuenta el hecho de que *el movimiento que se realiza a continuación debe elegirlo el oponente*, y por tanto, deberá propagarse hacia atrás al siguiente nivel. Supongamos que realizamos el movimiento B. Entonces el oponente debe elegir entre los movimientos E, F y G. El objetivo del oponente es **minimizar el valor de la función de evaluación**, por lo que puede esperarse que elija moverse a F. Esto significa que si hacemos el movimiento B, la verdadera posición en la que desembocamos en el siguiente movimiento es muy mala para nosotros. Esto es cierto aunque el nodo E represente una posible configuración que fuese buena para nosotros.

Pero ya que no nos toca a nosotros mover en este nivel, no podremos elegirlo. La Figura 2.28 muestra el resultado de propagar los nuevos valores hacia la raíz del árbol. En el nivel representado por la elección del oponente, se elige el menor valor y se propaga hasta la raíz. En el nivel que representa nuestra elección se ha elegido el nivel máximo.

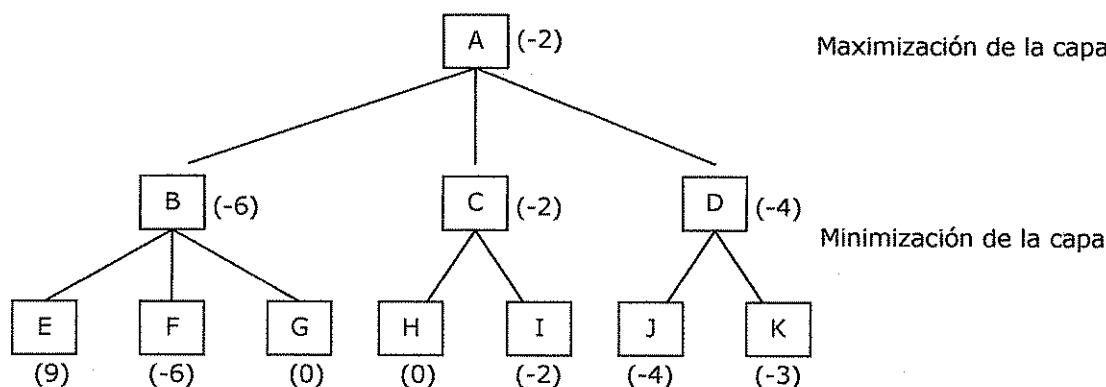


Figura 2.28

Propagación hacia atrás de los valores en una búsqueda de dos capas

Una vez que se han propagado hacia atrás los valores del segundo nivel, resulta claro que el movimiento correcto que podemos realizar en el primer nivel, dada la información disponible, es C, puesto que no hay nada que el oponente pueda hacer ahí para producir un valor peor que -2.

Este proceso puede repetirse para tantos niveles como lo permita el tiempo disponible, y las evaluaciones más precisas que se produzcan pueden usarse para elegir el movimiento correcto en el nivel más alto.

*La alternancia de maximización y minimización en capas alternas cuando las evaluaciones se envían de regreso a la raíz, se corresponde con las estrategias opuestas que siguen los dos jugadores y que da a este método el nombre de **minimax**.*

### 2.3.3. El procedimiento alfa-beta

#### Adición de la poda alfa-beta

Recuerde que el procedimiento **minimax** es un proceso primero en profundidad. Cada camino se explora tan lejos como lo permite el tiempo, la función de evaluación estática se aplica a las posiciones del juego en el último paso del camino, y el valor debe entonces propagarse hacia atrás en el árbol de nivel en nivel.

*Una de las ventajas de los procedimientos primero en profundidad es que a menudo puede mejorarse su eficiencia usando técnicas de ramificación y acotación, en las que pueden abandonarse rápidamente aquellas soluciones parciales que son claramente peores que otras soluciones conocidas.*

Describimos una aplicación directa de esta técnica en el problema del viajante de comercio. Para este problema, lo único que se necesitaba era recordar la longitud del mejor camino que se había encontrado hasta el momento. Si otro camino parcial sobrepasaba esa cota, se abandonaba. Pero, así como fue necesario modificar ligeramente nuestro procedimiento de búsqueda para tratar los jugadores, maximizador y minimizador, también es necesario modificar la estrategia de ramificación y acotación para incluir dos acotaciones, una para cada uno de los jugadores.

**Esta estrategia modificada se denomina poda alfa-beta.**

Requiere el mantenimiento de *dos valores umbral*, uno que representa la **cota inferior del valor** que puede asignarse en último término a un **nodo maximizante** (que llamaremos **alfa**)

y otro que represente la **cota superior del valor** que puede asignarse a un **nodo minimizante** (que llamaremos **beta**).

Para ver como funciona el *procedimiento alfa-beta*, consideremos el ejemplo mostrado en la Figura 2.29. Después de examinar el nodo F, sabemos que el oponente tiene garantizado un resultado de -5 o menos en C (puesto que el oponente es el jugador minimizador). Pero también sabemos que tenemos un resultado seguro de 3 o más en el nodo A, donde podemos llegar si movemos a B. Cualquier otro movimiento que produzca un resultado de menos de 3 es peor que el movimiento a B, y podemos ignorarlo. Con solo examinar F podemos asegurar que un movimiento en C es peor (será menor o igual que -5), sin necesidad de mirar el resultado del nodo G. Así pues, no necesitamos en absoluto explorar el nodo G. Naturalmente, puede parecer que la poda de un nodo no justifica el gasto de grabar los límites y examinarlos, pero si estuviésemos explorando un árbol de seis niveles, entonces no solo habríamos eliminado un nodo único, sino un árbol entero de tres niveles de profundidad.

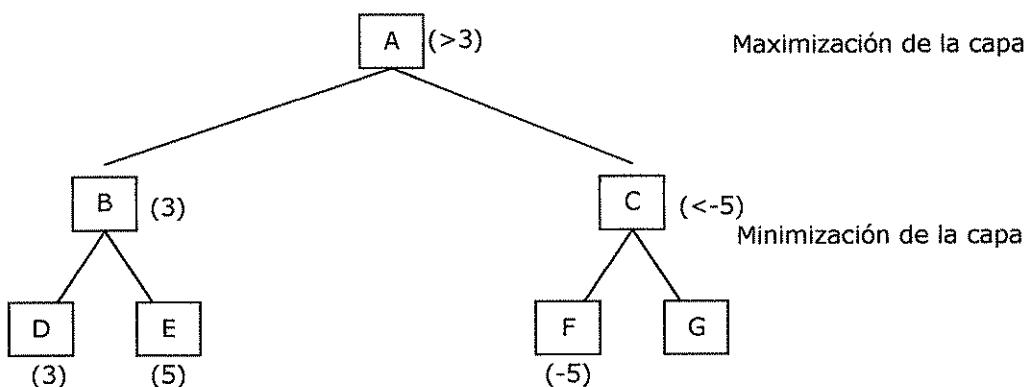


Figura 2.29

Una poda alfa

Para ver como pueden usarse los dos umbrales, alfa y beta, consideremos el ejemplo de la Figura 2.30. Al buscar este árbol, se explora el árbol entero encabezado por B, y descubrimos que en A podemos esperar un resultado de 3 como mínimo. Cuando este valor **alfa** se pase hacia F, nos permitirá evitar la exploración de L. Veamos por qué.

Después de examinar K, vemos que I proporciona un tanteo máximo de 0, lo que significa que F produce un mínimo de 0, pero esto es menor que el valor alfa de 3, por lo que no necesitamos considerar más ramas de I. El jugador maximizador ya sabe que no debe elegir C, y de ahí a I, puesto que si realiza ese movimiento el tanteo resultante no será mayor que 0, y en lugar de ello puede lograrse un tanteo de 3 moviendo a B.

#### **Veamos ahora como usar el valor de beta.**

Después de podar cualquier exploración posterior de I, se examina J, que produce un valor de 5, el cual se asigna a su vez como valor de F (puesto que es el máximo de 5 y 0). Este valor se convierte en el **valor de beta** en el nodo C. Nos garantiza que C será 5 o menos. A continuación debemos expandir G. En primer lugar se examina M que tiene un valor de 7, el cual se pasa a G como su valor provisional. Pero ahora se compara 7 con beta (5). Es mayor, y el jugador que tiene el turno del nodo C está tratando de minimizar. Por lo tanto ese jugador no elegirá G, que le conduciría a un resultado de 7 como mínimo, puesto que hay una alternativa de volver a F, lo que producirá un tanteo de 5. Así pues, no es necesario explorar ninguna de las otras ramas de G.

A partir de este ejemplo, se ve que en los niveles maximizantes podemos excluir un movimiento tan pronto como quede claro que su valor será menor que el umbral actual, mientras que en los niveles minimizantes la búsqueda terminará cuando se descubran valores mayores que el umbral actual.

Pero la exclusión de un movimiento posible del jugador maximizante significa realmente podar la búsqueda en un nivel minimizante. Veamos de nuevo el ejemplo de la Figura 2.30. Una vez determinado que C es un mal movimiento para A, no nos molestarímos en explorar G, o cualquier otro camino, en el nivel minimizante por debajo de C.

Por tanto, la forma en que se usan ahora alfa y beta consiste en que la búsqueda en un nivel minimizante puede terminar cuando se descubre un nivel menor que alfa, mientras que en un nivel maximizante la búsqueda puede terminar al descubrir un valor mayor que beta.

La acotación de la búsqueda en un nivel maximizante cuando se encuentra un valor alto puede parecer, al principio, opuesto a la intuición, pero si se piensa en que solo llegamos a un nodo concreto de un nivel maximizante si el jugador minimizante del nivel anterior lo elige, entonces tiene sentido.

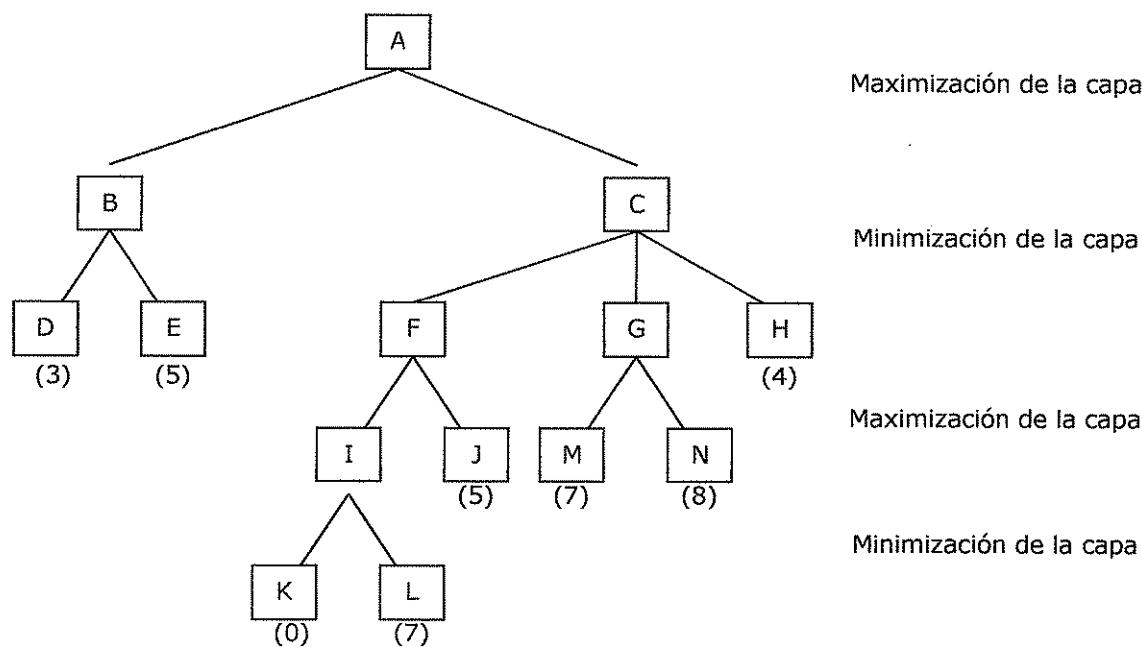


Figura 2.30

Podas alfa y beta

## 2.4. Búsqueda con Sistemas evolutivos

### 2.4.1. Conceptos de Genética

#### Qué son los genes

El descubrimiento de la naturaleza del material hereditario representaba un hito importante en el desarrollo de la *Genética*. Sin embargo, el estudio de la herencia había empezado mucho tiempo antes. Ya durante el siglo XVIII algunos investigadores se habían dedicado a observar la aparición de "monstruos" resultantes del cruzamiento entre especies animales diferentes, "monstruos" que, según decían, presentaban una mezcla de caracteres de los dos animales progenitores. Después, durante los siglos XVIII y XIX, se desarrollaron mucho, con finalidades económicas los trabajos de mejora de razas animales y vegetales.

Sin embargo, durante todo ese tiempo, los estudios sobre la herencia no podían considerarse como verdaderamente científicos. Sólo podemos hablar de ciencia cuando a partir de unos datos experimentales es posible elaborar unas reglas o leyes que generalmente permiten predecir comportamientos posteriores. En la genética premendeliana no ocurría así: se realizaban cruzamientos al azar, sin poder predecir las características de la descendencia.

En 1866 el fraile agostino austriaco Gregor Mendel, que había realizado experimentos con plantas de guisantes durante varios años, enunció las leyes de la herencia. Su éxito fue debido al excelente planteamiento de dos experimentos: en primer lugar, a la elección de un material adecuado, como es el guisante, para controlar los cruzamientos, y en segundo lugar, a haber simplificado al máximo los mismos, dedicándose a estudiar un solo carácter cada vez. De este modo, Mendel llegó a la conclusión de que todo carácter heredable depende de la presencia en las células del organismo de unos "**factores hereditarios**", transmisibles de padres a hijos y de los que todo organismo posee dos para cada carácter, uno procedente del padre y otro de la madre.

Consideremos un carácter humano, como es la coloración parda de los ojos. El que un individuo posea ojos pardos será debido a que posee en cada célula dos factores hereditarios determinantes del color pardo, procedentes uno del padre y otro idéntico de la madre. A su vez, cuando este individuo forme la célula reproductora que, junto con la procedente de un individuo del otro sexo, dará lugar al hijo, donará a la misma uno de aquellos "factores".

Las partículas concretas y finitas que Mendel denominó "*factores hereditarios*" comenzaron a ser denominadas **genes** a partir de 1910.

Actualmente se considera que un *gen* es un fragmento de DNA asociado a moléculas de proteína en los organismos superiores, que generalmente contiene información relativa a un carácter heredable del organismo.

### Cromosomas

La cantidad de DNA contenida en un organismo eucariote es muy superior a la contenida en un procariote. Un problema que se presentará en los primeros es el modo de transmisión del DNA a las células hijas.

Cuando una célula del eucariote se halla en reposo, entre una división celular y la siguiente no se advierte ninguna estructura visible en el interior de su núcleo; ni siquiera mediante el microscopio electrónico. El DNA se halla disperso en el interior del núcleo, aparentemente en estado de reposo divisional. Sin embargo, la situación verdadera es muy distinta: en este estadio, el *material genético* se está reproduciendo, de modo que cada molécula de DNA da origen a dos moléculas hijas, que son copias prácticamente exactas de la primera.

En cuanto la célula comienza su división, el núcleo desaparece y el DNA, hasta entonces invisible, comienza a apelotonarse de modo regular, formando unas estructuras muy densas, a modo de bastoncillos dobles, muchas veces en forma de X, denominadas **cromosomas**. El número de estos cromosomas presentes en las células de un determinado organismo es constante, salvo casos anómalos; así, el hombre posee 46 de ellos en sus células, al igual que la patata. A medida que la división celular progresó, los cromosomas se sitúan en el centro de la célula formando una figura a modo de estrella; seguidamente, cada cromosoma se divide en sentido longitudinal en dos mitades, yendo cada una a extremos opuestos de la célula. Por último, los bastoncillos se desapelotonan, los *cromosomas* dejan de ser visibles y se forman dos nuevos núcleos (correspondientes a las dos nuevas células), rodeando a los nuevos cromosomas.

Los cromosomas deben considerarse, pues, como las estructuras formadas por el DNA para transmitirse a las células hijas en el transcurso de la división celular.

Hay, no obstante, otro hecho fundamental a considerar: los cromosomas de una especie son iguales dos a dos. Así, los 46 cromosomas de la célula humana son, en realidad, 23 parejas de cromosomas homólogos. En cada pareja de cromosomas homólogos uno procede del padre y el otro de la madre. Vemos, pues, que existe un paralelismo entre los *genes* y los *cromosomas*: así como cada célula tiene dos cromosomas homólogos, uno procedente del padre y otro de la madre, también posee dos genes correspondientes a un carácter dado, procedente cada uno de un progenitor. Esto no es casual: los genes del organismo (constituidos por DNA) se disponen linealmente a lo largo de los cromosomas, de modo que cada gen se halla por duplicado en alguna de las parejas de homólogos. Con todo lo dicho, podemos tener ya una idea global de la relación DNA-gen-cromosoma: el cromosoma es una organización lineal de genes, formada por DNA y que únicamente se hace visible cuando, durante la división celular, su grado de engrosamiento es suficiente.

## Mutaciones I

Las diferencias existentes entre individuos de una misma especie y de especies diferentes se deben a cambios genéticos denominados ***mutaciones***. En la naturaleza, estas mutaciones aparecen de modo espontáneo y, sin duda, no todas ellas resultan favorables para el individuo. En algunos casos, es posible que los individuos sujetos a mutaciones se hallen en situación desfavorable respecto del resto de la población y, aunque no necesariamente sujetos a una eliminación rápida desaparecerán finalmente. En otros casos, la mutación representará para los individuos que la posean una mejora y, al reproducirse más fácilmente que el resto de la población, pasarán más o menos tarde a constituirla por entero.

La evolución de la vida en la Tierra ha sido debida a la existencia de ***mutaciones***. Estas, aunque en muchos casos son causa de muerte, en otros lo son de progreso, al permitir la aparición de genes que informan acerca de nuevas funciones. Así, a lo largo de millones de años ha sido posible una revolución que ha originado formas vivas tan diferentes como las existentes hoy. Esa ***evolución*** se produce frecuentemente en el sentido positivo de progreso, el cual es preciso concebirlo como una más eficaz relación entre organismo y medio ambiente. Sin embargo, las características del medio ambiente pueden cambiar hasta hacer que tal eficacia se anule. Por eso, las especies vivas no sólo aparecen, sino que también desaparecen cuando una situación favorable se torna desfavorable. Esto, que ha ocurrido durante toda la historia de la vida en la Tierra, ocurre también ahora y el hombre no se halla exento de la regla.

## Mutaciones II

Desde el punto de vista de la genética de poblaciones, las mutaciones son cambios heredables que ocurren en el genotipo. Estos cambios se producen por casualidad, en el sentido de que el ambiente no puede influir sobre el tipo de mutación producida, pero la tasa de mutaciones sí puede ser influída por factores ambientales (como la radiación) y también los distintos genes tienen tasas distintas de mutación por razones que se relacionarían con la composición química del gen en cuestión y con su posición en el cromosoma. La tasa media de mutaciones detectables en el fenotipo está comprendida entre 1 en 1000 y 1 en 1000000 de gametos por generación, según el alelo de que se trate. Si suponemos que existen unos 100000 genes (**pares de alelos**) en cada ser humano, cabe suponer que cada recién nacido es portador de dos mutaciones nuevas. Así, aunque la incidencia de mutaciones en cada gen en particular es pequeña, la cantidad de mutaciones nuevas por cada generación de la población es muy grande. Las mutaciones suelen considerarse la materia prima de los cambios evolutivos; ellas introducen variaciones sobre las que actúan otras fuerzas evolutivas, pero no determinan la dirección del cambio evolutivo.

## El lenguaje de la vida

Hasta 1950 se sabía que las características de los seres vivos vienen determinadas por los genes. Se sabía que éstos están constituidos por lo que se denomina proteínas y ácidos nucleicos. Pero hasta 1950, los *genes* son considerados como estructuras extremadamente complejas, en tres dimensiones, cada una de las cuales sería completamente diferente de las demás. La extraordinaria revelación de los años cincuenta fue demostrar que esta complejidad infinita es debida simplemente a la combinatoria de un número muy pequeño de unidades químicas, *de cuatro pequeñas moléculas*. Estas cuatro unidades se repiten por millones a lo largo de la fibra cromosómica. Se combinan y permutan hasta el infinito como las letras de un alfabeto a lo largo de un texto.

Del mismo modo que una frase constituye un segmento de un texto, de igual manera un *gen* corresponde a una cierta secuencia de *signos químicos*. En ambos casos, un símbolo aislado no representa nada; sólo la combinación de los signos adquiere un "sentido". En ambos casos, una secuencia dada, frase o gen, se inicia y se termina con signos específicos de "puntuación". La traducción de la secuencia nucleica en secuencia proteica es comparable a la traducción de un mensaje que llega, cifrado en Morse, pero que sólo tiene sentido una vez que ha sido

traducido al español, por ejemplo. Dicha traducción se efectúa mediante una "clave" que da la equivalencia de los "signos" entre los dos "alfabetos", nucleico y proteico.

La clave genética se conoce hoy perfectamente. Cada unidad proteica corresponde a un "*triplete*", es decir, una combinación particular de tres entre las cuatro unidades nucleicas. Como existen  $4^3 = 64$  combinaciones posibles de tres unidades nucleicas, el "diccionario" genético contiene 64 "palabras". Tres de estos tripletes aseguran la "puntuación", es decir, indican, en la cadena nucleica, el inicio y el fin de las "frases" que corresponden a las cadenas proteicas. Cada uno de los otros tripletes "representan" una de las unidades proteicas. Como el número de dichas unidades está limitado a 20, cada una de ellas responde a varios tripletes, a varios sinónimos en el diccionario, de lo que se deriva cierta flexibilidad en la escritura de la herencia.

Con todo ello se pone de manifiesto que todos los organismos, de la bacteria al hombre, son capaces de interpretar correctamente cualquier mensaje genético. La clave genética parece universal y su secreto conocido por todo el mundo viviente.

Existe, pues, una analogía muy sorprendente entre los dos sistemas, genético y lingüístico, ya que, por una parte, se observa una estricta colinearidad de los fenómenos del codaje y que, por otra parte, nos encontramos ante una combinatoria de elementos, fonemas o radicales químicos, que aislados no significan nada y que sólo toman un sentido cuando se combinan con otros elementos. Todavía pueden ponerse en evidencia analogías más profundas, ya que es posible, en ambos casos reducir las relaciones entre elementos, fonemas o radicales químicos a sistemas de oposición binaria, y que en ambos casos se encuentran niveles de construcción jerarquizados por unidades de rango inferior. Se trata, pues, de saber en qué medida todas estas semejanzas traducen o no algo más que una analogía. El destacado lingüista Roman Jakobson encuentra dicha analogía tan sorprendente que duda que sea debida al azar. Se pregunta si, de alguna forma, el sistema del lenguaje no se ha modelado sobre el de la herencia.

### Crossing Over

El *crossing over* es el intercambio de porciones de cromosomas homólogos y tiene lugar al comienzo de la meiosis.

Con el descubrimiento de los *crossover*, no sólo se empezó a comprender que los genes están en los cromosomas, como había supuesto Sutton, sino también que tienen que hallarse en determinados sitios o *loci* (singular, *locus*) en los cromosomas. Además, los alelos de cualquier gen en particular tienen que ocupar loci concordantes en cromosomas homólogos. De lo contrario, el intercambio de porciones de cromosomas ocasionaría un caos genético, en lugar de un intercambio exacto de *alelos*.

Sturtevant postuló:

- que los genes están dispuestos en una serie lineal en los cromosomas como las cuentas de un collar
- que los genes que están muy próximos se habrán de separar mediante *crossing over* con menos frecuencia que los genes más distantes
- que determinando, por lo tanto, las frecuencias de las *recombinaciones*, se podría establecer la secuencia de los genes a lo largo del cromosoma y las distancias relativas entre ellos.

### Migración: flujo de genes

Flujo de genes es el movimiento de *alelos* hacia y desde una población como consecuencia de la *inmigración* o *emigración* de individuos reproductivos o, en el caso de las plantas, la introducción de gametos (por medio del polen) procedentes de otras poblaciones. La inmigración puede introducir alelos nuevos en una población o modificar las frecuencias de los alelos. Su efecto global es reducir la diferencia entre las poblaciones, mientras que la

**selección natural** tiende a *acentuar las diferencias* al producir poblaciones más aptas para distintas condiciones locales. En consecuencia a menudo el flujo genético contrarresta la selección natural y, como veremos, también puede inhibir la especiación.

## Modalidades de la Evolución

La selección natural produce distintas modalidades de evolución. Puede originar *fenotipos* muy distintos en organismos íntimamente emparentados, fenotipos muy similares en organismos que son parientes lejanos, o los organismos mismos pueden convertirse en las fuerzas de selección por medio de sus interacciones con otras especies.

### Evolución divergente

La evolución divergente ocurre cuando una población queda aislada del resto de la especie y, a causa de determinadas presiones de selección, emprende un curso evolutivo distinto.

### Evolución convergente

Muchas veces los organismos que ocupan ambientes similares vienen a asemejarse entre ellos aunque filogenéticamente tengan un parentesco muy distante. Al estar sujetos a presiones de selección similares, exhiben adaptaciones similares.

### Evolución paralela

La evolución paralela se emplea para describir una situación en la cual los linajes han cambiado de maneras similares, de modo que los descendientes evolucionados se parecen tanto entre ellos como lo fueron sus antepasados.

### Coevolución

Cuando dos o más poblaciones interactúan de manera tan íntima que cada una de ellas ejerce una potente fuerza selectiva sobre la otra, ocurren ajustes simultáneos que producen coevolución.

## 2.4.2. Algoritmos Genéticos

Los **Algoritmos Genéticos (AG)** son métodos adaptativos que pueden ser utilizados para implementar búsquedas y problemas de optimización. Ellos están basados en los procesos genéticos de organismos biológicos, codificando una posible solución a un problema en un "**cromosoma**" compuesto por una cadena de bits o caracteres.

Estos cromosomas representan individuos que son llevados a lo largo de varias **generaciones**, en forma similar a las poblaciones naturales, evolucionando de acuerdo a los principios de **selección natural** y "**supervivencia**" del más apto, descritos por primera vez por Charles Darwin en su libro "Origen de las Especies". Emulando estos procesos, los Algoritmos Genéticos son capaces de "evolucionar" soluciones a problemas del mundo real.

En la naturaleza, los individuos compiten entre si por recursos tales como comida, agua y refugio. Adicionalmente, los animales de la misma especie normalmente antagonizan para obtener una pareja. Aquellos individuos que tengan más éxito tendrán probablemente un número mayor de descendientes, por lo tanto, mayores probabilidades de que sus genes sean propagados a lo largo de sucesivas generaciones. La combinación de características de los padres bien adaptados, en un descendiente, puede producir muchas veces un nuevo individuo mucho mejor adaptado que cualquiera de sus padres a las características de su medio ambiente.

Los Algoritmos Genéticos utilizan una analogía directa del fenómeno de evolución en la naturaleza. Trabajan con una **población de individuos**, cada uno representando una posible solución a un problema dado. A cada individuo se le asigna una **puntuación de adaptación**, dependiendo de que tan buena fue la respuesta al problema. A los más adaptados se les da la oportunidad de reproducirse mediante cruzamientos con otros individuos de la población, produciendo descendientes con características de ambos padres. Los miembros menos adaptados poseen pocas probabilidades de que sean seleccionados para la reproducción, y desaparecen.

Una nueva población de posibles soluciones es generada mediante la **selección** de los mejores individuos de la generación actual, emparejándolos entre ellos para producir un nuevo conjunto de individuos. Esta nueva generación contiene una proporción más alta de las características poseídas por los mejores miembros de la generación anterior. De esta forma, a lo largo de varias generaciones, las características buenas son difundidas a lo largo de la población mezclándose con otras. Favoreciendo el emparejamiento de los individuos mejor adaptados, es posible recorrer las áreas más prometedoras del espacio de búsqueda. Si el Algoritmo Genético ha sido diseñado correctamente, la población convergerá a una solución óptima o casi óptima al problema.

Los dos procesos que más contribuyen a la evolución son el **crossover** y la adaptación basada en la **selección**. La **mutación** también juega un papel significativo, pero determinar que tan importante sea su rol, continúa siendo una materia de debate (algunos se refieren a ella como un operador en background), ella no debe ser utilizada demasiado, ya que el Algoritmo Genético se puede convertir en una búsqueda al azar, pero su utilización asegura que ningún punto en el espacio de búsqueda tiene probabilidad 0 de ser examinado.

En la práctica, se puede implementar este modelo, utilizando matrices de bits o caracteres para representar los cromosomas. Operaciones sencillas de bits permiten efectuar el crossover, la mutación y otras operaciones. A pesar de que una gran cantidad de investigación ha sido realizada en cadenas de longitud variable y otras estructuras, la mayor parte del trabajo con AG ha sido enfocado en cadenas de caracteres de **longitud fija**. Se hace énfasis en este aspecto y en la necesidad de codificar la solución como una **cadena de caracteres**.

Generalmente, los AG son implementados siguiendo el siguiente ciclo:

1. Generar aleatoriamente la población inicial
2. Evaluar la adaptación de todos los individuos en la población.
3. Crear una nueva población efectuando operaciones como crossover, reproducción proporcional a la adaptación y mutaciones en los individuos cuya adaptación acaba de ser medida.
4. Eliminar la antigua población.
5. Iterar utilizando la nueva población, hasta que la población converja.

Cada iteración de este bucle es conocida como **generación**. La primera generación de este proceso es una población de individuos generados al azar. Desde ese punto, los operadores genéticos, en unión con la medida de adaptación, actúan para mejorar la población.

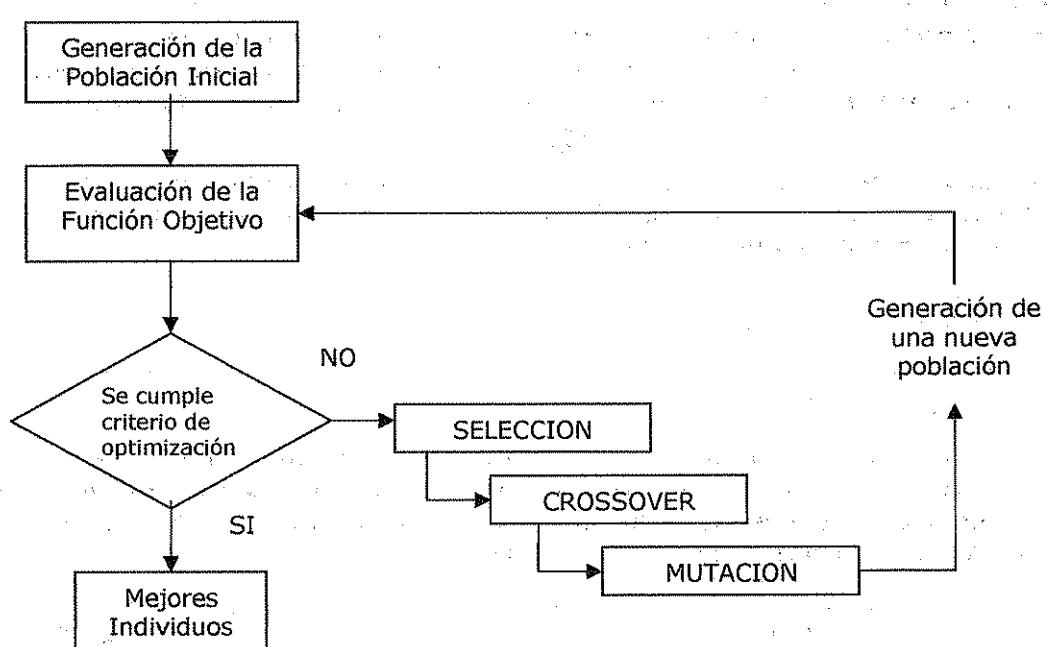


Figura 2.31

Los Algoritmos Genéticos no son la única técnica basada en una analogía de la naturaleza. Por ejemplo, las Redes Neurales están basadas en el comportamiento de las neuronas en el cerebro. Pueden ser utilizadas en una gran variedad de tareas de clasificación, como reconocimiento de patrones o proceso de imágenes. Actualmente está en investigación la utilización de Algoritmos Genéticos para el diseño de Redes Neurales.

El poder de los Algoritmos Genéticos proviene del hecho de que la técnica es robusta, y puede manejar exitosamente un amplio rango de problemas, incluso algunos que son difíciles de resolver por otros métodos. Los Algoritmos Genéticos no garantizan que encontrarán la solución óptima al problema, pero son generalmente buenos encontrando soluciones aceptables a problemas en corto tiempo. Donde existan técnicas especializadas para la resolución de problemas, estas superarán fácilmente a los Algoritmos Genéticos tanto en velocidad como en precisión.

El campo principal de aplicación es donde no existan este tipo de técnicas.

### **Diferencias entre los Algoritmos Genéticos y los métodos tradicionales.**

Los Algoritmos Genéticos tienen cuatro diferencias principales con los métodos más utilizados o conocidos de optimización y búsqueda:

- Trabajan con una codificación del conjunto de parámetros, no con estos directamente.
- Buscan simultáneamente la solución en una población de puntos, no en uno sólo.
- Utilizan la función objetivo (rendimiento), no derivadas u otro conocimiento auxiliar.
- Utilizan reglas de transición probabilísticas, y no determinísticas.

#### **2.4.3. Poblaciones**

##### **Codificación**

Las partes que relacionan un Algoritmo Genético con un problema dado son la codificación y la función de evaluación.

Si un problema puede ser representado por un **conjunto de parámetros** (*conocidos como genes*), estos pueden ser unidos para formar una **cadena de valores** (*cromosoma*), a este proceso se le llama **codificación**. En genética este conjunto representado por un cromosoma en particular es referido como **genotipo**, este contiene la información necesaria para construir un organismo, conocido como **fenotipo**.

Estos mismos términos se aplican en Algoritmos Genéticos, por ejemplo, si se desea diseñar un puente, el conjunto de parámetros especificando el diseño es el genotipo, y la construcción final es el fenotipo. La adaptación de cada individuo depende de su fenotipo, el cual se puede inferir de su genotipo, es decir, puede calcularse desde el cromosoma utilizando la función de evaluación.

Por ejemplo, si se tiene un problema de maximizar una función de tres variables,  $F(X,Y,Z)$ , se podría representar cada variable por un número binario de 10 bits, obteniéndose un cromosoma de 30 bits de longitud y 3 genes.

Existen varios aspectos relacionados con la codificación de un problema a ser tomados en cuenta en el momento de su realización:

- Se debe utilizar el alfabeto más pequeño posible para representar los parámetros, normalmente se utilizan dígitos binarios.
- Las variables que representan los parámetros del problema deben ser discretizadas para poder representarse con cadenas de bits, hay que utilizar suficiente resolución para asegurar que la salida tiene un nivel de precisión adecuado, se asume que la discretización es representativa de la función objetivo.
- La mayor parte de los problemas tratados con Algoritmos Genéticos son no lineales y muchas veces existen relaciones "ocultas" entre las variables que conforman la solución. Esta interacción es referida como **epistasis**, y es necesario tomarla en cuenta para una representación adecuada del problema. El tratamiento de los genotipos

inválidos debe ser tomado en cuenta para el diseño de la codificación. Supóngase que se necesitan 1200 valores para representar una variable, esto requiere al menos 11 bits, pero estos codifican un total de 2048 posibilidades, "sobrando" 848 patrones de bits no necesarios. A estos patrones se les puede dar un valor cero de adaptación, ser substituidos por un valor real, o eliminar el cromosoma.

#### 2.4.4. Operadores genéticos

Los Operadores genéticos son las distintas funciones que se aplican a las poblaciones, las cuales permiten obtener poblaciones nuevas:

Los tipos de operadores más usados son:

- Selección
- Crossover
- Mutación
- Migración
- Otros operadores

#### Selección

Determina como los individuos son elegidos para el apareamiento. Si se usa un método de selección que elija a los mejores individuos entonces la población convergerá hacia estos individuos pero éstos no representan la solución, entonces se seleccionan individuos, que a pesar de no ser los mejores, tienen algún tipo de material genético bueno en ellos.

Los métodos de selección más comúnmente usados son el método de la ruleta, torneos o ranking.

#### Crossover

Consiste en el intercambio de material genético entre dos cromosomas. El crossover es el principal operador genético, hasta el punto que se puede decir que no es un algoritmo genético si no tiene crossover, y sin embargo, puede serlo perfectamente sin mutación. La propuesta primaria del operador crossover es obtener material genético de las generaciones anteriores para las siguientes generaciones.

Para aplicar el crossover, entrecruzamiento o recombinación, se escogen aleatoriamente dos miembros de la población. No pasa nada si se emparejan dos descendientes de los mismos padres, ello garantiza la perpetuación de un individuo con buena puntuación. Sin embargo, si esto sucede demasiado a menudo, puede crear problemas, ya que toda la población puede aparecer dominada por los descendientes de algún gen.

La forma básica de este operador toma dos individuos y corta sus cromosomas en una posición seleccionada al azar, para producir dos segmentos anteriores y dos posteriores, los posteriores se intercambian para obtener dos cromosomas nuevos. Esto es conocido como *crossover de un punto*.

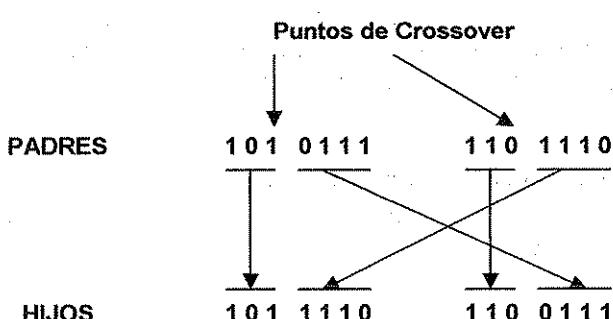


Figura 2.32.

Otros métodos de crossover son también aplicables, como los crossover de múltiples puntos, uniforme o cíclico.

### Mutación

La mutación es la alteración en forma aleatoria de un individuo de la población. Esto es necesario para que no se produzca una convergencia prematura y que todos los individuos de la población no tengan probabilidad cero de ser utilizados.

Este operador se aplica eventualmente a algunos individuos para cambiar aleatoriamente una parte de su material genético e introducir diversidad a la población.

No hace falta decir que no conviene abusar de la mutación. Es cierto que es un mecanismo generador de diversidad, y por lo tanto, la solución cuando un algoritmo genético está estancado, pero también es cierto que reduce el algoritmo a una búsqueda aleatoria.

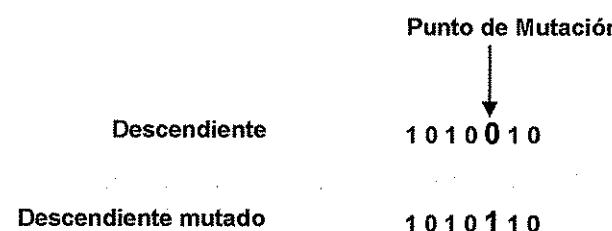


Figura 2.33

### Migración

La migración es el operador que genera un intercambio de individuos entre subpoblaciones. Este operador es aplicable en el caso de desarrollar una técnica de Algoritmos Genéticos Paralelos para la resolución de un problema.

Al recibir individuos de otras subpoblaciones (migración) se introduce la diversidad y se incrementa la presión de la selección (se aumenta la competencia por sobrevivir) en cada subpoblación. Esto es muy útil para evitar convergencias prematuras a óptimos locales. Sin embargo, si la subpoblación ha llegado a un estado de equilibrio, la introducción de nuevo material puede no ser efectiva porque es posible que cada subpoblación haya encontrado un nicho adecuado y se hayan formado especies distintas.

### 2.4.5. Función de evaluación

Dado un cromosoma, la función de evaluación consiste en asignarle un valor numérico de "**adaptación**", el cual se supone que es proporcional a la "**utilidad**" o "**habilidad**" del individuo representado. En muchas casos, el desarrollo de una función de evaluación involucra hacer una simulación, en otros, la función puede estar basada en el rendimiento y representar sólo una evaluación parcial del problema.

Adicionalmente debe ser rápida, ya que hay que aplicarla para cada individuo de cada población en las sucesivas generaciones, por lo cual, gran parte del tiempo de corrida de un algoritmo genético se emplea en la función de evaluación.

### Convergencia

Si el Algoritmo Genético ha sido correctamente implementado, la población evolucionará a lo largo de sucesivas generaciones de forma que la adaptación del mejor y el promedio general se incrementarán hacia el óptimo global.

La convergencia es la progresión hacia la uniformidad. Un gen ha convergido cuando el 95% de la población tiene el mismo valor. La población converge cuando todos los genes de cada individuo lo hacen.

Por ejemplo la figura siguiente muestra la convergencia representada por la varianza de una población a lo largo de sucesivas generaciones.

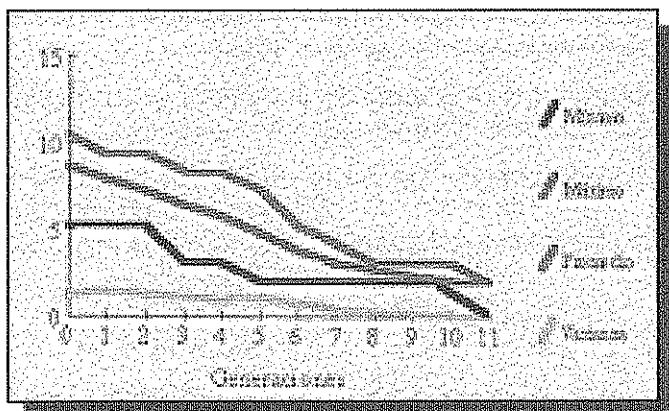


Figura 2.34

### Convergencia prematura

Un problema de los Algoritmos Genéticos dado por una mala formulación del modelo es aquel en el cual los *genes de unos pocos individuos* relativamente bien adaptados, pero no óptimos, pueden rápidamente *dominar la población*, causando que converja a un máximo local.

Una vez que esto ocurre, la habilidad del modelo para buscar mejores soluciones es *eliminada completamente*, quedando sólo la mutación como vía de buscar nuevas alternativas, y el algoritmo se convierte en una búsqueda lenta al azar.

Para evitar este problema, es necesario *controlar el número de oportunidades reproductivas de cada individuo*, tal que, no obtenga ni muy alta ni muy baja probabilidad. El efecto es comprimir el rango de adaptación y prevenir que un individuo "*super-adaptado*" tome control rápidamente.

### Finalización lenta

Este es un problema contrario al anterior, luego de muchas generaciones, la población habrá convergido, pero *no habrá localizado el máximo global*. La adaptación promedio será alta y habrá poca diferencia entre el mejor y el individuo promedio, por consiguiente será muy baja la tendencia de la función de adaptación a llevar el algoritmo hacia el máximo. Las mismas técnicas aplicadas en la convergencia prematura son utilizadas en este caso.

## Algoritmo Genético Canónico

El primer paso en la implementación de un algoritmo genético, una vez establecida la función objetivo, consiste en definir el tipo de la tira que conformará el cromosoma.

En los AG Canónicos, las mismas son **tiras binarias** de longitud L. Cada gen es un dígito 0 o 1.

Conocido **N**, número de cromosomas que compondrán las distintas poblaciones, se procede a crear la población inicial. Este proceso, en general se realiza en forma **random**. Cada gen de cada cromosoma toma el valor 0 o 1 en forma aleatoria.

A continuación, cada cromosoma es evaluado, usando la **función objetivo** y la **función fitness**. Ambas funciones, suelen ser intercambiables, aunque no hay que olvidar que, en realidad, se trata de funciones conceptualmente distintas.

En los AGC, la **función fitness** se define para cada cromosoma, como el cociente  **$f_i/F$** , donde  **$f_i$**  es el valor de la **función objetivo** correspondiente, mientras que **F** es el **promedio de los valores de la función objetivo** para cada tira de la población.

## Desarrollo paso a paso de un AG Canónico

Aplicaremos un AG simple a un problema particular de optimización, paso por paso. Consideremos el problema de **maximizar la función  $f(x) = x^2$** , para **x** variando entre **0** y **31**.

Para comenzar debemos codificar las variables de decisión de nuestro problema como tiras de longitud finita. Para este caso nos sirve codificar la variable **x** como enteros binarios sin signo de longitud 5, lo que nos permite obtener desde **00000** (0) hasta **11111** (31).

Fijada ya la función objetivo y definida la codificación de **x**, podemos simular una simple generación de un AG con selección, crossover y mutación.

Comenzaremos fijando una **población inicial de 4 miembros** de 5 dígitos binarios cada uno en forma aleatoria. Para este bajo número, podemos utilizar los valores obtenidos al arrojar una moneda. Supongamos que los resultados son los siguientes:

01101

11000

01000

10011

Con estos datos disponibles podemos construir una tabla donde reflejemos los componentes de la población, el valor de **x** que le corresponde, el valor de la función objetivo para cada uno.

Número de string	Población Inicial	Valor de x	$f(x) = x^2$	$f_i/F$	$f_i/F$	Veces seleccionados
1	01101	13	169	0,14	0,58	1
2	11000	24	576	0,49	1,97	2
3	01000	8	64	0,06	0,22	0
4	10011	19	361	0,31	1,23	1
Suma			1170	1,00	4,00	
Promedio			293	0,25	1,00	
Máximo			576	0,49	1,97	

La columna  **$f_i/F$**  nos da el peso porcentual que tiene cada componente de la población inicial en el aro de la ruleta, de donde puede apreciarse que después de ponerla en funcionamiento los componentes 1 y 4 son seleccionados en una ocasión cada uno, mientras que el 2 lo hace en 2 oportunidades y el 3 no es seleccionado. Esta población intermedia constituye lo que llamamos

**"mating pool"**. Esta primera etapa, creación de la población intermedia se denomina **Selección** (aplicando en este ejemplo el método de la Ruleta)

A continuación comenzaremos con el proceso de **crossover**. Para ello vamos extrayendo al azar pares de padres que serán sometidos a la crusa. En general, con los pares seleccionados y antes de la crusa se consulta un factor probabilístico llamado "**Pc: probabilidad de crusa**", que establece el usuario al comienzo de la ejecución, el que asumiremos como suele ser común **Pc = 0,6**. Esto significa que se genera un número al azar que puede valer 0 o 1 (este último con un 60% de probabilidad). Si el valor es 1, se procede al proceso de crossover. Para el ejemplo utilizaremos **Crossover de un punto**. En este caso se elige un número al azar entre 1 y 4 (en general **entre 1 y L-1**, si L es la longitud de la tira). En este punto se intercambian las colas de las tiras para crear un par de hijos.

Supongamos que **el punto de corte es 4** y que el par de padres seleccionados son el primero y el segundo. Por lo tanto los hijos engendrados serán:

0110   1	01100
1100   0	11001

En el próximo intento los padres 11000 (segundo) y 10011 (cuarto) con un punto de corte en 2, da los hijos 11011 y 10000. Con esto finaliza el Crossover de un punto.

Finalmente nos queda el proceso de **mutación** que viene controlado por el factor "**Pm: probabilidad de mutación**". Este, habitualmente, es un valor bajo, por ej: **0,001**. Con tal valor, y dado los escasos componentes de la población vamos a considerar que no se producirá mutación en esta generación.

Podemos, ahora, expresar estos nuevos resultados en la tabla siguiente:

Nueva población	Valor de x	$f(x) = x^2$
01100	12	144
11001	25	625
11011	27	729
10000	16	256
Suma		1754
Promedio		439
Máximo		729

Este proceso continúa de la misma manera y se detiene, entre otros criterios, cuando se lograron un número determinado de generaciones (por ejemplo, 10).

Comencemos a analizar lo que se pone de manifiesto en este ejercicio. El **promedio** (de 293 a 439) y el **máximo** (de 576 a 729) mejoran cuando se pasa de la primera generación a la segunda. Esto se logra combinando azar con una buena elección de padres (dado por sus fitness).

**Ejercicio**

Hacer un programa que utilice un Algoritmo Genético Canónico para buscar un máximo de la función:

$$f(x) = (x/\text{coef})^2 \text{ en el dominio } [0, 2^{30} - 1]$$

donde  $\text{coef} = 2^{30} - 1$

Teniendo en cuenta los siguientes datos:

- Probabilidad de Crossover = 0,75
- Probabilidad de Mutación = 0,05
- Población Inicial: 10 individuos
- Ciclos del programa: 20
- Método de Selección: Ruleta
- Método de Crossover: 1 Punto
- Método de Mutación: invertida

El programa debe mostrar finalmente el Cromosoma correspondiente al valor máximo obtenido y gráficas, usando EXCEL, de Máx, Mín y Promedio de la función objetivo por cada generación.

## 2.5. Planificación

### 2.5.1. Introducción

#### Un ejemplo de dominio: El mundo de los bloques

Las técnicas que se van a explicar pueden aplicarse a una gran variedad de dominios y de hecho así ha sido. Sin embargo, para poder hacer una comparación sencilla de los distintos métodos que se van a considerar, podría resultarnos útil observar el comportamiento de todos ellos en un único dominio que fuera lo suficientemente complejo como para que se necesitara usar todos los mecanismos, y lo suficientemente simple como para que se puedan seguir los ejemplos. Un dominio con estas características es el **mundo de los bloques**. En él, existe una superficie plana sobre la que se sitúan los bloques. Hay también varios bloques cúbicos todos ellos del mismo tamaño. Nosotros podemos situar unos bloques sobre otros. Tenemos un brazo robotizado que puede manipular los bloques. Las acciones que puede llevar a cabo incluyen:

- **DESAPILAR (A,B)**: Coger el bloque A que está situado sobre el bloque B. El brazo debe estar libre y el bloque A no debe tener bloques situados sobre él.
- **APILAR (A,B)**: Situar el bloque A sobre el bloque B. El brazo debe estar cogiendo a A y la superficie de B debe estar libre.
- **COGER(A)**: Coger el bloque A de la superficie y agarrarlo. El brazo debe estar libre y no debe existir nada sobre el bloque A.
- **BAJAR(A)**: Bajar el bloque A a la mesa (superficie plana). El brazo debe tener agarrando el bloque A.

Nótese que en el mundo que se acaba de describir, el brazo del robot solo puede coger un bloque a la vez. Además, como todos los bloques son del mismo tamaño, cada bloque solo puede tener como mucho un bloque situado sobre él.

Para poder especificar las condiciones bajo las cuales se puede llevar a cabo una acción y los resultados que ésta produce, es necesario utilizar los siguientes predicados:

- **SOBRE (A,B)**: El bloque A está sobre el bloque B.
- **SOBRELAMESA (A)**: El bloque A está sobre la mesa.
- **DESPEJADO (A)**: No hay nada sobre el bloque A.
- **AGARRADO (A)**: El brazo tiene agarrado el bloque A.
- **BRAZOLIBRE**: El brazo no está agarrando ningún bloque.

En el mundo de los bloques son ciertas varias sentencias lógicas. Por ejemplo:

$$\begin{aligned} [\exists x : \text{AGARRADO}(x)] &\rightarrow \neg \text{BRAZOLIBRE} \\ \forall x : \text{SOBRELAMESA}(x) &\rightarrow \neg \exists y : \text{SOBRE}(x,y) \\ \forall x : [\neg \exists y : \text{SOBRE}(y,x)] &\rightarrow \text{DESPEJADO}(x) \end{aligned}$$

La primera de esas sentencias simplemente indica que si el brazo está agarrando algo entonces no está vacío.

La segunda dice que si un bloque está sobre la mesa, no está sobre ningún bloque.

Por último, la tercera especifica que cualquier bloque sin otros situados sobre él, está despejado.

### 2.5.2. Componentes de un sistema de planificación

En los sistemas de resolución de problemas basados en técnicas elementales, era necesario llevar a cabo las siguientes funciones:

- Elegir la mejor regla para aplicar a continuación basándose en la mejor información heurística disponible.
- Aplicar la regla elegida para calcular el nuevo estado del problema que surge de su aplicación.
- Detectar cuando se ha llegado a una solución.
- Detectar callejones sin salida de forma que puedan abandonarse y que el esfuerzo del sistema se gaste en otras direcciones más fructíferas.

En sistemas más complejos, también debemos explorar técnicas para hacer cada una de estas tareas. Además, suele ser importante una quinta función:

- Detectar cuando se ha encontrado algo muy parecido a una solución correcta y emplear técnicas especiales para hacer que sea totalmente correcta.

Antes de explicar los diferentes métodos de planificación, es necesario revisar someramente las maneras de lograr estas cinco funciones.

#### Elección de las reglas a aplicar

La técnica más profunda para seleccionar reglas apropiadas que aplicar, *consiste en aislar el conjunto de diferencias existentes entre el objetivo deseado y el estado actual para poder identificar aquellas reglas que pueden reducir estas diferencias*. Si se encuentran varias reglas, puede usarse otro tipo de información heurística para poder elegir entre ellas. Esta técnica se basa en el método de análisis de medios y fines. Por ejemplo, si nuestro objetivo es tener una valla blanca que rodee nuestro jardincito y actualmente poseemos una valla marrón, deberíamos seleccionar operadores cuyos resultados fueran el cambio de color de un objeto. Si por otro lado, no disponemos de una valla, debemos considerar primero los operadores que construyan objetos de madera.

#### Aplicación de reglas

En los sencillos sistemas que se han explicado anteriormente, la aplicación de las reglas era sencilla. Simplemente las reglas especificaban el estado del problema que resultaba de su aplicación. Sin embargo, *ahora debemos ser capaces de trabajar con reglas que solo especifican una parte pequeña de un estado completo del problema*. Existen muchas formas de lograrlo.

*Una de estas formas consiste en describir para cada acción los cambios que realiza en la descripción del estado. Además, también son necesarias algunas sentencias que hagan que lo demás permanezca inalterado.*

Green (1969) propuso un ejemplo de este tipo de enfoque. En este sistema, *un estado se describe por un conjunto de predicados que representan los hechos que son ciertos en ese estado*. Cada estado diferente se representa explícitamente como parte de los predicados. Por ejemplo, la Figura 2.35 muestra un estado, llamado S0, que podría representar un problema sencillo en el mundo de los bloques.

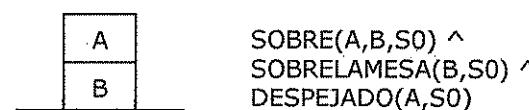


Figura 2.35

La manipulación de estas descripciones de los estados puede hacerse con un demostrador de teoremas por resolución. Así, por ejemplo, el efecto del operador **DESAPILAR(x,y)** podría describirse mediante el siguiente axioma.

[DESPEJADO(x,s) ^ SOBRE(x,y,s) →  
     [AGARRADO(x,HACER(DESAPILAR(x,y),s)) ^  
         DESPEJADO(y,HACER(DESAPILAR(x,y),s))]]

**HACER** es una función que especifica para un estado y una acción el nuevo estado que resulta de la ejecución de la acción.

El axioma dice que si DESPEJADO(x) y SOBRE(x,y) se cumplen en el estado s, entonces en el estado que resulta de HACER un DESAPILAR(x,y) a partir del estado s, se cumple que AGARRADO(x) y DESPEJADO(y).

Si ejecutamos DESAPILAR(A,B) en el estado S0 definido anteriormente, puede probarse, utilizando nuestras aserciones sobre S0 y sobre nuestro axioma para DESAPILAR, que en el estado que resulta de aplicar la operación de DESAPILAR (y lo denominamos estado S1) se cumple:

**AGARRADO(A,S1) ^ DESPEJADO(B,S1)**

Pero ¿qué más sabemos acerca de la situación del estado S1? Intuitivamente, sabemos que B todavía está sobre la mesa. Pero con lo que hemos visto hasta ahora, no podemos derivarlo. Para poder hacerlo, necesitamos también un conjunto de reglas denominadas **axiomas marco** (*frame axioms*), que describen los componentes del estado que no se ven afectados por cada operador. Por ejemplo, es necesario indicar que:

SOBRELAMESA(z,s) → SOBRELAMESA(z,HACER(DESAPILAR(x,y),s))

Este axioma dice que la relación **SOBRELAMESA** nunca se ve afectada por el operador **DESAPILAR**. También es necesario indicar que la relación **SOBRE** solo se ve afectada por el operador **DESAPILAR** si los bloques involucrados en la relación SOBRE son los mismos que los que involucra la operación DESAPILAR. Esto puede expresarse como:

[SOBRE(m,n,s) ^ ¬IGUAL(m,x)] →  
     SOBRE(m,n,HACER(DESAPILAR(x,y),s))

La ventaja de este enfoque es que se pueden llevar a cabo todas las operaciones necesarias en la descripción de los estados con un único y sencillo mecanismo como es el de resolución. Sin embargo, el precio que se paga es que el número de axiomas necesarios puede hacerse muy grande si las descripciones de los estados del problema son complejas. Por ejemplo, suponga que no solo estamos interesados en las posiciones de los bloques sino también en su color. Entonces, para cada operación (excepto posiblemente en PINTAR), sería necesario un axioma como el siguiente:

**COLOR(x,c,s) → COLOR(x,c,HACER(DESAPILAR(y,z),s))**

Para poder manipular dominios de problemas complejos, es necesario disponer de un mecanismo que no involucre un conjunto de axiomas marco demasiado grande.

En el sistema de resolución de problemas mediante robot **STRIPS** (Fikes y Nilsson, 1971) y en sus descendientes se usó un mecanismo de este tipo. Cada operador se describe mediante una lista de los nuevos predicados que el operador provoca que sean ciertos y una lista de los viejos predicados que el operador provoca que sean falsos. Estas dos listas se denominan lista **AÑADIR** y lista **BORRAR** respectivamente.

También debe especificarse una tercera lista para cada operador. Esta lista **PRECONDICIÓN** contiene aquellos predicados que deben ser ciertos para que pueda aplicarse el operador. Los axiomas marco del sistema de Green están explícitamente especificados en STRIPS. Cualquier operador no incluido ni en AÑADIR ni en BORRAR de un operador se asume que no se ve afectado por él. Esto significa que al especificar cada operador no es necesario considerar los aspectos del dominio que no están relacionados con él. Así, no es necesario decir nada acerca de la relación entre DESAPILAR y COLOR. Por supuesto, esto significa que debe usarse un mecanismo distinto a un sencillo demostrador de teoremas para poder calcular las descripciones de los estados después de que se lleven a cabo las operaciones.

Los operadores del estilo de STRIPS que se corresponden con las operaciones del mundo de los bloques que se han estado tratando aparecen en la Figura 2.36. Nótese que para reglas sencillas como estas, la lista PRECONDICIÓN con frecuencia es idéntica a la lista BORRAR. Para poder agarrar un bloque, el brazo debe estar libre: tan pronto como agarra el bloque, deja de estar libre. Pero las precondiciones no siempre son borradas. Por ejemplo, para que el brazo agarre un bloque, el bloque no debe tener otro situado sobre él. Después de que lo ha tomado, todavía sigue sin bloques sobre él. Esta es la razón de por qué las listas BORRAR y PRECONDICIÓN deben especificarse separadamente.

APILAR(x,y)
P: DESPEJADO(y) ^ AGARRADO(x)
B: DESPEJADO(y) ^ AGARRADO(x)
A: BRAZOLIBRE ^ SOBRE(x,y)
DESAPILAR(x,y)
P: SOBRE(x,y) ^ DESPEJADO(x) ^ BRAZOLIBRE
B: SOBRE(x,y) ^ BRAZOLIBRE
A: AGARRADO(x) ^ DESPEJADO(y)
COGER(x)
P: DESPEJADO(x) ^ SOBRELAMESA(x) ^ BRAZOLIBRE
B: SOBRELAMESA(x) ^ BRAZOLIBRE
A: AGARRADO(x)
BAJAR(x)
P: AGARRADO(x)
B: AGARRADO(x)
A: SOBRELAMESA(x) ^ BRAZOLIBRE

Figura 2.36

Al hacer implícitos los axiomas marco, se ha reducido enormemente la cantidad de información que hay que suministrar a cada operador. Esto significa, entre otras cosas, que cuando se introduce en el sistema un nuevo atributo posible a los objetos, no es necesario ir hacia atrás y añadir un nuevo axioma a cada uno de los operadores.

Pero ¿cómo podemos conseguir el efecto de utilizar los axiomas marco en el cálculo de descripciones completas de estados? El primer aspecto que sale a relucir es que para descripciones de estados complejas, la mayoría queda inalterada después de cada operación.

Pero si representamos el estado como una parte explícita de cada predicado, como en el sistema de Green, entonces toda la información debe deducirse de nuevo para cada estado. Para evitarlo, podemos abandonar el indicador explícito de estado a partir de predicados individuales y en su lugar simplemente actualizar una base de datos de predicados de forma que siempre describan el estado actual del mundo. Por ejemplo, si se comienza con la situación que se muestra en la Figura 2.35, podría describirse como

SOBRE(A,B) ^ SOBRELAMESA(B) ^ DESPEJADO(A)

Después de aplicar el operador DESAPILAR(A,B) nuestra descripción del mundo sería:

SOBRELAMESA(B) ^ DESPEJADO(A) ^ DESPEJADO(B) ^ AGARRADO(A)

Esto se deriva utilizando las listas AÑADIR y BORRAR especificadas como parte del operador DESAPILAR.

La simple actualización de una única descripción de estado funciona tan bien como llevar cuenta de los efectos de una secuencia dada de operadores. ¿Pero qué ocurre durante el proceso de búsqueda de la secuencia correcta de operadores? Si se explora una secuencia

incorrecta, se tiene que poder volver al estado original para intentarlo con otra secuencia diferente. Pero esto es posible siempre que la base de datos global describa el estado del problema en el nodo actual del grafo de búsqueda.

Todo lo que necesitamos hacer es almacenar en cada nodo los cambios que ha producido en la base de datos global cuando se pasó a través de ese nodo. Entonces, si se vuelve atrás hacia ese nodo, ya se pueden deshacer los cambios. Pero los cambios están exactamente en las listas AÑadir y Borrar de los operadores que se aplicaron para trasladarse de un nodo a otro. Así, necesitamos almacenar a lo largo de cada arco del grafo de búsqueda solo el operador que se aplicó.

En la Figura 2.37 se muestra un pequeño ejemplo de un árbol de búsqueda de este tipo y su correspondiente base de datos global. El estado inicial, descrito en la forma de STRIPS, es el que se muestra en la Figura 2.35. Nótese que podemos especificar no solo el operador (es decir, DESAPILAR) sino también sus argumentos, a fin de poder deshacer los cambios más tarde.

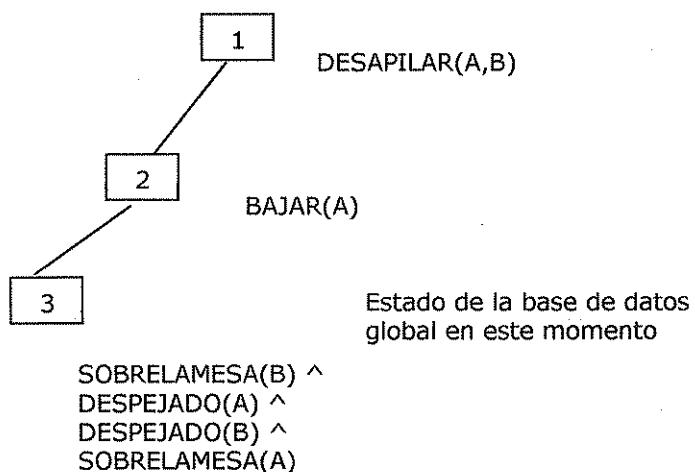


Figura 2.37

Suponga ahora que queremos explorar un camino diferente a partir del que se ha mostrado. En primer lugar se vuelve atrás a través del nodo 3 añadiendo cada uno de los predicados de la lista BORRAR de BAJAR en la base de datos global, y eliminar los elementos de la lista AÑadir de BAJAR. Después de hacerlo, la base de datos contiene:

SOBRELAMESA(B) ^ DESPEJADO(A) ^ DESPEJADO(B) ^ AGARRADO(A)

Como se esperaba, la descripción es idéntica a la que previamente se había calculado como resultado de aplicar DESAPILAR a la situación inicial. Si se repite el proceso usando las listas AÑadir y BORRAR de DESAPILAR, se deriva una descripción idéntica a la que se tenía cuando se comenzó.

Como para los dominios de problemas complejos resulta tan importante hacer implícitos los axiomas marco, todas las técnicas que veremos usan descripciones de los operadores disponibles siguiendo el estilo de STRIPS.

### Detección de una solución

*Se dice que un sistema de planificación ha tenido éxito al encontrar una solución a un problema cuando encuentra una secuencia de operadores que transforman el estado inicial del problema en un estado objetivo.*

¿Cómo puede saberse cuando ocurre este éxito? En los sistemas sencillos de resolución de problemas la respuesta es muy fácil, simplemente comparándolo con las descripciones del estado. Pero si no están representados explícitamente los estados completos, sino que lo están mediante un conjunto de propiedades relevantes, entonces el problema se hace más complejo. El modo de resolverlo depende de cómo estén representadas las descripciones de los estados. Con cualquier esquema de representación que se use es posible razonar con las representaciones para descubrir si una se empareja con la otra.

La lógica de predicados es una técnica de representación que ha servido como base para muchos de los sistemas de planificación que se han construido. Es adecuada por los mecanismos deductivos que proporciona. Suponga que como parte de un objetivo se tiene el predicado  $P(x)$ . Para ver si  $P(x)$  se satisface en un estado, se pregunta si podemos probar  $P(x)$  dadas las aserciones que describen el estado y los axiomas que definen el modelo del mundo (como por ejemplo, el hecho de que si el brazo está agarrando algo, entonces no está libre).

- Si podemos construir esta demostración, entonces el proceso de resolución del problema termina.
- Si no se puede, entonces debe proponerse una secuencia de operadores que podría resolverlo. Esta secuencia puede verificarse de la misma forma en que se hizo con el estado inicial preguntando si  $P(x)$  podía demostrarse a partir de los axiomas y de la descripción del estado derivado de la aplicación de los operadores.

### Detección de callejones sin salida

Cuando un sistema de planificación está buscando una secuencia de operadores que resuelva un problema concreto, debe ser capaz de detectar si está explorando un camino que nunca puede conducir a una solución (o al menos que no es probable que lo haga). Para detectar estos callejones sin salida pueden emplearse los mismos mecanismos de razonamiento que se usaron para detectar una solución.

- *Si el proceso de búsqueda razona hacia delante a partir del estado inicial, puede podar los caminos que conduzcan a un estado a partir del cual no se alcanza un estado objetivo.*

Por ejemplo, suponga que tenemos una cantidad fija de pintura: algo de blanco, algo de rosa y algo de rojo. Queremos pintar una habitación con las paredes rojo claro y el techo blanco. Podríamos conseguir el rojo claro añadiendo la pintura blanca a la roja. Pero entonces no podríamos pintar el techo de blanco. Por lo tanto, este intento debe abandonarse e intentar mezclar la pintura roja y la rosa. Pueden podarse también aquellos caminos que aunque no imposibilitan llegar a una solución, sí parece que no se encuentran muy cercanos a ella una vez que se exploran.

- *Si el proceso de búsqueda es hacia atrás a partir de un estado solución, se puede dar por finalizado un camino si se está seguro de que el estado inicial no puede alcanzarse o porque solo pueden hacerse pequeños progresos. En el razonamiento hacia atrás, cada objetivo se descompone en subobjetivos. Cada uno de ellos, además, puede descomponerse en subobjetivos adicionales.*

Algunas veces resulta sencillo detectar que no hay forma de que todos los subobjetivos de un conjunto dado puedan satisfacerse a la vez. Por ejemplo, el brazo del robot no puede estar libre y agarrando un bloque al mismo tiempo. Los caminos que intenten que estos dos objetivos sean ciertos simultáneamente pueden podarse inmediatamente. Otros caminos pueden podarse porque no conducen a ninguna parte. Por ejemplo, si al intentar satisfacer el objetivo A, el programa lo reduce a satisfacer los objetivos A, B y C, realmente se ha progresado poco. Se ha producido un problema más grande que el original, y el camino en cuestión debe abandonarse también.

### Identificación de soluciones casi correctas

Las clases de técnicas que se están explicando son con frecuencia útiles para resolver problemas *casi descomponibles*. Una buena forma de resolver estos problemas consiste en

asumir que son completamente descomponibles, resolver los subproblemas por separado, y verificar si al combinar las subsoluciones tenemos una solución al problema original. Por supuesto, si ocurre lo anterior no debe hacerse nada más.

Sin embargo, si no ocurre, se pueden hacer varias cosas.

- La más simple de todas es desechar la solución, buscar otra y esperar que resulte mejor que la anterior. Aunque es sencilla, esta estrategia puede conducir a malgastar una gran cantidad de esfuerzo.
- Un intento bastante mejor consiste en comparar la solución que resulta al ejecutarse la secuencia de operaciones correspondientes a la solución propuesta con el objetivo deseado. En la mayoría de los casos las diferencias entre las dos serán menores que las diferencias entre el estado inicial y el objetivo (asumiendo que la solución encontrada tiene partes buenas). En este punto, puede llamarse de nuevo al sistema de resolución de problemas y pedirle que encuentre una forma de eliminar estas nuevas diferencias.

La primera solución puede combinarse con esta segunda para formar una solución para el problema original.

- Otra forma todavía mejor de arreglar soluciones incompletas no consiste en intentar arreglarlas todas, sino en dejarlas especificadas de forma incompleta hasta que sea posible. Entonces, cuando esté disponible tanta información como sea posible, se completa la especificación de forma que no surjan conflictos.

Este enfoque puede denominarse como *estrategia de mínimo compromiso*. Puede aplicarse de varias formas. Una de ellas consiste en aplazar las decisiones en el orden en el que se llevan a cabo las operaciones. Así, en nuestro ejemplo anterior, en lugar de elegir arbitrariamente un orden para satisfacer un conjunto de precondiciones se dejará el orden sin especificar hasta casi al final. Entonces se mirarían los efectos de cada una de las subsoluciones para determinar las dependencias que existen entre ellas. Es en este punto cuando debe elegirse un ordenamiento.

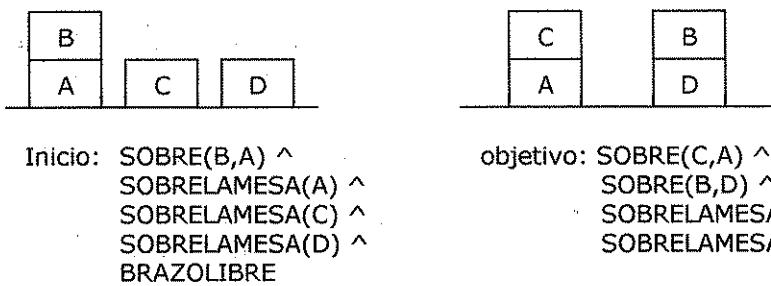


Figura 2.38

### 2.5.3. Planificación mediante una pila de objetivos

Una de las primeras técnicas que surgieron para componer objetivos que pueden interactuar, fue el uso de una **pila de objetivos**. Esto fue lo que se usó en **STRIPS**. En este método, el resolutor de problemas usa una pila que contiene tanto objetivos como operadores que deben proponerse para satisfacer estos objetivos.

El resolutor de problemas también usa una base de datos que describe la situación actual y un conjunto de operadores descritos mediante las listas PRECONDICIÓN, AÑADIR y BORRAR. Para ver como funciona este método analizaremos el ejemplo de la Figura 2.38.

Cuando se comienza con la resolución de este problema, la pila de objetivos es simplemente:

**SOBRE(C,A) ^ SOBRE(B,D) ^ SOBRELAMESA(A) ^ SOBRELAMESA(D)**

Pero queremos separar este problema en cuatro subproblemas, uno por cada componente del objetivo original. Dos de los subproblemas, **SOBRELAMESA(A)** y **SOBRELAMESA(D)**, son ya ciertos en el estado inicial. Por lo tanto solo tenemos que fijarnos en los dos restantes.

Al profundizar en el orden en que se desea atacar los subproblemas, deben crearse dos pilas de objetivos en este primer paso donde cada línea representa un objetivo de la pila y **SOBM** es una abreviatura de **SOBRELAMESA(A) ^ SOBRELAMESA(D)**:

SOBRE(C,A)  
SOBRE(B,D)  
SOBRE(C,A) ^ SOBRE(B,D) ^  
SOBM  
(1)

SOBRE(B,D)  
SOBRE(C,A)  
SOBRE(C,A) ^ SOBRE(B,D) ^  
SOBM  
(2)

1. En cada paso del proceso de resolución del problema se seguirá la pista del objetivo situado en la cima de la pila. Cuando se encuentra una secuencia de operadores que lo satisfacen, esta secuencia se aplica a la descripción del estado, obteniendo una nueva representación.
2. A continuación, se explora el objetivo situado en la cima de la pila y se intenta hacer que se satisfaga, comenzando a partir de la situación que se produjo como resultado de satisfacer el primer objetivo.
3. Este proceso continúa hasta que la pila de objetivos esté vacía.
4. Entonces, como última verificación, el objetivo original se compara con el estado final que surge de la aplicación de los operadores elegidos.
5. Si en este estado no se satisfacen algunas partes del objetivo (que puede que no ocurra si se alcanzó en un punto y más tarde se deshizo), entonces las partes no resueltas del subobjetivo se reinsertan en la pila y se repite el proceso.

Para continuar con el ejemplo que comenzamos antes, asuma que primero se intenta explorar la alternativa 1. La alternativa 2 conducirá también a la solución. De hecho, encuentra una tan fácilmente que no resulta muy interesante.

Al explorar la alternativa 1, verificamos primero si **SOBRE(C,A)** es cierto en el estado actual. Como no lo es, verificamos los operadores que pueden hacer que sea cierto. De los cuatro operadores que consideramos solo hay uno, **APILAR**, que debe llamarse con C y A. De esta forma, se sitúa **APILAR(C,A)** en la pila en lugar de **SOBRE(C,A)**, obteniendo:

**APILAR(C,A)**  
SOBRE(B,D)  
SOBRE(C,A) ^ SOBRE(B,D) ^ SOBM

**APILAR(C,A)** reemplaza a **SOBRE(C,A)** porque después de llevar a cabo APILAR se garantiza que **SOBRE(C,A)** se cumplirá.

Pero para poder aplicar **APILAR(C,A)**, deben satisfacerse sus precondiciones, de forma que deben establecerse como subobjetivos. De nuevo, debemos separar un objetivo compuesto:

#### **DESPEJADO(A) ^ AGARRADO(C)**

en sus componentes y elegir el orden en el que deben trabajar. En este punto, resulta adecuado **utilizar algún conocimiento heurístico**. **AGARRADO(x)** es muy fácil de lograr. Al menos es necesario dejar algo y coger el objeto deseado. Pero **AGARRADO** es también muy fácil de deshacer. Para poder realizar cualquier otra cosa, el robot necesitará utilizar el brazo. Así, si alcanzamos **AGARRADO** en primer lugar y después se intenta hacer algo más, es muy probable que lleguemos a un callejón sin salida.

Así utilizamos la **heurística** de que si uno de los objetivos que deben cumplirse simultáneamente es AGARRADO, éste debe realizarse el último. Esto produce una nueva pila de objetivos:

DESPEJADO(A)  
 AGARRADO(C)  
 DESPEJADO(A) ^ AGARRADO(C)  
**APILAR(C,A)**  
 SOBRE(B,D)  
 SOBRE(C,A) ^ SOBRE(B,D) ^ SOBM

Este tipo de información heurística debería estar en la lista PRECONDICIÓN comenzando los predicados en el orden en el que deberían realizarse.

A continuación se ve si DESPEJADO(A) es cierto. No lo es. El único operador que puede convertirlo en cierto es **DESAPILAR(B,A)**. Por lo tanto, se intenta aplicar. Esto produce la pila de objetivos:

SOBRE(B,A)  
 DESPEJADO(B)  
 BRAZOLIBRE  
 SOBRE(B,A) ^ DESPEJADO(B) ^ BRAZOLIBRE  
**DESAPILAR(B,A)**  
 AGARRADO(C)  
 DESPEJADO(A) ^ AGARRADO(C)  
**APILAR(C,A)**  
 SOBRE(B,D)  
 SOBRE(C,A) ^ SOBRE(B,D) ^ SOBM

Esta vez, cuando comparamos el elemento de la cima de la pila de objetivos, SOBRE(B,A), con el modelo del mundo, se ve que se satisface. Así, lo sacamos y consideramos el siguiente objetivo, DESPEJADO(B). Esto, aunque es cierto en el modelo del mundo, no está especificado explícitamente como uno de los predicados iniciales. Pero a partir de los predicados iniciales y del axioma del mundo de los bloques que dice que cualquier bloque sin bloques sobre él está despejado, un demostrador de teoremas puede concluir que DESPEJADO(B). Así, este subobjetivo también puede extraerse de la pila.

La tercera precondition para DESAPILAR(B,A) permanece. Es **BRAZOLIBRE**, y también es cierto en el modelo del mundo actual, por lo que puede eliminarse también de la pila.

El siguiente elemento de la pila es el objetivo combinado que representa todas las precondiciones de DESAPILAR(B,A). Lo verificamos para estar seguros de que se satisface en el modelo del mundo. Esto será así a no ser que se deshaga uno de sus componentes al intentar satisfacer otro. En este caso, no hay problemas y el *objetivo combinado se extrae de la pila*.

Ahora, el elemento de la cima de la pila es el operador **DESAPILAR(B,A)**. Hemos garantizado que todas sus precondiciones se satisfagan, por lo que puede aplicarse para producir un nuevo modelo del mundo a partir del cual continuará el proceso de resolución de problemas. Esto se logra usando las listas AÑADIR y BORRAR especificadas en DESAPILAR.

Mientras tanto, *almacenamos la información* de que DESAPILAR(B,A) ha sido el primer operador de la secuencia propuesta como solución. En este momento, la base de datos correspondiente al modelo del mundo es:

**SOBRELAMESA(A) ^ SOBRELAMESA(C) ^ SOBRELAMESA(D) ^ AGARRADO(B) ^ DESPEJADO(A)**

La pila de objetivos es:

AGARRADO(C)

DESPEJADO(A) ^ AGARRADO(C)

**APILAR(C,A)**

SOBRE(B,D)

SOBRE(C,A) ^ SOBRE(B,D) ^ SOBM

Ahora se intenta satisfacer el objetivo AGARRADO(C). Existen dos operadores que pueden hacer que AGARRADO(C) sea cierto: **COGER(C)** y **DESAPILAR(C,x)**, donde x podría ser cualquier bloque del que se pueda desapilar C. Sin más información, no podemos decir cual de los dos operadores es el apropiado, por lo que creamos dos ramas en el árbol de búsqueda que se corresponden con las siguientes pilas de objetivos:

SOBRELAMESA(C)  
DESPEJADO(C)  
BRAZOLIBRE  
SOBRELAMESA(C) ^  
DESPEJADO(C) ^  
BRAZOLIBRE  
**COGER(C)**  
DESPEJADO(A) ^ AGARRADO(C)  
**APILAR(C,A)**  
SOBRE(B,D)  
SOBRE(C,A) ^ SOBRE(B,D) ^  
SOBM

SOBRE(C,x)  
DESPEJADO(C)  
BRAZOLIBRE  
SOBRE(C,x) ^  
DESPEJADO(C) ^  
BRAZOLIBRE  
**DESAPILAR(C,x)**  
DESPEJADO(A) ^ AGARRADO(C)  
**APILAR(C,A)**  
SOBRE(B,D)  
SOBRE(C,A) ^ SOBRE(B,D) ^  
SOBM

Nótese que para la segunda alternativa, la pila de objetivos contiene la variable x, la cual aparece en tres sitios. Aunque x puede sustituirse por cualquier bloque, es importante que sea el mismo en las tres apariciones. Así, es importante que cuando se introduce una variable en la pila, el nombre sea diferente al de las variables que ya se encuentran en la pila. Además, una vez que se elige un objeto candidato para ligarlo con una variable, este enlace debe recordarse para que cuando se produzcan otras apariciones de la misma variable se relacione con el mismo objeto.

¿Cómo debería elegir nuestro programa entre las alternativas 1 y 2? Puede decirse que coger C(alternativa 1) es mejor que desapilarlo, ya que no está sobre nada. Para desapilar algo, primeramente debe estar apilado sobre algo. Aunque puede hacerse, es un esfuerzo malgastado. ¿Pero cómo podría saber esto nuestro programa? Supongá que se decide seguir por la alternativa 2 en primer lugar. Para satisfacer SOBRE(C,x), tenemos que **APILAR C** sobre el bloque x. Entonces la pila de objetivos sería:

DESPEJADO(x)  
AGARRADO(C)  
DESPEJADO(x) ^ AGARRADO(C)  
**APILAR(C,x)**  
DESPEJADO(C)  
BRAZOLIBRE  
SOBRE(C,x) ^ DESPEJADO(C) ^ BRAZOLIBRE  
**DESAPILAR(C,x)**

DESPEJADO(A)  $\wedge$  AGARRADO(C)  
**APILAR(C,A)**  
 SOBRE(B,D)  
 SOBRE(C,A)  $\wedge$  SOBRE(B,D)  $\wedge$  SOBM

Pero dese cuenta de que ahora una de las precondiciones de APILAR es **AGARRADO(C)**. Esto es lo que estamos intentando conseguir aplicando DESAPILAR, lo cual necesita aplicar APILAR para que la precondición SOBRE(C,x) se satisfaga.

Por lo tanto, volvemos a nuestro objetivo inicial. De hecho, ahora tenemos objetivos adicionales ya que se han añadido otros predicados a la pila. En este momento se determina que este camino es improductivo. Sin embargo, si el bloque C hubiera estado sobre otro bloque en el estado actual, SOBRE(C,x) se satisfaría inmediatamente sin necesidad de hacer un APILAR y este camino conduciría a una buena solución.

Ahora tenemos que volver a la alternativa 1, en la que se utilizaba COGER, para conseguir que el brazo agarrara a C. El elemento en la cima de la pila de objetivos es SOBRELAMESA(C), que como se satisface y, se elimina de la pila. El siguiente elemento es DESPEJADO(C), que también se satisface y se extrae de la pila. La siguiente precondición de COGER(C), es BRAZOLIBRE, que no se satisface ya que **AGARRADO(B)** es cierto.

Existen dos operadores que pueden hacer que BRAZOLIBRE sea cierto: APILAR(B,x) y BAJAR(B). Es decir, podemos situar B sobre la mesa o sobre otro bloque. ¿Cuál de las dos elegimos?

Si investigamos un poco, se ve que al final lo que queremos es conseguir B sobre D. Lo más eficaz sería situar B allí en este momento. Nuestro programa podría darse cuenta de esto comparando los elementos de las listas AÑADIR de los operadores del resto de la pila de objetivos. Si uno de los operadores provoca un efecto fortuito que hace cierto alguno de los objetivos, debería elegirse. De esta forma, elegimos aplicar **APILAR(B,D)** enlazando D con x en el operador APILAR. Esto hace que la pila de objetivos sea:

DESPEJADO(D)  
 AGARRADO(B)  
 DESPEJADO(D)  $\wedge$  AGARRADO(B)  
**APILAR(B,D)**  
 SOBRELAMESA(C)  $\wedge$  DESPEJADO(C)  $\wedge$  BRAZOLIBRE  
**COGER(C)**  
 DESPEJADO(A)  $\wedge$  AGARRADO(C)  
**APILAR(C,A)**  
 SOBRE(B,D)  
 SOBRE(C,A)  $\wedge$  SOBRE(B,D)  $\wedge$  SOBM

DESPEJADO(D) y AGARRADO(B) son ambas ciertas. Ahora la operación APILAR(B,D) puede realizarse y produce el siguiente modelo del mundo:

**SOBRELAMESA(A)  $\wedge$  SOBRELAMESA(C)  $\wedge$  SOBRELAMESA(D)  $\wedge$  SOBRE(B,D)  $\wedge$  BRAZOLIBRE**

En este momento, se satisfacen todas las precondiciones de COGER(C), por lo que puede ejecutarse. Entonces, todas las precondiciones de APILAR(C,A) son ciertas y puede, por lo tanto, ejecutarse.

Ahora ya podemos empezar a trabajar con la segunda parte de nuestro objetivo original, **SOBRE(B,D)**. Pero ya se ha satisfecho gracias a las operaciones realizadas para satisfacer el primer subobjetivo. Esto ocurrió porque al intentar elegir entre las posibles alternativas cuando el brazo estaba agarrando a B, se inspeccionó la pila de objetivos para ver si uno de los

operadores posibles provocaba efectos laterales, y se vio que era así. Por lo tanto, ahora extraemos SOBRE(B,D) de la pila de objetivos.

A continuación hay que realizar la última verificación consistente en que el objetivo combinado **SOBRE(C,A) ^ SOBRE(B,D) ^ SOBRELAMESA(A) ^ SOBRELAMESA(D)** se cumpla en cada una de sus partes, lo cual, por supuesto, se cumple. Entonces el resolutor de problemas puede ahora devolver como respuesta el plan:

- 1. DESAPILAR(B,A)**
- 2. APIALAR(B,D)**
- 3. COGER(C)**
- 4. APIALAR(C,A)**

En este sencillo ejemplo se ha visto la forma en que puede aplicarse la información heurística para guiar el proceso de búsqueda, intentando detectar caminos no provechosos y considerando ciertas interacciones entre objetivos que pueden ayudar a crear una buena solución en su conjunto. Sin embargo, para problemas de una mayor dificultad estos métodos no son adecuados.

#### 2.5.4. Planificación no lineal mediante fijación de restricciones

La anomalía de Sussman de la Figura 2.39 es un buen ejemplo que muestra la necesidad de un plan no lineal.

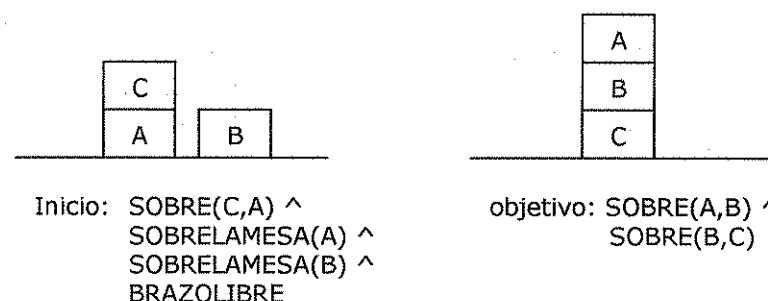


Figura 2.39

Existen dos maneras para comenzar la resolución de este problema, que corresponden a las pilas de objetivos:

$\text{SOBRE}(A,B)$   
 $\text{SOBRE}(B,C)$   
 $\text{SOBRE}(A,B) \wedge \text{SOBRE}(B,C)$

(1)

$\text{SOBRE}(B,C)$   
 $\text{SOBRE}(A,B)$   
 $\text{SOBRE}(A,B) \wedge \text{SOBRE}(B,C)$

(2)

Suponga que elegimos la alternativa 1 y comenzamos intentando conseguir A sobre B. Para ello se desapila C de A y se logra el primer subobjetivo **SOBRE(A,B)**. Ahora ya podemos comenzar a trabajar para satisfacer **SOBRE(B,C)**. Pero para hacerlo, tiene que desapilar A de B. Cuando se alcanza el objetivo **SOBRE(B,C)** y se intenta verificar el objetivo restante de la pila **SOBRE(A,B) ^ SOBRE(B,C)**, se descubre que no se satisface. Se ha deshecho **SOBRE(A,B)** en el proceso de alcanzar **SOBRE(B,C)**. La diferencia entre el objetivo y el estado actual es **SOBRE(A,B)**, que se añade a la pila por lo que hay que alcanzarlo de nuevo. Finalmente el objetivo se satisface.

Aunque este plan alcanza el objetivo deseado, no lo hace de una forma demasiado eficaz. Algo similar ocurre si se examinan los dos objetivos principales en orden opuesto. El método que se está usando no es capaz de encontrar una forma eficiente de resolver este problema.

El método de *planificación con pila de objetivos* aborda los problemas como objetivos conjuntos resolviendo por orden los objetivos uno cada vez. Este método genera un plan que contiene una secuencia de operadores que resuelven el primer objetivo, seguido por una secuencia completa para el segundo objetivo, etc. Pero como se ha visto, ***los problemas difíciles provocan interacciones entre los objetivos***. Los operadores que se utilizan para resolver un subproblema pueden interferir en la solución de un subproblema anterior.

La mayoría de los problemas necesitan un plan entrelazado en el que *se trabaje simultáneamente con múltiples subproblemas*. ***Este tipo de plan se denomina plan no lineal*** ya que no está compuesto por una secuencia lineal de subplanes completos.

Un buen plan para solucionar este problema es el siguiente:

1. Comenzar el trabajo con el objetivo SOBRE(A,B) despejando A y poniendo C sobre la mesa.
2. Alcanzar el objetivo SOBRE(B,C) apilando B sobre C.
3. Completar el objetivo SOBRE(A,B) apilando A sobre B.

La idea de ***fijación de restricciones*** es construir un plan mediante operadores incrementalmente hipotéticos, ordenamientos parciales entre operadores y enlaces entre variables y operadores. En un cierto instante del proceso de resolución del problema, se puede tener un conjunto de operadores útiles pero quizás no una idea muy clara sobre cómo ordenar estos operadores entre ellos.

*Una solución es un conjunto de operadores parcialmente ordenados y parcialmente instanciados para generar un plan intermedio, se convierte el orden parcial en un número de órdenes totales.*

#### 2.5.5. Planificación jerárquica

Para resolver problemas complicados, los resolutores de problemas tienen que generar planes muy extensos. Para poder hacerlo eficientemente, es importante ***poder eliminar algunos de los detalles del problema hasta que se encuentre una solución que resuelva los principales escollos***. Una forma de hacer esto es sustituir los detalles apropiados.

Los primeros intentos de lograrlo usaban macro-operadores en donde se construían operadores grandes a partir de otros más pequeños. Pero con este enfoque, no se eliminan los detalles de las descripciones de los operadores. En el sistema ABSTRIPS (Sacerdoti, 1974) se utilizó un enfoque algo mejor en el cual la planificación se realizaba con una *jerarquía de espacios de abstracción*, en cada uno de los cuales se ignoran las precondiciones de un nivel de abstracción más bajo.

Como ejemplo, suponga que quiere visitar a un amigo en Europa, pero tiene una cantidad limitada de dinero para gastar. Tendría sentido verificar primero el precio del billete de avión ya que encontrar un vuelo razonable en cuanto a precio será la parte más difícil del trabajo. Usted no se debería preocupar de cosas como llegar a la entrada, planificar la ruta al aeropuerto, o aparcar su coche hasta que no esté seguro de que tiene un vuelo que tomar.

El enfoque que usa ABSTRIPS para resolver un problema es el siguiente:

- En primer lugar resuelve el problema completamente, considerando solo aquellas precondiciones cuyo valor crítico sea el más alto posible. Estos valores reflejan la dificultad esperada para satisfacer una precondición. Para lograrlo, sigue el mismo procedimiento que STRIPS, pero simplemente ignora las precondiciones que caen por debajo de un cierto nivel crítico.

- Una vez hecho esto, utiliza el plan construido como el esbozo de un plan completo, y considera las precondiciones del siguiente nivel de criticidad más bajo.
- Entonces aumenta el plan con los operadores que satisfacen estas precondiciones. De nuevo, al elegir los operadores, ignora todas aquellas precondiciones cuya criticidad sea menor que el nivel que ahora se está considerando.

Este proceso se denomina *búsqueda primero en longitud* debido a que explora planes completos a un nivel de detalle antes de mirar los detalles de más bajo nivel de algunos de ellos.

La **asignación de valores de criticidad apropiados** es claramente un aspecto crucial para el éxito de este método de **planificación jerárquica**. Aquellas precondiciones que no tengan operadores que puedan satisfacerlas son las más críticas.

Por ejemplo, si intentamos resolver un problema que incluya el movimiento de un robot a lo largo de una casa y consideramos el operador PASAR-POR-LA-PUERTA, la precondición de que exista una puerta lo suficientemente ancha para que el robot pueda pasar a través de ella es lo más crítico, ya que si ocurre así (en una situación normal) nada de lo que podamos hacer puede lograr que no sea cierto. Pero la precondición de que la puerta esté abierta es de una criticidad menor si disponemos del operador ABRIR-PUERTA.

Para que un sistema de planificación jerárquica funcione con reglas del estilo de STRIPS, debe darse junto con las propias reglas, el valor de criticidad apropiado para cada término que pueda aparecer en la precondición. Dados estos valores, el proceso básico puede trabajar en gran parte de la misma forma en que lo hacen los sistemas de planificación no jerárquicos. Sin embargo, no se malgastarán esfuerzos en eliminar los detalles de planes que no estén cercanos a la resolución del problema.

## 2.5.6. Sistemas reactivos

Hasta ahora, se ha descrito un proceso de planificación deliberativo, en donde *el plan que resuelve una tarea completa se construye antes de actuar*. Sin embargo, existe un camino muy diferente que podría aproximarse al problema de decidir qué hacer. La idea de los **sistemas reactivos** consiste en **evitar planificar totalmente y, en lugar de eso, utilizar la situación observable como pista a la que simplemente reaccionar**.

*Un sistema reactivo debe tener acceso a algún tipo de base de conocimiento que describa las acciones que deben realizarse bajo ciertas circunstancias. Un sistema reactivo es muy diferente de los sistemas de planificación que se han explicado hasta ahora, porque elige solo una acción cada vez; no anticipa y selecciona una secuencia completa de acciones antes de realizar una primera acción.*

Uno de los sistemas reactivos más sencillos es un termostato. El trabajo de un termostato consiste en mantener constante la temperatura de una habitación. Uno podría imaginarse soluciones a este problema que necesiten cantidades significativas de planificación, teniendo en cuenta los cambios durante el día de la temperatura externa, cómo fluye el calor de una habitación a otra y otros muchos aspectos. Sin embargo, un termostato real no utiliza más que un par de sencillas reglas de situación-acción.

1. Si la temperatura de la habitación está k grados por encima de la temperatura deseada, entonces conectar el aire acondicionado.
2. Si la temperatura de la habitación está k grados por debajo de la temperatura deseada, entonces desconectar el aire acondicionado.

Resulta que los sistemas reactivos son capaces de mantener comportamientos sorprendentemente complejos, especialmente en tareas del mundo real tales como la navegación de robots.

La *principal ventaja* que presentan los sistemas reactivos frente a los planificadores tradicionales es que *funcionan de forma robusta en dominios difíciles de modelar con exactitud de forma completa*.

**Los sistemas reactivos evitan un modelado completo y basan sus acciones directamente en sus percepciones del mundo.**

En dominios complejos e impredecibles, la habilidad de planificar una secuencia fija de pasos a lo largo del tiempo es de un valor cuestionable.

*Otra ventaja de los sistemas reactivos es que son extremadamente sensibles, ya que evitan la explosión combinatoria que implica una planificación deliberativa.* Esto hace que sean muy atractivos para tratar con tareas en tiempo real como conducir y caminar.

*Por otra parte, como los sistemas reactivos no mantienen ningún modelo del mundo ni estructuras explícitas del objetivo, su rendimiento es limitado en este tipo de tareas.* Por ejemplo, parece poco probable que un sistema puramente reactivo sea capaz de jugar al ajedrez a alto nivel.

## **UNIDAD DIDACTICA 3**

**Representación del  
Conocimiento  
y Razonamiento**



## UNIDAD DIDACTICA 3

### EJE CONCEPTUAL

Representación del Conocimiento y Razonamiento

#### TEMAS

<b>3.1. El problema de la Representación del Conocimiento .....</b>	<b>5</b>
3.1.1. Correspondencia entre Conocimiento y RC .....	5
3.1.2. Propiedades de un buen sistema de RC.....	6
3.1.3. Modelos de Representación del Conocimiento .....	6
3.1.3.1. Conocimiento relacional simple .....	7
3.1.3.2. Conocimiento heredable .....	8
3.1.3.3. Conocimiento deductivo .....	9
3.1.3.4. Conocimiento procedimental .....	10
3.1.4. Problemas de la Representación del Conocimiento.....	10
3.1.5. El problema del Marco .....	13
<b>3.2. Lógica Simbólica.....</b>	<b>15</b>
3.2.1. La Lógica y el Lenguaje.....	15
3.2.1.1. Introducción .....	15
3.2.1.2. Naturaleza del Argumento .....	15
3.2.1.3. Verdad y Validez .....	17
3.2.1.4. Lógica Simbólica.....	18
3.2.2. Argumentos que contienen Enunciados Compuestos .....	20
3.2.2.1. Enunciados Simples y Compuestos .....	20
3.2.2.2. Enunciados Condicionales.....	24
3.2.2.3. Formas de Argumentos y Tablas de Verdad .....	26
3.2.2.4. Formas Sentenciales .....	30
3.2.3. El Método de Deducción .....	34
3.2.3.1. Prueba Formal de Validez .....	34
3.2.3.2. La Regla de Reemplazo .....	36
3.2.3.3. Demostración de la Invalidez .....	39
<b>3.3. Lógica de Predicados.....</b>	<b>41</b>
3.3.1. Introducción y concepto de Semidecidible .....	41
3.3.2. Representación de hechos simples en lógica .....	42
3.3.3. La Representación de las relaciones instancia y es-un.....	46
3.3.4. Representación de Funciones calculables y Predicados computables .....	47
3.3.5. Método de Resolución .....	50
3.3.6. Conversión a forma clausal.....	50
3.3.7. Las bases de la resolución .....	53
3.3.8. Resolución en lógica proposicional.....	53
3.3.9. El algoritmo de unificación .....	55
3.3.10. Resolución en lógica de predicados .....	56
<b>3.4. Representación del conocimiento mediante Reglas .....</b>	<b>59</b>
3.4.1. Comparación entre conocimiento procedimental y conocimiento declarativo.....	59
3.4.2. Programación Lógica .....	61
3.4.3. Diferencia entre Razonamientos hacia delante y hacia atrás .....	63
3.4.3.1. Sistemas de reglas encadenadas hacia atrás.....	63
3.4.3.2. Sistemas de reglas encadenadas hacia delante.....	64
3.4.3.3. Combinación del razonamiento hacia adelante y hacia atrás ....	64
<b>3.5. Razonamiento simbólico bajo incertidumbre.....</b>	<b>65</b>
3.5.1. Razonamiento No Monótono .....	65
3.5.1.1. Razonamiento por defecto .....	65
3.5.1.1.1. Lógica no monótona.....	66
3.5.1.1.2. Lógica por defecto .....	66
3.5.1.2. Razonamiento minimalista .....	67
3.5.1.2.1. La suposición de un mundo cerrado .....	67
3.5.1.2.2. Circunscripción .....	68
3.5.1.3. Cuestiones sobre la implementación .....	68
3.5.1.3.1. Implementación búsqueda primero en profundidad ....	69

3.5.1.3.2. Implementación búsqueda primero en anchura.....	70
3.5.2. Razonamiento Estadístico.....	71
3.5.2.1. Factores de certeza y sistemas basados en reglas.....	73
3.5.2.2. Redes Bayesianas.....	76
3.5.2.3. Teoría de Dempster-Shafer.....	76
<b>3.6. Estructuras de Ranura y Relleno Débiles.....</b>	<b>80</b>
3.6.1. Redes Semánticas .....	80
3.6.2. Marcos.....	82
<b>3.7. Estructuras de Ranura y Relleno Fuertes.....</b>	<b>83</b>
3.7.1. Dependencia Conceptual.....	83
3.7.2. Guiones .....	85
3.7.3. CYC.....	87

### 3.1. El problema de la Representación del Conocimiento

#### 3.1.1. Correspondencia entre Conocimiento y RC

Para resolver los complejos problemas con los que se enfrenta la Inteligencia Artificial, es necesario disponer tanto de una *gran cantidad de conocimiento* como de una serie de *mecanismos que permitan manipularlo* con el fin de obtener soluciones a nuevos problemas. En los programas de IA se ha representado el conocimiento (hechos) de muy distintas maneras. Pero, antes de pasar a un estudio detallado, debemos resaltar una característica que está presente en todas las representaciones, el hecho de que estamos manejando dos tipos de entidades:

- Hechos: verdades en un cierto mundo. Es aquello que queremos representar.
- Representaciones de los hechos en un determinado formalismo: estas son las entidades que realmente seremos capaces de manipular.

Una posible estructuración consiste en clasificar estas entidades en dos niveles distintos:

- El nivel del conocimiento, donde se describen los hechos (incluyendo el comportamiento y los objetivos de cada agente).
- El nivel simbólico, donde se describen los objetos del nivel del conocimiento en términos de símbolos manipulables por programas.

*La relación que se establece entre los hechos reales y la representación de los mismos se denomina correspondencia de la representación.* La representación hacia delante establece una correspondencia entre los hechos y sus representaciones, mientras que la representación hacia atrás establece la correspondencia inversa, desde las representaciones a los hechos.

Veamos un ejemplo sencillo donde utilizaremos la *lógica matemática* como *formalismo de representación*. Consideremos la frase:

**Spot es un perro.**

Esta frase representa un hecho que en lógica se puede escribir como:

**perro(Spot)**

Supongamos que también disponemos de un formalismo lógico para representar el hecho de que todos los perros tienen rabo:

**$\forall x: \text{perro}(x) \rightarrow \text{tienerabo}(x)$**

Y entonces, mediante el mecanismo deductivo de la lógica, se puede generar una nueva representación para el objeto:

**tienerabo(Spot)**

Utilizando una apropiada función de correspondencia hacia atrás, se puede generar la correspondiente frase en castellano:

**Spot tiene rabo.**

O bien se podría utilizar esta representación como detonante de una determinada acción o en la derivación de las representaciones de otros hechos.

*Es importante recordar que las funciones de correspondencia no suelen ser biunívocas.* De hecho no suelen ser ni siquiera funciones, sino relaciones de muchos a muchos.

*En algunas ocasiones, el uso de una buena representación hace que el mecanismo de razonamiento de un programa sea no solo correcto sino también trivial.*

### 3.1.2. Propiedades de un buen sistema de RC

Un buen sistema de representación del conocimiento en un dominio particular debe poseer las siguientes propiedades:

- **Suficiencia de la representación:** La capacidad de representar todos los tipos de conocimiento necesarios en el dominio.
- **Suficiencia deductiva:** La capacidad para manipular las estructuras de la representación con el fin de obtener nuevas estructuras que se correspondan con un nuevo conocimiento deducido a partir del antiguo.
- **Eficiencia deductiva:** La capacidad de incorporar información adicional en las estructuras de conocimiento con el fin de que los mecanismos de inferencia puedan seguir las direcciones más prometedoras.
- **Eficiencia en la adquisición:** La capacidad de adquirir nueva información con facilidad. El caso más simple es aquél en el que una persona inserta directamente el conocimiento en la base de datos. Idealmente, el programa sería capaz de controlar la adquisición de conocimiento por sí mismo.

Desgraciadamente todavía no se ha encontrado ningún sistema que optimice todos estos aspectos que sea aplicable a cualquier tipo de conocimiento. En consecuencia, existen múltiples técnicas para la representación del conocimiento. Muchos programas utilizan más de una técnica. En los próximos capítulos se describirán con detalle las más importantes de estas técnicas, pero, a modo de introducción, en este apartado se pasa revista a las principales ideas.

### 3.1.3. Modelos de Representación del Conocimiento

#### Espectro Sintáctico-Semántico de la Representación

Por un lado, hay **sistemas puramente sintácticos** en los que *no se tiene en cuenta el significado del conocimiento que está siendo representado*. Tales sistemas poseen reglas simples y uniformes para manipular la representación. No tienen en cuenta la información que contiene ésta. Por otro lado, existen **sistemas puramente semánticos** en los que no hay una forma unificada. Cada aspecto de la representación corresponde a una parte de información diferente y las reglas de inferencia son, por tanto, complicadas.

Se verán nueve estructuras declarativas diferentes para representar el conocimiento:

- Lógica de predicados
- Reglas de producción
- Sistemas no monótonos
- Sistemas de razonamiento estadístico
- Redes semánticas
- Marcos
- Dependencia conceptual
- Guiones
- CYC

De todas ellas, las **representaciones lógicas** (*lógica de predicados y sistemas no monótonos*) y las **estadísticas**, son las más puramente sintácticas. Sus reglas de inferencia son procedimientos estrictamente sintácticos que operan sobre fórmulas bien formadas (fbf) a pesar de lo que éstas representan. Los sistemas de **reglas de producción** son también, **principalmente sintácticos**.

Los intérpretes para estos sistemas normalmente usan únicamente información sintáctica para decidir qué reglas desestiman. De nuevo se ve la similitud entre lógica y reglas de producción como formas de representación y utilización del conocimiento. Pero es posible construir sistemas del tipo regla-producción, que tienen más semántica englobada en ellos. Por ejemplo,

en EMYCIN y otros sistemas que proporcionan un soporte explícito a los factores de certeza, las semánticas de dichos elementos son utilizadas por el intérprete de reglas para guiar este comportamiento.

Las **estructuras de ranura y relleno** están normalmente orientadas más a la semántica, aunque se extienden bastante profundamente dentro de este espectro. Las redes semánticas, como su propio nombre indica, están diseñadas para capturar las relaciones semánticas entre entidades, y normalmente se utilizan con un conjunto de reglas de inferencia que han sido especialmente diseñadas para manejar correctamente los tipos específicos de arcos presentes en la red. (Por ejemplo, las uniones del tipo es-un se tratan de un modo diferente que la mayoría del resto de enlaces). Los **sistemas de marcos** están normalmente más estructurados que las redes semánticas y contienen un conjunto incluso mayor de reglas de inferencia especializadas, incluyendo aquellas que llevan a cabo un array completo de reglas de inferencia por defecto, además de otros procedimientos como la verificación de la consistencia.

La **dependencia conceptual** va incluso más allá en cuanto a una representación semántica en lugar de una sintáctica. No solo proporciona la estructura abstracta de una representación, sino que además proporciona una indicación específica sobre los componentes que debe contener la representación (como las primitivas ACTs y las relaciones de dependencia). Así aunque las representaciones CD se pueden considerar como instancias de las redes semánticas, éstas pueden ser utilizadas por mecanismos de inferencia más potentes, que utilicen el conocimiento específico que contienen.

Y aunque los **guiones** se presentan muy similares a los marcos, en realidad se trata de marcos en los que las ranuras se han elegido cuidadosamente para representar la información que es útil cuando se razona acerca de situaciones. Esto hace posible que los procedimientos de manipulación de los guiones puedan aprovechar el conocimiento con el que están trabajando y así, poder resolver problemas más eficazmente. **CYC** utiliza tanto los marcos como la lógica (dependiendo del nivel en que utilicemos el conocimiento) para codificar los tipos específicos de conocimiento e inferencia asistida en el razonamiento de sentido común. El CYC es el más semántico de los sistemas que se han visto, ya que internamente proporciona el conocimiento más desarrollado para la manipulación de los diferentes tipos de estructuras del conocimiento.

### 3.1.3.1. Conocimiento relacional simple

El modo más sencillo de representar los hechos declarativos es mediante un *conjunto de relaciones del mismo tipo que las utilizadas en los sistemas de bases de datos*. En la Figura 3.1 se muestra un ejemplo de dichos sistemas relacionales. Se dice que esta representación es **simple** debido a la escasa capacidad deductiva que ofrece. Pero el conocimiento representado de esta forma puede servir como entrada a otros mecanismos de inferencia más potentes.

Por ejemplo, dado los hechos de la Figura 3.1 no es posible responder a una pregunta tan sencilla como "¿quién es el felino más rápido?". Pero dado un procedimiento para encontrar el felino más rápido, estos hechos permitirían que dicho procedimiento calcule una respuesta.

Nombre	Felino Salvaje	Promedio velocidad	Zoo
Julio	León	72	Luján
Claudio	Guepardo	99	BsAs
Tutio	Tigre	80	Luján

Figura 3.1

Los sistemas de bases de datos están diseñados para proporcionar el soporte adecuado al conocimiento relacional. Por tanto, no nos extenderemos más en explicaciones de este tipo de estructura de conocimiento. Los problemas prácticos que surgen cuando se intenta conectar un sistema de base de datos, que proporciona este tipo de soporte, con un sistema de representación del conocimiento, que incorpora otras capacidades que describiremos a continuación, ya han sido resueltos en diferentes productos comerciales.

### 3.1.3.2. Conocimiento heredable

El conocimiento relacional de la Figura 3.1 se compone de un conjunto de atributos que junto con unos valores asociados permite describir los objetos de la base de conocimiento. El conocimiento acerca de los objetos, de sus atributos y de sus valores no tiene que ser tan simple como el que se muestra en el ejemplo. En particular, es posible *extender la representación básica con unos mecanismos de inferencia que operen sobre la estructura de la representación*. Para que esto sea efectivo, la estructura se debe diseñar de acuerdo con el mecanismo de inferencia que se desee. **Una de las fórmulas más útiles de inferencia es la herencia de propiedades, donde los elementos de una clase heredan los atributos y los valores de otras clases más generales en los que están incluidos.**

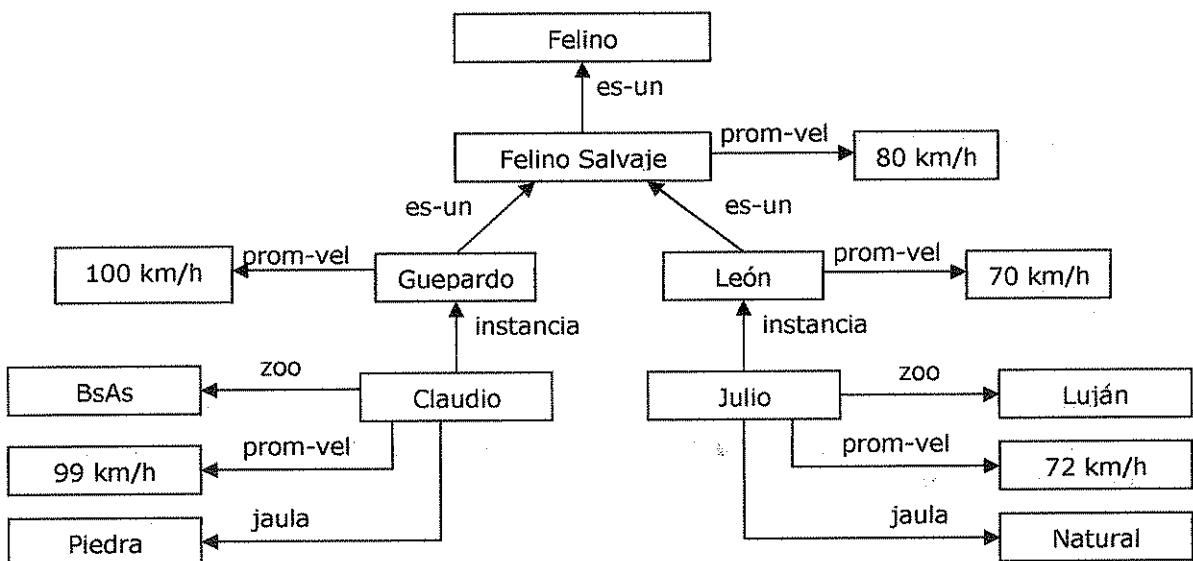


Figura 3.2

Para dar soporte a la **herencia de propiedades**, los objetos se deben organizar en **clases**, y las clases se deben disponer como una **jerarquía de generalizaciones**. En la Figura 3.2 aparece una estructura organizada de esta forma, en ella se ha introducido conocimiento acerca de felinos salvajes. Las **líneas** representan **atributos**; los **nodos recuadrados** representan **objetos y valores de los atributos de los objetos**. Estos valores, a su vez, también se pueden ver como objetos con atributos y valores, y así sucesivamente. Las **flechas** conectan **los objetos con sus valores a través de los correspondientes atributos**. La estructura que aparece en la figura es lo que se denomina una **estructura de ranura y relleno (slot-and-filler)**. También se puede denominar red semántica o colección de estructuras (frames). En este último caso, cada estructura representa la colección de atributos y valores asociados con un nodo en particular.

La Figura 3.3 muestra el nodo correspondiente a un león representado como una estructura. No hay que dejarse desanimar por la confusión terminológica. Es tan flexible el modo en que se puede utilizar ésta (y el resto de las estructuras que se describen en este apartado) para resolver cada uno de los problemas concretos de representación, que es difícil reservar palabras precisas para cada representación particular. En general, el uso del término sistema de estructuras (*frame system*) implica, de alguna manera, una mayor estructuración en los atributos y en el mecanismo de inferencia que cuando se utiliza el término red semántica.

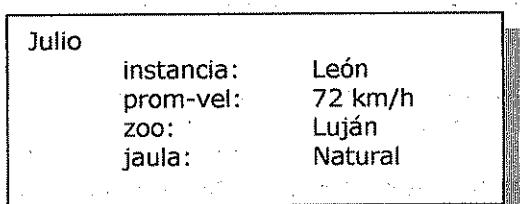


Figura 3.3

Lo que haremos aquí será dar una idea acerca de la manera en que sirven de soporte a la deducción mediante el conocimiento que contienen. Todos los objetos y la mayoría de los atributos que se utilizan en este ejemplo pertenecen al dominio de los felinos y no son significativos en general. Las dos excepciones son el atributo **es-un** (is a), que se utiliza para indicar *que una clase está contenida en otra*, y el atributo **instancia**, que se utiliza para indicar *pertenencia a una clase*.

*Estos dos atributos específicos (de utilidad general) son la base de la herencia de propiedades como una técnica de inferencia. Mediante esta técnica se puede acceder a la base de conocimiento y recuperar los hechos que han sido explícitamente almacenados en ella, así como otros hechos que se pueden deducir de los primeros.*

Una forma ideal de algoritmo de herencia de las propiedades se podría enunciar de la siguiente forma.

### Algoritmo: herencia de propiedades

Para acceder al valor V de un atributo A en una instancia I:

1. Encontrar I en la base de conocimiento.
2. Si el atributo A tiene algún valor asignado, devolver ese valor.
3. En caso contrario, comprobar si el atributo *instancia* tiene algún valor asignado. Si no lo tiene entonces fallar.
4. En caso contrario, ir al nodo identificado por ese valor y comprobar si allí existe algún valor para el atributo A. Si lo hay, devolverlo.
5. En caso contrario, repetir hasta que el atributo *es-un* no tenga valor asignado o hasta encontrar una respuesta:
  - a) Obtener el atributo *es-un* e ir a ese nodo.
  - b) Comprobar si el atributo A tiene algún valor. Si lo tiene, devolverlo.

Este procedimiento es una simplificación. No dice nada acerca de cómo proceder cuando hay más de un valor para los atributos *instancia* o *es-un*. Pero aún así describe el mecanismo básico de la herencia.

#### 3.1.3.3. Conocimiento deductivo

La herencia de propiedades es una forma muy potente de inferencia, pero no es la única. En algunas ocasiones es necesario echar mano de **toda la potencia de la lógica tradicional** y aún más) para describir las inferencias apropiadas. En la Figura 3.4 se muestra un ejemplo de la **lógica de predicados de primer orden** aplicada a la representación de conocimiento adicional acerca de los felinos salvajes.

$$\forall x: \text{felino\_salvaje}(x) \wedge \text{hambriento}(x) \wedge \text{prom\_vel}(x, 72) \wedge \text{presa}(y) \wedge \text{prom\_vel}(y, 20) \rightarrow \text{come}(x, y)$$

Figura 3.4

Por supuesto, este conocimiento será inútil a menos que se disponga de un mecanismo de inferencia que lo pueda aprovechar (de la misma forma que el conocimiento por omisión del ejemplo anterior habría sido inútil sin el algoritmo que permitía recorrer la base de conocimiento).

El **procedimiento de inferencia** necesario en este caso será uno que implemente las **reglas lógicas de la inferencia**. Existen muchos de tales mecanismos, algunos de los cuales razonan hacia adelante a partir de los hechos hasta llegar a las conclusiones, mientras que otros razonan hacia atrás desde las conclusiones buscadas hasta los hechos de partida. Entre los más utilizados de estos procedimientos se encuentra el de **resolución** que utiliza una estrategia de prueba por contradicción.

### 3.1.3.4. Conocimiento procedimental

Hasta ahora los ejemplos sobre la base de conocimiento acerca de los felinos salvajes se han centrado en hechos declarativos relativamente estáticos. Pero existe otro tipo de conocimiento, **operacional o procedimental**, igualmente útil, que especifica qué hacer cuando se da una determinada situación. El conocimiento procedimental se puede representar en los programas de muchas maneras distintas. La manera más habitual consiste en especificarlo como **un código** (*en algún lenguaje de programación como PROLOG que hace algo*). La máquina utiliza el conocimiento cuando ejecuta el código para llevar a cabo una determinada tarea. Un ejemplo de esto se ve en la Figura 3.5

```
come (X,Y) :- felino_salvaje (X), hambriento (X), prom_vel(X,72),
           presa (Y), prom_vel (Y,20)
```

Figura 3.5

### 3.1.4. Problemas de la Representación del Conocimiento

Antes de embarcarnos en la discusión de los mecanismos específicos que se utilizan para la representación de los distintos tipos de conocimiento acerca del mundo real, es necesario señalar brevemente una serie de cuestiones presentes en todos ellos:

#### A. Atributos importantes

**¿Existen atributos tan genéricos que aparezcan en prácticamente todos los dominios de aplicación? Si es así, es necesario asegurarse de que sean tratados adecuadamente en cada uno de los mecanismos que se propongan. Si esos atributos existen ¿Cuáles son?**

Hay dos atributos con una especial significación, cuyo uso nos han sido presentados anteriormente: **instancia** y **es-un**. Estos atributos son importantes debido a que en ellos se apoya la herencia de propiedades. Reciben diferentes nombres en los sistemas de IA, pero el nombre es lo de menos, lo importante es que dichos atributos representan la *pertenencia a una clase* y la *inclusión de una clase en otra*, y que *la inclusión de clases es transitiva*.

#### B. Relaciones entre atributos

**¿Se pueden establecer relaciones relevantes entre los atributos de los objetos?**

Los atributos que se utilizan para describir a los objetos pueden ser a su vez entidades representables. ¿Qué propiedades poseen independientemente del conocimiento específico que codifiquen? Hay cuatro propiedades que merece la pena mencionar aquí:

- Inversos
- Existencia en una jerarquía es-un
- Técnicas para el razonamiento acerca de los valores
- Atributos univalueados

#### Inversos

Las entidades del mundo se pueden relacionar de muy diversas maneras. Pero desde el momento en que decidimos describir esas relaciones como atributos, nos restringimos a una perspectiva según la cual nos fijamos en un objeto y tratamos de establecer las relaciones que existen entre él y los otros. Los atributos son esas relaciones.

Por ejemplo, en el caso de la siguiente representación:

zoo(Julio,Luján)

se puede interpretar igualmente como una afirmación acerca de Julio o acerca de Luján. Su uso efectivo dependerá del resto de los hechos que contenga el sistema.

Otra aproximación consiste en utilizar atributos que fijen un determinado punto de vista, pero utilizándolos por parejas, de manera que ***uno sea el inverso del otro***. De esta manera, tendremos dos hechos:

- uno asociado con Julio  
zoo = Luján
- otro asociado con Luján  
felinos = Julio, ....

Esta es la aproximación que se toma en los sistemas de redes semánticas y los sistemas basados en marcos. Cuando se utiliza, suele ir acompañada de una herramienta para la adquisición de conocimiento que obliga a la *declaración conjunta de los atributos inversos*.

### **Existencia en una jerarquía es-un**

De la misma forma que existen clases de objetos y subconjuntos más específicos de esas clases, también se puede hablar de *atributos* y *especializaciones de los atributos*. Considérese, por ejemplo, el atributo felinos\_salvajes. Este atributo es en realidad una especialización del atributo felinos, que a su vez, es una especialización del atributo mamíferos. Este tipo de relaciones de ***generalización-especialización*** referidas a los atributos tienen la misma misión que cuando se aplican a los demás conceptos – *servir de soporte a la herencia*. En el caso de los atributos, la información que se hereda consiste en cosas tales como restricciones sobre los valores que un atributo puede tomar o mecanismos para el cómputo de dichos valores.

### **Técnicas para el razonamiento acerca de los valores**

En ocasiones los valores de los atributos se especifican explícitamente durante la creación de una base de conocimiento. Pero generalmente el sistema debe razonar sobre valores que no ha recibido explícitamente. Hay varios tipos de información que pueden jugar un determinado papel en este razonamiento, incluyendo:

- Información acerca del tipo del valor. El valor de altura debe ser un número medido en una unidad de longitud.
- Restricciones sobre el valor. La edad de una persona no puede ser mayor que la de ninguno de sus progenitores.
- Reglas para el cómputo de un valor cuando sea necesario. Reglas hacia atrás.
- Reglas que describen las acciones que se deberían llevar a cabo en el caso de que se llegase a conocer un determinado valor. Reglas hacia adelante.

### **Atributos univaluados**

Un tipo de atributo, no por específico menos útil, es aquel que ***solo puede tomar un único valor***. Por ejemplo, un león determinado solo puede pertenecer a un único zoo, y tiene una única velocidad promedio. Cuando se pretenda definir un nuevo valor para uno de estos atributos que ya tuviera otro valor asignado previamente, solo puede ser por dos razones. O bien se ha producido un cambio en el mundo o bien existe una contradicción en la base de conocimiento que es necesario resolver.

### C. Selección de la granularidad de la representación

#### ¿A qué nivel se debe representar el conocimiento?

Independientemente del mecanismo de representación que se elija, es necesario responder a la siguiente cuestión: *“¿A qué nivel de detalle se debería representar el mundo?”* Un buen ejemplo servirá de ilustración al problema. Supongamos que estamos interesados en representar el hecho:

##### **Juan vislumbró a Susana**

Una posible representación sería:

vislumbró(agente(Juan), objeto(Susana))

Con esta representación sería sencillo responder a la pregunta:

¿Quién vislumbró a Susana?

Pero supongamos que queremos saber si:

##### **¿Vio Juan a Susana?**

La respuesta, obviamente, es que “sí”, pero no es una respuesta que podamos obtener a partir del único hecho conocido hasta ahora. Por supuesto se podrían añadir otros hechos como:

vislumbró(x,y) → vio(x,y)

Otra posible solución consistiría en representar explícitamente el hecho de que vislumbrar es en realidad una forma particular de ver. Se podría escribir algo así como:

vio(agente(Juan),  
      objeto(Susana),  
      duración(breve))

El problema de *elegir la granularidad de la representación adecuada no es fácil*. Está claro que mientras más bajo sea el nivel que escogamos, más sencillo será razonar para ciertos casos, a costa de un proceso de inferencia más complejo para obtener esa representación a partir del lenguaje natural y de un mayor espacio de almacenamiento, puesto que muchas inferencias se representarán muchas veces.

### D. La representación de conjuntos de objetos

#### ¿Cómo se deben representar los conjuntos de objetos?

La posibilidad de representar conjuntos de objetos es importante por varias razones.

- En primer lugar, existen *propiedades que se verifican en conjuntos de objetos pero no así en los elementos particulares de los conjuntos*. Por ejemplo, “En Australia hay más ovejas que personas”.
- La otra razón por la que es importante disponer de algún medio de representación de conjuntos es que *cuando existe una propiedad que verifican todos los elementos de un conjunto, es más eficiente asociar esa propiedad al conjunto que asociarla individualmente a cada uno de sus componentes*.

Hay dos formas de definir un conjunto y sus elementos.

- La primera consiste en **enumerar todos los elementos**. Es lo que se denomina una *definición por extensión*.
- La segunda consiste en **dar una determinada regla**, de forma que cuando se evalúa un determinado objeto, da como resultado verdadero o falso según que el objeto pertenezca o no al conjunto. Tales reglas son denominadas *definiciones por comprensión*.

Por ejemplo, una definición por extensión del conjunto de los planetas habitados por personas de nuestro sistema solar es {Tierra}. La definición por comprensión sería:

{x: planeta\_del\_sistema\_solar(x) ^ posición\_respecto\_al\_sol(x,3)}

De esta forma, es muy fácil determinar cuando son iguales dos conjuntos definidos por extensión, pero no así cuando los conjuntos se definen por comprensión.

Sin embargo, las representaciones por comprensión permiten la definición de conjuntos infinitos y de conjuntos en los que no se conocen explícitamente todos sus elementos.

La segunda propiedad es que los conjuntos descritos por comprensión se pueden definir dependientes de parámetros modificables, como el tiempo o la localización espacial.

## E. Búsqueda de la estructura adecuada a cada circunstancia

### Dada una base de conocimiento muy extensa ¿Cómo acceder a los fragmentos relevantes en cada momento?

Una vez que se ha encontrado una estructura de conocimiento candidata a resolver el problema en curso, es necesario establecer una correspondencia detallada entre ambos. Los detalles del proceso de correspondencia dependerán de la representación que se utilice. Puede consistir en establecer las ligaduras adecuadas entre los objetos y las variables, o puede que sea necesario hacer comparaciones entre atributos. En cualquier caso, a medida que se localicen los valores que satisfagan las restricciones impuestas por la estructura de conocimiento, aquellos irán ocupando sus correspondientes lugares dentro de la estructura.

Al intentar acceder a los fragmentos relevantes de la base de conocimiento, en cada momento es importante tener en cuenta los siguientes puntos:

- Tomar aquellas partes de la estructura seleccionada que se hayan conseguido identificar en la presente situación y utilizarlas en la búsqueda de las posibles alternativas.
- Obviar el fallo de la estructura actual y seguir utilizándola. Por ejemplo, una silla con solo tres patas puede que simplemente esté rota, o que haya otro objeto delante que impida ver la cuarta pata.
- Utilice enlaces que conecten las estructuras y que sugieran posibles direcciones de búsqueda.
- Si las estructuras de conocimiento están almacenadas como una jerarquía es-un, ascender por la jerarquía hasta encontrar una estructura lo bastante general como para no entrar en contradicción con la situación en curso.

#### 3.1.5. El problema del Marco

En este capítulo se han descrito diversos métodos de representación del conocimiento que permiten la construcción de descripciones de estados complejos en un programa de búsqueda.

*Existe una cuestión adicional relacionada con la representación eficiente de las secuencias de estados que se generan en un proceso de búsqueda.*

Esta puede ser una tarea difícil cuando se trate de problemas complejos o que presenten estructuras extrañas.

#### Considérese el mundo de los robots domésticos.

Existe un gran número de objetos y relaciones a representar, de modo que en las descripciones de los estados se deben incluir hechos tales como **encima(Planta12,Mesa34)**, **debajo(Mesa34,Ventana13)** y **dentro(Mesa34,Habitación15)**. Una posible estrategia consiste en almacenar la descripción de cada estado como una lista de tales hechos. Pero qué ocurre durante el proceso de resolución de problemas si cada una de dichas descripciones es muy larga?

*La mayoría de los hechos no cambiará de un estado a otro, a pesar de lo cual se representarán en cada uno de los nodos y la memoria se llenará rápidamente.*

Y lo que es más, **se empleará la mayor parte del tiempo en la creación de estos nodos y en la copia de estos hechos** – la mayor parte de los cuales no cambia – de unos nodos a otros.

Por ejemplo, en el mundo del robot, se emplearía mucho tiempo repitiendo en cada nodo **debajo(Suelo, Techo)**. Y todo esto además de resolver el verdadero problema que consiste en determinar los cambios que se deberían producir de un nodo al siguiente.

*Al problema de la representación de los hechos que cambian, así como de aquellos que no lo hacen, es a lo que se conoce como **problema del marco** (frame problem) (McCarthy y Hayes, 1969).*

En determinados dominios, el único problema consiste en la representación de todos los hechos. En otros, en cambio no es trivial determinar cuales son los que cambian. Por ejemplo, en el mundo del robot, se puede tener una planta encima de una mesa que está debajo de una ventana. Supongamos que se desplaza la mesa al centro de la habitación. Se deberá inferir que la planta también está ahora en el centro de la habitación pero que la ventana no.

Para llevar a cabo este tipo de razonamiento, hay algunos sistemas donde se utilizan explícitamente unos axiomas denominados **axiomas del marco**, que **describen las cosas que cambian cuando se aplica un determinado operador sobre el estado n para alcanzar el estado n+1**. (Todo aquello que cambia forma parte del propio operador).

Así, en el dominio del robot, se escribirían axiomas como éste:

$$\text{Color}(x,y,s_1) \wedge \text{desplaza}(x,s_1,s_2) \rightarrow \text{color}(x,y,s_2)$$

que se puede leer como, "Si  $x$  es de color  $y$  en el estado  $s_1$  y se aplica la operación de desplazamiento de  $x$  en el estado  $s_1$  para llegar al estado  $s_2$ , entonces el color de  $x$  en  $s_2$  sigue siendo  $y$ ".

Desgraciadamente en los dominios complejos es necesario utilizar un enorme número de axiomas como éstos. Otra aproximación consiste en suponer que solo cambia aquello que debe. Donde por "debe" se entiende que los cambios vendrán dados explícitamente por los axiomas que describen al operador o que se deducen de manera lógica de algún cambio explícito. Esta idea de circunscribir el conjunto de cosas inusuales es muy potente; se puede utilizar como una solución parcial al problema estructural y como un modo de razonamiento con conocimiento incompleto.

### 3.2. Lógica Simbólica

#### 3.2.1. La Lógica y el Lenguaje

##### 3.2.1.1. Introducción

###### ¿Qué es la Lógica?

Es fácil hallar respuestas a la pregunta "¿Qué es la Lógica?" Según Charles Peirce, "Se han dado casi un centenar de definiciones de ella". Pero Peirce continúa diciendo:

*"Sin embargo, se concederá generalmente que su problema central es la clasificación de los argumentos, de modo que todos los que sean malos se pongan de un lado y los que sean buenos del otro. . ."*

**El estudio de la Lógica, es el estudio de los métodos y principios usados al distinguir entre los argumentos correctos (buenos) y los argumentos incorrectos (malos).**

Con esta definición no se intenta implicar, desde luego, que uno puede hacer la distinción sólo si ha estudiado lógica. Pero el estudio de ésta ayudará a distinguir entre los argumentos correctos e incorrectos, y lo hará de varias maneras. Ante todo, en el estudio propio de la lógica, ésta se aborda como un arte y como una ciencia.

- Aquí, como en cualquier parte, *la práctica ayudará a alcanzar la perfección.*
- En segundo lugar, el estudio de la lógica, especialmente la lógica simbólica, como el estudio de cualquier ciencia exacta *incrementará la capacidad de razonamiento.*
- Y por último, el estudio de la lógica dará al estudiante ciertas *técnicas para probar la validez de todos los argumentos*, incluyendo los suyos. Este conocimiento tiene valor porque cuando los errores son de fácil detección es menos probable que se cometan.

La lógica se ha definido con frecuencia como la **ciencia del razonamiento**. Esta definición, aunque da una clave a la naturaleza de la lógica, no es muy exacta. El razonamiento es la clase especial de pensamiento llamada **inferencia**, en la que **se sacan conclusiones partiendo de premisas**. Como pensamiento, sin embargo, no es campo exclusivo de la lógica, sino parte también de la materia de estudio del psicólogo. Los psicólogos que examinan el proceso del razonamiento lo encuentran en extremo complejo y altamente emocional, consistente en torpes procedimientos de prueba y error iluminados por súbitas – y a veces en apariencia inconsecuentes – visiones internas. Todos son de importancia para la psicología.

**Pero el lógico no se interesa en el proceso real del razonamiento. A él le importa la corrección del proceso completado.**

Su pregunta siempre es: ¿se sigue la conclusión alcanzada de las premisas usadas o supuestas? Si las premisas son un fundamento adecuado para aceptar la conclusión, si afirmar que las premisas son verdaderas garantiza el afirmar la verdad de la conclusión, entonces el razonamiento es correcto. De otra manera es incorrecto.

Los métodos y técnicas del lógico se han desarrollado primordialmente con el objeto de aclarar la distinción. El lógico se interesa en todo razonamiento, sin atender al contenido mismo, sino sólo desde este punto de vista especial.

##### 3.2.1.2. Naturaleza del Argumento

La **inferencia** es una actividad en la que se afirma una proposición sobre la base de otra u otras proposiciones aceptadas como el punto de partida del proceso. *Al lógico no le concierne el proceso de inferencia, sino las proposiciones iniciales y finales de ese proceso de las relaciones entre ellas.*

**Las proposiciones son o verdaderas o falsas**, y en esto difieren de las preguntas, órdenes y exclamaciones. Los gramáticos clasifican las formulaciones lingüísticas de las proposiciones, preguntas, órdenes, y exclamaciones, en oraciones declarativas, interrogativas, imperativas y exclamatorias, respectivamente. Estas nociones son familiares.

Es costumbre distinguir entre las oraciones declarativas y las proposiciones que se afirman al pronunciar aquellas.

- La distinción se hace resaltar observando que **una oración declarativa es siempre parte de un lenguaje**, lengua en que se dice o se describe, mientras que **las proposiciones no son privativas de ninguna de las lenguas en que se expresa**.
- Otra diferencia es que **la misma oración articulada en diferentes contextos puede afirmar diferentes proposiciones**. (por ej. la oración "tengo hambre", puede ser proferida por personas diferentes haciendo aserciones diferentes).

La misma clase de distinción puede establecerse entre las oraciones y los enunciados. Puede hacerse el mismo enunciado utilizando palabras diferentes, y la misma oración puede ser dicha en contextos diferentes para hacer enunciados diferentes. Los términos "enunciados" y "proposición" no son sinónimos exactos, pero en los escritos de los lógicos se usa más o menos en el mismo sentido.

Aquí se usarán los dos términos. En los capítulos siguientes usaremos también el término "enunciado" y el término "proposición" refiriéndonos a las oraciones en las que se expresan los enunciados (y las proposiciones). En cada caso, el significado quedará claro por el contexto.

A cada *inferencia* corresponde un *argumento*, y de estos argumentos trata la lógica primordialmente.

**Un argumento puede definirse como un grupo cualquiera de proposiciones o enunciados de los cuales se afirma que hay uno que se sigue de los demás, considerando éstos como fundamento de la verdad de aquél.**

La palabra argumento también tiene otros significados en su uso cotidiano, pero en la lógica tiene el sentido técnico explicado. En los capítulos que siguen usaremos también la palabra argumento en un sentido derivado para referirnos a una oración cualquiera o colección de oraciones en que está formulado o expresado un argumento. Cuando así lo hagamos, presupondremos que la claridad del contexto permite asegurar que al pronunciar esas oraciones se hacen enunciados únicos o se afirman proposiciones únicas.

Todo argumento tiene una estructura, en cuyo análisis usualmente se emplean los términos "**premisa**" y "**conclusión**".

**La conclusión de un argumento es la proposición afirmada basándose en las otras proposiciones del argumento y estas otras proposiciones que se afirman como fundamento o razones para la aceptación de la conclusión son las premisas de ese argumento.**

Notemos que "premisa" y "conclusión" son términos relativos, en el sentido de que la misma proposición puede ser premisa en un argumento y conclusión en otro. Así, todos los hombres son mortales es premisa en el argumento

**Todos los hombres son mortales.**

Sócrates es un hombre.

Por lo tanto, Sócrates es mortal.

y conclusión en el argumento

Todos los animales son mortales.

Todos los hombres son animales.

**Luego, todos los hombres son mortales.**

**Toda proposición puede ser premisa o conclusión dependiendo del contexto.**

Es una premisa cuando se presenta en un argumento en el que se le supone para demostrar alguna otra proposición y es una conclusión cuando se presenta en un argumento que se pretende la demuestra basándose en las otras proposiciones que se suponen .

Es costumbre distinguir entre **argumentos deductivos e inductivos**.

En todos los argumentos se pretende que las premisas proporcionen algún fundamento para la verdad de sus conclusiones, pero sólo en un **argumento deductivo** se pretende que sus premisas proveen un fundamento **absolutamente concluyente**.

Los términos técnicos “**válido**” e “**inválido**” se usan en lugar de “correcto” e “incorrecto” al caracterizar los argumentos deductivos.

**Un argumento deductivo es válido cuando sus premisas y conclusiones están relacionadas de modo tal que es absolutamente imposible que las premisas sean verdaderas, a menos que la conclusión lo sea también.**

La tarea de la lógica deductiva es la que aclara la naturaleza de la relación que existe entre premisas y conclusión en un argumento válido, y proporcionar las técnicas de discriminación entre los válidos y los inválidos.

En los argumentos inductivos sólo se pretende que sus premisas proporcionen **algún fundamento para sus conclusiones**. Ni el término “válido” ni su opuesto “inválido” se aplican con propiedad a los argumentos inductivos. Los argumentos inductivos difieren entre sí en el **grado de verosimilitud o probabilidad que sus premisas confieren a sus conclusiones**, y se les estudia en la lógica inductiva. Pero en este libro nos ocuparemos solamente de los argumentos deductivos y usaremos la palabra “**argumento**” en referencia exclusiva a los argumentos deductivos.

### 3.2.1.3. Verdad y Validez

**La verdad y falsedad** caracterizan las proposiciones o los enunciados, y puede decirse, en sentido derivado, que caracterizan las oraciones declarativas en que se les formula. Pero los argumentos no se caracterizan propiamente por cuanto que son verdaderos o falsos.

**Por otro lado, la validez y la invalidez** caracterizan los argumentos más bien que las proposiciones o los enunciados.

Hay una conexión entre la validez o invalidez de un argumento y la verdad o falsedad de sus premisas y conclusión, pero esta conexión no es de ningún modo una conexión simple.

Algunos argumentos válidos solamente contienen proposiciones verdaderas, como, por ejemplo,

Todos los murciélagos son mamíferos.  
Todos los mamíferos tienen pulmones.  
Luego, todos los murciélagos tienen pulmones.

Pero un argumento puede contener proposiciones falsas exclusivamente y ser válido a pesar de todo, como, por ejemplo,

Todas las truchas son mamíferos.  
Todos los mamíferos tienen alas.  
Luego, todas las truchas tienen alas.

**Este argumento es válido porque si sus premisas fuesen verdaderas su conclusión tendría que ser verdadera también**, aunque de hecho son falsas.

Estos dos ejemplos muestran que, aunque algunos argumentos válidos tienen conclusiones verdaderas, no todos las tienen verdaderas.

**La validez de un argumento no garantiza la verdad de su conclusión.**

Cuando consideramos el argumento

Si soy presidente entonces soy famoso.  
Yo no soy presidente.  
Por tanto, yo no soy famoso.

podemos ver que aunque tanto las premisas como la conclusión son verdaderas, es un argumento inválido. Su invalidez se hace obvia al compararlo con otro argumento de la misma forma:

Si Rockefeller es presidente, entonces es famoso.  
Rockefeller no es presidente.  
Luego, Rockefeller no es famoso.

Este argumento es claramente inválido, puesto que sus premisas son verdaderas pero su conclusión es falsa. Los dos últimos ejemplos muestran que aun cuando algunos argumentos inválidos tienen conclusiones falsas no todos las tienen falsas.

**La falsedad de su conclusión no garantiza la invalidez de un argumento. Pero la falsedad de su conclusión sí garantiza que o el argumento es inválido o por lo menos una de sus premisas es falsa.**

Hay dos condiciones que debe satisfacer un argumento para establecer la verdad de su conclusión.

**Debe ser válido y todas sus premisas deben ser verdaderas.**

Al lógico sólo atañe una de estas condiciones. Determinar la verdad o falsedad de las premisas es tarea de la investigación científica en general, pues las premisas pueden tratar de cualquier asunto. Pero determinar la validez o invalidez de los argumentos es el campo especial de la lógica deductiva. Al lógico le interesa la cuestión de la validez aún para argumentos cuyas premisas puedan ser falsas.

Podría cuestionarse la legitimidad de ese interés. Podría sugerirse que se confinara nuestra atención sólo a los argumentos de premisas verdaderas. Pero es frecuentemente necesario depender de la validez de argumentos cuyas premisas son de verdad desconocida. Los científicos modernos investigan sus teorías deduciendo conclusiones de las mismas que predicen el comportamiento de fenómenos observables en el laboratorio o el observatorio.

La conclusión se pone a prueba directamente por observación y, si es verdadera esto tiende a confirmar la teoría de donde se dedujo, pero si es falsa queda refutada la teoría. En uno y otro caso el científico tiene un interés vital en la validez del argumento por el que la conclusión puesta a prueba se deduce de la teoría investigada porque si el argumento es inválido, su procedimiento es inútil. Lo que precede es una descripción sobreimplificada del método científico, pero sirve para mostrar que las cuestiones de validez son importantes aún en argumentos de premisas falsas.

### 3.2.1.4. Lógica Simbólica

Se ha explicado que a la lógica le conciernen los argumentos y que éstos contienen proposiciones o enunciados como sus premisas y conclusiones. Estas últimas no son entidades lingüísticas, como las oraciones declarativas, sino más bien son lo que las oraciones declarativas típicamente afirman al ser articuladas.

Sin embargo, la comunicación de proposiciones y argumentos requiere el uso del lenguaje, y esto complica nuestro problema. Los argumentos formulados en inglés o cualquier otro lenguaje natural son de difícil evaluación debido a la vaga y equívoca naturaleza de las palabras en que se expresan, la ambigüedad de su construcción, sus expresiones idiomáticas, que pueden interpretarse mal, y su estilo metafórico agradable por un lado, pero engañoso por otro. Sin embargo la resolución de estas dificultades no es el problema central para el lógico, porque aún ya resueltas queda todavía el problema de decidir la validez o la invalidez del argumento.

Para evitar las dificultades periféricas ligadas al lenguaje ordinario, los trabajadores de las ciencias han desarrollado **vocabularios técnicos especializados**. El científico economiza el espacio y el tiempo requeridos para la escritura de sus reportes y teorías adoptando símbolos especiales para expresar ideas que de otra manera requerirían una larga sucesión de palabras familiares para su formulación. Esto tiene la ventaja adicional de reducir la cantidad de atención requerida, puesto que cuando una oración o ecuación se alarga demasiado se hace más difícil captar su significado. La introducción del símbolo exponente en las matemáticas permite expresar la ecuación

$$A \times A \times A \times A \times A \times A \times A = B \times B \times B \times B \times B \times B$$

más breve e intelígiblemente como

$$A^{12} = B^7$$

Una ventaja semejante se ha logrado usando las fórmulas gráficas en la química orgánica; y el lenguaje de cualquier ciencia avanzada se ha visto enriquecido por innovaciones simbólicas similares.

***La lógica también ha desarrollado un sistema de notación técnica especial.***

Aristóteles hacía uso de ciertas abreviaciones para facilitar sus investigaciones, y la lógica simbólica moderna ha crecido con la introducción de otros muchos símbolos especiales. La diferencia entre la lógica nueva y la antigua es más una cuestión de grado que de naturaleza, pero la diferencia de grado es tremenda.

La lógica simbólica moderna es incomparablemente más poderosa como herramienta de análisis y deducción a través del desarrollo de un lenguaje técnico propio.

*Los símbolos especiales de la lógica moderna nos permiten exhibir con mayor claridad las estructuras lógicas de argumentos cuya formulación puede quedar oscura en el lenguaje ordinario.*

Es una tarea más fácil la de dividir los argumentos en válidos e inválidos cuando es les expresa con el lenguaje simbólico especial, pues en éste no se dan los problemas periféricos de vaguedad, ambigüedad, peculiaridades idiomáticas y metáforas. La introducción y utilización de símbolos especiales sirve no solo para facilitar la evaluación de los argumentos, sino también para aclarar la naturaleza de la inferencia deductiva.

Los símbolos especiales de la lógica se adaptan mucho mejor que el lenguaje ordinario a la obtención de las inferencias. Su superioridad en este respecto es comparable a aquella de que gozan los numerales arábigos sobre los más antiguos numerales romanos, tratándose de la computación. Es fácil multiplicar 148 por 47, pero muy difícil computar el producto de CXLVIII y XLVII. De manera semejante, la obtención de inferencias y la evaluación de los argumentos se ve grandemente facilitada con la adopción de una notación lógica especial.

### 3.2.2. Argumentos que contienen Enunciados Compuestos

#### 3.2.2.1. Enunciados Simples y Compuestos

Todos los **enunciados** pueden dividirse en dos clases: **simples** y **compuestos**.

- Un enunciado **simple** es uno que no contiene otro enunciado como parte componente.
- Todo enunciado **compuesto** contiene otro enunciado como componente.

Por ejemplo, "Las pruebas de armas nucleares en la atmósfera serán interrumpidas o este planeta se hará inhabitable" es un enunciado compuesto cuyos componentes son los dos enunciados simples "Las pruebas de armas nucleares en la atmósfera serán interrumpidas" y "este planeta será inhabitable".

Las partes componentes de un enunciado compuesto pueden a su vez ser enunciados compuestos, desde luego.

Ahora veremos algunas de las maneras diferentes de combinar los enunciados en enunciados compuestos:

**A.** El enunciado "Las rosas son rojas y las violetas son azules" es una **conjunción**, un enunciado compuesto que se forma insertando la palabra "**y**" entre los dos enunciados. Dos enunciados así combinados se llaman enunciados **conyuntos**.

Sin embargo, la palabra "**y**" tiene otros usos, como en el enunciado "Cástor y Pólux eran gemelos" que no es compuesto, sino un enunciado simple que afirma cierta relación. Introducimos el punto **,** como un símbolo especial para combinar enunciados conjuntivamente. Usándolo, la conjunción precedente se escribe "Las rosas son rojas. Las violetas son azules".

Si  $p$  y  $q$  son dos enunciados cualesquiera su conjunción se escribe  $p.q$ .

Cada enunciado es o verdadero o falso, de modo que se puede hablar del *valor de la verdad* de un enunciado, siendo el valor de verdad de un enunciado verdadero, *verdadero* y el valor de verdad de un enunciado falso, *falso*.

Hay dos amplias categorías en las que pueden dividirse los enunciados compuestos de acuerdo con que exista o no una conexión necesaria entre el valor de verdad del enunciado compuesto y los valores de verdad de sus enunciados componentes.

El valor de verdad del enunciado compuesto "Smith cree que el plomo es más pesado que el zinc" es completamente independiente del valor de verdad de su enunciado componente simple "el plomo es más pesado que el zinc", pues las personas tienen creencias correctas tanto como creencias equivocadas.

Por otro lado, hay una conexión necesaria entre el valor de verdad de una conjunción y los valores de verdad de sus enunciados conyuntos.

**Una conjunción es verdadera si sus conyuntos son ambos verdaderos, pero es falsa en cualquier otra circunstancia.**

**Cualquier enunciado compuesto cuyo valor de verdad está determinado completamente por los valores de verdad de sus enunciados componentes es un enunciado compuesto función de verdad.**

Los únicos enunciados compuestos que aquí consideraremos serán enunciados compuestos función de verdad. Por lo tanto, en el resto de este libro usaremos el término "enunciado simple" para referirnos a cualquier enunciado que no sea compuesto función de verdad.

Como las conjunciones son enunciados compuestos función de verdad nuestro símbolo es un conectivo de función de verdad (o veritativo funcional, como también se dice).

Dados dos enunciados  $p$  y  $q$  hay solamente cuatro conjuntos de valores de verdad para ellos, y en cada caso el valor de verdad de su conjunción  $p.q$  está determinado de manera única. Los cuatro casos posibles pueden exhibirse como a continuación:

- En el caso  $p$  es verdadero y  $q$  es verdadero,  $p \cdot q$  es verdadero;
- En el caso  $p$  es verdadero y  $q$  es falso,  $p \cdot q$  es falso;
- En el caso  $p$  es falso y  $q$  es verdadero,  $p \cdot q$  es falso;
- En el caso  $p$  es falso y  $q$  es falso,  $p \cdot q$  es falso.

Al representar los valores de verdad verdadero y falso con las letras "T" y "F", respectivamente, la manera en que el valor de verdad de una conjunción queda determinado por los valores de verdad de sus conyuntos se muestra de manera más concisa por medio de una *tabla de verdad*, como sigue:

$p$	$q$	$p \cdot q$
T	T	T
T	F	F
F	T	F
F	F	F

Ya que especifica el valor de verdad de  $p \cdot q$  en cada caso posible, esta tabla de verdad se puede tomar como *definición* del símbolo *punto*. Otras palabras tales como "aunque", "sin embargo", etc., y hasta la coma y el punto y coma, se utilizan también para conjuntar dos enunciados en un compuesto y todos ellos pueden traducirse indiferentemente como el símbolo punto en lo que respecta a los valores de verdad.

**B.** El enunciado "No es el caso que el plomo sea más pesado que el oro" también es compuesto siendo la **negación** (o el *contradictorio*) de su enunciado compuesto único "el plomo es más pesado que el oro". Introducimos el símbolo " $\sim$ ", llamado una *tilde*, para simbolizar la negación. Hay frecuentemente otras formulaciones en lenguaje ordinario, de una negación. Así, si  $L$  simboliza el enunciado "el plomo es más pesado que el oro", los enunciados diferentes "no es el caso que el plomo sea más pesado que el oro", "es falso que el plomo sea más pesado que el oro", "el plomo no es más pesado que el oro", "no es verdad que el plomo sea más pesado que el oro", "el plomo no es más pesado que el oro", se simbolizan todos indiferentemente como  $\sim L$ .

Más generalmente, si  $p$  es cualquier enunciado su negación se escribe  $\sim p$ .

Como la negación de un enunciado verdadero es un enunciado falso y la negación de un enunciado falso es uno verdadero, podemos tomar la siguiente tabla de verdad como definición del símbolo tilde:

$p$	$\sim p$
T	F
F	T

**C.** Cuando dos enunciados se combinan disyuntivamente insertando la palabra "o" entre ellos, el enunciado compuesto que resulta es una **disyunción** (o *alternación*) y los dos enunciados así combinados se llaman **disyuntos** (o *alternativos*).

La palabra "o" tiene dos sentidos diferentes, uno de los cuales es la clara intención en el enunciado "se perderá derechos a recompensas en caso de enfermedad o desempleos". Aquí la intención es obviamente cancelar el derecho a premios no sólo para las personas enfermas y las personas desempleadas sino también para las personas que están enfermas y desempleadas.

Este sentido de la palabra "o" se denomina **débil o inclusivo**.

En donde la precisión sea esencial, como los contratos y otros documentos legales, este sentido se hace explícito usando la frase "y/o".

Es otro el sentido de "o" que se intenta dar en el menú de un restaurante escribiendo "té o café", queriendo decir que por el precio estipulado el cliente puede tomar café o té pero no ambos.

**Este segundo sentido de "o" es llamado fuerte o exclusivo.**

En donde la precisión es esencial y se quiere dar el sentido exclusivo a la palabra "o" suele agregarse la frase "pero no ambos".

**Una disyunción que usa el "o" exclusivo afirma que por lo menos uno de los disyuntivos es verdadero, pero no ambos son verdaderos.**

El significado común parcial que al menos un disyunto es verdadero, es el significado todo de una disyunción inclusiva y parte del significado de una disyunción exclusiva.

En latín la palabra "vel" expresa el **sentido inclusivo** de la palabra "o" y la palabra "aut" expresa el **sentido exclusivo**.

Es costumbre usar la primera letra de "vel" para simbolizar "o" en su sentido inclusivo. Si p y q son dos enunciados cualesquiera, su **disyunción débil o inclusiva** se escribe  $p \vee q$ . El símbolo de " $\vee$ ", denominado una cuña (o una ve), es un conectivo de función de verdad y se define por la tabla de verdad siguiente:

p	q	$p \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

Un argumento que obviamente es válido y contiene una disyunción es el siguiente Silogismo Disyuntivo:

Las Naciones Unidas serán reforzadas o habrá una tercera guerra mundial.

Las Naciones Unidas no serán reforzadas.

Luego habrá una tercera guerra mundial.

Es evidente que un Silogismo Disyuntivo es válido en cualquiera de las interpretaciones de la palabra "o", esto es, sin atención a que su primera premisa afirma una disyunción inclusiva o exclusiva. Es usualmente difícil, y a veces imposible, descubrir cuál es el sentido de la palabra "o" que se intenta dar en una disyunción. Pero el argumento válido típico que tiene una disyunción como premisa es, como el Silogismo Disyuntivo, válido en cualquier interpretación de la palabra "o".

**Por lo tanto, efectuamos una simplificación al traducir cualquier ocurrencia de la palabra "o" en el símbolo lógico " $\vee$ " – sin atención al sentido que se quiera dar a "o" –.**

Desde luego, en donde se establezca explícitamente que la disyunción es exclusiva, usando la frase adicional "pero no ambos", por ejemplo tenemos el aparato simbólico para simbolizar este sentido, como se explica más adelante.

El uso de los paréntesis, corchetes y llaves para la puntuación de las expresiones matemáticas es familiar. La expresión "6 + 9 / 3", no determina un número único, aunque si la puntuación aclara cómo agrupar los números que la constituyen, denota 5 o 9. La puntuación es necesaria también para resolver la ambigüedad en el lenguaje de la lógica simbólica, porque los enunciados compuestos son susceptibles de combinaciones para formar enunciados más complicados.

Hay ambigüedad en  $p \cdot q \vee r$ , que podrían ser o la conjunción de p con q  $\vee$  r, o por otro lado la disyunción de p . q con r. Estos dos sentidos diferentes los dan sin ambigüedad las puntuaciones diferentes:  $p.(q \vee r)$  y  $(p \cdot q) \vee r$ . En el caso en que p y q sean falsos ambos y r verdadero, la primera expresión puntuada es falsa (pues su primer enunciado conjunto es falso), pero la segunda expresión puntuada es verdadera (pues su segundo enunciado disyunto es verdadero). Aquí, la diferencia de puntuación hace toda la diferencia entre verdad y falsedad. En la lógica simbólica, como en las matemáticas, usamos paréntesis, corchetes y llaves para la puntuación. Sin embargo, para reducir el número de signos de puntuación requerido estableceremos el convenio simbólico de que en cualquier expresión la tilde se aplicará a la componente más pequeña permitida por la puntuación. De este modo, la ambigüedad de  $\sim p \vee q$ , que podría significar o  $(\sim p) \vee q$  o  $\sim (p \vee q)$ , queda resuelta por nuestro convenio para significar la primera de estas, pues la tilde puede (y en consecuencia por nuestro convenio lo hace) aplicarse a la primera componente p y no a la expresión más larga p  $\vee$  q.

No todas las **conjunciones** se formulan explícitamente colocando la palabra "y" entre oraciones completas, como en "*Carlitos es limpio y Carlitos es encantador*". De hecho, ésta se expresaría más naturalmente como "*Carlitos es limpio y encantador*", y "*Juan y Carolina subieron a la colina*" es la manera más natural de expresar la conjunción "*Juan subió a la colina y Carolina subió a la colina*".

Lo mismo con las **disyunciones**: "*o Alicia o Beatriz serán elegidas*" expresa más brevemente la proposición que alternativamente se formula como "*Alicia será elegida o Beatriz será elegida*"; y "*Carlota será secretaria o tesorera*" expresa de manera un tanto más breve la misma proposición que "*o Carlota será secretaria o Carlota será tesorera*".

La **negación** de una disyunción se expresa a menudo usando la frase "*ni-ni*". Así, la disyunción "*Alicia o Beatriz serán elegidas*" queda negada por el enunciado "*ni Alicia ni Beatriz serán elegidas*". La disyunción se simbolizaría como A  $\vee$  B, y su negación como  $\sim(A \vee B)$  o como  $(\sim A) \cdot (\sim B)$ , que son fórmulas equivalentes. Negar que al menos uno de los enunciados es verdadero es asegurar que ambos enunciados son falsos.

La palabra "*ambos*" tiene varias funciones. Una de ellas es solo cuestión de énfasis. Decir "*Ambos Juan y Carolina subieron a la colina*" es solo para recalcar que los dos hicieron lo que se dice que hicieron al decir "*Juan y Carolina subieron a la colina*". Una función más útil de la palabra "*ambos*" es de puntuación. "*Ambos ... y ... no son ...*" se usa para expresar lo mismo que "*Ni ... ni ... es ...*". En oraciones tales el orden que guardan las palabras "*ambos*" y "*no*" es de mucha significación. Hay una gran diferencia entre:

Alicia y Beatriz no serán ambas elegidas

y

Alicia y Beatriz ambas no serán elegidas.

La primera se simboliza como  $\sim(A \cdot B)$ , la última como  $\sim(A) \cdot \sim(B)$ .

Finalmente, hay que observar que la frase "*a menos que*" puede también usarse en la expresión de la disyunción de dos enunciados.

Así, "*Nuestros recursos pronto se agotarán, a menos que se procesen más materiales de desecho*" puede expresarse también como "*O se procesan más materiales de desecho o se agotarán pronto nuestros recursos*" y se simboliza como M  $\vee$  E.

Como una **disyunción exclusiva** asegura que al menos uno de los disyuntos es verdadero pero no ambos, podemos simbolizar la disyunción exclusiva de dos enunciados p y q cualesquiera simplemente como  $(p \vee q) \cdot \sim(p \cdot q)$ .

Así, podemos simbolizar las conjunciones, las negaciones y las disyunciones inclusivas y exclusivas. Todo enunciado compuesto construido a partir de enunciados simples por aplicación repetida de conectivos de función de verdad, tendrá valores de verdad completamente determinados por los valores de verdad de esos enunciados simples.

Por ejemplo, si A y B son enunciados verdaderos y X y Y son falsos, el valor de verdad del enunciado compuesto  $\sim[(\sim A \vee X) \vee \sim(B \cdot Y)]$  puede encontrarse de la manera siguiente. Como A es verdadero,  $\sim A$  es falso, y como X es falso, también la disyunción  $(\sim A \vee X)$  es falsa. Dado que Y es falso, la conjunción  $(B \cdot Y)$  es falsa y su negación  $\sim(B \cdot Y)$  es verdadera. De este modo,

la disyunción ( $\sim A \vee X$ )  $\vee \sim(B \cdot Y)$  es verdadera, y su negación que es el enunciado original, es falsa. Este procedimiento paso a paso, iniciado en las componentes (más) internas nos permite, siempre, determinar el valor de verdad de un enunciado compuesto función de verdad partiendo de los valores de verdad de sus enunciados simples componentes.

### 3.2.2.2. Enunciados Condicionales

El enunciado compuesto "Si el tren se retrasa entonces perderemos nuestro transbordo" es un **condicional** (o un hipotético, una implicación o un enunciado implicativo).

El enunciado componente situado entre el "si" y el "entonces" es llamado el **antecedente** (o el implicante o prótasis), y el componente que sigue al "entonces" es el **consecuente** (o el implicado o apódosis).

Un **condicional** no afirma que su antecedente sea verdadero o que su consecuente lo sea; solo afirman que **si su antecedente es verdadero, entonces su consecuente es también verdadero**, o sea, que su antecedente implica su consecuente. La clave del significado de un condicional es la relación de implicación que se asegura que existe entre su antecedente y su consecuente, en ese orden.

Si examinamos un cierto número de condicionales diferentes veremos que pueden afirmar diferentes implicaciones.

- En el condicional "Si a todos los gatos les gusta el hígado y Dina es un gato, entonces a DINA le gusta el hígado", el consecuente se sigue **lógicamente** del antecedente.
- Por otro lado, en el condicional "Si la figura es un triángulo, entonces tiene tres lados", el consecuente se sigue del antecedente por la **definición** misma de "triángulo".
- Pero la verdad del condicional "Si el oro se sumerge en agua regia, entonces el oro se disuelve" no es cuestión de lógica ni de definición. Aquí la conexión afirmada es causal y debe descubrirse **empíricamente**.

Este ejemplo muestra que hay diferentes clases de implicaciones que constituyen diferentes tipos de sentidos de la frase "si-entonces". Observadas estas diferencias, ahora buscamos un significado común identificable, algún **significado parcial común** a éstos que, como hemos aceptado, son diferentes tipos de condicionales.

Nuestra discusión de "si-entonces" correrá paralela a nuestra previa discusión de la palabra "o".

Primero, señalamos dos sentidos diferentes de esa palabra.

Segundo, notamos que había un **significado parcial común**: el hecho de que al menos un disyunto sea verdadero, se vio que estaba involucrado tanto en el "o" inclusivo como en el exclusivo.

Tercero, introdujimos el símbolo especial "v" para representar este sentido parcial común (que era todo el significado de "o" en su sentido inclusivo).

Cuarto, observamos que, dado que argumentos como el Silogismo Disyuntivo son válidos en cualquier interpretación de la palabra "o", simbolizar cualquier ocurrencia de la palabra "o" por el símbolo cuña preserva la validez de tales argumentos. Y como nos interesan los argumentos desde el punto de vista de la determinación de su validez, esta traducción de la palabra "o" en "v" que puede abstraer o ignorar parte de su significado en algunos casos, es enteramente adecuada para nuestros propósitos actuales.

Un **significado parcial común** de estas diferentes clases de enunciados condicionales surge cuando preguntamos cuáles serían circunstancias suficientes para establecer la falsedad de un condicional. ¿En qué circunstancias acordaríamos que el condicional "Si el oro se sumerge en agua regia entonces el oro se disuelve" es falso?. Claramente, el enunciado es falso en el caso de que se sumerja el oro en esta solución y no se disuelva.

**Cualquier condicional de antecedente verdadero y consecuente falso debe ser falso.**

Luego, cualquier condicional **si  $p$  entonces  $q$**  se sabe que es **falso** en el caso de que la conjunción  $p \cdot \neg q$  sea conocida **verdadera**, esto es, en caso de que el antecedente sea verdadero y su consecuente falso.

**Para que el condicional sea verdadero la condición indicada deberá ser falsa.**

En otras palabras, **para que cualquier condicional si  $p$  entonces  $q$  sea verdadero,  $\neg(p \cdot \neg q)$** , la negación de la conjunción de su antecedente con la negación de su consecuente, **también debe ser verdadera**. Luego, podemos considerar esta última como parte del significado del condicional.

Introducimos un nuevo símbolo " $\supset$ ", llamado herradura, para representar el **significado parcial común en todos los enunciados condicionales**, definiendo " $p \supset q$ " como una abreviación de  $\neg(p \cdot \neg q)$ . La herradura es un conectivo de función de verdad, cuya significación exacta queda indicada por la tabla de verdad siguiente:

$p$	$q$	$\neg q$	$p \cdot \neg q$	$\neg(p \cdot \neg q)$	$p \supset q$
T	T	F	F	T	T
T	F	T	T	F	F
F	T	F	F	T	T
F	F	T	F	T	T

En ésta, la primera y segunda columnas representan todos los valores de verdad posibles para los enunciados componentes  $p$  y  $q$ , y las columnas tercera, cuarta y quinta representan etapas sucesivas al determinar el valor de verdad del enunciado compuesto  $\neg(p \cdot \neg q)$  en cada caso. La sexta columna es idénticamente la misma que la quinta, puesto que las fórmulas que las encabezan por definición expresan la misma proposición. El símbolo de herradura no debe pensarse que representa el significado del "si-entonces", o la relación de implicación, sino más bien un **factor parcial común de las diferentes clases de implicaciones** significadas por la frase "si-entonces".

Podemos considerar esta **herradura** como **símbolo de una clase especial, extremadamente débil, de implicación**, y nos resulta conveniente hacerlo así, pues algunas maneras de leer " $p \supset q$ " son "si  $p$  entonces  $q$ ", " $p$  implica  $q$ " o " $p$  solo si  $q$ ". La implicación débil simbolizada " $\supset$ " se llama **implicación material**, y su nombre especial indica que es una noción especial, que no debe confundirse con las otras clases de implicación más usuales.

Algunos enunciados condicionales en el lenguaje ordinario afirman meramente implicaciones materiales como, por ejemplo, "Si Rusia es una democracia entonces yo soy Napoleón". Es claro que la implicación afirmada aquí no es lógica, ni definitoria, ni causal. No se pretende ninguna "conexión real" entre lo que afirma el antecedente y lo que se afirma en el consecuente. Esta clase de condicional se usa ordinariamente como un método enfático o humorístico de negar la verdad de su antecedente, pues típicamente contiene un enunciado notorio o ridículamente falso como consecuente. Cualquier afirmación tal respecto a los valores de verdad se simboliza adecuadamente usando el conectivo función de verdad " $\supset$ ".

Aunque la mayor parte de los enunciados condicionales afirman más que una implicación meramente material entre el antecedente y el consecuente, ahora proponemos simbolizar cualquier ocurrencia de "**si-entonces**" mediante el conectivo de función de verdad " $\supset$ ". Debe admitirse que esta simbolización abstracta e ignora parte del significado de casi todos los enunciados condicionales. Pero la proposición puede justificarse sobre la base de que la validez de los argumentos válidos que involucran condicionales se preserva cuando los condicionales se consideran como implicaciones materiales solamente, como se establecerá en las siguientes secciones.

Los enunciados condicionales pueden expresarse en toda una variedad de formas. Un enunciado de la forma "si  $p$  entonces  $q$ " podría igualmente bien expresarse como "si  $p$ ,  $q$ ", " $q$  si  $p$ ", "que  $p$  implica que  $q$ ", "que  $p$  trae consigo que  $q$ ", " $p$  solo si  $q$ ", "que  $p$  es una condición

suficiente que  $q$ ", o, como "que  $q$  es una condición necesaria que  $p$ ", y cualquiera de estas formulaciones se simbolizará mediante  $p \supset q$ .

### 3.2.2.3. Formas de Argumentos y Tablas de Verdad

En esta sección desarrollamos un **método puramente mecánico** para probar la **validez de argumentos que contienen enunciados compuestos de función de verdad**. Ese método está íntimamente relacionado con la técnica familiar de *refutación por analogía lógica* que se usó en el primer capítulo para demostrar la invalidez del argumento

Si yo soy presidente entonces soy famoso.  
Yo no soy presidente.  
Luego yo no soy famoso.

Este argumento se mostró que era inválido construyendo otro argumento de la misma forma:

Si Rockefeller es presidente entonces él es famoso.  
Rockefeller no es presidente.  
Luego Rockefeller no es famoso.

que obviamente es inválido, pues sus premisas son verdaderas, pero su conclusión falsa.

Cualquier argumento se prueba que es inválido si es posible construir otro argumento de exactamente la misma forma con premisas verdaderas y una conclusión falsa. Esto refleja el hecho de que *la validez y la invalidez son características puramente formales de los argumentos*:

*dos argumentos cualesquiera que tienen la misma forma o son válidos ambos o ambos son inválidos, independientemente de las diferencias de su contenido.*

La noción de dos argumentos que tienen exactamente la misma forma es una noción que merece mayor examen.

Es conveniente, al discutir las *formas de los argumentos*, usar letras minúsculas de la parte media del alfabeto, "p", "q", "r", "s", ... como **variables sentenciales**, que se definen simplemente como *letras por las cuales, o en lugar de las cuales, se pueden sustituir enunciados*.

Ahora definimos una **forma argumental** como cualquier **arreglo de símbolos que contiene variables sentenciales**, de modo que al sustituir enunciados por las variables sentenciales – siendo siempre el mismo enunciado el que reemplaza a la misma variable – el resultado es un argumento.

Por precisión, establecemos el convenio de que en cualquier forma argumental, "p" será la primera variable sentencial que ocurre en el mismo, "q" será la segunda, "r" la tercera y así sucesivamente.

*Cualquier argumento que sea resultado de la sustitución de enunciados en lugar de variables sentenciales de una forma argumental, se dice que tiene esa forma o que es una instancia de sustitución de esa forma argumental.*

Si simbolizamos el enunciado simple "Las Naciones Unidas serán reforzadas" con U, y el enunciado simple "Habrá una tercera guerra mundial" con W, entonces el Silogismo Disyuntivo antes presentado puede simbolizarse como

$$(1) \quad \begin{array}{l} U \vee W \\ \sim U \\ \therefore W \end{array}$$

Tiene la forma

$$(2) \quad \begin{array}{l} p \vee q \\ \sim p \\ \therefore q \end{array}$$

de la cual resulta reemplazando las variables sentenciales  $p$  y  $q$  por los enunciados  $U$  y  $W$ , respectivamente. Pero esa no es la única forma de la cual es una instancia de sustitución. El mismo argumento se obtiene reemplazando las variables sentenciales  $p$  y  $q$  y  $r$  en la forma argumental

$$(3) \quad \begin{array}{c} p \\ q \\ \therefore r \end{array}$$

por los enunciados  $U \vee W$ ,  $\sim U$  y  $W$ , respectivamente.

*Definimos la forma específica de un argumento dado, como aquella forma argumental de la cual resulta el argumento reemplazando cada variable sentencial por un enunciado simple diferente.*

Así, la forma específica del argumento (1) es la forma argumental (2). Aunque la forma argumental (3) es una forma del argumento (1), no es la forma específica del mismo.

*La técnica de refutación por analogía lógica puede ahora describirse más precisamente. Si la forma específica de un argumento dado puede mostrarse que tiene una instancia de sustitución con premisas verdaderas y conclusión falsa, entonces el argumento dado es inválido.*

Los términos "válido" e "inválido" pueden extenderse para aplicarse a formas argumentales tanto como a argumentos. Una forma argumental inválida es una que tiene cuando menos una instancia de sustitución con premisas verdaderas y una conclusión falsa.

La técnica de refutación por analogía lógica presupone que todo argumento del cual la forma específica es una forma argumental inválida es un argumento inválido. Toda forma argumental que no sea inválida es válida; una forma argumental válida es una que no tiene instancia de sustitución con premisas verdaderas y conclusión falsa. Cualquier argumento dado puede probarse que es válido si se puede mostrar que la forma específica del argumento dado es una forma argumental válida.

**Para determinar la validez o invalidez de una forma argumental debemos examinar todas las instancias de sustitución posibles de ella para ver si algunas tienen premisas verdaderas y conclusiones falsas.**

Los argumentos de los que aquí nos ocupamos solamente contienen enunciados simples y enunciados función de verdad compuestos con aquellos, y solo nos interesan los valores de verdad, de sus premisas y conclusiones. Podemos obtener todas las instancias de sustitución posibles cuyas premisas y conclusiones tienen diferentes variables sentenciales en la forma argumental que se prueba. Estas pueden disponerse de la manera más conveniente en una tabla de verdad, con una columna inicial o guía para cada variable sentencia que aparece en la forma argumental. Así, para probar la validez de la forma del Silogismo Disyuntivo

$$\begin{array}{l} p \vee q \\ \sim p \\ \therefore q \end{array}$$

construimos la siguiente tabla de verdad:

<b>p</b>	<b>q</b>	<b>p v q</b>	<b>~p</b>
T	T	T	F
T	F	T	F
F	T	T	T
F	F	F	T

Cada renglón de esta tabla representa una clase completa de instancias de sustitución. Las T y las F en las dos columnas iniciales representan los valores de verdad de enunciados que pueden sustituirse por las variables p y q en la forma argumental. Estos valores determinan los valores de verdad en las otras columnas, la tercera de las cuales está encabezada por la *primera "premisa"* de la forma argumental y la cuarta por la *segunda "premisa"*. El encabezado de la segunda columna es la *conclusión* de la forma argumental. Un examen de esta tabla de verdad revela que cualesquiera que sean los enunciados sustituidos por las variables p y q, el argumento resultante no puede tener premisas verdaderas y una conclusión falsa, pues el tercer renglón representa el único caso posible en que ambas premisas son verdaderas y ahí la conclusión también es verdadera.

Como las **tablas de verdad** proporcionan un **método puramente mecánico o efectivo de decisión de la validez o invalidez de cualquier argumento** del tipo general aquí considerado, ahora podemos justificar nuestra propuesta de simbolizar todos los enunciados condicionales por medio del conectivo de función de verdad " $\rightarrow$ ". La justificación para tratar todas las implicaciones como si fueran meramente implicaciones materiales es que los argumentos válidos que contienen enunciados condicionales siguen siendo válidos cuando estos condicionales se interpretan como afirmando implicaciones materiales solamente.

Las tres más simples y más intuitivamente válidas formas de argumento que involucran enunciados condicionales son

**Modus Ponens**      Si p entonces q  
                           P  
                           ∴ q

**Modus Tollens**      Si p entonces q  
                           ~q  
                           ∴ ~p

y el **Silogismo Hipotético** Si p entonces q  
                           Si q entonces r  
                           ∴ Si p entonces r

El que sigan siendo válidos cuando sus condicionales se interpretan como aseveraciones de implicaciones materiales, es un hecho que fácilmente se establece por tablas de verdad. La validez de **Modus Ponens** se muestra con la misma tabla de verdad que define el símbolo herradura:

p	q	p → q
T	T	T
T	F	F
F	T	T
F	F	T

Aquí las dos premisas se representan por las columnas tercera y primera y la conclusión por la segunda. Solo el primer renglón representa instancias de sustitución en las que ambas premisas son verdaderas, y en ese renglón la conclusión también es verdadera.

La validez de **Modus Tollens** se muestra por medio de la siguiente tabla:

<b>p</b>	<b>q</b>	<b><math>p \rightarrow q</math></b>	<b><math>\neg q</math></b>	<b><math>\neg p</math></b>
T	T	T	F	F
T	F	F	T	F
F	T	T	F	T
F	F	T	T	T

Aquí solamente el cuarto renglón representa instancias de sustitución en las que ambas premisas (las columnas tercera y cuarta) son verdaderas, y ahí la conclusión (quinta columna) también es verdadera.

Como el **Silogismo Hipotético** contiene tres enunciados distintos para variables sentenciales distintas, su tabla de verdad debe tener tres columnas iniciales y requerirá ocho renglones para alistar todas las posibles instancias de sustitución:

<b>p</b>	<b>q</b>	<b>r</b>	<b><math>p \rightarrow q</math></b>	<b><math>q \rightarrow r</math></b>	<b><math>p \rightarrow r</math></b>
T	T	T	T	T	T
T	T	F	T	F	F
T	F	T	F	T	T
T	F	F	F	T	F
F	T	T	T	T	T
F	T	F	T	F	T
F	F	T	T	T	T
F	F	F	T	T	T

Al construirla, las tres columnas iniciales representan todos los arreglos posibles de valores de verdad para los enunciados sustituidos en lugar de las variables sentenciales **p**, **q** y **r**, la cuarta columna se llena con referencia a la primera y la segunda, la quinta con referencia a la segunda y la tercera, y la sexta con referencia a la primera y la tercera. Las premisas son ambas verdaderas solo en los renglones primero, quinto, séptimo y octavo, y en estos renglones la conclusión también es verdadera. Esto basta para mostrar que el Silogismo Hipotético es válido cuando sus condicionales se simbolizan mediante el símbolo herradura. Todas las dudas que queden respecto a la afirmación de que los argumentos válidos que contienen condicionales siguen siendo válidos cuando sus condicionales se interpreten como afirmando meramente implicaciones materiales puede aclararlas el lector al construir, simbolizar y probar sus propios ejemplos mediante tablas de verdad.

Para probar la validez de una forma argumental mediante una tabla de verdad, es necesaria una tabla con una columna inicial o guía separada para cada variable sentencial diferente y un renglón separado para cada posible asignación de valores de verdad a las variables sentenciales involucradas. Así pues, probar una forma argumental que contiene **n** variables sentenciales distintas requiere una tabla de verdad con  $2^n$  renglones. Al construir tablas de verdad es conveniente fijar un patrón uniforme de inscripción de las **T** y las **F** en sus columnas iniciales o guía. En este libro nos apegaremos a la práctica de ir simplemente alternando las **T** y las **F** hacia abajo en la columna inicial extrema derecha, alternando pares de **T** con pares de **F** hacia abajo en la columna directamente a su izquierda, después alternando grupos de cuatro **T** con grupos de cuatro **F**, ..., y al llegar a la columna extrema izquierda ponemos **T** en todas su mitad superior y **F** en toda su mitad inferior.

Hay dos formas de argumento inválidas que tienen un parecido superficial con las formas válidas Modus Ponens y Modus Tollens.

Estas son:

$$\begin{array}{c} p \supset q \\ q \\ \therefore p \end{array} \quad \text{y} \quad \begin{array}{c} p \supset q \\ \sim p \\ \therefore \sim q \end{array}$$

y se conocen con el nombre de **Falacias de Afirmación del Consecuente y Negación del Antecedente**, respectivamente.

La invalidez de ambas puede mostrarse en una misma tabla de verdad:

p	q	$p \supset q$	$\sim p$	$\sim q$
T	T	T	F	F
T	F	F	F	T
F	T	T	T	F
F	F	T	T	T

Las dos premisas en la Falacia de Afirmación del Consecuente encabezan las columnas segunda y tercera, y son verdaderas en el primer y en el tercer renglón. Pero la conclusión, que encabeza la primera columna, es falsa en el tercer renglón, lo que muestra que la forma de argumentar tiene una instancia de sustitución con premisas verdaderas y conclusión falsa y, por lo tanto, es inválida. Las columnas tres y cuatro son las encabezadas por las premisas en la Falacia de Negación del Antecedente, que son verdaderas en los renglones tercero y cuarto. Su conclusión encabeza la columna cinco y es falsa en el tercer renglón, lo que muestra que la segunda forma argumental también es inválida.

Hay que recalcar que aunque una forma de argumento válido tiene solo argumentos válidos como instancias de sustitución, una forma de argumento inválido puede tener instancias de sustitución válidas tanto como inválidas. Así que para demostrar que un argumento dado es inválido hay que demostrar que la forma específica de ese argumento es inválida.

### 3.2.2.4. Formas Sentenciales

La introducción de variables sentenciales en la sección precedente nos permitió definir las formas argumentales en general y la forma específica de un argumento dado.

Ahora definimos una **forma sentencial** como **cualquier sucesión de símbolos conteniendo variables sentenciales**, de modo que al sustituir enunciados por las variables sentenciales – reemplazando la misma variable sentencial por el mismo enunciado en toda la secuencia – el resultado es un enunciado.

Otra vez, para precisar, convendremos en que en cualquier forma sentencial “p” será la primera variable sentencial que aparece, “q” será la segunda en ocurrir, “r” la tercera y así sucesivamente. Todo enunciado que resulta de la sustitución de las variables sentenciales por enunciados en una forma sentencial, se dirá que tiene esa forma o que es una instancia de sustitución de ella.

Así como distinguimos la forma específica de un argumento dado, así también distinguiremos la forma específica de un enunciado dado como la forma sentencial de la que resulta el enunciado poniendo en el lugar de cada variable sentencial un enunciado simple diferente.

Así, por ejemplo, si A, B y C son enunciados simples diferentes, el enunciado compuesto  $A \supset (B \vee C)$  es una instancia de sustitución de la **forma sentencial**  $p \supset q$  y también de la **forma sentencial**  $p \supset (q \vee r)$ , pero solo esta última es la **forma específica del enunciado** dado.

Aunque los enunciados “Balboa descubrió el Océano Pacífico” (B) y “Balboa descubrió el Océano Pacífico o no lo descubrió” ( $B \vee \sim B$ ) ambos son verdaderos, descubrimos su verdad de

maneras enteramente diferentes. La verdad de B es cuestión histórica y se aprende por medio de la investigación empírica. Aún más, podrían haber ocurrido cosas que hicieran a B falso; nada necesario hay respecto a la verdad de B. Pero la verdad del enunciado  $B \vee \sim B$  puede saberse independientemente de toda investigación empírica y ningún suceso puede hacerlo falso porque es una verdad necesaria.

El enunciado  $B \vee \sim B$  es una **verdad formal**, una instancia de sustitución de una forma sentencial cuyas instancias de sustitución todas son verdaderas.

**Una forma sentencial que solo tiene instancias de sustitución verdaderas se dice tautológica, o que es una tautología.**

La forma específica de  $B \vee \sim B$  es  $p \vee \sim p$  y se prueba que es tautológica mediante la siguiente tabla de verdad:

P	$\sim p$	$p \vee \sim p$
T	F	T
F	T	T

En la columna que encabeza la forma sentencial de que se trata solo hay valores **T**, luego **todas sus instancias de sustitución son verdaderas**. Cualquier enunciado que es una instancia de sustitución de una forma sentencial tautológica es formalmente verdadero y también se le llama tautológico, o una tautología.

Similarmente, aunque los enunciados "Cortés descubrió el Pacífico" (C) y "Cortés descubrió el Pacífico y Cortés no descubrió el Pacífico" ( $C \cdot \sim C$ ) ambos son falsos, descubrimos su falsedad de dos maneras enteramente diferentes.

El primero simplemente ocurre que es falso y eso se descubre empíricamente; mientras que el segundo es necesariamente falso y eso puede saberse independientemente de toda investigación empírica. El enunciado  **$C \cdot \sim C$  es formalmente falso**, es una instancia de sustitución de una forma sentencial cuyas instancias de sustitución todas son falsas.

**Un enunciado, se dice que contradice, o que es una contradicción de otro enunciado, cuando es lógicamente imposible que ambos sean verdaderos.**

En este sentido la contradicción es una relación entre enunciados. Pero hay otro sentido, relacionado del mismo término.

Cuando es lógicamente imposible que un enunciado particular sea verdadero, ese enunciado mismo es llamado autocontradicitorio o una autocontradicción. Más simplemente, se dice que tales enunciados son contradictorios o contradicciones, y ésta es la terminología que aquí usaremos.

**Una forma sentencial que solo tiene instancias de sustitución falsas se dice que es una contradicción o que es contradictoria, y los mismos términos se aplican a sus instancias de sustitución.**

La **forma sentencial  $p \cdot \sim p$**  se prueba que es una **contradicción** por el hecho de que en su tabla de verdad solo hay valores **F** en la columna que encabeza.

**Enunciados y formas sentenciales que no son ni tautológicas ni contradictorias se dice que son contingentes o que son contingencias.**

Por ejemplo,  $p$ ,  $\sim p$ ,  $p \vee q$ ,  $p \cdot q$ ,  $p \supset q$  son formas sentenciales contingentes; y  $B$ ,  $C$ ,  $\sim B$ ,  $B \cdot C$ ,  $B \vee C$ , son enunciados contingentes. El término es apropiado, pues sus valores de verdad no están formalmente determinados, sino que *son dependientes o contingentes de la situación*.

Fácilmente se demuestra que  $p \supset (q \supset p)$  y  $\sim p \supset (p \supset q)$  son tautologías. Al expresarlas en lenguaje ordinario como "Un enunciado verdadero es implicado por cualquier enunciado" y como "Un enunciado falso implica cualquier enunciado" parecen bastante extrañas. Algunos escritores las han llamado las paradojas de la implicación material. Pero si se tiene presente que el símbolo herradura es un conectivo función de verdad que representa la implicación

material y no la "implicación en general" o clases más usuales como son la implicación lógica o la implicación causal, entonces dichas formas sentenciales tautológicas no son sorprendentes en lo absoluto. Y al corregir esas engañosas formulaciones del castellano insertando la palabra "materialmente" antes de "implicado" e "implica", entonces desaparece el aire paradójico. La implicación material es una noción técnica y la motivación del lógico al introducirla y usarla es la enorme simplificación que resulta en su tarea de discriminar entre los argumentos válidos y los inválidos.

**Dos enunciados se dicen materialmente equivalentes cuando tienen el mismo valor de verdad, y simbolizamos el enunciado de que son materialmente equivalentes insertando el símbolo " $\equiv$ " entre ellos.**

Tratándose de un conectivo función de verdad, el símbolo tres barras se define con la siguiente tabla de verdad:

p	q	$p \equiv q$
T	T	T
T	F	F
F	T	F
F	F	T

Decir que dos enunciados son materialmente equivalentes es decir que materialmente el uno implica el otro, como es fácil de verificar con una tabla de verdad. Así, el símbolo de las tres barras puede leerse "es materialmente equivalente con" o "si y solo si".

Un enunciado de la forma  $p \equiv q$  se llama bicondicional. Dos enunciados se dicen lógicamente equivalentes cuando el bicondicional que expresa su equivalencia material es una tautología. El "principio de la Doble Negación", expresado como  $p \equiv \sim p$ , con una tabla de verdad se demuestra que es tautológico.

Hay dos equivalencias lógicas que expresan importantes interrelaciones de las conjunciones, disyunciones y negaciones. Como una conjunción afirma que sus dos conyuntos son verdaderos, su negación solo necesita afirmar que uno de los dos conyuntos es falso. Luego, negar la conjunción  $p \cdot q$  equivale a afirmar la disyunción de las negaciones de  $p$  y  $q$ . Este enunciado de equivalencia se simboliza como  $\sim(p \cdot q) \equiv (\sim p \vee \sim q)$  y se demuestra que es una **tautología** mediante la tabla de verdad:

p	q	$p \cdot q$	$\sim(p \cdot q)$	$\sim p$	$\sim q$	$\sim p \vee \sim q$	$\sim(p \cdot q) \equiv (\sim p \vee \sim q)$
T	T	T	F	F	F	F	T
T	F	F	T	F	T	T	T
F	T	F	T	T	F	T	T
F	F	F	T	T	T	T	T

De manera semejante, como una disyunción meramente afirma que al menos un disyunto es verdadero, negarla es afirmar que ambos son falsos. Negar la disyunción  $p \vee q$  equivale a afirmar la conjunción de las negaciones de  $p$  y  $q$ . Se simboliza como  $\sim(p \vee q) \equiv (\sim p \cdot \sim q)$ , y con una tabla de verdad fácilmente se prueba que es una **tautología**.

Estas dos equivalencias se conocen como **Teoremas de De Morgan**, por el lógico matemático inglés Augustus De Morgan (1806-1871), y en lenguaje ordinario pueden enunciarse de manera

comprendida como: La negación de la  $\left\{ \begin{array}{l} \text{conjunción} \\ \text{disyunción} \end{array} \right\}$  de dos enunciados

es lógicamente equivalente a la  $\left\{ \begin{array}{l} \text{disyunción} \\ \text{conjunción} \end{array} \right\}$  de sus negaciones.

*Dos formas sentenciales se dicen lógicamente equivalentes si, no importando qué enunciados se sustituyan por sus variables sentenciales – poniendo el mismo enunciado en lugar de la misma variable sentencial en ambas formas sentenciales –, los pares resultantes de enunciados son equivalentes.*

Así, el Teorema de De Morgan afirma que  $\sim(p \vee q)$  y  $\sim p \cdot \sim q$  son formas sentenciales lógicamente equivalentes. Por el Teorema de De Morgan y el principio de la Doble Negación  $\sim(\sim p \cdot \sim q)$  y  $\sim p \vee \sim q$  son lógicamente equivalentes, luego cualquiera puede tomarse como definición de  $p \supset q$ ; la segunda es la elección más usual.

**A todo argumento corresponde un enunciado condicional cuyo antecedente es la conjunción de las premisas del argumento y cuyo consecuente es la conclusión del argumento.**

**Ese condicional correspondiente es una tautología si y solo si el argumento es válido.**

Así la forma argumental válida

$$\begin{array}{c} p \vee q \\ \sim p \\ \therefore q \end{array}$$

corresponde la forma sentencial tautológica  $[(p \vee q) \cdot \sim p] \supset q$ ;

y a la forma argumental inválida

$$\begin{array}{c} p \supset q \\ q \\ \therefore p \end{array}$$

corresponde la forma sentencial no tautológica  $[(p \supset q) \cdot q] \supset p$ .

Una forma argumental es válida si y solo si su tabla de verdad tiene el valor **T** bajo su conclusión en cada renglón en que haya el valor **T** bajo todas sus premisas. Como puede aparecer una **F** en la columna encabezada por el condicional correspondiente solo donde haya **T** bajo todas las premisas y **F** bajo la conclusión, es claro que solo puede haber el valor **T** bajo un condicional que corresponde a una forma argumental válida. Si un argumento es válido, el enunciado de que la conjunción de sus premisas implica su conclusión es una tautología.

Una versión alternativa de la prueba de la tabla de verdad de una forma argumental sentencial es la siguiente, que corresponde a la tabla de verdad precedente:

	(p)	(q)	(p $\wedge$ q)	( $\neg$ p)	( $\neg$ q)	( $\neg$ p $\vee$ q)	(p $\supset$ q)	(p $\supset$ q) $\wedge$ q	(p $\supset$ q) $\supset$ p
(1)	T	T	T	T	F	T	F	F	T
(2)	T	F	F	F	T	F	T	T	F
(3)	F	T	F	T	F	T	T	F	T
(4)	F	F	F	T	T	F	T	T	F
(5)									
(6)									
(7)									
(8)									
(9)									
(10)									

Aquí las columnas (2), (4), (7), (10) son las columnas iniciales o guía. La columna (3) se llena con referencia a las columnas (2) y (4) y la columna (1) con referencia a la columna (3). La columna (6) se llena con referencia a la columna (7), la columna (9) se llena con referencia a la columna (10) y entonces la columna (8) con referencia a las columnas (6) y (9). Finalmente, la columna (5) se llena con referencia a las columnas (1) y (8). El hecho de que su conectivo principal tenga solo valores **T** en su columna de la tabla de verdad, establece que **la forma sentencial probada es una tautología**.

### 3.2.3. El Método de Deducción

#### 3.2.3.1. Prueba Formal de Validez

Cuando los argumentos contienen *más de dos o tres enunciados simples diferentes* como componentes, se hace *difícil y tedioso utilizar tablas de verdad para probar su validez*.

Un método más conveniente de establecer la validez de algunos argumentos es deducir las conclusiones de sus premisas por una secuencia de argumentos más cortos y más elementales que ya se conoce que son válidos. Considérese, por ejemplo, el siguiente argumento en el que aparecen enunciados simples diferentes:

- el procurador general ha impuesto una censura estricta o si Black envió la carta que escribió, entonces Davis recibió un aviso.
- Si nuestras líneas de comunicación no se han interrumpido por completo, entonces si Davis recibió un aviso, entonces Emory fue informado del asunto.
- Si el procurador general ha impuesto una censura estricta, entonces nuestras líneas de comunicación se han interrumpido por completo.
- Nuestras líneas de comunicación no se han interrumpido por completo.
- Por tanto, si Black envió la carta que escribió, entonces Emory fue informado del asunto.

Se puede traducir en nuestro simbolismo como

$$\begin{aligned} A \vee (B \supset D) \\ \sim C \supset (D \supset E) \\ A \supset C \\ \sim C \\ \therefore B \supset E \end{aligned}$$

Establecer la validez de este argumento por medio de una tabla de verdad requeriría una tabla de treinta y dos renglones. Pero podemos probar el argumento dado como válido deduciéndolo de sus premisas por una secuencia de solamente cuatro argumentos cuya validez se ha señalado ya.

- De la tercera y cuarta premisas,  $A \supset C$  y  $\sim C$ , válidamente inferimos  $\sim A$  por Modus Tollens.
- De  $\sim A$  y la primera premisa  $A \vee (B \supset D)$ , válidamente inferimos  $B \supset D$ , por un Silogismo Disyuntivo.
- De la segunda y cuarta premisas,  $\sim C \supset (D \supset E)$  y  $\sim C$ , válidamente se infiere  $D \supset E$  por Modus Ponens.
- Y finalmente, de estas dos últimas conclusiones (o subconclusiones),  $B \supset D$  y  $D \supset E$ , válidamente inferimos  $B \supset E$  por un Silogismo Hipotético.

**Que su conclusión se deduce de sus premisas usando argumentos válidos exclusivamente, prueba que el argumento original es válido.**

Aquí las formas argumentales válidas elementales Modus Ponens (M.P.), Modus Tollens (M.T.), el Silogismo Disyuntivo (D.S.), y el Silogismo Hipotético (H.S.) se usan como Reglas de Inferencia por medio de las cuales se deducen válidamente las conclusiones a partir de las premisas.

Una manera más formal y más concisa de escribir esta prueba de validez es hacer una lista de las premisas y de los enunciados deducidos de ellas en una columna, con las "justificaciones" para estos últimos escritas a un lado de los mismos. En cada caso, la "justificación" para un enunciado especifica los enunciados precedentes a partir de los cuales, y la regla de inferencia por medio de la cual, el enunciado en cuestión fue deducido. Es conveniente poner la conclusión a la derecha de la última premisa, separada de la misma por una línea diagonal que automáticamente señala que todos los enunciados que están por arriba de la misma son premisas.

La prueba formal de validez para el argumento dado puede escribirse como

1.  $A \vee (B \supset D)$
2.  $\sim C \supset (D \supset E)$
3.  $A \supset C$
4.  $\sim C / \therefore B \supset E$
5.  $\sim A \quad 3, 4, M.T.$
6.  $B \supset D \quad 1, 5, D.S.$
7.  $D \supset E \quad 2, 4, M.P.$
8.  $B \supset E \quad 6, 7, H.S.$

**Una prueba formal de validez para un argumento** dado se define como una sucesión de enunciados, cada uno de los cuales es una premisa de ese argumento o se sigue de los precedentes por un argumento válido elemental, y tal que el último enunciado de la secuencia es la conclusión del argumento cuya validez se está demostrando.

Esta definición debe completarse y hacerse más precisa especificando qué es lo que va a contar como "argumento válido elemental". Primero definimos un argumento válido elemental como cualquier argumento que es una instancia de sustitución de una forma de argumento válida, y después presentamos una lista de solo nueve formas de argumento suficientemente obvias para ser vistas como **formas de argumento válidas elementales** y aceptadas como **Reglas de Inferencia**.

Una cuestión que hay que recalcar es que cualquier instancia de sustitución de una forma de argumento válida elemental es un argumento válido elemental. Así, el argumento

$$\begin{array}{l} \sim C \supset (D \supset E) \\ \sim C \\ \therefore (D \supset E) \end{array}$$

es un argumento válido elemental porque es una instancia de sustitución de la forma de argumento válida elemental Modus Ponens (M.P.). Resulta de

$$\begin{array}{l} p \supset q \\ p \\ \therefore q \end{array}$$

sustituyendo  $\sim C$  por  $p$  y  $(D \supset E)$  por  $q$ , así que es de esa forma aun cuando Modus Ponens no es la forma específica del argumento dado.

**Iniciamos** nuestro desarrollo del **método de deducción** presentando una lista de solo nueve formas de argumento válidas elementales que pueden usarse al construir pruebas formales de validez:

### REGLAS DE INFERENCIA

#### 1. Modus Ponens (M.P.)

$$\begin{array}{l} p \supset q \\ p \\ \therefore q \end{array}$$

#### 6. Dilema Destructivo (D.D.)

$$\begin{array}{l} (p \supset q) \cdot (r \supset s) \\ \sim q \vee \sim r \\ \therefore \sim p \vee \sim r \end{array}$$

#### 2. Modus Tollens (M.T.)

$$\begin{array}{l} p \supset q \\ \sim q \\ \therefore \sim p \end{array}$$

#### 7. Simplificación (Simp.)

$$\begin{array}{l} p \cdot q \\ \therefore p \end{array}$$

## 3. Silogismo Hipotético (H.S.)

$$p \supset q$$

$$q \supset r$$

$$\therefore p \supset r$$

## 8. Conjunción (Conj.)

$$p$$

$$q$$

$$\therefore p \cdot q$$

## 4. Silogismo Disyuntivo (D.S.)

$$p \vee q$$

$$\sim p$$

$$\therefore q$$

## 9. Adición (Ad.)

$$p$$

$$\therefore p \vee q$$

## 5. Dilema Constructivo (C.D.)

$$(p \supset q) \cdot (r \supset s)$$

$$p \vee r$$

$$\therefore q \vee s$$

Estas nueve reglas de inferencia son **formas válidas elementales de argumentos** cuya validez fácilmente se establece mediante tablas de verdad. Pueden usarse para construir pruebas formales de validez para una amplia clase de argumentos más complicados. Los nombres de la lista son estándar en su mayor parte, y el uso de sus abreviaciones permite presentar las pruebas formales con un mínimo de escritura.

### 3.2.3.2. La Regla de Reemplazo

Hay muchos argumentos válidos de función de verdad cuya validez no se puede probar usando solamente las nueve Reglas de Inferencia dadas hasta aquí. Por ejemplo, una prueba formal de validez del argumento obviamente válido

$$A \cdot B$$

$$\therefore B$$

requiere Reglas de Inferencia adicionales.

Ahora bien, los únicos enunciados compuestos que nos interesan aquí son los enunciados compuestos función de verdad. Luego, si se reemplaza una parte cualquiera de un enunciado compuesto por una expresión que es lógicamente equivalente a la parte reemplazada, el valor de verdad del enunciado que resulta es el mismo que el del enunciado original. A esto se le llama, algunas veces la **Regla de Reemplazo**, y otras, la del Principio de Extensionalidad.

Adoptamos la **Regla de Reemplazo** como un **principio adicional de inferencia**. Nos permite inferir de cualquier enunciado el resultado de reemplazar todo o parte de ese enunciado por otro enunciado lógicamente equivalente a la parte reemplazada. Así, usando el principio de la Doble Negación (D.N.), que afirma la equivalencia lógica de  $p$  y  $\sim\sim p$ , podemos inferir, de  $A \supset \sim\sim B$  cualquiera de los enunciados,

$$A \supset B, \sim\sim A \supset \sim\sim B, A \supset \sim\sim\sim B, \text{ o } \sim(A \supset \sim\sim B)$$

por la Regla de Reemplazo.

Para hacer más definida esta regla, damos ahora una lista de equivalencias lógicas con las que puede usarse. Estas equivalencias constituyen **nuevas Reglas de Inferencia** que es posible usar para probar la validez de argumentos. Las numeraremos consecutivamente después de las nueve reglas ya enunciadas.

**Regla de Reemplazo:** Cualquiera de las siguientes expresiones lógicamente equivalentes puede reemplazar a la otra en donde ocurran:

10. Teoremas de De Morgan (De M.):	$\sim(p \cdot q) \equiv (\sim p \vee \sim q)$
	$\sim(p \vee q) \equiv (\sim p \cdot \sim q)$
11. Comutación (Comm.):	$(p \vee q) \equiv (q \vee p)$
	$(p \cdot q) \equiv (q \cdot p)$
12. Asociación (Asoc.):	$[p \vee (q \vee r)] \equiv [(p \vee q) \vee r]$
	$[p \cdot (q \cdot r)] \equiv [(p \cdot q) \cdot r]$
13. Distribución (Dist.):	$[p \cdot (q \vee r)] \equiv [(p \cdot q) \vee (p \cdot r)]$
	$[p \vee (q \cdot r)] \equiv [(p \vee q) \cdot (p \vee r)]$
14. Doble Negación (D.N.):	$p \equiv \sim\sim p$
15. Transposición (Trans.):	$(p \supset q) \equiv (\sim q \supset \sim p)$
16. Implicación Material (Impl.):	$(p \supset q) \equiv (\sim p \vee q)$
17. Equivalencia Material (Equiv.):	$(p \equiv q) \equiv [(p \supset q) \cdot (q \supset p)]$
	$(p \equiv q) \equiv [(p \cdot q) \vee (\sim p \cdot \sim q)]$
18. Exportación (Exp.):	$[(p \cdot q) \supset r] \equiv [(p \supset (q \supset r))]$
19. Tautología (Taut.):	$p \equiv (p \vee p)$
	$p \equiv (p \cdot p)$

Ahora puede escribirse una prueba formal de validez para el argumento dado al principio del párrafo 3.2:

- |          |                |
|----------|----------------|
| 1. A.B / | $\therefore B$ |
| 2. B.A   | 1, Comm.       |
| 3. B     | 2, Simp.       |

Algunas formas de argumento, aunque muy elementales y perfectamente válidas, no se incluyen en nuestra lista de diecinueve Reglas de Inferencia. Aunque el argumento

A.B

$\therefore B$

es obviamente válido, su forma

p.q

$\therefore q$

no está incluida en nuestra lista. Por tanto, B no se sigue de A.B por ningún argumento válido elemental según los define nuestra lista. Puede, sin embargo, deducirse usando dos argumentos válidos elementales como mostramos antes. Podríamos agregar la forma de argumento intuitivamente válida

p.q

$\therefore q$

a nuestra lista, claro está; pero si agrandáramos nuestra lista de esta manera llegaríamos a tener una lista demasiado larga y, por tanto, no manejable.

La lista de las Reglas de Inferencia contiene numerosas redundancias. Por ejemplo, Modus Tollens podría salir de la lista sin realmente debilitar la maquinaria, pues todo paso deducido usándola puede serlo usando otras Reglas de la lista. Pero Modus Tollens es un principio de inferencia tan común e intuitivo que se le ha incluido, y otros han sido incluidos por conveniencia también, a pesar de su redundancia lógica.

***La prueba de que una sucesión dada de enunciados es una demostración formal, es efectiva.***

Es decir, por observación directa se podrá deducir si cada renglón siguiente a las premisas se sigue o no de los renglones que le preceden mediante alguna de las Reglas de Inferencia dadas. No es necesario "pensar": ni pensar sobre la validez de la deducción de cada renglón. Aún en donde falte la "justificación" de un enunciado, a un lado del mismo, hay un procedimiento finito, mecánico, para decidir si la deducción es legítima. Cada renglón viene precedido por solamente un número finito de renglones y solo se han adoptado un número finito de Reglas de Inferencia. Aunque toma tiempo, puede verificarse por inspección si el renglón en cuestión se sigue de algún renglón, o par de renglones precedentes mediante alguna Regla de Inferencia de nuestra lista.

***Hay una diferencia importante entre las primeras nueve y las últimas diez Reglas de Inferencia.***

- *Las primeras nueve pueden aplicarse a renglones enteros de una demostración.*

De modo A puede inferirse de A.B por simplificación solo si A.B constituye un renglón completo. Pero ni A ni  $A \supset C$  se siguen de  $(A.B) \supset C$  por simplificación o cualquier otra Regla de Inferencia. A no es consecuencia porque A puede ser falso y  $(A.B) \supset C$  verdadero.  $A \supset C$  no es consecuencia porque si A es verdadero y B y C ambos son falsos,  $(A.B) \supset C$  es verdadero mientras que  $A \supset C$  es falso.

- *Por otro lado, cualquiera de las diez últimas Reglas de Inferencia puede aplicarse a renglones enteros o partes de renglones.*

No solo puede inferirse el enunciado  $A \supset (B \supset C)$  del renglón entero  $(A.B) \supset C$  por Exportación, sino del renglón  $[(A.B) \supset C] \vee D$  podemos inferir  $[(A \supset B) \supset C] \vee D$  por Exportación. La Regla de Reemplazo autoriza que expresiones lógicamente equivalentes especificadas se reemplacen entre sí donde ocurran aún en donde no constituyan renglones enteros de una demostración.

Pero las nueve primeras Reglas de Inferencia solo pueden usarse tomando como premisas renglones enteros de una demostración.

En ausencia de reglas mecánicas para la construcción de demostraciones formales de validez, pueden darse algunas sugerencias y métodos prácticos.

- La primera es simplemente *empezar deduciendo conclusiones de las premisas mediante las Reglas de Inferencia dadas*. Al tener más y más subconclusiones de éstas como premisas para nuevas deducciones, mayor es la probabilidad de que se vea como deducir la conclusión del argumento que se quiere demostrar que es válido.
- Otra sugerencia es *tratar de eliminar enunciados que ocurren en las premisas, pero no en la conclusión*. Esta eliminación puede llevarse a cabo solamente de acuerdo con las Reglas de Inferencia. Pero las Reglas contienen muchas técnicas para eliminar enunciados. La simplificación es una de ellas: con ésta, el conyunto derecho de una conjunción puede simplemente quitarse, a condición de que la conjunción sea un renglón entero en la demostración. Y por Comutación puede hacerse derecho al enunciado conyunto izquierdo de una conjunción para eliminarlo por Simplificación. El término "medio" q puede eliminarse por un Silogismo Hipotético dadas dos premisas o subconclusiones de los patrones  $p \supset q$  y  $q \supset r$ . La distribución es una regla útil para transformar una disyunción de la forma  $p \vee (q.r)$  en la conjunción  $(p \vee q).(p \vee r)$  cuyo conjunto de la derecha  $p \vee r$  puede entonces eliminarse por Simplificación.
- Otro método práctico es *introducir por Adición un enunciado que ocurre en la conclusión, pero no en las premisas*.
- Otro método es el de *proceder hacia atrás desde la conclusión buscando algún enunciado o par de enunciados de los cuales se le pudiera deducir mediante algunas de las Reglas de Inferencia, y entonces tratar de deducir esos enunciados intermedios, y así sucesivamente, hasta llegar a algunos que sean deducibles de las premisas*.

### 3.2.3.3. Demostración de la Invalidad

#### No Completud de las Diecinueve Reglas

Las diecinueve Reglas de Inferencia presentadas hasta el momento son incompletas, lo que quiere decir que hay argumentos válidos función de verdad cuya validez no es demostrable usando tan solo esas diecinueve Reglas.

#### La Regla de Demostración Condicional

A continuación **introducimos una nueva regla para usarla en el método de deducción: la regla de Demostración Condicional**. En esta sección la nueva regla se aplicará tan solo a argumentos cuyas conclusiones son enunciados condicionales.

- A todo argumento le corresponde un enunciado condicional cuyo antecedente es la conjunción de las premisas del argumento y cuyo consecuente es la conclusión del argumento.
- Como se ha señalado, un argumento es válido si y solo si su correspondiente condicional es una tautología.

Si un argumento tiene un enunciado condicional como conclusión, que podemos simbolizar  $A \supset C$ , entonces si simbolizamos la conjunción de sus premisas como P, el argumento es válido si y solo si el condicional

$$(1) \quad P \supset (A \supset C)$$

es una tautología.

Si podemos deducir la conclusión  $A \supset C$  de las premisas conjuntas en P por una secuencia de argumentos válidos elementales, habremos así demostrado la validez del argumento y que el condicional asociado (1) es una tautología. Por el principio de Exportación, (1) es lógicamente equivalente a

$$(2) \quad (P.A) \supset C$$

Pero (2) es el condicional asociado con un argumento un tanto diferente. Este segundo argumento tiene como premisas todas las premisas del primer argumento, además de una premisa adicional que es el antecedente de la conclusión del primer argumento. Y la conclusión del segundo argumento es el consecuente de la conclusión del primer argumento.

Ahora bien, si deducimos la conclusión del segundo argumento, C, de las premisas conjuntas en P.A por una sucesión de argumentos válidos elementales, habremos demostrado que su enunciado condicional asociado (2) es una tautología. Pero como (1) y (2) son lógicamente equivalentes esto demuestra que (1) es una tautología también, de donde se sigue que el argumento original con una premisa menos y la conclusión condicional  $A \supset C$  también es válido. Ahora la regla de Demostración Condicional nos permite inferir la validez de cualquier argumento

$$\begin{array}{l} P \\ \therefore A \supset C \end{array}$$

de una demostración formal de validez para el argumento

$$\begin{array}{l} P \\ A \\ \therefore C \end{array}$$

Dado cualquier argumento cuya conclusión es un enunciado condicional, una demostración de su validez usando la regla de Demostración Condicional, es decir, una demostración condicional de su validez, se construye suponiendo que el antecedente de su conclusión es una premisa adicional y luego deduciéndolo el consecuente de su conclusión por una sucesión de argumentos válidos elementales.

Así, una demostración condicional de validez para el argumento

$$\begin{aligned} & (A \vee B) \supset (C \cdot D) \\ & (D \vee E) \supset F \\ & \therefore A \supset F \end{aligned}$$

puede escribirse como

- |                                     |                            |
|-------------------------------------|----------------------------|
| 1. $(A \vee B) \supset (C \cdot D)$ |                            |
| 2. $(D \vee E) \supset F$           | $/ \therefore A \supset F$ |
| 3. <b>A</b>                         | <b>F (C.P.)</b>            |
| 4. $A \vee B$                       | 3, Ad.                     |
| 5. $C \cdot D$                      | 1, 4, M.P.                 |
| 6. $D \cdot C$                      | 5, Comm.                   |
| 7. $D$                              | 6, Simpl.                  |
| 8. $D \vee E$                       | 7, Ad.                     |
| 9. $F$                              | 2, 8, M.P.                 |

Aquí la diagonal de separación, así como el símbolo "por lo tanto" de los tres puntos, y el "**C.P.**", indican que se está usando la regla de la Demostración Condicional.

### La Regla de Demostración Indirecta

El método de **Demostración Indirecta**, a menudo llamado método de **Demostración por Reducción al absurdo**, es familiar para todos los que hayan estudiado la geometría elemental. Al deducir sus teoremas, Euclides suele empezar suponiendo lo opuesto de lo que se propone demostrar.

*Si este supuesto conduce a una contradicción o "se reduce a un absurdo" entonces el supuesto debe ser falso, y su negación – el teorema que se desea demostrar – debe ser verdadero.*

Una demostración indirecta de validez para un argumento dado se construye suponiendo, como premisa adicional, la negación de su conclusión y deduciéndo entonces una contradicción explícita del conjunto aumentado de las premisas. Así, una demostración indirecta de validez para el argumento

$$\begin{aligned} & A \supset (B \cdot C) \\ & (B \vee D) \supset E \\ & D \vee A \\ & \therefore E \end{aligned}$$

puede escribirse como a continuación

- |                            |                                    |
|----------------------------|------------------------------------|
| 1. $A \supset (B \cdot C)$ |                                    |
| 2. $(B \vee D) \supset E$  |                                    |
| 3. $D \vee A$              | $/ \therefore E$                   |
| 4. $\sim E$                | <b>IP (Demostración Indirecta)</b> |
| 5. $\sim(B \vee D)$        | 2, 4, M.T.                         |
| 6. $\sim B \cdot \sim D$   | 5, De M.                           |
| 7. $\sim D \cdot \sim B$   | 6, Comm.                           |
| 8. $\sim D$                | 7, Simpl.                          |
| 9. $A$                     | 3, 8, D.S.                         |
| 10. $B \cdot C$            | 1, 9, M.P.                         |
| 11. $B$                    | 10, Simpl.                         |
| 12. $\sim B$               | 6, Simpl.                          |
| 13. $B \cdot \sim B$       | 11, 12, Conj.                      |

El renglón 13 es una **contradicción** explícita, luego, **la demostración es completa**, pues la validez del argumento original se sigue por la regla de Demostración Indirecta.

### 3.3. Lógica de Predicados

#### 3.3.1. Introducción y concepto de Semidecidible

*El aspecto más atractivo del formalismo lógico es que proporciona de manera inmediata un método muy potente para la obtención de nuevo conocimiento a partir del antiguo: la deducción matemática. En este formalismo se puede concluir la verdad de un aserto sin otra cosa que demostrar que es consecuencia de lo ya conocido.*

De esta manera la idea de prueba, según se entiende en matemáticas como un método riguroso de demostración de una proposición que se cree cierta, se puede extender a la deducción como un medio de obtener respuestas a preguntas y soluciones a problemas.

Uno de los primeros dominios donde se aplicó la IA fue la demostración automática de teoremas, que tenía como objetivo la demostración de proposiciones en distintas áreas de la matemática. Por ejemplo, el Logic Theorist (Newell et al., 1963) demostraba teoremas del primer capítulo de los Principia Matemática de Whitehead y Russell (1950). Otro demostrador de teoremas (Gelernter et al., 1963) trabaja sobre teoremas de geometría. La demostración de teoremas matemáticos sigue siendo un área de investigación activa en IA. Pero, la utilidad de las técnicas matemáticas se ha extendido más allá del ámbito tradicional de las matemáticas. La matemática no resulta muy diferente de cualquier otra labor intelectual compleja en lo que se refiere a la necesidad de mecanismos de deducción fiables y de una gran cantidad de conocimiento heurístico para el control de lo que, de otra forma, sería un problema de búsqueda intratable.

*Una de las razones más importantes para utilizar el formalismo lógico es que si se representa el conocimiento como sentencias lógicas, entonces se dispone de un buen mecanismo para razonar con ese conocimiento.*

Determinar la validez de una proposición en lógica proposicional es algo inmediato, aunque pueda ser computacionalmente costoso.

Por lo tanto, antes de adoptar la **lógica de predicados** como un medio apropiado para la representación del conocimiento, sería bueno preguntarse *si también ofrece un mecanismo adecuado para razonar con ese conocimiento*.

A primera vista la respuesta parece afirmativa. *Es posible deducir nuevas sentencias a partir de las antiguas. Sin embargo, y por desgracia, no se dispone de un procedimiento de decisión*, ni siguiera de uno exponencial. Existen procedimientos que permitirán encontrar la prueba de un teorema dado si es que en realidad se trata de un teorema. Pero no está garantizado que estos procedimientos paren cuando la sentencia no sea un teorema.

En otras palabras, *aunque la lógica de predicados de primer orden no es decidible, sí es semidecidible*. Un procedimiento muy sencillo consiste en aplicar sistemáticamente las reglas de inferencia sobre los axiomas, siguiendo un cierto orden, comprobando cada uno de los teoremas generados para determinar si es el que se pretende demostrar. Sin embargo, este método no es especialmente eficiente por lo que sería aconsejable encontrar uno mejor.

A pesar de que los resultados negativos, tales como el hecho de que no exista ningún procedimiento de decisión para la lógica de predicados, no suelen tener gran importancia en una ciencia como la IA, que se ocupa de buscar métodos positivos para hacer las cosas, este resultado negativo en particular resulta de utilidad, puesto que permite afirmar que en la búsqueda de un procedimiento de prueba eficiente, basta con encontrar uno que demuestre teoremas, aunque no esté garantizada la parada ante algo que no sea un teorema. Y el hecho de que no pueda existir un procedimiento de decisión que pare con todas las entradas posibles, no significa que no pueda existir uno que pare en la mayoría de los casos cuando de lo que se trate sea de resolver problemas del mundo real.

*Por lo tanto, la lógica de predicados, a pesar de su indecidibilidad teórica, puede resultar útil como medio de representación y manipulación de ciertos tipos de conocimiento que pueden ser necesarios en un sistema de IA.*

### 3.3.2. Representación de hechos simples en lógica

En primer lugar exploraremos el uso de la *lógica proposicional* como medio de representación del tipo de conocimiento acerca del mundo que puede ser necesario en un sistema de IA. *El atractivo de la lógica proposicional consiste en su simplicidad y en la disponibilidad de un procedimiento de decisión.* Es muy sencillo representar los hechos del mundo real como proposiciones lógicas escritas en forma de *fórmulas bien formadas (fbf)* de la lógica proposicional, como se muestra en la Figura 3.6.

Está lloviendo. LLUEVE
Está soleado. SOLEADO
Hace viento. VENTOSO
Llueve, por lo tanto, no hace sol. LLUEVE → ¬SOLEADO

Figura 3.6

Mediante estas proposiciones se podría deducir, por ejemplo, que no hace sol a partir del hecho de que está lloviendo. Supóngase que se desea representar el hecho obvio que se enuncia en la conocida frase:

Sócrates es un hombre.

Se podría escribir:

SOCRATESHOMBRE

Pero si además se quisiera representar el hecho:

Platón es un hombre.

habría que escribir algo así como:

PLATONHOMBRE

que no tiene nada en común con la anterior, de manera que no es posible llegar a ninguna conclusión acerca de las similitudes entre Sócrates y Platón. Sería mucho mejor si estos hechos se representasen como:

**HOMBRE(SÓCRATES)**

**HOMBRE(PLATON)**

puesto que *ahora la estructura del conocimiento está reflejada en la propia estructura de la representación*. Pero para ello es necesario utilizar predicados que se aplican sobre argumentos. Sería aún más difícil representar la también conocida frase:

Todos los hombres son mortales.

Que se podría representar como:

**HOMBREMORTAL**

Pero de esta forma no se consigue expresar el hecho de que todos y cada uno de los hombres son mortales.

Para hacerlo, *es necesario utilizar la lógica de predicados de primer orden (o simplemente lógica de predicados) como medio de representación del conocimiento, dado que permite representar cosas que no serían representables de forma razonable utilizando la lógica proposicional.*

Mediante la lógica de predicados se pueden representar hechos del mundo real como sentencias escritas en forma de fbf.

A continuación se presenta un ejemplo específico donde se utiliza la lógica de predicados como medio de representación del conocimiento. Considérese las siguientes sentencias:

1. Marco era un hombre.
2. Marco era un pompeyano.
3. Todos los pompeyanos eran romanos.
4. César fue un gobernante.
5. Todos los romanos o eran leales a César o le odiaban.
6. Todo el mundo es leal a alguien.
7. La gente solo intenta asesinar a los gobernantes a los que no es leal.
8. Marco intentó asesinar a César.

Los hechos descritos en esas frases se pueden representar como un conjunto de fbf de la lógica de predicados de la siguiente manera:

1. Marco era un hombre.

#### **hombre(Marco)**

En esta representación se recoge el hecho fundamental de la humanidad de Marco.

Sin embargo, se pierden ciertos detalles de la frase en lenguaje natural, como el concepto de tiempo pasado. Según el uso que se pretenda hacer del conocimiento, esta omisión será aceptable o no. Para este sencillo ejemplo es suficiente.

2. Marco era un pompeyano.

#### **pompeyano(Marco)**

3. Todos los pompeyanos eran romanos.

$\forall x: \text{pompeyano}(x) \rightarrow \text{romano}(x)$

4. César fue un gobernante.

#### **gobernante(César)**

Aquí se ignora el hecho de que los nombres propios no se suelen referir a un único individuo, puesto que hay mucha gente que comparte el mismo nombre.

En ocasiones, para decidir a quién se refiere una frase en particular de entre un grupo de personas con el mismo nombre, puede ser necesaria una gran cantidad de conocimiento y de razonamiento.

5. Todos los romanos o eran leales a César o le odiaban.

$\forall x: \text{romano}(x) \rightarrow \text{leal}(x, \text{César}) \vee \text{odia}(x, \text{César})$

En lenguaje natural, la palabra “o” se puede referir a la o-conjuntiva de la lógica o a la o-disyuntiva (XOR). Aquí se ha utilizado la interpretación conjuntiva. Es posible argumentar que esta frase en realidad tiene un significado disyuntivo. Para expresarlo se escribiría:

$\forall x: \text{romano}(x) \rightarrow$

$[(\text{leal}(x, \text{César}) \vee \text{odia}(x, \text{César})) \wedge \neg(\text{leal}(x, \text{César}) \wedge \text{odia}(x, \text{César}))]$

6. Todo el mundo es leal a alguien

$\forall x: \exists y: \text{leal}(x, y)$

Un problema importante que se plantea cuando se intenta convertir frases en lenguaje natural a sentencias lógicas, es el ámbito de los cuantificadores. Con esta frase se quiere decir, como se ha supuesto al escribir la anterior fórmula lógica, que cada persona tiene alguien a quien él o ella es leal, probablemente alguien diferente para cada cual. O quiere decir que hay alguien a quien todo el mundo es leal (que se escribiría como  $\exists y: \forall x: leal(x,y)$ ). Normalmente solo una de las interpretaciones parece plausible, y es ésa la que se suele tomar.

7. La gente solo intenta asesinar a los gobernantes a los que no es leal.

$\forall x: \forall y: persona(x) \wedge gobernante(y) \wedge intenta\_asesinar(x,y) \rightarrow \neg leal(x,y)$

Esta frase también es ambigua. Significa que a los únicos gobernantes a los que la gente intenta asesinar son aquellos a los que no son leales (la interpretación que se ha utilizado), o significa que lo único que la gente intenta hacer es asesinar a gobernantes a los que no son leales.

En la representación de esta frase se ha decidido escribir "*intenta asesinar*" como un único predicado. De esta forma se obtiene una representación sencilla que permite razonar acerca de intentos de asesinato. Pero con esta representación, no sería fácil establecer conexiones entre intentar un asesinato e intentar cualquier otra cosa o entre un intento de asesinato y un verdadero asesinato. Si fuese necesario establecer tales conexiones habría que escoger una representación diferente.

8. Marco intentó asesinar a César.

**intenta\_asesinar(Marco,César)**

Los anteriores ejemplos dan una idea clara de la dificultad de convertir frases en lenguaje natural a sentencias lógicas.

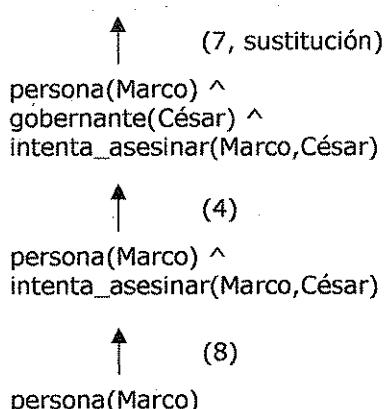


Figura 3.7

Supongamos que se desea utilizar los anteriores asertos para responder a la siguiente pregunta:

**¿Era Marco leal a César?**

Parece que por medio de 7 y 8 se puede demostrar que Marco no era leal a César (ignorando de nuevo la distinción entre tiempo pasado y presente). Intentemos ahora obtener una prueba formal razonando a la inversa desde el objetivo a alcanzar:

$\neg leal(\text{Marco}, \text{César})$

Para probar dicho objetivo es necesario utilizar las reglas de inferencia que permiten transformarlo en otro objetivo (o posiblemente conjunto de objetivos) que a su vez puedan ser transformados, y así sucesivamente, hasta que no quede ningún objetivo por satisfacer.

Para realizar este proceso es posible que sea necesario utilizar un grafo Y-O cuando haya caminos alternativos para satisfacer un objetivo dado. Por simplicidad solo se muestra uno de los posibles caminos. *La Figura 3.7 muestra un intento de prueba del objetivo mediante la reducción del conjunto de objetivos necesarios aún sin satisfacer, a un conjunto vacío.* El intento falla, ya que no se puede satisfacer el objetivo persona(Marco) con los asertos disponibles.

*El problema es que, a pesar de que se sabe que Marco era un hombre, no se puede deducir que era una persona.* Se debe añadir la representación de otro hecho al sistema:

9. Todos los hombres son personas.

$\forall x: \text{hombre}(x) \rightarrow \text{persona}(x)$

De esta forma se satisface el último objetivo y se obtiene una prueba de que **Marco no era leal a César.**

En este ejemplo se hallan presentes tres importantes cuestiones involucradas en el proceso de conversión de frases en lenguaje natural a sentencias lógicas y el posterior uso de estas sentencias en la deducción de otras nuevas:

- Muchas frases en lenguaje natural son ambiguas (por ejemplo, 5, 6 y 7 del ejemplo anterior). La selección de la interpretación correcta puede ser difícil.
- Normalmente hay más de una forma de representar el conocimiento (como ya se vio en las frases 1 y 7). La simplicidad de la representación es una cualidad deseable, pero puede ocurrir que de esa forma se penalice cierto tipo de razonamiento. La conveniencia de la representación que se escoga para un conjunto de frases dependerá del uso que se quiera hacer del conocimiento contenido en dichas frases.
- Aún en los casos más sencillos, no es probable que en un conjunto de frases dado, esté contenida toda la información necesaria para razonar sobre el tema en cuestión. Para poder razonar de manera efectiva a partir de un conjunto de sentencias, normalmente es necesario utilizar un conjunto de sentencias adicionales que representan aquellos hechos que, por lo general no se suelen especificar, por demasiado obvios.

*Se puede presentar un problema adicional cuando no se conocen de antemano las sentencias que se va a intentar deducir.* En el ejemplo anterior el objetivo era responder a la pregunta "¿era Marco leal a César?". Pero ¿Cómo puede decidir un programa cual de estas dos sentencias demostrar?

$\text{leal}(\text{Marco}, \text{César})$

$\neg \text{leal}(\text{Marco}, \text{César})$

1. Se puede intentar diferentes cosas como razonar a partir de los axiomas y ver qué respuesta se obtiene, en lugar de utilizar la estrategia empleada en el ejemplo anterior de razonar hacia atrás a partir del objetivo propuesto. El problema de razonar hacia delante a partir de los axiomas es que es probable que el factor de ramificación provoque que no se obtenga ninguna respuesta en un tiempo razonable.
2. Una segunda posibilidad consistiría en utilizar algún tipo de reglas heurísticas que permitiesen decidir cual es la respuesta más probable y entonces intentar demostrarla. Si después de un tiempo razonable no se encontrase una demostración, se intentaría con la otra respuesta. Es importante esta idea de limitar el esfuerzo, puesto que cualquier procedimiento de prueba que se utilice puede no parar cuando se enfrenta con algo que no es un teorema.
3. Otra posibilidad sería tratar de demostrar ambas respuestas simultáneamente y parar cuando una de las dos demostraciones tuviese éxito. Sin embargo, aún así, si no se dispone de suficiente información para responder a la pregunta, es posible que el programa no pare nunca.

4. Una cuarta estrategia consiste en intentar demostrar la verdad de una respuesta y la falsedad de la contraria y utilizar la información obtenida en cada proceso como ayuda en el otro.

### 3.3.3. La Representación de las relaciones instancia y es-un

Los atributos *instancia* y *es-un* juegan un papel importante en el modelo de razonamiento guiado por la herencia de propiedades. Si se revisa la manera en que se ha representado el conocimiento acerca de Marco y César en el ejemplo anterior, se observará que no se han utilizado estos atributos. Ciertamente no se han utilizado predicados con esos nombres. ¿Por qué no? La respuesta es que, a pesar de que no se han utilizado explícitamente los predicados *instancia* y *es-un*, sí se han empleado las relaciones que representan, la pertenencia a una clase y la inclusión de una clase en otra.

La Figura 3.8 muestra la representación lógica, escrita de tres formas distintas, de las primeras cinco frases del apartado anterior.

1. hombre(Marco) 2. pompeyano(Marco) 3. $\forall x: \text{pompeyano}(x) \rightarrow \text{romano}(x)$ 4. gobernante(César) 5. $\forall x: \text{romano}(x) \rightarrow \text{leal}(x, \text{César}) \vee \text{odia}(x, \text{César})$
1. instancia(Marco, hombre) 2. instancia(Marco, pompeyano) 3. $\forall x: \text{instancia}(x, \text{pompeyano}) \rightarrow \text{instancia}(x, \text{romano})$ 4. instancia(César, gobernante) 5. $\forall x: \text{instancia}(x, \text{romano}) \rightarrow \text{leal}(x, \text{César}) \vee \text{odia}(x, \text{César})$
1. instancia(Marco, hombre) 2. instancia(Marco, pompeyano) 3. es-un(pompeyano, romano) 4. instancia(César, gobernante) 5. $\forall x: \text{instancia}(x, \text{romano}) \rightarrow \text{leal}(x, \text{César}) \vee \text{odia}(x, \text{César})$ 6. $\forall x: \forall y: \forall z: \text{instancia}(x, y) \wedge \text{es-un}(y, z) \rightarrow \text{instancia}(x, z)$

Figura 3.8

En la **primera parte** de la figura aparece una representación de la que ya hemos hablado. En esta representación, *la pertenencia a una clase se representa con predicados monarios* (como *romano*) que se refieren a cada una de las clases. Asertar *P(x)* como cierto es equivalente a afirmar que *x* es una instancia (o un elemento) de *P*.

En la **segunda parte** de la figura aparece una representación donde se utiliza explícitamente el predicado *instancia*. *Este es un predicado binario, cuyo primer argumento es un objeto y cuyo segundo argumento es la clase a la que ese objeto pertenece. Sin embargo, en esta representación no se utiliza explícitamente el predicado es-un.* En la tercera sentencia de la segunda parte de la figura aparece un ejemplo de cómo se representa la relación que se establece entre dos subclases cuando una está incluida en la otra, como por ejemplo la clase de los pompeyanos en la clase de los romanos. La regla de implicación que ahí aparece se lee como que si un objeto es una instancia de la subclase pompeyano entonces también será una instancia de la superclase *romano*. Nótese que esta regla es equivalente a la definición de la relación de inclusión en la teoría de conjuntos.

Por último, en la figura se muestra una **tercera representación** donde se utilizan explícitamente los predicados *instancia* y *es-un*. Con el uso del predicado *es-un* se simplifica la representación de la tercera sentencia, pero hace necesaria la definición de un nuevo axioma (definido con el número 6). Este axioma adicional describe como se puede combinar una relación de instancia con una relación *es-un* para dar lugar a una nueva relación de instancia. Sin embargo, este axioma es de carácter general y no es necesario definirlo para cada nueva relación *es-un*.

Estos ejemplos pretenden ilustrar dos ideas diferentes.

- La primera, bastante específica, nos permite afirmar que no es necesario representar explícitamente las relaciones de pertenencia a una clase e inclusión de una clase en otra por medio de los predicados instancia y es-un. De hecho, normalmente resulta engoroso hacerlo de esta forma y se suele optar por la utilización de predicados monarios que representan a las clases.
- La segunda idea es más general y viene a decir que normalmente es posible obtener varias representaciones de un determinado hecho en un sistema de representación dado, ya sea éste la lógica o cualquier otro. La elección dependerá, en parte del tipo de deducciones que se pretenda agilizar, y en parte del gusto de cada cual. La norma que siempre se debe respetar es el uso consistente de una determinada representación en toda la base de conocimiento. Dado que las reglas de inferencia se construyen para ser aplicadas sobre una forma de representación en concreto, es necesario que todo el conocimiento sobre el que esas reglas son aplicables esté representado de la forma que las reglas necesiten. El uso de representaciones inconsistentes provoca muchos errores en los mecanismos de razonamiento de los programas basados en conocimiento. La moraleja es, simplemente, que se debe ser cuidadoso.

### 3.3.4. Representación de Funciones calculables y Predicados computables

En el ejemplo del apartado anterior, todos los hechos simples se representaban como combinaciones de predicados aislados, tales como:

`intenta_asesinar(Marco,César)`

Esta es una buena solución cuando el número de hechos es pequeño o cuando los propios hechos carecen de una estructura definida de forma que no existen demasiadas alternativas.

Pero supongamos que se desea expresar hechos simples tales como las siguientes relaciones mayor-que y menor-que:

`mayor(1,0) menor(0,1)  
mayor(2,1) menor(1,2)  
mayor(3,2) menor(2,3)`

⋮  
⋮  
⋮

Está claro que **sería tedioso representar todos estos hechos, uno a uno, explícitamente**. Entre otras cosas porque los hay infinitos. Pero aún en el caso de que solo considerásemos el subconjunto finito que es posible representar en, digamos, una palabra máquina por número, sería muy ineficiente representarlos todos explícitamente cuando, en lugar de ello, es posible **calcularlos en el momento necesario**.

Por tanto, resulta muy conveniente **extender la representación** para poder expresar estos **predicados computables**. Independientemente del procedimiento de demostración que se utilice, cuando se encuentre con un predicado de este tipo, en lugar de buscarlo explícitamente en la base de datos o intentar deducirlo mediante algún tipo de razonamiento, bastará con **invocar un procedimiento, especificado previamente**, que devolverá un valor de verdadero o falso.

También suele ser útil disponer de **funciones computables** además de predicados computables. De esta forma sería posible evaluar el predicado:

**`mayor(2+3,1)`**

Para ello es necesario evaluar en primer lugar el valor de la función suma con los argumentos 2 y 3, para luego evaluar el predicado mayor con los argumentos 5 y 1.

En el siguiente ejemplo se muestra la utilidad de esta idea de funciones y predicados computables. También se hace uso del concepto de **igualdad**, de modo que sea posible sustituir un objeto por otro que sea igual a él, cuando este cambio resulte de alguna utilidad en el proceso de prueba.

Considérese el siguiente conjunto de hechos que, otra vez, se refieren a Marco:

1. Marco era un hombre.

**hombre(Marco)**

De nuevo no tenemos en consideración el tiempo verbal.

2. Marco era pompeyano.

**pompeyano(Marco)**

3. Marco nació en el 40 D.C.

**nace(Marco,40)**

Por simplicidad no representaremos D.C. explícitamente, de la misma forma que se suele omitir en las conversaciones de la vida diaria. Si en algún momento fuese necesario representar fechas A.C., entonces habría que decidir la manera de hacerlo, por ejemplo, utilizando números negativos. Nótese que la representación de una frase no tiene que ser una copia de la frase en sí, siempre que sea posible pasar de la fase a su representación y viceversa. Esto permite elegir una representación tal como números positivos y negativos, con la que un programa puede trabajar fácilmente.

4. Todos los hombres son mortales.

$\forall x: \text{hombre}(x) \rightarrow \text{mortal}(x)$

5. Todos los pompeyanos murieron cuando el volcán entró en erupción en el 79 D.C.

**erupción(volcan,79) ^  $\forall x: [\text{pompeyano}(x) \rightarrow \text{murió}(x,79)]$**

Claramente en esta sentencia se representan los dos hechos enunciados en la frase anterior. También se podría asertar otro hecho que no aparece en la representación, a saber, que la erupción del volcán causó la muerte de los pompeyanos. Normalmente se suele suponer una relación de causalidad entre efectos que se producen al mismo tiempo siempre que esa suposición sea plausible.

La interpretación de esta frase presenta un problema adicional que consiste en determinar el referente de la expresión "el volcán". Hay más de un volcán en el mundo. Claramente aquí nos referimos al Vesuvio, que está cerca de Pompeya y que entró en erupción en el 79 D.C. En general para resolver estas referencias se necesita tanto una gran cantidad de razonamiento como una gran cantidad de conocimiento adicional.

6. Ningún mortal vive más de 150 años.

$\forall x: \forall t_1: \forall t_2: \text{mortal}(x) \wedge \text{nace}(x, t_1) \wedge \text{mayor}(t_2 - t_1, 150) \rightarrow \text{muerto}(x, t_2)$

Esta frase se podría expresar de diversas maneras. Por ejemplo, se podría definir una función edad y asertar que su valor nunca puede ser mayor de 150. Sin embargo, la representación escogida aquí es más sencilla y suficiente para este ejemplo.

7. Estamos en 2004.

**ahora = 2004**

Aquí se hace uso de la idea de que una cantidad se puede sustituir por otra de igual valor.

Supongamos ahora que se desea responder a la pregunta "**¿Está Marco vivo?**". Echando un rápido vistazo a los asertos es fácil llegar a la conclusión de que hay más de un modo posible de deducir una respuesta. Bien podemos demostrar que **Marco está muerto porque lo mató la erupción del volcán**, o bien podemos demostrar que **está muerto porque si no lo estuviera tendría más de 150 años**, lo cual sabemos que es imposible.

Pero en cuanto intentemos seguir rigurosamente alguna de estas dos líneas de razonamiento descubriremos, como en el ejemplo anterior, la **necesidad de utilizar conocimiento adicional**. Por ejemplo, en las sentencias se habla de la muerte, pero no se dice nada que la relacione con la vida, que es el concepto que aparece en la pregunta. Por tanto, es necesario añadir los siguientes hechos:

8. Estar vivo significa no estar muerto.

$$\forall x: \forall t: [\text{vivo}(x, t) \rightarrow \neg\text{muerto}(x, t)] \wedge [\neg\text{muerto}(x, t) \rightarrow \text{vivo}(x, t)]$$

Esto no es estrictamente correcto, pues el hecho de no estar muerto solo implica estar vivo si se refiere a objetos animados. (Las sillas no están ni vivas ni muertas). De nuevo, obviaremos este detalle por ahora. Es muy extraño que dos representaciones signifiquen exactamente lo mismo desde todos los puntos de vista.

9. Si alguien muere, está muerto para siempre.

$$\forall x: \forall t_1: \forall t_2: \text{murió}(x, t_1) \wedge \text{mayor}(t_2, t_1) \rightarrow \text{muerto}(x, t_2)$$

Esta representación expresa el hecho de que se está muerto todos los años a partir del año en que se murió. No se tiene en cuenta si se está muerto el año en que se murió.

- 1. hombre(Marco)
- 2. pompeyano(Marco)
- 3. nace(Marco, 40)
- 4.  $\forall x: \text{hombre}(x) \rightarrow \text{mortal}(x)$
- 5.  $\forall x: [\text{pompeyano}(x) \rightarrow \text{murió}(x, 79)]$
- 6. erupción(volcan, 79)
- 7.  $\forall x: \forall t_1: \forall t_2: \text{mortal}(x) \wedge \text{nace}(x, t_1) \wedge \text{mayor}(t_2 - t_1, 150) \rightarrow \text{muerto}(x, t_2)$
- 8. ahora = 2004
- 9.  $\forall x: \forall t: [\text{vivo}(x, t) \rightarrow \neg\text{muerto}(x, t)] \wedge [\neg\text{muerto}(x, t) \rightarrow \text{vivo}(x, t)]$
- 10.  $\forall x: \forall t_1: \forall t_2: \text{murió}(x, t_1) \wedge \text{mayor}(t_2, t_1) \rightarrow \text{muerto}(x, t_2)$

Figura 3.9

#### Un conjunto de hechos acerca de Marco

En la Figura 3.9 aparecen todos los hechos definidos hasta ahora. Intentemos ahora responder a la pregunta “¿Está Marco vivo?” buscando una demostración de:

$$\neg\text{vivo}(\text{Marco}, \text{ahora})$$

El término vacío que aparece al final de la Figura 3.10 indica que la lista de condiciones que queda por probar está vacía con lo cual la demostración ha terminado con éxito.

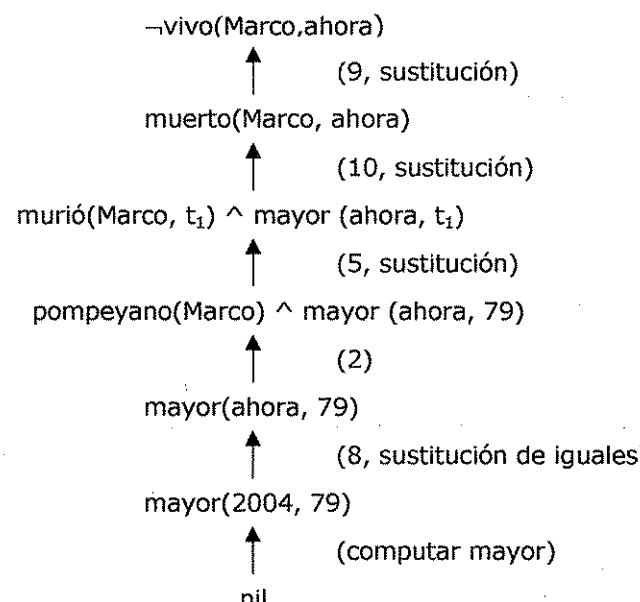


Figura 3.10

Una forma de demostrar que Marco está muerto

### 3.3.5. Método de Resolución

Desde el punto de vista computacional sería muy útil disponer de un *procedimiento de demostración* que en una sola operación llevase a cabo los distintos procesos involucrados en el razonamiento con sentencias de la lógica de predicados. La **Resolución** cumple este requisito y debe su eficiencia al hecho de que trabaja sobre *sentencias que han sido transformadas a una forma canónica*, descrita más abajo, muy conveniente.

**El procedimiento de resolución obtiene demostraciones por refutación. Es decir, para probar una proposición (demostrar su validez) se intenta demostrar que su negación lleva a una contradicción con las proposiciones conocidas (es decir, es insatisfacible).**

Esta aproximación difiere de la técnica utilizada hasta ahora, donde se intentaba generar la demostración de un teorema aplicando las reglas conocidas hasta llegar a alguno de los axiomas. La explicación del mecanismo de resolución será más directa una vez que se haya descrito la forma estandar en que se representan las proposiciones, y por lo tanto, la posponemos hasta entonces.

### 3.3.6. Conversión a forma clausal

Supongamos que sabemos que todos los romanos que conocen a Marco o bien odian a César o bien piensan que cualquiera que odie a otro está loco. Esta sentencia se puede representar con la siguiente fbf:

$$\forall x: [\text{romano}(x) \wedge \text{conoce}(x, \text{Marco})] \rightarrow [\text{odia}(x, \text{César}) \vee (\forall y: \exists z: \text{odia}(y, z) \rightarrow \text{cree\_loco}(x, y))]$$

El uso de esta fórmula en una demostración requeriría establecer unas correspondencias muy complicadas. Una vez que se haya establecido la correspondencia con una parte de ella, como por ejemplo  $\text{cree\_loco}(x, y)$ , será necesario realizar las operaciones adecuadas con el resto de la fórmula, tanto en la parte donde se incluye ese fragmento como en aquellas donde no.

Si la fórmula fuese más simple este proceso sería mucho más sencillo. Sería más sencillo trabajar con esta fórmula si:

- Fuera más plana, es decir, si hubiese menos anidamiento entre las expresiones.
- Los cuantificadores estuvieran separados de la fórmula, de modo que no fuera necesario tenerlos en consideración.

La **forma normalizada conjuntiva** (Davis y Putnam, 1960) **tiene ambas propiedades**. Por ejemplo, la fórmula dada anteriormente acerca de los sentimientos de los romanos que conocen a Marco se representaría en forma normalizada conjuntiva como:

$$\neg \text{romano}(x) \vee \neg \text{conoce}(x, \text{Marco}) \vee \\ \neg \text{odia}(x, \text{César}) \vee \neg \text{odia}(y, z) \vee \text{cree\_loco}(x, z)$$

Puesto que *existe un algoritmo para convertir cualquier fbf a la forma normalizada conjuntiva*, no perdemos la generalidad si empleamos un procedimiento de demostración (como la resolución) que opere solamente con fórmulas expresadas de esta forma.

De hecho, *para que la resolución funcione*, es necesario ir un poco más allá. *Es necesario convertir el conjunto de fórmulas bien formadas a un conjunto de cláusulas*, donde una **cláusula** se define como **una fbf en forma normalizada conjuntiva que no contiene ninguna conectiva ^**.

Para ello se convierte cada fbf a forma normalizada conjuntiva y, a continuación, se divide esa expresión en cláusulas, una por cada conjunción. Cuando se aplique el procedimiento de prueba, las conjunciones se considerarán agrupadas. Para transformar una fbf a forma clausal es necesario seguir los siguientes pasos:

### Algoritmo: Conversión a forma clausal

1. Eliminar las implicaciones,  $\rightarrow$ , usando el hecho de que  $a \rightarrow b$  es equivalente a  $\neg a \vee b$ .

Realizando esa transformación en las fbf dadas anteriormente obtenemos:

$$\forall x: \neg[\text{romano}(x) \wedge \text{conoce}(x, \text{Marco})] \vee$$

$$[\text{odia}(x, \text{César}) \vee (\forall y: \neg(\exists z: \text{odia}(y, z)) \vee \text{cree\_loco}(x, y))]$$

2. Reducir el ámbito de las negaciones,  $\neg$ , a un único término, usando el hecho de que  $\neg(\neg p) = p$ , las leyes de Morgan [según las cuales  $\neg(a \wedge b) = \neg a \vee \neg b$  y  $\neg(a \vee b) = \neg a \wedge \neg b$ ], y las correspondencias normales entre cuantificadores [ $\neg\forall x: P(x) = \exists x: \neg P(x)$  y  $\neg\exists x: P(x) = \forall x: \neg P(x)$ ]. Realizando esta transformación en las fbf del paso 1 se obtiene:

$$\forall x: [\neg\text{romano}(x) \vee \neg\text{conoce}(x, \text{Marco})] \vee$$

$$[\text{odia}(x, \text{César}) \vee (\forall y: \forall z: \neg\text{odia}(y, z) \vee \text{cree\_loco}(x, y))]$$

3. Normalizar las variables de forma que cada cuantificador esté ligado a una variable.

Puesto que las variables son solo nombres sin un valor concreto, este proceso no puede afectar al valor de verdad de la fbf. Por ejemplo, la fórmula

$$\forall x: P(x) \vee \forall x: Q(x)$$

se convertiría en

$$\forall x: P(x) \vee \forall y: Q(y)$$

Este paso es una preparación para el siguiente.

4. Mover todos los cuantificadores a la izquierda de la fórmula sin cambiar su orden relativo. Esto es posible gracias a que no existe ningún conflicto entre los nombres de las variables. Realizando esta operación sobre la fórmula del paso 2, se obtiene:

$$\forall x: \forall y: \forall z: [\neg\text{romano}(x) \vee \neg\text{conoce}(x, \text{Marco})] \vee$$

$$[\text{odia}(x, \text{César}) \vee (\neg\text{odia}(y, z) \vee \text{cree\_loco}(x, y))]$$

En este momento la fórmula es lo que se conoce como **fórmula normal prenex**.

*Consiste en un prefijo de cuantificadores seguido por una matriz que está libre de cuantificadores.*

5. Eliminar los cuantificadores existenciales. En una fórmula donde se incluye una variable cuantificada existencialmente se afirma que existe un valor que puede sustituir a la variable, y que hace verdadera la fórmula. Es posible eliminar el cuantificador sustituyendo la variable por una referencia a una función que produzca el valor deseado. Puesto que no se conoce necesariamente la forma de producir ese valor, se debe crear un nuevo nombre de función para cada sustitución. No se hace ninguna afirmación sobre esas funciones excepto que deben existir. Así, por ejemplo, la fórmula:

$$\exists y: \text{Presidente}(y)$$

puede transformarse en la fórmula

$$\text{Presidente}(\mathbf{S1})$$

Donde S1 es una función sin argumentos que produce de algún modo un valor que satisface el predicado Presidente.

Si surgen cuantificadores existenciales dentro del ámbito de cuantificadores universales, los valores que satisfagan el predicado pueden depender de los valores de las variables cuantificadas universalmente. Por ejemplo, en la fórmula:

$$\forall x: \exists y: \text{padre\_de}(y, x)$$

el valor de y que satisface `padre_de` depende del valor concreto de x. Por lo tanto, se deben generar funciones con el mismo número de argumentos que el número de cuantificadores universales que afecten a la expresión que se esté tratando. Así, este ejemplo se transformaría en:

$$\forall x: \text{padre\_de}(\mathbf{S2}(x), x)$$

Estas funciones que generamos se llaman **funciones de Skolem**. Aquellas que no tienen argumentos se llaman a veces **constantes de Skolem**.

6. Eliminar el prefijo. En este punto, *todas las variables que quedan están cuantificadas universalmente*, por lo que el prefijo puede ser simplemente ignorado, y cualquier procedimiento de demostración que usemos puede suponer simplemente que cualquier variable con la que se encuentre, está cuantificada universalmente. Ahora la fórmula producida en el paso 4 aparece como:

$$\begin{aligned} & [\neg \text{romano}(x) \vee \neg \text{conoce}(x, \text{Marco})] \vee \\ & \quad [\text{odia}(x, \text{César}) \vee (\neg \text{odia}(y, z)) \vee \text{cree_loco}(x, y)] \end{aligned}$$

7. Convertir la matriz en una conjunción de disyunciones. Como en este ejemplo no aparece ninguna conectiva Y, basta con utilizar la propiedad asociativa de la conectiva lógica O [es decir,  $a \vee (b \vee c) = (a \vee b) \vee c$ ] y quitar simplemente los paréntesis, para obtener:

$$\begin{aligned} & \neg \text{romano}(x) \vee \neg \text{conoce}(x, \text{Marco}) \vee \\ & \quad \text{odia}(x, \text{César}) \vee \neg \text{odia}(y, z) \vee \text{cree_loco}(x, y) \end{aligned}$$

Sin embargo, con frecuencia es también necesario utilizar la propiedad distributiva  $[(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)]$ . Por ejemplo, la fórmula:

$$(\text{invierno} \wedge \text{llevarbotas}) \vee (\text{verano} \wedge \text{llevarsandalias})$$

se convierte después de una aplicación de la regla en:

$$\begin{aligned} & [\text{invierno} \vee (\text{verano} \wedge \text{llevarsandalias})] \\ & \quad \wedge [\text{llevarbotas} \vee (\text{verano} \wedge \text{llevarsandalias})] \end{aligned}$$

y, después de una segunda aplicación, que es necesaria, porque aún quedan conjunciones unidas por la conectiva O, en

$$\begin{aligned} & (\text{invierno} \vee \text{verano}) \wedge \\ & (\text{invierno} \vee \text{llevarsandalias}) \wedge \\ & (\text{llevarbotas} \vee \text{verano}) \wedge \\ & (\text{llevarbotas} \vee \text{llevarsandalias}) \end{aligned}$$

8. Crear una cláusula por cada conjunción. Para que una fbf sea cierta, todas las cláusulas que se han generado a partir de ella deben ser ciertas. Cuando se trabaja con varias fórmulas bien formadas, es posible combinar el conjunto de cláusulas generadas por cada una de ellas para representar los mismos hechos que representaban las fórmulas originales.
9. Normalizar las variables que aparecen en el conjunto de cláusulas generadas en el paso 8. Con esto se pretende que no haya dos cláusulas que hagan referencia a la misma variable, para lo cual es necesario renombrar a las variables adecuadamente. Esta transformación se apoya en el hecho:

$$(\forall x: P(x) \wedge Q(x)) = \forall x: P(x) \wedge \forall x: Q(x)$$

Así, puesto que cada cláusula es una conjunción separada y todas las variables están cuantificadas universalmente, no es necesario dar valor a una variable cuantificada universalmente (es decir, sustituirla por un valor concreto). Pero, en general, queremos mantener las cláusulas en su forma más general durante tanto tiempo como sea posible. Al instanciar una variable, queremos conocer el mínimo número de sustituciones que deben realizarse para preservar el valor de verdad del sistema.

**Después de aplicar todo este procedimiento a un conjunto de fórmulas bien formadas tendremos un conjunto de cláusulas, cada una de las cuales será una disyunción de literales. Estas cláusulas serán las que utilice el procedimiento de resolución para generar demostraciones.**

### 3.3.7. Las bases de la resolución

**El procedimiento de resolución es un proceso iterativo simple en el cual, en cada paso, se comparan (resuelven) dos cláusulas llamadas cláusulas padres, produciendo una nueva cláusula que se ha inferido de ellas.**

La nueva cláusula representa la forma en que las dos cláusulas padres interaccionan entre sí. Supongamos que hay dos cláusulas en el sistema:

invierno v verano

$\neg$ invierno v frío

Recordemos que esto significa que ambas cláusulas deben ser ciertas (es decir, aunque las cláusulas parecen independientes, en realidad están agrupadas).

Ahora vemos que siempre deberá ser cierta una de las aserciones invierno o  $\neg$ invierno. Si invierno es cierto, entonces frío debe ser cierto para garantizar la verdad de la segunda cláusula. Si  $\neg$ invierno es verdad, entonces verano debe ser cierto para garantizar la verdad de la primera cláusula. Así pues, a partir de esas dos cláusulas se puede deducir:

**verano v frío**

*Esta es la deducción que hará el procedimiento de resolución.*

- La resolución opera tomando dos cláusulas tales que cada una contenga el mismo literal, en este ejemplo, invierno. **El literal debe estar en forma positiva en una cláusula y en forma negativa en la otra.**
- El resolvente se obtiene combinando todos los literales de las dos cláusulas padres excepto aquellos que se cancelan.
- Si la cláusula producida es la cláusula vacía, es que se ha encontrado una contradicción.

Por ejemplo, las dos cláusulas:

invierno

$\neg$ invierno

producirán la cláusula vacía. Si existe una contradicción, se encontrará en algún momento. Naturalmente, si no existe ninguna contradicción, es posible que el procedimiento nunca termine, aunque como veremos, a menudo existen formas de detectar que no existe contradicción.

### 3.3.8. Resolución en lógica proposicional

Para aclarar como funciona la resolución, presentaremos en primer lugar el procedimiento de resolución para *lógica proposicional*. Posteriormente lo expandiremos para incluir la lógica de predicados.

En *lógica proposicional*, el procedimiento para producir una demostración por resolución de la proposición P respecto a un conjunto de axiomas F es el siguiente:

#### Algoritmo: Resolución de proposiciones

1. Convertir todas las proposiciones de F a forma clausal.
2. Negar P y convertir el resultado a forma clausal. Añadir la cláusula resultante al conjunto de cláusulas obtenidas en el paso 1.
3. Hasta que se encuentre una contradicción o no se pueda seguir avanzando repetir:
  - a. Seleccionar dos cláusulas. Llamarlas cláusulas padres.
  - b. Resolverlas juntas. La cláusula resultante, llamada **resolvente**, será la disyunción de todos los literales de las cláusulas padres con la siguiente excepción: si existen pares de literales L y  $\neg$ L de forma que una de las cláusulas padre contenga L y la otra contenga  $\neg$ L, entonces se eliminarán tanto L como  $\neg$ L del resolvente.

- c. Si el resolvente es la cláusula vacía, es que se ha encontrado una contradicción. Si no lo es, añadirla al conjunto de cláusulas disponibles.

**Veamos un ejemplo sencillo.** Supongamos que se nos dan los axiomas mostrados en la primera columna de la Figura 3.11 y queremos demostrar R. En primer lugar, se convierten los axiomas a forma clausal, tal como se muestra en la segunda columna de la figura. A continuación negamos R, obteniendo  $\neg R$  que ya está en forma clausal. A continuación se empieza a elegir pares de cláusulas para resolverlas.

Axiomas de partida	Convertidos a forma clausal	
P	P	(1)
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$	(2)
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$	(3)
T	$\neg T \vee Q$	(4)
	T	(5)

Figura 3.11

Aunque puede resolverse cualquier par de cláusulas, solamente aquellos pares que contengan literales complementarios, producirán un resolvente que tenga alguna probabilidad de conducir al objetivo de obtener la cláusula vacía (que se representa como una caja). Es posible, por ejemplo, generar la secuencia de resolventes que se muestra en la Figura 3.12. Se empieza resolviendo con la cláusula  $\neg R$ , puesto que es una de las cláusulas que deben estar involucradas en la contradicción que estamos intentando hallar.

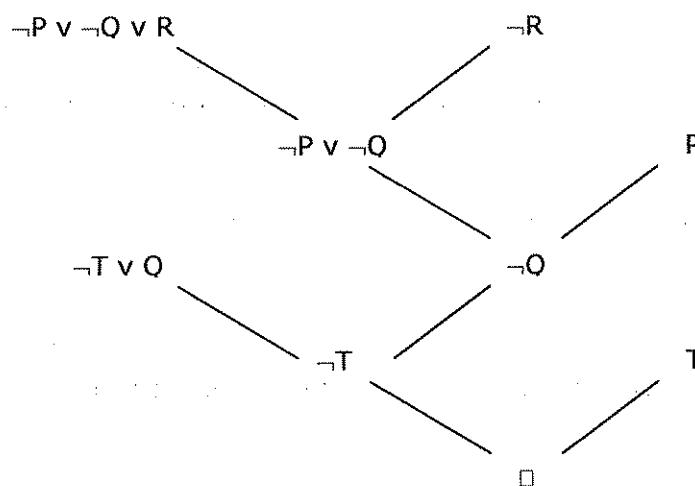


Figura 3.12

La resolución se puede ver como un proceso que toma un conjunto de cláusulas que se suponen ciertas y, basándose en la información que las otras proporcionan, genera nuevas cláusulas que representan restricciones a la satisfacibilidad de las cláusulas originales. Surge una contradicción cuando una cláusula se vuelve tan restringida que no hay forma de hacerla verdadera. Esto se indica con la generación de la cláusula vacía.

### 3.3.9. El algoritmo de unificación

En lógica proposicional es fácil determinar que dos literales no pueden ser ciertos al mismo tiempo. Basta con buscar  $L$  y  $\neg L$ .

En **lógica de predicados** el proceso de correspondencia es más complicado puesto que **se deben considerar los argumentos de los predicados**.

Por ejemplo,  $\text{hombre}(\text{John})$  y  $\neg \text{hombre}(\text{John})$  es una contradicción, mientras que  $\text{hombre}(\text{John})$  y  $\neg \text{hombre}(\text{Spot})$  no lo es. Así pues, para detectar las contradicciones, se necesita un procedimiento de emparejamiento que compare dos literales y descubra si existe un conjunto de sustituciones que los haga idénticos.

*Existe un procedimiento recursivo directo, denominado **algoritmo de unificación** que realiza exactamente esto.*

La idea básica de la unificación es muy sencilla.

- Para unificar dos literales, en primer lugar se comprueba si los predicados coinciden. Si es así, seguimos adelante, si no, no hay forma de unificarlos, sean cuales sean sus argumentos. Por ejemplo, los literales:

intenta\_asesinar(Marco, César)

odia(Marco,César)

no son unificables.

- Si los predicados concuerdan, a continuación se van comprobando los argumentos de dos en dos. Si el primero concuerda, podemos continuar con el segundo, y así sucesivamente. Para comprobar cada par de argumentos, basta con llamar recursivamente al procedimiento de unificación. Las reglas de emparejamiento son sencillas. Aquellos predicados o constantes que sean diferentes no se pueden emparejar; aquellos que sean idénticos pueden hacerlo.

Una *variable* se puede emparejar con otra *variable*, con *cualquier constante* o con *un predicado*, con la restricción de que el predicado no debe contener ninguna instancia de la variable con la que se está emparejando.

La única complicación en este procedimiento es que se debe encontrar una única sustitución consistente para todo el literal, y no sustituciones separadas para cada parte del mismo. Para hacerlo, se debe tomar cada sustitución que encontramos y aplicarla al resto de literales antes de continuar intentando unificarlos.

Por ejemplo, supongamos que queremos unificar las expresiones:

$P(x,x)$

$P(y,z)$

Las dos ocurrencias de  $P$  se emparejan sin problema. A continuación comparamos  $x$  e  $y$ , y decidimos que si sustituimos  $y$  por  $x$ , se pueden emparejar. Escribiremos esa sustitución como

$y/x$

(Naturalmente, se podría haber sustituido  $x$  por  $y$ , puesto que ambos son nombres de variables sin más significado. El algoritmo simplemente elegirá una de las dos sustituciones). Pero ahora, si continuamos y emparejamos  $x$  con  $z$ , obtendremos la sustitución  $z/x$ . Pero no es posible sustituir, a la vez,  $y$  y  $z$  por  $x$ , por lo que la sustitución no es consistente.

Una vez encontrada la primera sustitución  $y/x$  es necesario realizar dicha sustitución en el resto de literales, para obtener:

$P(y,y)$

$P(y,z)$

A continuación intentamos unificar los argumentos y y z, lo que logramos con la sustitución z/y. Con esto se completa el proceso y la sustitución resultante es la composición de las dos sustituciones encontradas. La composición se escribe como:

$$(z/y)(y/x)$$

siguiendo la notación estandar para la composición de funciones.

El **objetivo del procedimiento de unificación** es descubrir, al menos una sustitución que permita el emparejamiento de dos literales. Normalmente, si existe una de dichas sustituciones, existirán muchas más. Por ejemplo, los literales:

$$\text{odia}(x,y)$$

$$\text{odia}(\text{Marco}, z)$$

Pueden unificarse con cualquiera de las siguientes sustituciones:

$$(\text{Marco}/x, z/y)$$

$$(\text{Marco}/x, y/z)$$

$$(\text{Marco}/x, \text{César}/y, \text{César}/z)$$

$$(\text{Marco}/x, \text{Polonio}/y, \text{Polonio}/z)$$

Las dos primeras son equivalentes, excepto por la diferencia léxica. Pero las dos segundas, aunque producen un emparejamiento, producen también una sustitución que es más restrictiva de lo que sería estrictamente necesario para el emparejamiento.

Puesto que la sustitución final producida por el proceso de unificación será usada por el procedimiento de resolución, sería conveniente generar el unificador más general posible.

### 3.3.10. Resolución en lógica de predicados

Ahora disponemos de una forma sencilla para determinar si dos literales son **contradicторios** – *lo son si uno de ellos puede unificarse con la negación del otro*.

Así por ejemplo, hombre(x) y  $\neg\text{hombre}(\text{Spot})$  son contradictorios, puesto que hombre(x) y hombre(Spot) son unificables. Esto corresponde a la idea intuitiva que dice que hombre(x) no es cierto para cualquier valor de x si se conoce la existencia de algún x, por ejemplo Spot, para el cual es falso hombre(x). Así, para usar la resolución sobre expresiones de la lógica de predicados, se utiliza el algoritmo de unificación para localizar partes de literales que se cancelen mutuamente.

También es necesario utilizar el unificador obtenido mediante el algoritmo de unificación para generar la cláusula resolvente. Por ejemplo, supongamos que queremos resolver las dos cláusulas:

1. hombre(Marco)
2.  $\neg\text{hombre}(x_1) \vee \text{mortal}(x_1)$

El literal hombre(Marco) se puede unificar con el literal hombre( $x_1$ ) con la sustitución Marco/  $x_1$ , que nos dice que para  $x_1 = \text{Marco}$ ,  $\neg\text{hombre}(\text{Marco})$  es falso. Pero no podemos cancelar simplemente las dos apariciones del literal hombre tal como hacíamos en lógica proposicional, generando el resolvente mortal( $x_1$ ).

La cláusula 2 dice que para un  $x_1$  dado, se cumple  $\neg\text{hombre}(x_1) \vee \text{mortal}(x_1)$ . Por lo cual, para que sea cierta, tan solo podemos afirmar que mortal(Marco) debe ser cierto. No es necesario que mortal( $x_1$ ) sea cierto para cualquier  $x_1$ , puesto que para algunos valores de  $x_1$ ,  $\neg\text{hombre}(x_1)$  podría ser cierto, haciendo que mortal( $x_1$ ) sea irrelevante para la certeza de la cláusula completa.

Por lo tanto, el resolvente generado a partir de las cláusulas 1 y 2 debe ser mortal(Marco), que se obtiene aplicando el resultado del proceso de unificación al resolvente. El proceso de resolución puede continuar a partir de ahí para determinar si mortal(Marco) conduce a una contradicción con otras cláusulas disponibles.

Este ejemplo ilustra la importancia de normalizar separadamente las variables durante el proceso de conversión de las expresiones a forma clausal. Suponiendo que se haya hecho la normalización, es fácil determinar como debe usarse el unificador para realizar las sustituciones que den lugar al resolvente. Si hay dos ocurrencias de la misma variable, se les debe aplicar la misma sustitución.

Ahora ya podemos exponer el algoritmo de resolución para lógica de predicados suponiendo un conjunto de sentencias F y una sentencia P que debemos demostrar:

### **Algoritmo: Resolución**

1. Convertir todas las sentencias de F a forma clausal.
2. Negar P y convertir el resultado a forma clausal. Añadirlo al conjunto de cláusulas obtenidas en 1.
3. Hasta que se encuentre una contradicción o no pueda realizarse ningún progreso o se haya realizado una cantidad de esfuerzo predeterminada, repetir:
  - a. Seleccionar dos cláusulas. Llamarlas cláusulas padres.
  - b. Resolverlas. El resolvente será la disyunción de todos los literales de ambas cláusulas padres, una vez realizadas las sustituciones apropiadas, con la siguiente excepción: si existe un par de literales T1 y  $\neg T_2$  tales que una de las cláusulas padres contenga T1 y la otra contenga  $T_2$  y si T1 y  $T_2$  son unificables, entonces ni T1 ni  $T_2$  deben aparecer en el resolvente. Llamaremos a T1 y  $T_2$  literales complementarios. A continuación utilizar la sustitución producida por la unificación para crear el resolvente. Si existe más de una pareja de literales complementarios, en el resolvente solo se eliminará uno de ellos.
  - c. Si el resolvente es la cláusula vacía, se ha encontrado una contradicción. Si no lo es, añadirla al conjunto de cláusulas sobre las que se está aplicando el procedimiento.

Si la elección de cláusulas a resolver en cada paso se hace siguiendo unas ciertas formas sistemáticas, el procedimiento de resolución encontrará una contradicción, si ésta existe.

Volvamos a nuestra discusión sobre Marco y veamos como se puede utilizar la resolución para demostrar nuevas cosas acerca de él. Consideraremos en primer lugar el conjunto de sentencias del ejemplo anterior convertidas a forma clausal.

1. hombre(Marco)
2. pompeyano(Marco)
3.  $\neg \text{pompeyano}(x_1) \vee \text{romano}(x_1)$
4. gobernante(César)
5.  $\neg \text{romano}(x_2) \vee \text{leal}(x_2, \text{César}) \vee \text{odia}(x_2, \text{César})$
6.  $\text{leal}(x_3, f_1(x_3))$
7.  $\neg \text{hombre}(x_4) \vee \neg \text{gobernante}(y_1) \vee \neg \text{intenta\_asesinar}(x_4, y_1) \vee \neg \text{leal}(x_4, y_1)$
8.  $\text{intenta\_asesinar}(\text{Marco}, \text{César})$

En la Figura 3.13 se muestra una demostración por resolución de la sentencia:

**odia(Marco,César)**

Naturalmente, podrían haberse generado muchos más resolventes de los que aparecen en la figura, pero se han utilizado las técnicas heurísticas descritas anteriormente para guiar la búsqueda. Nótese que lo que aquí se ha hecho es, en esencia, razonar hacia atrás desde la sentencia que queremos demostrar que es una contradicción, a través de un conjunto de conclusiones intermedias hasta la conclusión final de inconsistencia.

Supongamos que el verdadero objetivo al demostrar el aserto:

odia(Marco, César)

era responder a la pregunta "*¿Odiaba Marco a César?*". En este caso, podríamos haber intentado igualmente demostrar la sentencia:

$\neg$ odia(Marco, César)

Para lo cual, se debería añadir:

odia(Marco, César)

al conjunto de cláusulas disponibles antes de iniciar el proceso de resolución. Pero notamos inmediatamente que no existe ninguna cláusula que contenga un literal de la forma  $\neg$ odia. Puesto que el proceso de resolución solo puede generar nuevas cláusulas formadas por combinaciones de los literales de cláusulas ya existentes, sabemos que no puede generarse tal cláusula, por lo que concluimos que odia(Marco, César) no producirá una contradicción con los asertos conocidos. Este es un ejemplo del tipo de situaciones en las que el procedimiento de resolución puede detectar que no existe contradicción.

**Demostrar:** odia(Marco, César)

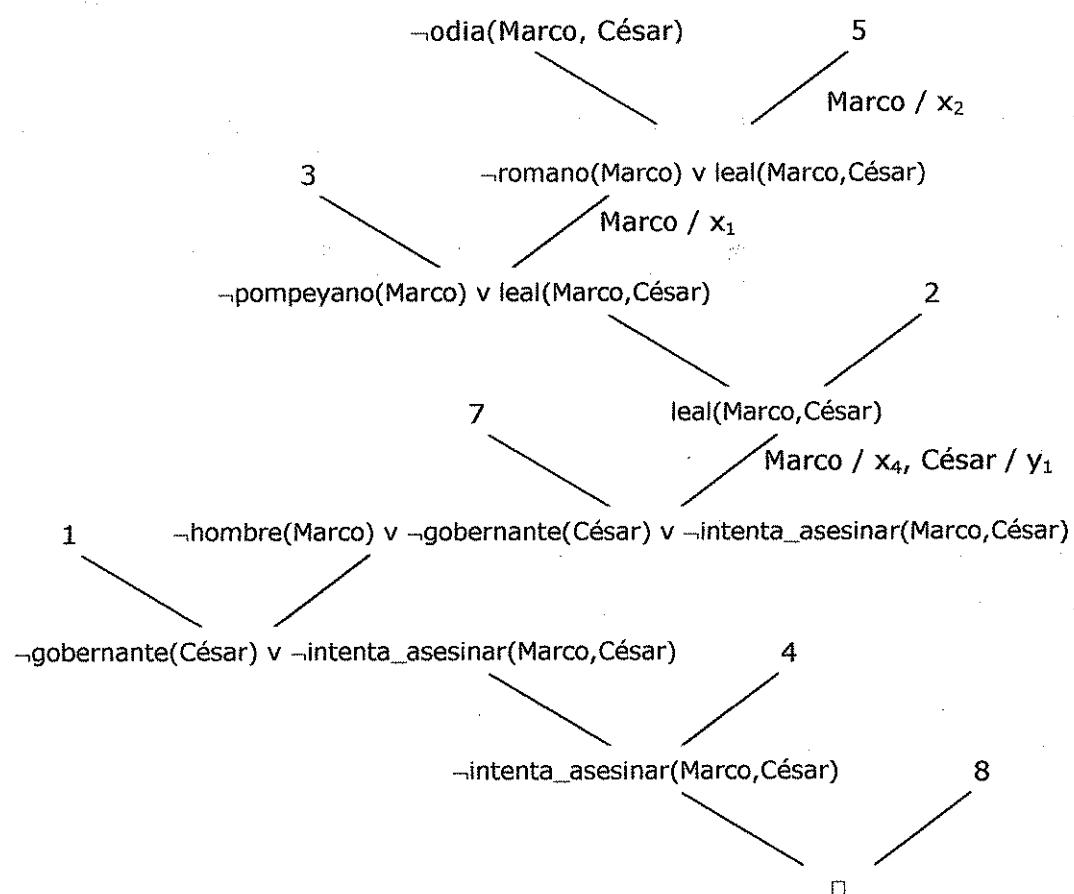


Figura 3.13

Demostración por Resolución

### 3.4. Representación del conocimiento mediante Reglas

Hablaremos del uso de reglas para codificar el conocimiento. Esto representa un campo de estudio verdaderamente importante, ya que los sistemas de razonamiento basados en reglas desempeñan un importante papel en la evolución de la Inteligencia Artificial, *transformándola de una ciencia de laboratorio en una significativa ciencia comercial*.

Hasta ahora se ha hablado de las reglas como la base de los programas de búsqueda. Pero ya se hizo hincapié en el modo en que el conocimiento acerca del mundo estaba representado en las reglas. En particular se ha estado considerando que el conocimiento de control de la búsqueda se encontraba completamente separado de las propias reglas.

Esto no se seguirá considerando así, ya que a partir de ahora se considerará *un conjunto de reglas para representar tanto el conocimiento acerca de las relaciones en el mundo, como el conocimiento para resolver problemas utilizando el contenido de las reglas*.

#### 3.4.1. Comparación entre conocimiento procedimental y conocimiento declarativo

Ya que el estudio de la representación del conocimiento se ha centrado hasta ahora en el uso de aserciones lógicas, se utilizará la lógica como punto de partida en el estudio de los sistemas basados en reglas.

**Una representación declarativa**, es aquella en la que el conocimiento está especificado, pero en la manera en que dicho conocimiento debe ser usado no viene dado.

Para utilizar una representación declarativa se debe aumentar ésta con un programa que especifique lo que debe hacerse con el conocimiento y de qué modo debe hacerse.

Por ejemplo, *un conjunto de aserciones lógicas se puede combinar con un demostrador de teoremas por resolución, para dar lugar a un programa completo de resolución de problemas*.

Existe un modo diferente de considerar las aserciones lógicas, *considerarlas como un programa*, en lugar de cómo datos para un programa. Desde este punto de vista, *las sentencias de implicación definen los caminos de razonamiento legitimado*, así como las aserciones atómicas proporcionan los puntos de partida (o, si el razonamiento se realiza hacia atrás, los puntos de finalización) de dichos caminos. Estos caminos de razonamiento definen construcciones tradicionales de control, como if-then-else, definen los caminos de ejecución de los programas tradicionales.

En otras palabras, se pueden considerar las aserciones lógicas como representaciones procedimentales del conocimiento.

**Una representación procedimental**, es aquella en la que la información de control necesaria para utilizar el conocimiento se encuentra embebida en el propio conocimiento.

Para utilizar una representación procedimental se necesita aumentarla con un intérprete que siga las instrucciones dadas por el conocimiento.

Realmente, considerar las aserciones lógicas como código no es una idea excesivamente novedosa, dado que todos los programas constituyen realmente los datos para otros programas que los interpretan (o compilan) y los ejecutan.

**La verdadera diferencia entre el punto de vista declarativo del conocimiento y el procedimental** radica en *dónde se encuentra la información de control*.

Por ejemplo, considérese la siguiente base de conocimiento:

hombre(Marco Antonio)  
 hombre(César)  
 persona(Cleopatra)  
 $\forall x: \text{hombre}(x) \rightarrow \text{persona}(x)$

Ahora imagínese que se intenta extraer de esta base de conocimiento la respuesta a la siguiente cuestión:

$\exists y: \text{persona}(y)$

Se desea ligar y con un valor particular para el cual persona es verdad. La base de conocimiento que tenemos justifica alguna de las siguientes respuestas:

$y = \text{Marco Antonio}$   
 $y = \text{César}$   
 $y = \text{Cleopatra}$

Ya que existe más de un valor que satisface el predicado, y que solo se necesita un valor, la respuesta a dicha pregunta dependerá del orden en que se examinen las diferentes aserciones durante la búsqueda de una respuesta.

**Si se consideran las aserciones como declarativas**, éstas no dirán, por sí mismas, nada acerca de cómo van a ser examinadas. Si por el contrario se les considera como **procedimentales**, sí lo harán.

Por supuesto, los programas no deterministas son posibles (por ejemplo las construcciones de programación concurrente y paralela). Por tanto, se pueden considerar estas aserciones como un programa no determinista cuya salida simplemente no está definida. En caso de hacer esto se consigue una representación "procedimental", que realmente no contiene más información que la que contiene la forma "declarativa".

Pero la mayoría de los **sistemas que consideran el conocimiento como procedimental**, no hacen esto en realidad. La razón es que si al menos el procedimiento se va a ejecutar o en una máquina secuencial o bien sobre alguna de las máquinas paralelas más conocidas, deben tomarse algunas decisiones en relación con el **orden en que se examinarán las aserciones**.

No existe hardware disponible que trate la aleatoriedad. Por tanto, si el intérprete debe disponer de un modo para decidir, no existe una razón real para no especificarlo como parte de la definición del lenguaje y así poder definir el significado de un programa cualquiera en dicho lenguaje.

Por ejemplo, se puede especificar que *las aserciones serán examinadas en el orden en que aparecen en el programa, y que la búsqueda se realizará primero en profundidad*, lo que quiere decir que si se establece un nuevo subobjetivo, se convertirá en el objetivo prioritario inmediatamente, mientras que otros caminos únicamente serán examinados si el nuevo objetivo falla. Al hacer esto, las aserciones que se dieron anteriormente describen un programa que dará como respuesta:

$y = \text{Cleopatra}$

Para ver claramente las diferencias entre las representaciones declarativas y las procedimentales, considérese las siguientes aserciones:

$\text{hombre}(\text{Marco Antonio})$   
 $\text{hombre}(\text{César})$   
 $\forall x: \text{hombre}(x) \rightarrow \text{persona}(x)$   
 $\text{persona}(\text{Cleopatra})$

Observándolas desde un *punto de vista declarativo*, forman la misma base de conocimiento que se tenía anteriormente. El sistema da las mismas respuestas y ninguna de ellas se selecciona explícitamente.

Pero considerándolo **procedimentalmente**, y utilizando el modelo de control que se utilizó para obtener Cleopatra como respuesta, se observa que **ésta es una base de conocimiento diferente**, ya que ahora la respuesta a nuestra pregunta es Marco Antonio. Esto ocurre porque la primera afirmación que se alcanza con el objetivo de persona es la regla de inferencia  $\forall x: \text{hombre}(x) \rightarrow \text{persona}(x)$ . Esta regla establece un subobjetivo que consiste en encontrar un hombre. De nuevo, las afirmaciones son examinadas desde el principio, pero ahora Marco Antonio, satisface el subobjetivo, y por tanto también el objetivo principal. Así que Marco Antonio será la respuesta emitida.

Es importante recordar que, aunque ya se ha dicho que una representación procedural codifica la información de control en la base de conocimiento, esto solo ocurre en el caso en que el intérprete de la base de conocimiento reconozca dicha información de control. Por lo tanto, se pueden haber obtenido diferentes respuestas a la pregunta persona, dejando la base de conocimiento original intacta y cambiando el intérprete de forma que examine las afirmaciones empezando por la última hasta la primera. Siguiendo este esquema de control. César será la respuesta correcta.

Siempre ha existido una gran controversia en la IA acerca de si las estructuras de representación del conocimiento declarativas son mejores o peores que las procedimentales.

Realmente no se ha llegado a una conclusión clara para esta cuestión. Como se observa en el estudio que se ha hecho, la distinción entre las dos formas es a menudo muy confusa. En lugar de intentar responder cual de los dos enfoques es mejor, lo que se intentará es describir el modo en que los formalismos de reglas y los intérpretes se pueden combinar para solucionar problemas.

### 3.4.2. Programación Lógica

**La programación lógica es un paradigma de los lenguajes de programación**, en el cual **las aserciones lógicas se consideran como programas**, como ya se describió en el apartado anterior.

Actualmente se utilizan diferentes sistemas de programación lógica, el más popular de ellos es el **PROLOG**. Un programa PROLOG está descrito como **una serie de aserciones lógicas**, cada una de las cuales es una **cláusula de Horn**.

**Una cláusula de Horn es una cláusula que tiene, como mucho, un literal positivo.**

Así  $p, \neg p \vee q, p \rightarrow q$  son cláusulas de Horn. La última realmente no parece una cláusula y además parece que tiene dos literales positivos. Pero recuerde que esto se puede convertir en  $\neg p \vee q$ , lo que constituye una cláusula de Horn bien formada.

Como se verá más adelante; cuando las cláusulas de Horn están escritas en los programas PROLOG, realmente se parecen más a la primera forma (una implicación con, como mucho, un literal a la derecha del signo de implicación), que a la cláusula que realmente se ha producido.

La cuestión es que los programas PROLOG están compuestos solo por cláusulas de Horn, y no por expresiones lógicas arbitrarias, lo que da lugar a dos importantes consecuencias.

1. La primera es que, debido a la representación uniforme, se puede escribir un sencillo y potente intérprete que resulte eficaz.
2. La segunda consecuencia es, si cabe, más importante, ya que la **lógica de los sistemas de cláusulas de Horn es decidible** (al contrario de la lógica de predicados de primer orden completa).

La estructura de control que el intérprete PROLOG impone en un programa PROLOG, es la misma que se utilizó al principio de este capítulo para encontrar las respuestas Cleopatra y Marco Antonio.

La entrada de un programa es un objetivo que debe ser probado. Se aplica un *razonamiento hacia atrás* para intentar demostrar el objetivo dadas las aserciones del programa.

El programa se lee de arriba hacia abajo y de izquierda a derecha, y la búsqueda será primero en profundidad con vuelta atrás.

```

 $\forall x: \text{mascota}(x) \wedge \text{pequeño}(x) \rightarrow \text{animal\_de\_compañía}(x)$ 
 $\forall x: \text{gato}(x) \vee \text{perro}(x) \rightarrow \text{mascota}(x)$ 
 $\forall x: \text{caniche}(x) \rightarrow \text{perro}(x) \wedge \text{pequeño}(x)$ 
 $\text{caniche(peluso)}$ 

```

### Representación lógica

```

animal_de_compañía(X) :- mascota(X), pequeño(X).
mascota(X) :- gato(X).
mascota(X) :- perro(X).
perro(X) :- caniche(X).
pequeño(X) :- caniche(X).
caniche(peluso).

```

### Representación en PROLOG

Figura 3.14

La Figura 3.14 muestra un ejemplo de una sencilla base de conocimiento representada en una *notación lógica estándar*, y después en *PROLOG*. Ambas representaciones contienen dos tipos de afirmaciones denominadas **hechos**, que únicamente contienen constantes (es decir sin variables) y **reglas** que contienen variables.

- Los hechos representan sentencias acerca de objetos específicos.
- Las reglas representan sentencias acerca de las diferentes clases de objetos.

Obsérvese que existen algunas diferencias sintácticas superficiales entre las representaciones lógicas y las representaciones PROLOG, como por ejemplo:

1. En la lógica, las variables están específicamente cuantificadas. En PROLOG, la cuantificación se realiza de un modo implícito por la forma en que las variables son interpretadas. En PROLOG se hace que todas las variables comiencen con letras mayúsculas, y que todas las constantes comiencen con letras minúsculas o números.
2. En la lógica existen símbolos explícitos para "y" (^) y "o" (v). En PROLOG existe un símbolo explícito para y (,), pero no existe un símbolo explícito para o. En lugar de esto, la disyunción se expresa mediante una lista de sentencias alternativas, y cualquiera de ellas puede proporcionar una base para una conclusión.
3. En la lógica, las implicaciones de la forma "p implica q" se escriben como  $p \rightarrow q$ . En PROLOG, la misma implicación se escribe "hacia atrás", como : q :- p. Esta forma es natural en PROLOG, ya que el intérprete siempre trabaja hacia atrás sobre un objetivo, lo que da lugar a que cada regla comience con el componente que debe emparejarse en primer caso. Este primer componente se llama cabeza de la regla.

La principal **diferencia entre la representación de la lógica y la de PROLOG**, es que **el intérprete PROLOG fija una estrategia de control**, y por tanto, las aserciones en un programa PROLOG definen un camino de búsqueda concreto para así encontrar una respuesta a cualquier pregunta.

En contraste con esto, las aserciones lógicas definen únicamente el conjunto de respuestas que las justifica; por sí mismas no dicen nada sobre como se debe elegir entre todas las respuestas, si es que existe más de una.

La estrategia de control PROLOG básica, que se acaba de sugerir, es simple. Comienza con una sentencia problema, que en este caso es considerado como el objetivo a probar; busca las aserciones que pueden probar el objetivo; considera hechos que prueban el objetivo directamente, y además considera cualquier regla cuya cabeza se empareje con el objetivo. Para decidir cuando puede aplicar una regla o un hecho al problema que le ocupa utiliza el **procedimiento de unificación estándar**. Razonará hacia atrás desde ese objetivo, hasta

que encuentre el modo de terminar con las aserciones en el programa. Considera los caminos utilizando la **estrategia de la búsqueda primero en profundidad**, así como utilizando la **vuelta atrás**. En cada punto en el que debe elegir, considera las opciones siguiendo el orden en que éstas aparecen en el programa. Si el objetivo tiene más de una parte de conjuntiva, comprueba las partes en el orden en que éstas aparecen, propagando los enlaces de las variables que determinó la unificación.

### 3.4.3. Diferencia entre Razonamientos hacia delante y hacia atrás

El objeto de un procedimiento de búsqueda es descubrir un camino a través de un espacio problema partiendo de un estado inicial y finalizando en un estado objetivo. Mientras que PROLOG únicamente efectúa la búsqueda a partir de un estado objetivo en realidad existen dos direcciones hacia las que se puede dirigir dicha búsqueda:

- Hacia delante, a partir de los estados iniciales.
- Hacia atrás, partiendo de los estados objetivos.

Razonamiento hacia delante a partir de los estados iniciales. Se comienza por construir un árbol de secuencias de movimientos que se pueden presentar como soluciones, empezando por la configuración inicial en la raíz del árbol. Se generará el siguiente nivel del árbol encontrando todas las reglas cuyos lados izquierdos se relacionen con el nodo raíz, y que utilicen sus lados derechos para crear nuevas configuraciones. Se creará el siguiente nivel tomando cada nodo que se haya generado en el nivel anterior, y aplicándolo a todas las reglas cuyos lados izquierdos se relacionen con éste. Se continuará así hasta que se consiga una configuración que se empareje con el estado objetivo.

Razonamiento hacia atrás a partir de los estados objetivo. Se comienza construyendo un árbol de secuencias de movimientos que ofrezcan soluciones empezando con la configuración objetivo en la raíz del árbol. Se generará el siguiente nivel del árbol encontrando todas las reglas cuyos lados derechos estén ligadas con el nodo raíz. Estas serán todas las reglas que, si son las únicas que se aplican, generarán el estado que se desea. Se utilizará el lado izquierdo de las reglas para generar los nodos en este segundo nivel del árbol. Se generará el siguiente nivel del árbol tomando cada nodo del nivel previo, y encontrando todas las reglas cuyo lado derecho esté ligado con éste. Entonces se utilizarán los correspondientes lados izquierdos para generar los nuevos nodos. Se continuará hasta que se genere un nodo que se empareja con el estado inicial. Este método de razonamiento hacia atrás, a partir del estado final deseado, se denomina algunas veces **razonamiento dirigido al objetivo**.

#### 3.4.3.1. Sistemas de reglas encadenadas hacia atrás

Los sistemas de reglas encadenadas hacia atrás, de los cuales el PROLOG es un ejemplo, resultan muy eficaces para la **resolución de problemas dirigidos al objetivo**.

Por ejemplo, un sistema interrogador probablemente utilizará un encadenamiento hacia atrás para razonar acerca de las respuestas a las preguntas del usuario.

En el PROLOG, las reglas están restringidas a ser cláusulas de Horn. Esto permite crear rápidamente un índice, ya que todas las reglas que se utilizan para deducir un hecho dado, comparten la misma cabeza de regla. Las reglas se emparejan a través del procedimiento de unificación. La unificación intenta encontrar un conjunto de restricciones para las variables, y así igualar un subobjetivo con las cabezas de algunas reglas. Las reglas, en un programa PROLOG se emparejan en el orden en el que aparecen.

Otros sistemas de encadenamiento hacia atrás permiten reglas más complejas. En MYCIN, por ejemplo, las reglas pueden ser aumentadas con factores de certeza probabilísticos para reflejar el hecho de que unas reglas son más fiables que otras.

### 3.4.3.2. Sistemas de reglas encadenadas hacia delante

En lugar de dirigirse por objetivos, algunas veces se desea ser *dirigido por la información que se va incorporando*. Por ejemplo, supóngase que sentimos calor cerca de nuestra mano. Seguramente se tenderá a retirar la mano de ahí. Mientras que esto se puede considerar como un comportamiento dirigido a un objetivo, se modela de un modo más natural por medio de ciclos de reconocimiento de actos, característicos de los sistemas de reglas encadenadas hacia adelante. En algunos de estos sistemas, los lados izquierdos de las reglas se emparejan con la descripción del estado. Las reglas que se emparejan vuelcan sus aserciones de su parte derecha en el estado, y así el proceso se repite sucesivamente.

### 3.4.3.3. Combinación del razonamiento hacia adelante y hacia atrás

A veces, determinados aspectos de un problema se manejan más fácilmente utilizando el encadenamiento hacia adelante, mientras que otros se solucionan de un modo más sencillo utilizando el encadenamiento hacia atrás.

Considérese por ejemplo un programa de diagnósticos médicos basado en el encadenamiento hacia adelante. Este puede aceptar, aproximadamente, una veintena de hechos acerca de la condición del paciente, entonces trabajará hacia adelante con dichos hechos para intentar deducir la naturaleza y la causa de la enfermedad.

Ahora supóngase que en un momento dado el lado izquierdo de una de las reglas esté "casi satisfecho", por ejemplo si nueve de las diez precondiciones que tuviera fuesen ya conocidas, en este caso resultaría mejor aplicar un razonamiento hacia atrás para satisfacer la décima precondición de un modo directo, en lugar de esperar a que el encadenamiento hacia adelante sustituya al hecho por accidente. También puede ocurrir que la décima condición requiera más pruebas médicas. En ese caso el encadenamiento hacia atrás se puede utilizar para interrogar al usuario.

Saber si es posible utilizar las mismas reglas tanto para el razonamiento hacia adelante, como para el razonamiento hacia atrás, también depende de la propia forma de las reglas.

- Si, tanto las partes derechas como las izquierdas contienen aserciones puras, entonces el encadenamiento hacia adelante puede emparejar las aserciones en el lado izquierdo de una regla, y añadir a la descripción del estado las aserciones del lado derecho.
- Pero si se permiten los procedimientos arbitrarios como partes derechas de las reglas, entonces las reglas no serán reversibles. Algunos lenguajes de producción solo permiten reglas reversibles, mientras que otros no.

Cuando se utilizan reglas irreversibles se debe establecer un compromiso de búsqueda, al mismo tiempo que se escriben las reglas. Pero como ya se sugirió antes, resulta útil pensar en hacerlo de todos modos, ya que permite al que escribe las reglas añadir algún conocimiento de control a estas propias reglas.

### 3.5. Razonamiento simbólico bajo incertidumbre

#### 3.5.1. Razonamiento No Monótono

Hasta ahora, se han descrito técnicas de razonamiento para un modelo del mundo completo, consistente e inalterable.

Desafortunadamente, en muchos dominios de problemas no es posible crear tales modelos. Se han propuesto varios marcos lógicos y métodos computacionales para poder manipular estos problemas. Veremos dos enfoques:

- El **razonamiento no monótono**, en el cual los axiomas y/o las reglas de inferencia se extienden para que sea posible razonar con información incompleta. Sin embargo, esos sistemas preservan la propiedad de que en un determinado momento, una sentencia puede pensarse que es cierta, puede pensarse que es falsa, o puede pensarse que no es ninguna de las dos.
- **Razonamiento estadístico**, en el que se extiende la representación para permitir algún tipo de medida numérica sobre la certeza (en lugar de simplemente CIERTO o FALSO) para asociar a cada sentencia.

Los sistemas convencionales de razonamiento, como la lógica de predicados de primer orden, están diseñados para trabajar con información que cumple tres importantes propiedades:

- *La información es completa con respecto al dominio de interés.* En otras palabras, todos los hechos necesarios para resolver el problema o están presentes en el sistema o pueden derivarse de ellos mediante reglas convencionales de la lógica de primer orden.
- *La información es consistente.*
- *La única forma en que puede cambiar la información es que se añadan nuevos hechos conforme estén disponibles.* Si estos nuevos hechos son consistentes con todos los demás hechos que ya se han afirmado, entonces **ninguno de los hechos pertenecientes al conjunto que eran ciertos pueden refutarse**. Esta propiedad se denomina **monotonía**.

Desafortunadamente, si no se verifica alguna de estas propiedades, los sistemas de razonamiento basados en la lógica convencional son inadecuados.

*Los sistemas de razonamiento no monótono, por otro lado, se diseñan para que puedan resolver problemas en los que quizás no aparezca alguna de estas propiedades.*

##### 3.5.1.1. Razonamiento por defecto

Se quiere usar el *razonamiento no monótono* para llevar a cabo lo que comúnmente se denomina **razonamiento por defecto**.

**Se pretende llegar a unas conclusiones basadas en lo que es más probable que sea cierto.**

En este apartado se explican dos enfoques para lograrlo.

- Lógica no monótona.
- Lógica por defecto.

A continuación se describen dos clases comunes de razonamiento no monótono que pueden definirse en estas lógicas:

- Abducción
- Herencia

### 3.5.1.1.1. Lógica no monótona

La **lógica no monótona** es un sistema que **proporciona una base para razonar por omisión**, en donde el lenguaje de la lógica de predicados de primer orden se **aumenta con un operador modal M**, que se lee como "**es consistente**". Por ejemplo, la fórmula:

$$\forall x, y: \text{Parientes}(x,y) \wedge M \text{ está_de_acuerdo}(x,y) \rightarrow \text{defenderá}(x,y)$$

se lee "*Para todo x e y, si x e y son parientes y si el hecho de que x se haya puesto de acuerdo con y es consistente con el resto de las suposiciones, entonces se concluye que x defenderá a y*".

### 3.5.1.1.2. Lógica por defecto

La **lógica por defecto** es una **lógica alternativa para llevar a cabo un razonamiento basado en omisiones** en la que **se introduce un nuevo tipo de reglas de inferencia**.

Este enfoque permite reglas de inferencia de la forma:

**A : B**

**C**

esta regla debe leerse así: "*si A es probable y es consistente asumir B, entonces se concluye que C*".

Como se puede ver, el propósito es muy similar al de las expresiones no monótonas que se usaban en la Lógica no monótona. Sin embargo, existen algunas **diferencias importantes entre las dos teorías**.

- La primera de ellas es que en la Lógica por defecto, las nuevas reglas de inferencia se usan como base para calcular un conjunto de extensiones plausibles de la base de conocimiento. Cada extensión se corresponde con una extensión consistente máxima de la base de conocimiento. La lógica entonces admite como teorema cualquier expresión válida en alguna extensión. Si es necesario decidirse entre las extensiones para poder resolver el problema, debe proporcionarse algún otro mecanismo.
- Una segunda diferencia importante entre estas dos teorías es que en la Lógica por defecto, las expresiones no monótonas son reglas de inferencia en lugar de expresiones del lenguaje. Es decir, no pueden manipularse mediante otras reglas de inferencia. Esto conduce a algunos resultados no esperados. Por ejemplo, dadas las dos reglas:

**A : B**

**B**

**$\neg A : B$**

**B**

sin ninguna aserción sobre A, no se puede llegar a ninguna conclusión sobre B, ya que no se aplica ninguna regla de inferencia.

## Abducción

La lógica estándar lleva a cabo deducciones. Dados dos axiomas:

$$\forall x: A(x) \rightarrow B(x)$$

$$A(C)$$

Puede concluirse que  $B(C)$  por deducción.

Pero ¿qué ocurre si se toma la implicación al revés? Por ejemplo, suponga que el axioma dado es:

$$\forall x: \text{Sarampión}(x) \rightarrow \text{Manchas}(x)$$

El axioma dice que si se tiene sarampión esto implica que aparecen manchas rojas.

Pero suponga que lo que se observa son las manchas rojas. Podría ser bueno concluir que se tiene un sarampión. Pero esta conclusión no está permitida por las reglas de la lógica estándar y aunque puede ser incorrecto, es posible que sea la mejor suposición que pueda hacerse.

*La derivación de conclusiones de esta forma es otra manera de razonamiento por defecto. Denominamos a este método **razonamiento por abducción**.*

El proceso de razonamiento por abducción puede describirse con más precisión de la siguiente forma, "Dadas dos fbf ( $A \rightarrow B$ ) y ( $B$ ), para cualquier expresión  $A$  y  $B$ , si es consistente asumir  $A$ , hacerlo".

El razonamiento abductivo no es un tipo de lógica del estilo de la Lógica por defecto y la Lógica no monótona. En realidad, puede describirse sobre cualquiera de ellas.

## Herencia

El razonamiento no monótono se utiliza con mucha frecuencia en la herencia de los valores de los atributos desde la descripción prototípica de una clase hacia las entidades individuales que pertenecen a la clase.

### 3.5.1.2. Razonamiento minimalista

Hasta ahora se ha hablado sobre métodos generales que proporcionan formas de describir cosas que son ciertas en general. Ahora se muestran métodos para referirse a un tipo muy específico y útil de cosas que son ciertas en general.

Estos métodos se basan en *alguna variante de la idea de modelo mínimo*. Aunque existen algunas definiciones diferentes sobre qué constituye un modelo mínimo, para nuestro propósito se dirá que **un modelo es mínimo si no existen otros modelos en los que sean ciertas menos cosas**.

La idea que hay detrás del uso de *modelos mínimos* como base para el *razonamiento no monótono sobre el mundo* es la siguiente:

**"Existen muchas menos sentencias ciertas que falsas.** Si algo es relevante y cierto, tiene sentido asumir que pertenece a nuestra base de conocimiento. Por lo tanto, asuma que las únicas sentencias ciertas son aquellas que necesariamente deben ser ciertas para que se mantenga la consistencia de la base de conocimiento".

#### 3.5.1.2.1. La suposición de un mundo cerrado

*La suposición de un mundo cerrado sugiere una sencilla forma de razonamiento minimalista.*

La suposición de un mundo cerrado dice que los únicos objetos que satisfacen un predicado  $P$  son aquellos que deben hacerlo. La suposición de un mundo cerrado es particularmente poderosa como base para razonar con bases de datos, las cuales se asume que son completas con respecto a las propiedades que describen.

Por ejemplo, se puede asumir sin peligro alguno que una base de datos sobre personal puede listar todos los empleados de una empresa. Si alguien pregunta si Gomez trabaja para la empresa, se puede responder "no" a no ser que aparezca explícitamente en la lista como un empleado.

Aunque la suposición de un mundo cerrado es a la vez sencilla y poderosa, *puede dar errores en la generación de respuestas apropiadas*, ya que esta suposición no es siempre cierta en el mundo; algunas partes del mundo no son realmente "*posibles de cerrar*". Esto se observa cuando se sacan conclusiones sobre ciertos hechos y luego se introducen nuevos hechos, que no estaban presentes con anterioridad en la base de conocimiento. La suposición de un mundo cerrado producirá resultados apropiados exactamente en la misma medida en que sea cierta la suposición de que todos los hechos positivos relevantes están presentes en la base de conocimiento.

### 3.5.1.2.2. Circunscripción

Aunque la suposición de un mundo cerrado captura parte de la idea de que algo que no debe ser necesariamente cierto debería ser asumido como falso, no la captura toda. Posee dos limitaciones esenciales:

- Opera sobre predicados individuales sin considerar las interacciones entre los predicados definidos en la base de conocimiento.
- Asume que todos los predicados tienen listadas todas sus instancias.

Para manipular estos problemas, se han propuesto distintas teorías sobre la ***circunscripción***.

En todas estas teorías, ***se añaden nuevos axiomas a la base de conocimiento existente***.

El efecto de estos axiomas consiste en forzar una interpretación mínima sobre una parte seleccionada de la base de conocimiento. En particular, cada axioma específico describe una forma de delimitar (es decir, de circunscribir) el conjunto de valores para los que un axioma particular de la teoría original sea cierto. Suponga, como ejemplo, que se tiene la sencilla aserción:

$$\forall x: \text{Adulto}(x) \wedge \neg \text{AB}(x, \text{aspecto}1) \rightarrow \text{Sabe\_leer}(x)$$

Nos gustaría circunscribir AB, puesto que nos gustaría aplicarlo únicamente a aquellos individuos a los que se les aplica.

En definitiva, lo que queremos hacer es decir algo, lo que debe ser el predicado AB. Para saber de qué se trata, es necesario conocer para qué valores se hace cierto.

### 3.5.1.3. Cuestiones sobre la implementación

Es importante tener en cuenta que no hay una correspondencia exacta entre las lógicas que se han descrito y las implementaciones que se van a explicar.

Las técnicas de implementación de Razonamiento No Monótono se pueden dividir en dos clases, dependiendo del enfoque que se da al problema del control de la búsqueda:

- **Primero en profundidad**, en la que se sigue un único camino, el más prometedor, hasta que surge alguna parte de conocimiento que fuerza a abandonar este camino e intentar otro..
- **Primero en anchura**, en donde se consideran igual de prometedoras todas las posibilidades. Todas ellas se consideran como un grupo, y se van eliminando algunas de ellas conforme se dispone de nuevos hechos. A veces puede ocurrir que solo uno de ellos (o un número muy pequeño) resulte ser consistente con todo lo demás que se conoce.

La resolución de un problema puede hacerse mediante un razonamiento hacia delante o mediante un razonamiento hacia atrás. La resolución de un problema que utiliza conocimiento incierto no es una excepción. Como consecuencia de todo esto se pueden definir dos enfoques:

- **Razonar hacia delante a partir de lo que se conoce**. Las conclusiones que se derivan de forma no monótona se manipulan de la misma forma que las que se derivan de forma monótona. Los sistemas de razonamiento no monótono que soportan este tipo de razonamiento permiten que las reglas estandar de encadenamiento hacia delante se extiendan con cláusulas **a-no-ser-que**, que proporcionan la base del razonamiento por defecto. El control (incluyendo la elección de la interpretación por defecto) se trata de la misma forma que todas las demás decisiones de control que realiza el sistema.
- **Razonar hacia atrás para determinar si alguna expresión P es cierta** (o quizás para encontrar un conjunto de vínculos entre las variables que hacen que sea cierto). Los sistemas de razonamiento no monótono que soportan este tipo de razonamiento pueden proporcionar alguna o todas de las siguientes características:
  - Que permite cláusulas por defecto (a-no-ser-que) en las reglas hacia atrás.
  - Que soporte algún tipo de debate en el que se intente producir argumentos tanto a favor de P como en su contra.

### 3.5.1.3.1. Implementación búsqueda primero en profundidad

#### Vuelta atrás dirigida por dependencias

Si se usa un enfoque primero en profundidad para el razonamiento no monótono, probablemente ocurrirá lo siguiente: necesitamos conocer un hecho, F, el cual no puede derivarse monótonamente a partir de lo que ya se conoce, pero sí puede derivarse haciendo alguna suposición A que parezca plausible. Así, una vez hecha la suposición A, se deriva F y entonces a partir de F se derivan los hechos adicionales G y H. Más tarde derivamos otros hechos M y N, aunque completamente independientes de A y F. Un poco más tarde, aparecen nuevos hechos que invalidan A.

Es necesario anular nuestra prueba de F, además de las de G y H ya que dependen de F. Pero ¿qué pasa con M y N? No dependen de F, por lo que no es lógico que deban invalidarse. Pero si se usa una vuelta atrás convencional, debe volverse hacia atrás en las conclusiones conforme éstas han sido derivadas. Por lo tanto, en la vuelta atrás se llega a M y N, por lo que se deshacen, con el fin de llegar a F, G, H y A.

Para empezar a tratar este problema, es necesaria una noción completamente diferente de la vuelta atrás, que debe basarse en las **dependencias lógicas** en lugar de en el orden cronológico en que se produjeron las decisiones. A este nuevo método lo denominamos **vuelta atrás dirigida por dependencias**, en contraste con el de **vuelta atrás cronológica** que se ha usado hasta ahora.

Antes de entrar en detalle en el funcionamiento de la vuelta atrás dirigida por dependencias, merece la pena indicar que aunque una de sus grandes motivaciones es el tratamiento del razonamiento no monótono, resulta útil también en los programas de búsqueda convencionales. Esto no es demasiado sorprendente si se considera que un programa de búsqueda primero en profundidad crea una nueva rama en el espacio de búsqueda una vez hecha alguna estimación "no muy precisa" sobre algo. Si eventualmente la rama es inadecuada, entonces se sabe que al menos una de las estimaciones que se han hecho era incorrecta. Esta estimación podría estar a lo largo de la rama.

En la vuelta atrás cronológica se asume que se trata de la suposición que se ha hecho más recientemente, por lo que se vuelve a ese punto para intentar alguna otra alternativa. Sin embargo, en ocasiones se dispone de información adicional que ayuda a encontrar la estimación incorrecta. Entonces, sería adecuado retractarse únicamente de esa estimación y dejar intacto todo lo demás que hubiera sucedido hasta entonces. Esto es exactamente lo que hace la vuelta atrás dirigida por dependencias.

Si se quiere usar una vuelta atrás dirigida por dependencias, es necesario realizar las siguientes acciones:

- Asociar a cada nodo una o más justificaciones. Cada justificación se corresponde con un proceso de derivación que conduce al nodo. (Como es posible que un nodo se derive de distintas formas, debe permitirse la posibilidad de que existan múltiples justificaciones). Cada justificación debe contener una lista con todos los nodos (hechos, reglas, suposiciones) de los que depende la derivación.
- Proporcionar un mecanismo que cuando se produzca una contradicción entre el nodo y su justificación genere el conjunto "malas" de suposiciones que están debajo de la justificación. El conjunto "malas" se define como el mínimo conjunto de suposiciones tales que si se elimina algún elemento de este conjunto, la justificación no será más válida y el nodo inconsistente deja de ser creíble.
- Proporcionar un mecanismo que considere el conjunto "malas" y elija una suposición para retirar.
- Proporcionar un mecanismo que propague el resultado de la retirada de una suposición. Este mecanismo debe convertir en inválidas todas las justificaciones que dependan, aunque sea indirectamente, de la suposición retirada.

## Sistemas de mantenimiento de la verdad basados en justificaciones (JTMS)

La idea de un sistema de mantenimiento de la verdad (truth maintenance system) o TMS surge como una forma de proporcionar la habilidad de trabajar con una vuelta atrás dirigida por dependencias para poder soportar el razonamiento no monótono.

Un TMS *permite conectar las aserciones mediante una red de dependencias del tipo hoja de cálculo*. Un JTMS o Sistema de mantenimiento de la verdad basado en Justificaciones, no conoce nada sobre la estructura de las aserciones en sí mismas. El único papel del sistema es servir como libro de anotaciones para un sistema de resolución de problemas, que a su vez, le proporciona tanto las aserciones como las dependencias entre las aserciones.

## Sistemas de mantenimiento de la verdad basados en la lógica (LTMS)

Un LTMS, Sistema de mantenimiento de la verdad basado en la Lógica es muy similar a un JTMS. Pero se diferencia en un aspecto importante.

En un JTMS, el TMS trata los nodos de la red como átomos, lo cual significa que no hay relaciones entre ellos excepto aquellas que se sitúan explícitamente en las justificaciones. En particular, un JTMS no tiene problemas en etiquetar simultáneamente a P y  $\neg P$ . No se detectará una contradicción de forma automática.

En un LTMS, por otro lado, se pueden detectar contradicciones de este tipo automáticamente.

### 3.5.1.3.2. Implementación búsqueda primero en anchura

## Sistemas de mantenimiento de la verdad basados en Suposiciones (ATMS)

Una forma alternativa de implementar el razonamiento no monótono lo constituyen los Sistemas de mantenimiento de la verdad basados en Suposiciones (ATMS) (Assumption-based truth maintenance systems). Tanto en un JTMS como en un LTMS se sigue una única línea de razonamiento en cada momento, y cuando es necesario cambiar las suposiciones del sistema, surge una vuelta atrás dirigida por dependencias.

***En un ATMS, se mantienen en paralelo varios caminos alternativos. La vuelta atrás se evita a expensas del mantenimiento de múltiples contextos, cada uno de los cuales se corresponde con un conjunto de suposiciones consistentes.***

En los sistemas basados en ATMS, al evolucionar el razonamiento, *el universo de contextos consistentes va podándose conforme se detectan contradicciones*. Los contextos consistentes que quedan se usan para etiquetar las suposiciones, de forma que indiquen el contexto en el que cada aserción tiene una justificación válida. Las aserciones que no tienen una justificación válida en algún contexto consistente se pueden podar por consideración del resolutor del problema. Conforme el conjunto de contextos consistentes se va haciendo cada vez más pequeño, el conjunto de aserciones que el resolutor de problemas puede creer de forma consistente, se reduce.

Esencialmente, mientras que un sistema ATMS funciona en anchura, considerando todos los posibles contextos a la vez, los sistemas JTMS y LTMS funcionan en profundidad.

### 3.5.2. Razonamiento Estadístico

Hasta ahora, se han descrito varias técnicas de representación que pueden utilizarse para modelar los sistemas de creencias en los que, en un momento dado, o se determina que un hecho en particular es o bien cierto o bien falso, o no se hace ninguna consideración al respecto.

En algunas resoluciones de problemas, sin embargo, puede resultar adecuado **describir las creencias sobre las que no se tiene certeza**, pero en las que **existen algunas evidencias que las apoyan**. Considere este tipo de problemas divididos en dos grupos:

- El primero de ellos está formado por problemas en los que se da una cierta aleatoriedad. Los juegos de cartas como el bridge o el blackjack son dos buenos ejemplos de este tipo de problemas. A pesar de que en estos problemas no es posible hacer una predicción sobre el mundo con absoluta certeza; si se dispone de conocimiento sobre las probabilidades de los distintos resultados, y sería deseable poder utilizar dicho conocimiento.
- El segundo tipo de problemas podría, en principio modelarse mediante las técnicas descritas en el capítulo anterior. En estos problemas, el mundo no es aleatorio, sino que se comporta "normalmente" hasta que surge algún tipo de excepción. La dificultad estriba en el hecho de que son muchas las posibles excepciones que se pueden producir, y deben enumerarse explícitamente (usando técnicas como AB y A-NO-SER-QUE). La mayoría de las tareas catalogadas como de sentido común pertenecen a este tipo de problemas, por ejemplo el razonamiento experto involucrado en los diagnósticos médicos. Para este tipo de problemas, puede ser muy útil algún tipo de medida estadística tal como las funciones que logran hacer un resumen del mundo. En lugar de tener que enumerar todas las posibles excepciones que se pueden producir, es mejor utilizar un resumen numérico que indique la frecuencia con la que es de esperar que aparezca una excepción de un cierto tipo.

### La probabilidad y el Teorema de Bayes

En muchos sistemas de resolución de problemas un objetivo importante consiste en reunir evidencias sobre la evolución del sistema y modificar su comportamiento sobre la base de las mismas. *Para modelar este comportamiento se necesita una teoría estadística de la evidencia.*

*Las estadísticas bayesianas constituyen esta teoría.*

El concepto fundamental de las estadísticas bayesianas es el de la probabilidad condicionada:

$$P(H|E)$$

La expresión anterior se lee como sigue: *la probabilidad de la hipótesis H dado que se observe la evidencia E.*

Para calcularla, es necesario tener en cuenta la *probabilidad previa de H* (la probabilidad que se le asignaría a H si no existe evidencia) y la parte en la que *E proporciona evidencia de H*. Para lograrlo, es necesario definir un universo que contenga un conjunto exhaustivo y mutuamente excluyente de **H<sub>i</sub>**, entre los que se intenta discriminar.

Sea,

**P(H<sub>i</sub>|E)** = la probabilidad de que la hipótesis H<sub>i</sub> sea verdad dada la evidencia E.

**P(E|H<sub>i</sub>)** = la probabilidad de que se observe la evidencia E dada la hipótesis i como verdadera.

**P(H<sub>i</sub>)** = la probabilidad a priori de que la hipótesis i sea cierta, independientemente de cualquier evidencia específica. Estas probabilidades se denominan probabilidades previas o a priori.

**K** = el número total de posibles hipótesis.

El teorema de Bayes se enuncia así:

$$P(H_i | E) = \frac{P(E | H_i).P(H_i)}{\sum_{n=1}^k P(E | H_n).P(H_n)}$$

Suponga, por ejemplo, que se está interesado en examinar la evidencia geológica de un lugar concreto para determinar si sería un buen lugar para hacer una excavación y encontrar un cierto mineral.

Si se conocen las probabilidades previas de aparición de cada uno de los minerales y también se conocen las probabilidades de que si un mineral aparece, entonces se observen ciertas características físicas, entonces puede utilizarse la fórmula de Bayes para calcular, a partir de las evidencias que se reúnan, la probabilidad de que aparezcan los distintos minerales.

En realidad esto es lo que se hace en el programa **PROSPECTOR**, el cual se ha usado con éxito como ayuda a la localización de depósitos de distintos minerales, incluyendo cobre y uranio.

La clave para poder utilizar el teorema de Bayes como base para razonar bajo incertidumbre consiste en **saber exactamente que es lo que dice**.

Especificamente, cuando se dice  $P(A|B)$  se está describiendo *la probabilidad de A condicionada a que la única evidencia que se tiene es B*. Si existe otra evidencia relevante, debe considerarse también. Suponga, por ejemplo, que se está resolviendo un problema de diagnóstico médico.

Considere las siguientes afirmaciones:

S: el paciente tiene manchas rojas,

M: el paciente tiene sarampión,

F: el paciente tiene fiebre alta.

Sin otra evidencia adicional, la presencia de manchas rojas sirve como evidencia de sarampión. La evidencia de la fiebre también sirve, ya que el sarampión suele provocar fiebre. Pero suponga que ya se sabe que el paciente tiene sarampión. En este caso, la evidencia adicional de que tiene manchas rojas en la piel no nos dice nada sobre la probabilidad de que tenga fiebre. Alternativamente, tanto las manchas rojas como la fiebre, por separado, constituirían una evidencia a favor del sarampión.

***Si se presentan las dos cosas, es necesario tenerlas ambas en cuenta para calibrar el peso total de la evidencia.***

***Sin embargo, como las manchas y la fiebre no son síntomas independientes, no pueden sumarse sus efectos.***

En lugar de esto, es necesario representar explícitamente la probabilidad condicionada que surja de su conjunción. En general dado un cuerpo de evidencia previo y alguna nueva observación E, es necesario hacer el siguiente cálculo:

$$P(H | E, e) = P(H | E) \cdot \frac{P(e | E, H)}{P(e | E)}$$

Desafortunadamente, en un mundo arbitrariamente complejo, el tamaño del conjunto de probabilidades combinadas que se necesitan para calcular esta función, crece como una función de la forma  $2^n$ , donde n es el número de proposiciones diferentes que es necesario considerar.

Esto hace que el teorema de Bayes sea inaplicable por diversos motivos:

- El problema de la adquisición de conocimiento es inabarcable. Son necesarias demasiadas probabilidades. Además de esto, existe la evidencia empírica sustancial de que las personas son malas estimadoras de probabilidades.
- El espacio que se necesitaría para almacenar todas las probabilidades es demasiado grande.
- El tiempo empleado en calcular las probabilidades es demasiado grande.

A pesar de todos estos problemas, las estadísticas bayesianas proporcionan una base atractiva para los sistemas que razonan bajo incertidumbre. Como resultado de todo esto, se han desarrollado distintos mecanismos que hacen uso de su potencialidad, pero que al mismo tiempo hacen que sea tratable. En el resto de este capítulo, se explican tres de estos mecanismos:

- Incorporación de factores de certeza a las reglas.
- Redes bayesianas.
- Teoría de Dempster-Shafer

### 3.5.2.1. Factores de certeza y sistemas basados en reglas

En esta sección se describe una forma práctica de compromiso sobre un sistema bayesiano puro.

El enfoque que se va a explicar surgió en el sistema **MYCIN**, en el cual se intenta recomendar las terapias apropiadas para pacientes con infecciones bacterianas. El sistema interactúa con el médico en la adquisición de los datos clínicos necesarios. MYCIN es un ejemplo de un **sistema experto**, debido a que realiza tareas que normalmente se encomiendan a un experto humano.

Este sistema se basa en el **uso de razonamiento probabilístico**. MYCIN representa la mayor parte de su conocimiento sobre diagnósticos en forma de un **conjunto de reglas**.

***A cada regla se le asocia un factor de certeza, que representa una medida sobre la evidencia que existe de que la conclusión sea el conseciente de la regla en el caso de que se describa el antecedente de la misma.***

Una regla de MYCIN típica se parecería a ésta:

**Si: (1) la cepa del organismo es gram-positivo, y  
(2) la morfología del organismo es coco, y  
(3) los organismos crecen de forma arracimada,  
entonces hay una buena probabilidad (0.7) de que  
el organismo sea un staphylococcus.**

MYCIN utiliza las reglas para hacer un **razonamiento hacia atrás** de los datos clínicos disponibles **a partir del objetivo** de encontrar organismos significativos causantes de enfermedades. Una vez que encuentra la identidad de tales organismos intenta seleccionar una terapia para tratar la enfermedad. Para poder comprender la forma en la que MYCIN utiliza información incierta, debe responderse a dos cuestiones: "¿Cuál es el significado de los factores de certeza?" y "¿Cómo combina MYCIN las estimaciones sobre la certeza de cada una de las reglas para hacer una estimación de la certeza de sus conclusiones?".

Debe también responderse a otra cuestión que surge de la ya descrita intratabilidad del razonamiento puramente bayesiano, que es: "¿Qué compromisos se realizan en la técnica MYCIN y qué riesgos llevan asociados?". En el resto de esta sección se responde a todos estos interrogantes.

Comencemos con una sencilla respuesta a la primera de las preguntas (a la que se volverá más tarde para dar una respuesta más detallada).

Un factor de certeza ( $FC[h,e]$ ) se define en términos de dos componentes:

- **MB[h,e]**. Una medida (entre 0 y 1) de la *creencia* de que la hipótesis  $h$  proporciona la evidencia  $e$ . MB da una medida sobre hasta qué punto la evidencia soporta la hipótesis. Es cero si la evidencia no soporta la hipótesis.
- **MD[h,e]**. Una medida (entre 0 y 1) sobre la *incredulidad* de que la hipótesis  $h$  proporciona la evidencia  $e$ . MD da una medida de hasta qué punto la evidencia soporta la negación de la hipótesis. Es cero si la evidencia soporta la hipótesis.

A partir de estas dos medidas, se puede definir el factor de certeza como sigue:

$$FC[h,e] = MB[h,e] - MD[h,e]$$

En cada regla basta un único número para definir tanto el valor de MB como el de MD, y por lo tanto, también el de FC, ya que *cada regla de MYCIN se corresponde como una parte de la evidencia y cada parte de la evidencia o bien soporta o bien niega una hipótesis (pero nunca ambas cosas)*.

**Los factores de certeza de las reglas de MYCIN los proporcionan los expertos que escriben las reglas.**

Reflejan las valoraciones del experto sobre la fortaleza con que la evidencia soporta la hipótesis. *Sin embargo, en el proceso de razonamiento de MYCIN, los factores de certeza tienen que combinarse para reflejar el uso de las múltiples partes de la evidencia y las múltiples reglas que se aplican para resolver el problema.*

La Figura 3.15 ilustra tres formas de combinación que es necesario considerar.

En la Figura 3.15(a), todas las reglas proporcionan la evidencia que relaciona una única hipótesis. En la Figura 3.15(b), es necesario considerar nuestra creencia como una colección de distintas proposiciones tomadas juntas. En la Figura 3.15(c), la salida de una regla proporciona la entrada de la siguiente.

*¿Qué fórmulas deberían utilizarse para plasmar estas combinaciones?* Antes de responder a esta cuestión, es necesario primero describir algunas propiedades que sería adecuado que cumplieran las funciones de combinación:

- Las funciones de combinación deberían ser commutativas y asociativas, ya que el orden en el que se reúnen las evidencias es arbitrario.
- Hasta que no se alcance la certeza, las evidencias adicionales que confirman deben incrementar MB (y de forma similar, con las evidencias que restan confirmación y MD).
- Si las inferencias inciertas se encadenan juntas, el resultado debe ser de menor certeza que cada una de las inferencias por separado.

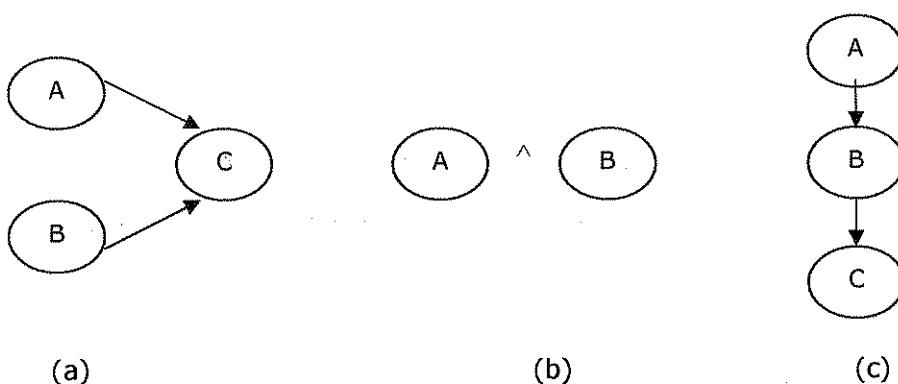


Figura 3.15

1. Si se supone que todas estas propiedades son deseables, considere en primer lugar la Figura 3.15(a), en la que **varias partes de evidencia se combinan para determinar el factor de certeza de una hipótesis**. Las medidas sobre la creencia o no creencia de una hipótesis dadas dos observaciones  $s_1$  y  $s_2$  se calculan de la siguiente forma:

$$\text{MB}[h, s_1 \wedge s_2] = \begin{cases} 0 & \text{si } \text{MD}[h, s_1 \wedge s_2] = 1 \\ \text{MB}[h, s_1] + \text{MB}[h, s_2] \cdot (1 - \text{MB}[h, s_1]) & \text{en caso contrario} \end{cases}$$

$$\text{MD}[h, s_1 \wedge s_2] = \begin{cases} 0 & \text{si } \text{MB}[h, s_1 \wedge s_2] = 1 \\ \text{MD}[h, s_1] + \text{MD}[h, s_2] \cdot (1 - \text{MD}[h, s_1]) & \text{en caso contrario} \end{cases}$$

Una forma de plasmar estas fórmulas en castellano consiste en que la medida sobre la creencia en  $h$  es 0 si no se cree en  $h$  con certeza.

En caso contrario, la medida sobre la creencia en  $h$ , dadas dos observaciones, es la medida sobre la creencia dada solo por una observación más algún incremento debido a la segunda observación. Este incremento se calcula tomando primero la diferencia entre 1 (certeza) y la creencia dada por la primera observación. Esta diferencia es la mayor que puede añadir la segunda observación. La diferencia se escala mediante la creencia sobre  $h$  dada solo la segunda observación.

De forma similar puede darse una explicación parecida para la fórmula que calcula la incredulidad.

A partir de MB y MD, puede calcularse FC. Nótese que si se unen varias fuentes de corroboración de la evidencia, el valor absoluto de FC se incrementa. Si se introduce una evidencia conflictiva, el valor absoluto de FC disminuye.

*Un sencillo ejemplo muestra como operan estas funciones.* Suponga que se tiene una observación inicial que confirma nuestra creencia en  $h$  con  $\text{MB} = 0.3$ . Entonces  $\text{MD}[h, s_1] = 0$  y  $\text{FC}[h, s_1] = 0.3$ . A continuación se hace una segunda observación que confirma  $h$  con un valor de  $\text{MB}[h, s_2] = 0.2$ .

Entonces:

$$\text{MB}[h, s_1 \wedge s_2] = 0.3 + 0.2 \cdot 0.7 \\ = 0.44$$

$$\text{MD}[h, s_1 \wedge s_2] = 0.0$$

$$\text{FC}[h, s_1 \wedge s_2] = 0.44$$

En este ejemplo se observa como una evidencia que tan solo sirve para apoyar levemente una cierta suposición, puede acumularse y producir incrementos mayores en los factores de certeza.

2. Considere ahora la Figura 3.15(b), en donde es necesario calcular el **factor de certeza de una combinación de hipótesis**. En particular, esto es necesario cuando se necesita conocer el factor de certeza de un antecedente de una regla que contiene varias cláusulas (como, por ejemplo, en la regla del estafilococo). El cálculo de la combinación de factores de certeza puede hacerse a partir de MB y MD.

Las fórmulas que usa MYCIN para la conjunción y disyunción de dos hipótesis son:

$$\text{MB}[h_1 \wedge h_2, e] = \min(\text{MB}[h_1, e], \text{MB}[h_2, e])$$

$$\text{MB}[h_1 \vee h_2, e] = \max(\text{MB}[h_1, e], \text{MB}[h_2, e])$$

**MD se calcula de forma análoga.**

3. Finalmente considere la Figura 3.15(c), en donde **las reglas se encadenan de forma que el resultado de la incertidumbre que sale de una regla es la entrada de la otra**. La solución para este problema también tendrá en cuenta el caso en el que tenga que asignarse a las entradas iniciales una medida sobre su incertidumbre.

Este caso podría darse fácilmente en aquellas situaciones en donde la evidencia es el resultado de algún experimento o algún test de laboratorio, de forma que los resultados no son completamente exactos. En estos casos, el factor de certeza de la hipótesis debe tener en cuenta tanto la intensidad con la que la evidencia parece indicar la hipótesis como el nivel de confianza en la evidencia.

MYCIN define el encadenamiento de reglas como sigue. Sea  $MB' [h,s]$  la medida de la creencia sobre  $h$  estando completamente segura la validez de  $s$ . Sea  $e$  la evidencia que nos lleva a creer en  $s$  (por ejemplo, las lecturas de los instrumentos del laboratorio o los resultados de aplicar otras reglas).

Entonces:

$$MB [h,s] = MB' [h,s] \cdot \max (0, FC[s,e])$$

Como en MYCIN los factores de certeza iniciales son estimaciones que proporcionan los expertos que escriben las reglas, no es realmente necesario dar una definición más precisa del significado de  $FC$  aparte de la ya mencionada.

### 3.5.2.2. Redes Bayesianas

Los Factores de Certeza representan un mecanismo de reducción de la complejidad de los sistemas de razonamiento bayesiano mediante la realización de algunas aproximaciones del formalismo.

*Las Redes Bayesianas constituyen un enfoque alternativo al de factores de certeza, en el que el formalismo de razonamiento bayesiano se preserva y se confía en la modularidad del mundo que se intenta modelar.*

La idea principal consiste en que para describir el mundo real no es necesario utilizar una tabla de probabilidades enorme en la que se listan las probabilidades de todas las combinaciones concebibles de sucesos.

La mayoría de los sucesos son condicionalmente independientes de la mayoría de los demás, por lo que no deben considerarse sus interacciones (y por lo tanto no se necesitan calcular todas las probabilidades).

**En lugar de esto, se puede usar una representación más local en donde se describen grupos de sucesos que interactúen.**

### 3.5.2.3. Teoría de Dempster-Shafer

Hasta ahora se han descrito diversas técnicas de forma que en todas ellas se consideraban proposiciones individuales y se asignaba a cada una de ellas una estimación (es decir, un único número) del grado de creencia que se garantizaba dada la evidencia.

En este apartado, se considera una técnica alternativa denominada **Teoría de Dempster-Shafer**.

*Este nuevo enfoque considera conjuntos de proposiciones y les asigna a cada uno de ellos un intervalo:*

**[Creencia, Verosimilitud]**

con el que debe indicarse el grado de creencia. **La creencia** (que normalmente se denota por Bel, belief) mide **la fuerza de la evidencia a favor de un conjunto de proposiciones**.

El rango va de 0 (que indica evidencia nula) a 1 (que denota certeza).

La verosimilitud ( $Pl$ , plausibility) se define como:

$$Pl(s) = 1 - Bel(\neg s)$$

Su rango también va desde 0 hasta 1 y **mide el alcance con que la evidencia a favor de  $\neg s$  deja espacio para la creencia en  $s$ .**

En particular, si se tiene evidencia cierta a favor de  $\neg s$ , entonces  $Bel(\neg s)$  es 1 y  $Pl(s)$  es 0. Lo anterior también indica que el único posible valor  $Bel(s)$  es 0.

*El intervalo creencia-verosimilitud que se ha definido mide no solo el nivel de creencia sobre algunas proposiciones, sino también la cantidad de información que se tiene.*

Suponga que se consideran tres hipótesis rivales: A, B y C. Si no se tiene información, para cada una de ellas se dice que la probabilidad de que sean ciertas está en el rango [0,1].

Conforme se acumula evidencia, el intervalo va estrechándose, representando el incremento de confianza con que se sabe la probabilidad de cada hipótesis. Esto contrasta con el enfoque bayesiano puro, en donde probablemente se empezaría por asignar las probabilidades a priori equitativamente entre las hipótesis, de forma que para cada una de ellas  $P(h) = 0,33$ .

Con los intervalos se clarifica el hecho de que no se posee información al comenzar. En el enfoque bayesiano esto no es así, ya que se podría terminar con los mismos valores en la probabilidad si se reúnen volúmenes de evidencia de forma que tomados juntos sugieran que los tres valores aparecen con la misma frecuencia. Esta diferencia puede resultar importante si una de las decisiones que necesita hacer el programa consiste en ver si se reúne más evidencia o se actúa sobre la base de la que ya existe.

## Sistemas Semánticos para Representación del Conocimiento

Las buenas representaciones son la clave de una buena resolución de problemas

***En general, una representación es un conjunto de convenciones sobre la forma de describir un tipo de cosas.***

Una descripción aprovecha las convenciones de una representación para describir alguna cosa en particular.

El hallar la representación apropiada es una parte fundamental de la resolución de un problema. Considere, por ejemplo, el siguiente problema para niños:

### **El granjero, la zorra, el pollo y el grano**

Un granjero quiere cruzar un río llevando consigo una zorra silvestre, un pollo gordo y un saco de granos de trigo. Por desgracia, su bote es tan pequeño que solo puede transportar una de sus pertenencias en cada viaje. Peor aún, la zorra, si no se le vigila, se come al pollo, y el pollo, si no se lo cuida, se come el trigo; de modo que el granjero no debe dejar a la zorra sola con el pollo o al pollo solo con el trigo. ¿Qué se puede hacer?

Descrito en español, la resolución del problema se lleva unos cuantos minutos porque es preciso separar las restricciones relevantes de los detalles irrelevantes. El español no es una buena representación.

Sin embargo, descrito de manera más apropiada, el problema no toma tiempo alguno porque se puede trazar una línea del principio al final en la Figura 3.16 de manera instantánea. El trazado de dicha línea resuelve el problema porque cada dibujo representa un arreglo seguro para el granjero y sus pertenencias en las orillas del río, y cada conexión entre los dibujos representa un cruce válido. El dibujo es una buena descripción ya que las situaciones permitidas y los cruces legales quedan claramente definidos y no existen detalles irrelevantes.

Para hacer un diagrama así, primero se construye un nodo por cada forma en que el granjero y sus tres pertenencias pueden ocupar los dos márgenes del río. Debido a que el granjero y sus pertenencias pueden encontrarse, cada uno, en cualquier lado del río, existen  $2^{1+3} = 16$  arreglos, diez de los cuales son seguros en el sentido de que nadie es comido. Los seis arreglos no seguros colocan a la zorra, el pollo y el trigo en uno u otro lado, o al pollo y al trigo en uno y otro lado, o a la zorra y al pollo en uno y otro lado.

El segundo y último paso es dibujar un enlace para cada viaje permitido. Por cada par ordenado de arreglos existe un enlace que los conecta si y solo si los dos arreglos cumplen con dos condiciones: primera, el granjero cambia de lado; y segunda, a lo sumo una de las pertenencias del granjero cambia de lado. Debido a que existen diez arreglos permitidos, hay  $10 \times 9 = 90$  pares ordenados, pero solo 20 de ellos satisfacen las condiciones requeridas por los enlaces.

*Es evidente que la descripción nodo y enlace es una buena descripción con respecto al problema planteado, ya que resulta fácil de hacer y, una vez que se tiene, el problema resulta simple de resolver.*

La idea importante que ilustra este problema es que una buena descripción, desarrollada de acuerdo con las convenciones de una buena representación, es una puerta abierta para la resolución del problema; una mala descripción, que utiliza una mala representación, es un obstáculo que impide la resolución del problema.

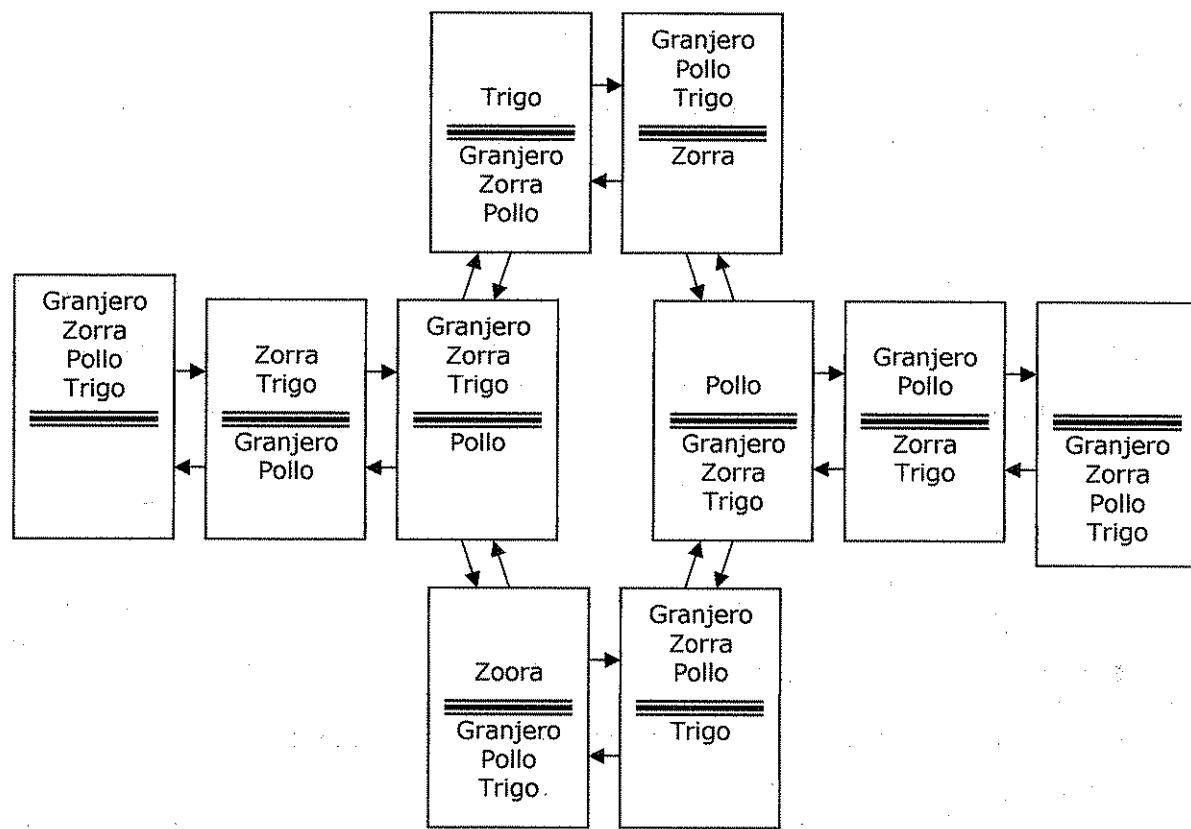


Figura 3.16

### 3.6. Estructuras de Ranura y Relleno Débiles

Recuérdese que estas estructuras se introdujeron en un principio como un dispositivo para soportar adecuadamente la herencia a lo largo de los enlaces es-un e instancia. Este es un importante aspecto de estas estructuras.

**La herencia monótona se puede desarrollar más eficazmente con estas estructuras que con la lógica pura, y la herencia no monótona puede soportarse muy fácilmente. La razón por la que la herencia se ejecuta de un modo sencillo, es que en los sistemas de ranura y relleno el conocimiento está estructurado como un conjunto de entidades y todos sus atributos.**

Se describirán dos enfoques de este tipo de estructuras: las redes semánticas (semantic nets) y los marcos (frames).

Se hablará sobre las propias representaciones, así como sobre las técnicas para razonar con ellas. De todos modos, no se hablará demasiado acerca del conocimiento específico que deben contener estas estructuras.

A estas estructuras de "conocimiento pobre" se les denominarán "*débiles*". En las estructuras de ranura y relleno "*fuertes*" se establecen mayores compromisos en relación con el contenido de las representaciones.

#### 3.6.1. Redes Semánticas

La idea principal que hay debajo de las redes semánticas es que la **información contenida en ellas se representa como un conjunto de nodos conectados unos con otros mediante un conjunto de arcos etiquetados que representan las relaciones entre los nodos.**

En la Figura 3.17 se muestra un fragmento de una red semántica típica.

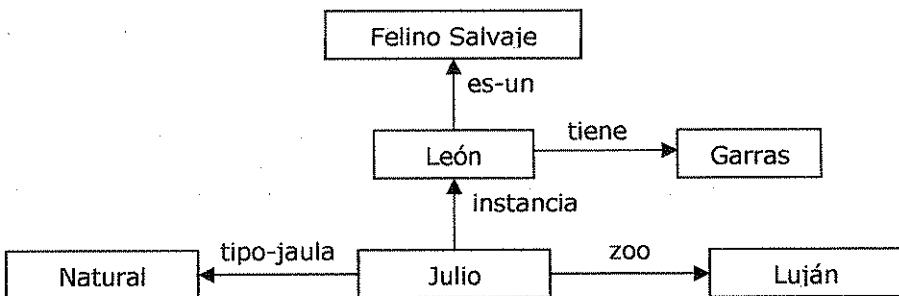


Figura 3.17

Esta red contiene ejemplos tanto de relaciones es-un como de relaciones instancia, así como algunas otras relaciones más específicas del dominio como zoo y tipo-jaula. En esta red se puede utilizar la herencia para derivar la relación adicional:

**tiene(Julio, Garras)**

#### Búsqueda de intersección

Una de las primeras formas de usar las redes semánticas antiguas fue para *encontrar relaciones entre objetos*, dividiendo la activación a partir de cada uno de los dos nodos y observando donde se encontraba dicha activación.

Este proceso se llama **búsqueda de intersección**. Utilizando este proceso es posible usar la red de la Figura 3.17 de manera que se puedan responder preguntas tales como "*¿Cuál es la conexión entre Luján y Natural?*".

Esta clase de razonamiento utiliza una de las grandes ventajas de las estructuras de ranura y relleno sobre las representaciones puramente lógicas, ya que **tienen la ventaja de la**

**organización del conocimiento basado en entidades**, que proporcionan las representaciones de ranura y relleno.

Sin embargo, para poder contestar a preguntas más estructuradas son necesarias redes con una estructuración más alta. En los siguientes apartados se ampliará y refinará nuestra noción de red para que éstas puedan soportar un razonamiento más sofisticado.

### Representación de predicados no binarios

Las redes semánticas se pueden considerar como un modo natural de representar las relaciones que podrían aparecer como instancias de los **predicados binarios** en la lógica de predicados. Por ejemplo, algunos de los arcos de la Figura 3.17 se podrían representar en lógica como:

- es\_un(León, Felino Salvaje)**
- instancia(Julio, León)**
- zoo(Julio, Luján)**
- tipo-jaula(Julio, Natural)**

Pero el conocimiento expresado en predicados de mayor arididad, también se puede expresar en redes semánticas. Ya se ha visto que muchos de los **predicados unarios** de la lógica se pueden considerar como predicados binarios, utilizando algunos predicados de propósito muy general, como puede ser es-un e instancia. Así, por ejemplo,

- **hombre(Marco)**

se podría reescribir como:

- instancia(Marco, hombre)**

y de este modo se ve que es mucho más fácil hacer la representación en una red semántica.

Los predicados de **tres o más argumentos** también pueden convertirse a forma binaria creando un nuevo objeto que represente todo el predicado, y después introduciendo predicados binarios para describir la relación con este nuevo objeto de cada uno de los argumentos originales.

Supóngase que se sabe:

- Marcador(Rosario Central, Newells, 3-1)**

Esto se puede representar en una red semántica creando un nodo que represente el juego específico, y relacionar después las tres partes de la información con dicho nodo. La Figura 3.18 muestra la red que surge al hacer esto.

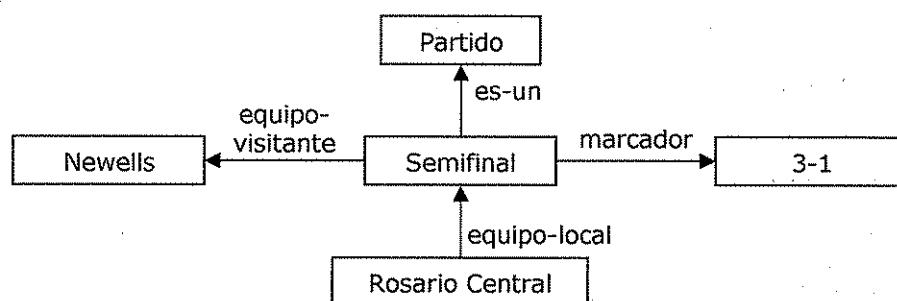


Figura 3.18

### 3.6.2. Marcos

Un marco (frame) es una colección de atributos, normalmente llamados ranuras (slots), con valores asociados (y posibles restricciones entre los valores), que describe alguna entidad del mundo.

Un marco único, tomado independientemente, no suele ser útil. En lugar de eso se construyen sistemas de marcos a partir de colecciones de marcos conectados unos con otros en virtud del hecho de que el valor de un atributo de un marco puede ser a su vez otro marco.

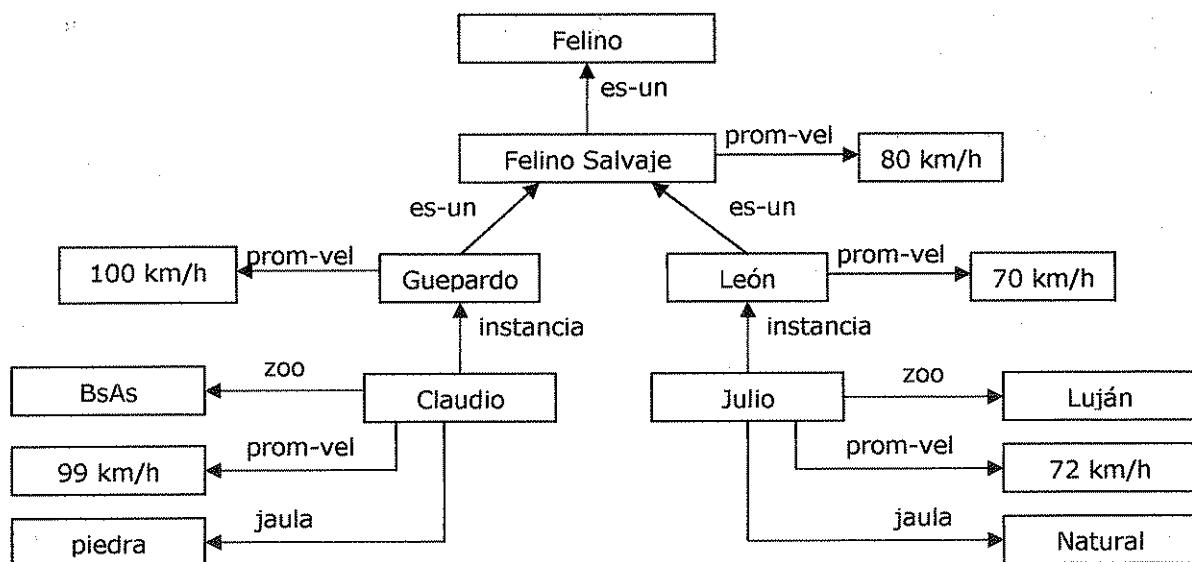


Figura 3.19

### Los marcos como conjuntos e instancias

La **teoría de conjuntos** proporciona una buena base para comprender los sistemas de marcos. Aunque no todos los sistemas de marcos se definen de este modo, aquí será así. Con este enfoque, **cada marco representa, ya una clase (un conjunto), ya una instancia (un elemento de la clase)**. Para ver como funciona esto, se considerará el sistema de marcos que se muestra en la Figura 3.20, basado en la red semántica de la Figura 3.19.

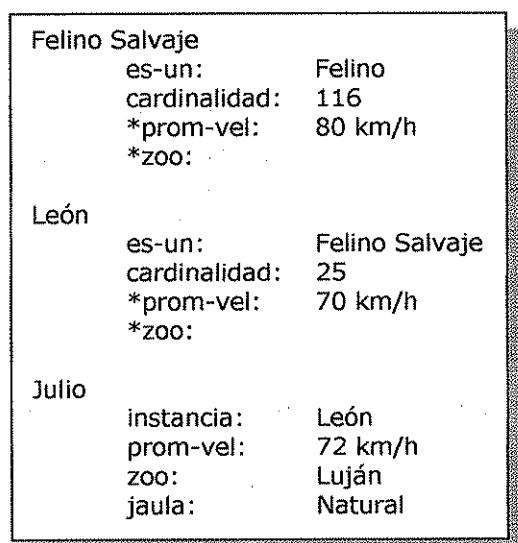


Figura 3.20

En este ejemplo los marcos Felino Salvaje y León son todas las clases. El marco Julio es una instancia.

La **relación es-un** que se ha estado utilizando sin una definición precisa, **es en realidad la relación subconjunto**. El conjunto de los Leones es un subconjunto de los Felinos Salvajes. El conjunto de los Felinos Salvajes es un subconjunto de los Felinos.

La **relación instancia** corresponde con la relación **elemento-de**. Julio es un elemento del conjunto de los Leones. Así también es un elemento del superconjunto de Felinos Salvajes. La transitividad de es-un, que se estudió por encima en la descripción de herencia de propiedades, deriva directamente de la transitividad de la relación subconjunto.

Tanto la **relación es-un**, como la **relación instancia**, tienen **atributos inversos** que se denominan **subclases** y **todas las instancias**. Realmente no tiene importancia escribirlos explícitamente en las instancias, a menos que sea necesario referirse a ellos. Se tendrá en cuenta que el sistema de marcos los mantiene automáticamente, bien de un modo explícito o bien calculándolos si es necesario.

Debido a que una clase representa un conjunto, existen dos clases de atributos que se pueden asociar con ésta. Existen atributos acerca del conjunto en sí mismo, y también atributos para ser heredados por cada elemento del conjunto. Se indicará la diferencia entre estos dos tipos asociando al segundo un asterisco (\*). Por ejemplo, la clase de los Leones tiene como cardinal 25 (es decir, hay 25 leones). El atributo prom-vel es heredado por los individuos pertenecientes a dicha clase, por lo tanto de esta manera se trasladan propiedades hereditarias desde las superclases hacia sus instancias. El atributo zoo no tiene valor ingresado por lo cual se observa que mediante el sistema de marcos se pueden también definir prototipos.

### 3.7. Estructuras de Ranura y Relleno Fuertes

*Las redes semánticas y los sistemas de marcos implementan estructuras muy generales para representar el conocimiento.*

La Dependencia Conceptual, los Guiones y los CYC, **implementan poderosas teorías** sobre la forma en que los programas de IA pueden representar y utilizar el conocimiento sobre situaciones comunes.

#### 3.7.1. Dependencia Conceptual

**La Dependencia Conceptual (DC) es una teoría sobre la representación del tipo de conocimiento sobre los eventos que normalmente aparecen en las frases de lenguaje natural.**

El objetivo consiste en representar el conocimiento de alguna forma que:

- Facilite extraer inferencias de las frases.
- Sea independiente del lenguaje en el que están las frases originalmente.

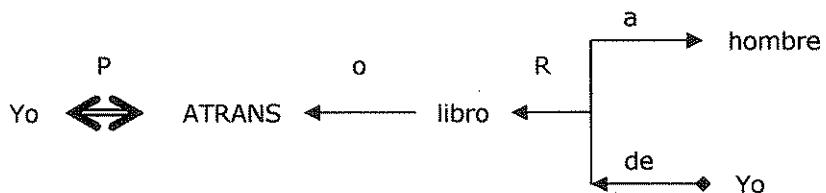
Debido a estos dos requisitos, *la representación en DC de una frase no se construye con primitivas que se corresponden con las palabras que aparecen en la frase, sino con primitivas conceptuales que pueden combinarse para formar el significado de las palabras en cualquier lenguaje concreto*.

Esta teoría la describió Shank (1973) y se ha implementado en varios programas que leen y comprenden texto en lenguaje natural.

Al contrario que en las redes semánticas, que proporcionan solo una estructura en la que pueden situarse nodos que representan información a cualquier nivel, **la Dependencia Conceptual proporciona tanto una estructura como un conjunto específico de primitivas, a un nivel concreto de granularidad**, en las que puede construirse representaciones de trozos particulares de información.

La Figura 3.21 muestra un ejemplo sencillo de la forma en que se representa el conocimiento en DC para la frase

### Yo le di un libro al hombre



donde los símbolos tienen los siguientes significados:

- Las flechas indican direcciones de la dependencia
- La flecha doble indica los tipos de enlaces entre el actor y la acción
- P indica tiempo pasado
- ATRANS es una de las acciones primitivas utilizadas por la teoría. Indica transferencia de posesión
- O indica la relación OBJECT CASE
- R indica la relación RECIPIENT CASE

Figura 3.21

En DC, las representaciones de las acciones se construyen a partir de un conjunto de **acciones primitivas**. Aunque existen diferencias significativas sobre el conjunto exacto de acciones primitivas que proporcionan las distintas implementaciones de DC, un típico conjunto es el siguiente:

<b>ATRANS</b>	Transferencia de una relación abstracta (dar)
<b>PTRANS</b>	Transferencia de una localización física de un objeto (ir)
<b>PROPEL</b>	Aplicación de fuerza física (empujar)
<b>MOVE</b>	Movimiento de una parte del cuerpo por su dueño (patear)
<b>GRASP</b>	Asimiento de un objeto por un actor (empuñar)
<b>INGEST</b>	Ingestión de un objeto por parte de un animal (comer)
<b>EXPTEL</b>	Expulsión de algo del cuerpo de un animal (llorar)
<b>MTRANS</b>	Transferencia de información mental (decir)
<b>MBUILD</b>	Construcción de información nueva aparte de la vieja (decidir)
<b>SPEAK</b>	Producción de sonidos (hablar)
<b>ATTEND</b>	Concentración de un órgano sensorial hacia un estímulo (escuchar)

Un segundo conjunto de bloques construidos de DC es el conjunto de las **dependencias permitidas entre las conceptualizaciones** descritas en una frase. Existen cuatro categorías conceptuales primitivas a partir de las cuales pueden construirse estructuras de dependencias. Estas son:

<b>ACT</b>	Acciones
<b>PP</b>	Objetos (Productores de imágenes)
<b>AA</b>	Modificadores de acciones (asistentes de acciones)
<b>PA</b>	Modificadores de PP (asistentes de imágenes)

Además, las estructuras de dependencia son en sí mismas conceptualizaciones y pueden servir como componentes de estructuras de dependencias más grandes.

Las dependencias entre conceptualizaciones se corresponden con las relaciones semánticas entre los conceptos subyacentes. La Figura 3.22 proporciona una lista de algunas de ellas.

La primera columna contiene las reglas; la segunda contiene ejemplos de su uso, y la tercera contiene una versión en español de cada ejemplo.

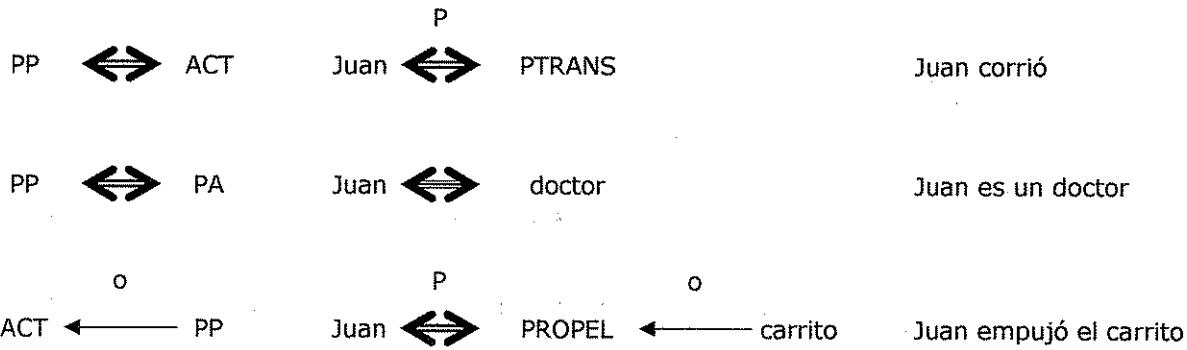


Figura 3.22

### 3.7.2. Guiones

DC es un mecanismo para representar y razonar sobre eventos. Sin embargo, los eventos raramente ocurren por separado. En este apartado se presenta un mecanismo de *representación del conocimiento de secuencias comunes de eventos*.

**Un guión (script) es una estructura que describe una secuencia estereotipada de eventos en un contexto concreto.**

Un guión está formado por un conjunto de ranuras. Asociada a cada ranura puede estar alguna información sobre qué tipo de valores puede contener, así como un valor por defecto que puede usarse si no se dispone de ninguna otra información. Hasta ahora la definición de un guión parece muy similar a la de marco. En este nivel de detalle las dos estructuras son idénticas. Sin embargo, debido al papel especializado que juega un guión, podemos hacer algunas afirmaciones más precisas sobre su estructura.

La Figura 3.23 muestra parte de un guión típico, el guión sobre un restaurante (tomado de Schank y Abelson, 1977). La figura ilustra los componentes importantes de un guión.

<b>Entry conditions</b>	Condiciones que, en general, deben satisfacerse antes de que puedan ocurrir los eventos que se describen en el guión.
<b>Result</b>	Condiciones que, en general, deberán ser ciertas después de que ocurran los eventos que se describen en el guión.
<b>Props</b>	Ranuras que representan objetos que aparecen involucrados en los eventos descritos en el guión.
<b>Roles</b>	Ranuras que representan a gente que aparece involucrada en los eventos descritos en el guión.
<b>Track</b>	La variación específica sobre un patrón más general representado por este guión concreto.
<b>Scenes</b>	La secuencia de eventos que ocurre. Los eventos se representan con un formalismo de dependencias conceptuales.

Los guiones resultan útiles porque en el mundo real aparecen *patrones en la ocurrencia de los eventos*. Estos patrones surgen debido a las relaciones de causalidad entre los eventos. Los agentes llevarán a cabo una acción de forma que entonces son capaces de realizar otra. Los eventos que se describen en un guión forman una gigantesca cadena causal. El comienzo de la cadena es el conjunto de condiciones de entrada, las cuales hacen que los primeros eventos del guión puedan ocurrir. El final de la cadena es el conjunto de resultados los cuales pueden capacitar posteriores eventos o secuencias de ellos (posiblemente descritos por otro guión).

Por medio de esta cadena, los eventos se conectan tanto con eventos anteriores que los hacen posibles, como con eventos posteriores, a los cuales capacita.

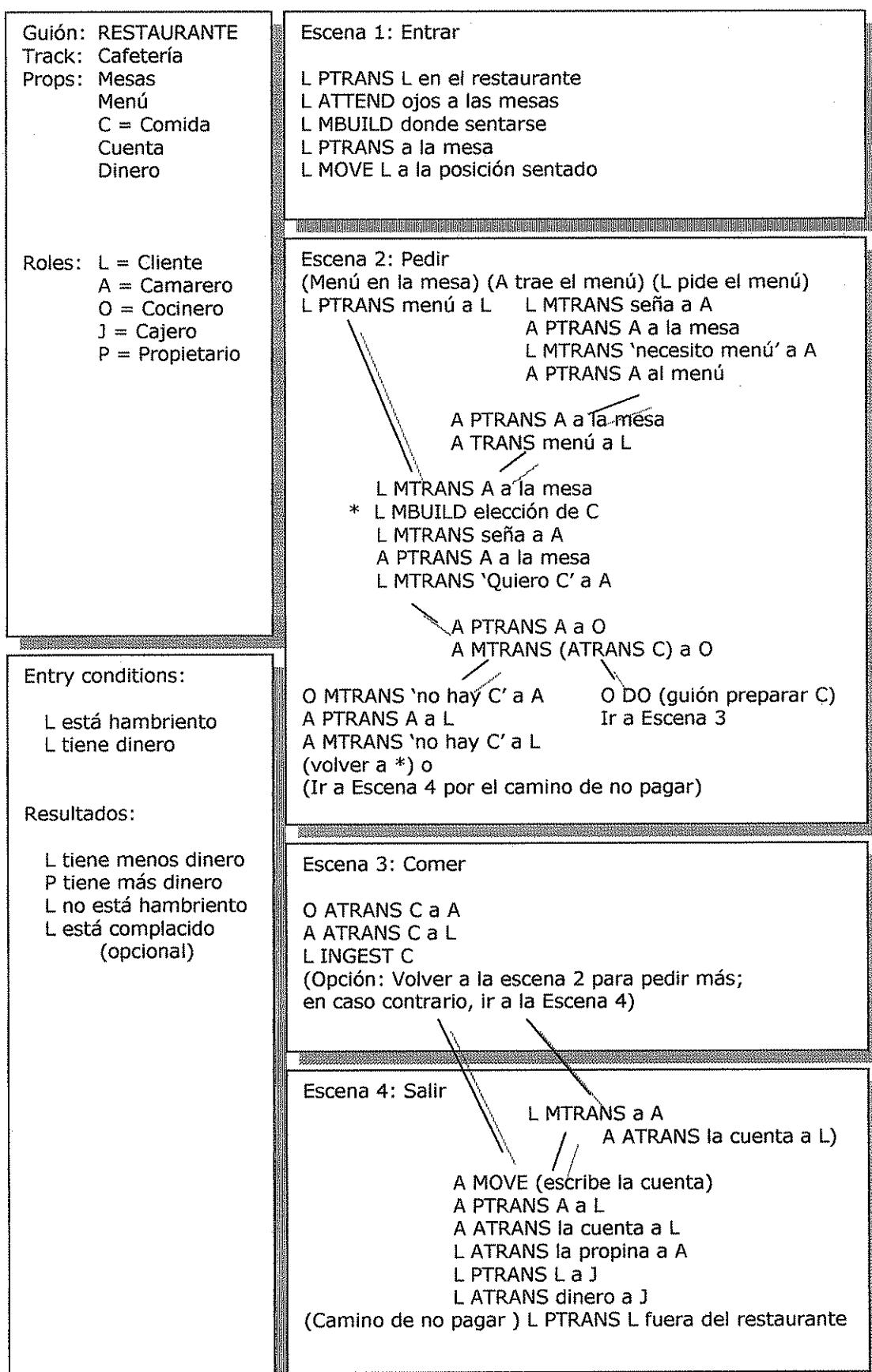


Figura 3.23

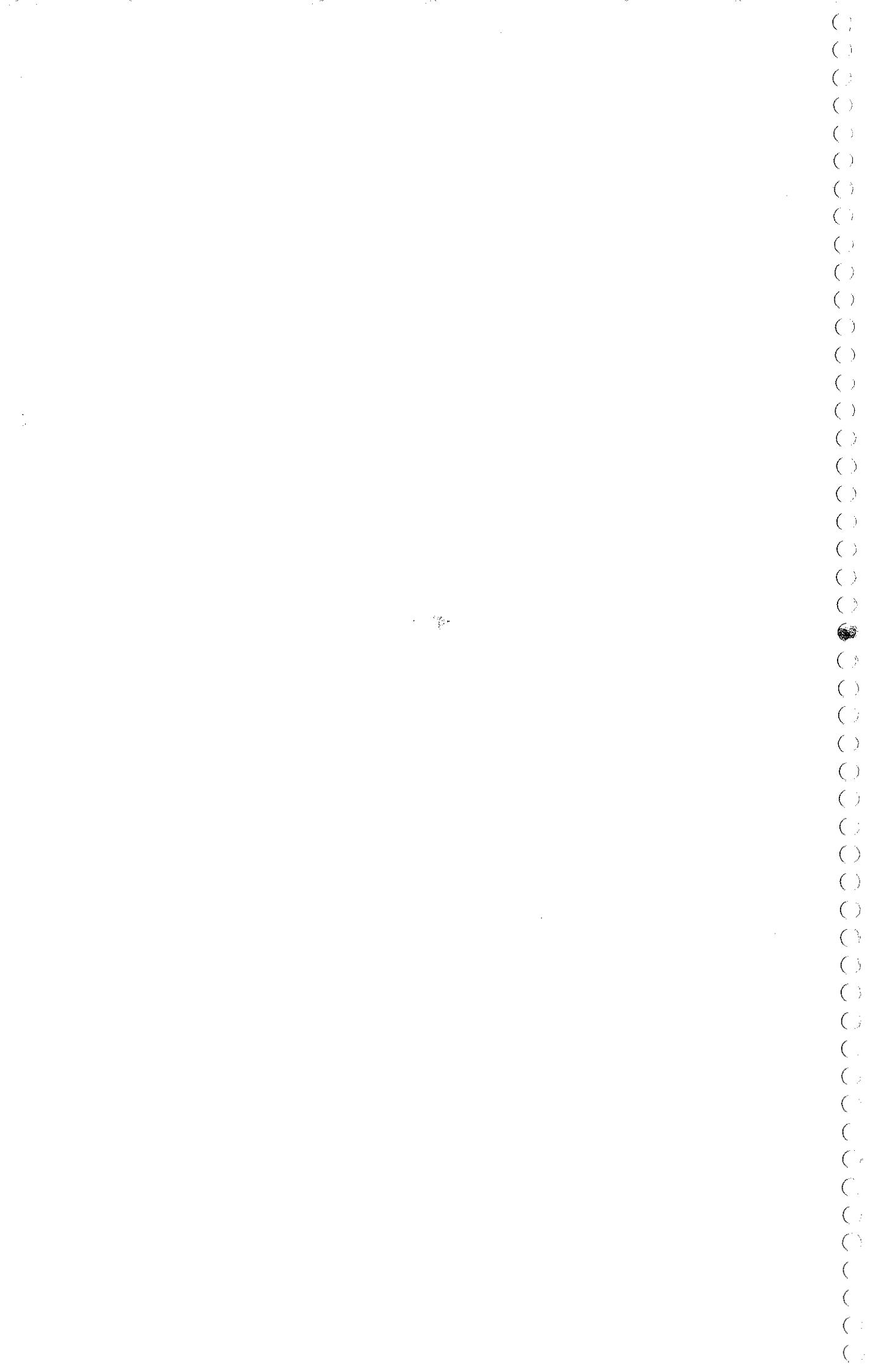
### 3.7.3. CYC

**CYC es un proyecto de una gran base de conocimiento cuyo propósito es el de capturar el conocimiento humano de sentido común.**

El objetivo de CYC es codificar el amplio cuerpo de conocimiento que es tan obvio que resulta fácil olvidar indicarlo explícitamente. Esta base de conocimiento podría combinarse con bases de conocimiento especializadas para producir sistemas que sean menos frágiles que la mayoría de los que se disponen en la actualidad.

Como DC, CYC representa una teoría concreta para describir el mundo y como DC, puede usarse para tareas de IA tales como la comprensión del lenguaje natural.

Sin embargo, CYC es más comprensible; mientras que DC proporciona una teoría concreta para la representación de eventos, CYC contiene representaciones de eventos, objetos, actitudes y muchas otras. Además CYC se preocupa especialmente de aspectos de escala, esto es, qué ocurre cuando construimos bases de conocimiento que contienen millones de objetos.



## **UNIDAD DIDACTICA 4**

**Ingeniería  
del Conocimiento**



**UNIDAD DIDACTICA 4**  
**EJE CONCEPTUAL**  
Ingeniería del Conocimiento

**TEMAS**

<b>4.1. PROLOG (Programación Lógica) .....</b>	<b>4</b>
4.1.1. Una breve historia del Prolog .....	4
4.1.2. ¿Para qué sirve Prolog?.....	4
4.1.3. Lenguaje Procedural vs. Lenguaje Declarativo .....	4
4.1.4. Inteligencia Artificial .....	5
<b>4.2. Relación con la Lógica .....</b>	<b>6</b>
4.2.1. Hechos .....	6
4.2.2. Variables.....	7
4.2.3. Reglas .....	7
4.2.4. Cláusulas .....	9
4.2.5. Preguntas .....	9
4.2.6. Conjunciones y Backtracking .....	9
<b>4.3. Estructura de un Programa en Prolog.....</b>	<b>11</b>
4.3.1. Composición de un programa. Cláusulas. Predicados. Dominios .....	11
4.3.2. Ejemplos.....	14
<b>4.4. Ingeniería del Conocimiento .....</b>	<b>16</b>
4.4.1. Organización de una Base de Conocimientos .....	16
4.4.2. Encontrando el Experto.....	16
4.4.3. Verificaciones de la BC.....	17
<b>4.5. Definición de Sistema Experto.....</b>	<b>17</b>
4.5.1. Definición funcional .....	18
4.5.2. Definición estructural.....	18
<b>4.6. Amazonas de Sistemas Expertos.....</b>	<b>20</b>
<b>4.7. Aplicaciones de los Sistemas Expertos .....</b>	<b>21</b>
Ventajas de la aplicación de Sistemas Expertos .....	21
Fases de la "inserción social" de los Sistemas Expertos .....	22
<b>4.8. Sistemas Expertos más conocidos .....</b>	<b>22</b>

#### 4.1. PROLOG (Programación Lógica)

**PROLOG** es un lenguaje de programación que se utiliza para resolver problemas en los que entran en juego objetos y relaciones entre objetos. Actualmente se ha convertido en el **principal entorno de programación para inteligencia artificial (IA)**, una de las principales áreas de aplicación de las computadoras que está emergiendo. También se puede hacer virtualmente cualquier cosa con el PROLOG como podría hacerlo con cualquier otro lenguaje de programación, incluyendo juegos, contabilidad, gráficos y simulación. No es siempre el lenguaje más práctico o eficiente para algunas aplicaciones, pero pueden realizarse con él.

Para los programadores que investigan en IA, el PROLOG ofrece un método diferente al empleado por los lenguajes más familiares, tales como BASIC, COBOL, PASCAL y C.

##### 4.1.1. Una breve historia del Prolog

PROLOG significa "*PROgramming LOGic*", es decir *programación basada en la lógica* y es un lenguaje de programación de computadoras que fue inventado alrededor de 1970 por Alain Colmerauer y sus colegas de la Universidad de Marsella, Francia. Rápidamente el PROLOG se convirtió en el lenguaje principal para IA en Europa, mientras que LISP (otro lenguaje de programación usado para IA) se usaba principalmente por los programadores de los Estados Unidos. A finales de los años '70 comenzaron a aparecer versiones de PROLOG para microcomputadoras. Uno de los compiladores de PROLOG más populares fue el MICROPROLOG, pero éste no ofrece la riqueza de predicados.

No existió mucho interés por el PROLOG hasta que los científicos japoneses lanzaron su famoso proyecto de la quinta generación con el objetivo de diseñar nuevas computadoras y software. De repente, la gente comenzó a mirar de otra forma el PROLOG y sus posibilidades.

##### 4.1.2. ¿Para qué sirve Prolog?

Los lenguajes de computadoras son raramente buenos para todos los tipos de problemas. FORTRAN es usado principalmente por los científicos y matemáticos, mientras que COBOL, es usado principalmente en el mundo comercial. A las implementaciones del PROLOG le falta la habilidad para manejar problemas sobre "números" o "procesamiento de texto".

En su lugar, PROLOG está diseñado para manejar "**problemas lógicos**" (es decir, problemas en los que se necesitan tomar decisiones de una forma ordenada), intenta hacer que la computadora "razone" la forma de encontrar una solución. Es particularmente adecuado para diferentes tipos de problemas de inteligencia artificial.

##### 4.1.3. Lenguaje Procedural vs. Lenguaje Declarativo

La mayoría de los lenguajes de computadoras personales, BASIC, PASCAL, COBOL, etc., han sido **procedurales**. Tales lenguajes permiten al programador decirle a la computadora *lo que tiene que hacer, paso a paso, procedimiento por procedimiento, hasta alcanzar una conclusión*.

El **PROLOG** no es procedural, es **declarativo**, necesita que se declaren reglas y hechos sobre símbolos específicos y luego se le pregunte sobre si un objetivo concreto se deduce lógicamente a partir de los mismos.

Mientras que un lenguaje procedural le exige que introduzca el recipiente y los ingredientes, un lenguaje declarativo solo le pide los ingredientes y el objetivo. Se declara la situación con la que quiere trabajar y donde quiere ir, **el propio lenguaje realiza el trabajo de decidir como alcanzar dicho objetivo**.

La diferencia entre lenguaje declarativo y procedural es una de las razones por la que la implementación de un lenguaje como PROLOG es una herramienta tan buena para desarrollar aplicaciones en IA, especialmente cuando se lo compara con otros lenguajes.

*Al trabajar con un lenguaje declarativo se da información sobre un tema determinado, se definen las relaciones que existen entre estos datos y finalmente se construyen preguntas o cuestionamientos sobre todo el paquete, quedándole al lenguaje la tarea de elaborar las conclusiones mediante un razonamiento lógico.*

#### 4.1.4. Inteligencia Artificial

Determinar qué es un programa inteligente implica que se conoce lo que significa **inteligencia**:

*capacidad o habilidad para percibir hechos y proposiciones y sus relaciones y razonar sobre ellos. Esencialmente significa pensar.*

Esta definición implica solamente inteligencia humana, no admite la posibilidad de que una máquina pueda pensar, ya que los programas no hacen la misma tarea de la misma forma que una persona. **Que un programa sea inteligente requiere que actúe intelligentemente, como un ser humano.**

**Un programa inteligente exhibe un comportamiento similar al de un humano cuando se enfrenta a un problema similar.**

No es necesario que el programa resuelva concretamente o intente resolver el problema de la misma manera que un humano.

Obsérvese que **el programa no necesita pensar como un humano**, pero **debe actuar como tal**.

Es difícil establecer una fecha de comienzo para lo que es comúnmente llamado IA. El primer paso se le atribuye a Alan Turing por su invención de la computadora de programas almacenados. Determinó que un programa podía ser almacenado como dato en la memoria de la computadora y ejecutado más tarde, anteriormente las computadoras fueron máquinas dedicadas que debían ser recableadas para diferentes problemas. El almacenamiento de programas permitía entonces cambiar la función de la computadora fácil y rápidamente.

El término inteligencia artificial se imputa a Marvin Minsky, investigador del MIT, quien escribió un artículo titulado "Pasos de la Inteligencia Artificial" (Enero 1961), que explicaba la posibilidad de hacer pensar a las computadoras.

Al final de los años '70 se habían alcanzado varios éxitos, tales como *el procesamiento de lenguaje natural, representación del conocimiento y resolución de problemas en áreas específicas de la IA*.

Los dos problemas más significativos de IA son los **sistemas expertos** y el **procesamiento de lenguaje natural**. A saber:

#### Sistema Experto

*Es un programa de computadora que contiene conocimientos acerca de un determinado campo y cuando es interrogado responde como un experto humano.*

Contiene información (*una base de conocimientos*) y una herramienta para comprender las preguntas y responder la respuesta correcta examinando la base (*un motor de inferencia*).

El PROLOG tiene incorporado estructuras para la creación de bases de conocimientos y un motor de inferencia.

#### Procesamiento de lenguaje natural

El procesamiento de lenguaje natural **es la técnica que fuerza a las computadoras a entender el lenguaje humano.**

Los científicos que lo estudian esperan crear un hardware y software que permita escribir "*llevar el archivo del prolog a la carpeta prolog*" y haga que la computadora siga dichas directrices.

El PROLOG puede usar la idea de una base de conocimientos y un motor de inferencias para dividir el lenguaje humano en diferentes partes y relaciones y así intentar comprender su significado detectando *palabras clave*.

## 4.2. Relación con la Lógica

Como su nombre lo indica, el **PROLOG se basa en manipulaciones lógicas, posibilita al programador especificar sus problemas en forma lógica**, en lugar de en términos de construcciones convencionales de programación sobre lo que debe hacer la computadora y en que momento.

Si queremos analizar como se relaciona PROLOG con la lógica debemos establecer primero que es lo que significa lógica: la **lógica** se desarrolló, originalmente como **una forma de representar argumentos de manera que fuera posible comprobar si éstos eran válidos o no**.

Se puede utilizar la lógica para expresar objetos, relaciones entre objetos y como pueden inferirse de forma válida algunos objetos a partir de otros. Por ejemplo, cuando decimos "*Eduardo tiene una PC*" estamos expresando una relación entre el objeto "*Eduardo*" y otro "*una PC*"; además, la relación tiene un orden específico: *ies Eduardo quien tiene una PC y no una PC quien tiene a Eduardo!*. Cuando realizamos la pregunta **¿Tiene Eduardo una PC?** Lo que estamos haciendo es indagar sobre una relación.

**PROLOG trabaja con lógica proposicional también conocida como lógica de predicados o cálculo proposicional.**

PROLOG hace que la computadora maneje la parte de inferir, **tiene un motor de inferencia incorporado que automáticamente busca los hechos y construye conclusiones lógicas**.

La programación de computadoras en PROLOG consiste en:

- Declarar algunos hechos sobre los objetos y sus relaciones.
- Definir algunas reglas sobre los objetos y sus relaciones.
- Hacer preguntas sobre los objetos y sus relaciones.

Podemos considerar a PROLOG como un almacén de hechos y reglas, que utiliza éstos para responder las preguntas, proporciona los medios para realizar inferencias de un hecho a otro.

Se puede considerar a PROLOG como un lenguaje coloquial, lo cual significa que el programador y la computadora sostienen una especie de conversación.

### 4.2.1. Hechos

La primera forma de combinar un objeto y una relación es usarlos para definir un hecho, la sintaxis de PROLOG es:

**relación (objeto).**

Supongamos que queremos decir a PROLOG el hecho de que "*a Eduardo le gusta la PC*", en PROLOG debemos escribir:

**le\_gusta\_a (eduardo,pc)**

Observe que el objeto está entre paréntesis y la relación le precede, lo que quiere decir que este objeto tiene esa relación. La **relación** se conoce como el **predicado** y el **objeto** como el **argumento**.

Los siguientes puntos son importantes:

- Los nombres de todos los objetos y relaciones deben comenzar con una letra minúscula.
- Primero se escribe la relación y luego los objetos separándolos mediante comas y encerrados entre paréntesis.
- Al final del hecho debe ir un punto (el carácter ".").
- El carácter "\_" en el nombre del predicado indica que todo es una única palabra para una relación.
- Dos hechos coinciden si sus predicados son lo mismo (se escriben de igual forma) y si cada uno de sus correspondientes argumentos son iguales entre sí.

Al definir relaciones entre objetos utilizando hechos, debemos prestar atención al orden en el que se escriben los objetos entre paréntesis. De hecho, el orden es arbitrario, pero debemos ponernos de acuerdo en un orden determinado y ser luego consecuentes con este orden.

Por ejemplo, en el hecho `le_gusta_a (eduardo,pc)` hemos puesto el "gusta" como el primero de los objetos entre paréntesis y el objeto "es gustado" en segundo lugar. Así el hecho `le_gusta_a (eduardo,pc)` no es lo mismo que `le_gusta_a (pc,eduardo)`, mientras que el primero nos indica que a eduardo le gusta la PC, el segundo nos dice que a la PC le gusta eduardo.

Los nombres de los objetos y relaciones son completamente arbitrarios. En vez de un término como `le_gusta_a (eduardo,pc)` podríamos representar lo mismo simplemente con `a(b,c)` y recordar que a significa `le_gusta_a`, b significa `eduardo` y c significa `PC`.

#### 4.2.2. Variables

En PROLOG no solo se pueden nombrar determinados objetos, sino que también **se pueden utilizar nombres como X que representen objetos a los que el mismo PROLOG les dará valor**, este tipo de nombres es lo que se llama **variables**.

Cuando el lenguaje PROLOG utiliza una determinada variable ésta puede estar instanciada o no instanciada. El primer caso se da cuando existe un objeto determinado representado por la variable, en caso contrario, una variable no está instanciada cuando, todavía no se conoce lo que representa.

Las variables deben comenzar con una letra mayúscula.

Cuando se intenta establecer una relación que contenga una variable, PROLOG efectuará una búsqueda recorriendo todos los hechos que él tiene almacenados para encontrar un objeto que pueda ser representado por la variable. Por ejemplo, cuando preguntamos *¿Le gusta X eduardo?*, PROLOG buscará entre todos sus hechos para encontrar cosas que le gusten a eduardo.

Una variable X, no nombra un objeto en particular en sí mismo sino que se puede utilizar para representar objetos que no podemos nombrar. Por ejemplo no podemos nombrar un objeto como algo que le gusta a eduardo, de forma que PROLOG adopta una forma de expresar esto:

En vez de preguntar:

`le_gusta_a (eduardo,Algo que le gusta a eduardo).`

Podemos utilizar:

`le_gusta_a (eduardo,X).`

Observación:

Las variables pueden tener nombres más largos. Ej: `Algoquelegustaaeduardo`.

A veces es necesario utilizar una variable aunque su nombre no se utilice nunca, supongamos que queremos averiguar si a alguien le gusta eduardo, pero no estamos interesados en saber quien, entonces utilizaremos la variable anónima. Esta variable es un único carácter de subrayado.

`le_gusta_a (_,_eduardo).`

Se utiliza para evitar el tener que imaginar continuamente diferentes nombres de variables cuando no se van a utilizar en ningún otro sitio de la cláusula.

#### 4.2.3. Reglas

En PROLOG se usa una regla cuando se quiere significar que un hecho depende de un grupo de otros hechos. Por ejemplo, si queremos afirmar que a eduardo le gustan todas las PCs del mercado, habría que escribir hechos por separado, así:

```

le_gusta_a (eduardo, ibm).
le_gusta_a (eduardo, compaq).
le_gusta_a (eduardo, acer).
le_gusta_a (eduardo, falcon).

```

Y esto para cada una de las PCs que tengamos.

En PROLOG una regla consiste en una cabeza y un cuerpo. Estas partes se encuentran separadas mediante el símbolo ":-", que está compuesto de un signo de dos puntos ":" y de un guión "-". El ":-" se pronuncia "si".

La cabeza describe qué hecho es el que la regla intenta definir, mientras que el cuerpo describe la conjunción de objetivos que deben satisfacer, uno tras otro, para que la cabeza sea cierta.

```

es(compaq,pc).
es(falcon,pc).
es(coupe,auto).
.
.
.
le_gusta_a(eduardo,objeto):-es(objeto,pc).

```

Una forma más simple de hacerlo es decir que a Eduardo le gusta cualquier objeto siempre que éste sea una PC. Este hecho se da en forma de una regla sobre lo que le gusta a Eduardo, en vez de dar la relación de todas las PC que le gustan a Eduardo.

La regla es mucho más compacta que una lista de hechos. ***Las reglas hacen que el PROLOG pase de ser solo un diccionario o una base de datos, en el que se pueda buscar, a ser una máquina lógica, pensante.***

Ejemplos de reglas:

**Marco compra vino si es más barato que la cerveza.**

**X es un pájaro si: (con uso de variables)**

**X es un animal y  
X tiene plumas.**

***Una regla es una afirmación general sobre objetos y sus relaciones.***

Así podemos permitir que una variable represente un objeto diferente en cada uso diferente de la regla. Por ejemplo:

- a Marco le gusta cualquiera al que le guste el vino, o en otras palabras,
- a Marco le gusta algo si a esto le gusta el vino, o con variables,
- a Marco le gusta X si a X le gusta el vino.

El ejemplo anterior se escribe en PROLOG de la forma siguiente:

```
le_gusta_a(marco,X) :- le_gusta_a(X,vino).
```

#### 4.2.4. Cláusulas

Utilizaremos la palabra **cláusula** siempre que nos refiramos a **un hecho o a una regla**.

Existen dos formas de dar información a PROLOG sobre un predicado dado, como *le\_gusta\_a*. Podemos darle tanto hechos como reglas, en general, un predicado está definido por una mezcla de hechos y reglas. A uno y otras se los denomina como cláusulas de un predicado. Por ejemplo consideremos la regla:

una persona puede robar una cosa si la persona es un ladrón y le gusta la cosa y la cosa es valiosa.

En PROLOG sería:

```
puede_robear(X,Y) :- ladron(X),le_gusta_a(X,Y),valiosa(Y).
```

El predicado *puede\_robear* significa que alguna persona X puede robar alguna cosa Y.

Esta cláusula depende de las cláusulas *ladron*, *le\_gusta\_a* y *valiosa*.

#### 4.2.5. Preguntas

Una vez que tengamos algunos hechos podemos hacer algunas preguntas acerca de ellos. En PROLOG una pregunta se representa igual que un hecho, salvo que delante se pone un símbolo especial: ?-

Por ejemplo:

```
?- tiene(eduardo,pc).
```

lo que podemos interpretar como *¿tiene eduardo una pc?*

**Cuando se hace una pregunta PROLOG efectúa una búsqueda por toda la base de datos, localizando hechos que coincidan con el hecho en cuestión** (Ya se explicó en puntos importantes de los hechos, cuando dos hechos coinciden). Si se encuentra uno que coincida se responderá *si* (*yes*), por el contrario, si no se encuentra, la respuesta será *no*.

En PROLOG la respuesta no se utiliza para determinar que no existe nada que coincida con la pregunta realizada.

#### 4.2.6. Conjunciones y Backtracking

En el caso de que se quiera contestar preguntas sobre relaciones un poco más complejas, como *¿se gustan José y María?*, primero se debería preguntar si José gusta de María y luego, si la respuesta fue *yes*, si María gusta de José. **Esto representa un problema con dos objetivos separados**. El "y" expresa el interés de la **conjunción** de los dos objetivos: lo que se quiere hacer es satisfacer ambos, primero uno y luego el otro. Esto en PROLOG sería:

```
?- le_gusta_a(jose,maria), le_gusta_a(maria,jose).
```

La coma se lee "y" y separa los objetivos de una pregunta. Cuando PROLOG tiene que satisfacer objetivos, lo hará objetivo por orden, buscando los coincidentes en la base de datos. Deberá ser necesario que se cumplan todos, en orden.

Combinando las conjunciones con el uso de variables se puede contestar por ejemplo:

*¿Hay algo que le guste a José y María?*

Lo que se haría sería buscar primero si hay algún A que le guste a María y luego si ese A también le gusta a José. Que se expresa:

```
?- le_gusta_a(maria,A), le_gusta_a(jose,A).
```

PROLOG buscará el primer objetivo en la base, si lo encuentra marcará el lugar e intentará satisfacer el segundo, si lo logra se marcará también dicha posición en la base de conocimientos, lo que determina que se encontró la solución.

Si por el contrario el segundo objetivo no se logra, PROLOG intentará resatisfacer el objetivo anterior (buscar otra solución). Lo que hace es **volver hacia atrás e intentar resatisfacer su anterior objetivo comenzando, no desde el principio de la base, sino de su correspondiente marca de posición** y luego satisfacer el segundo objetivo.

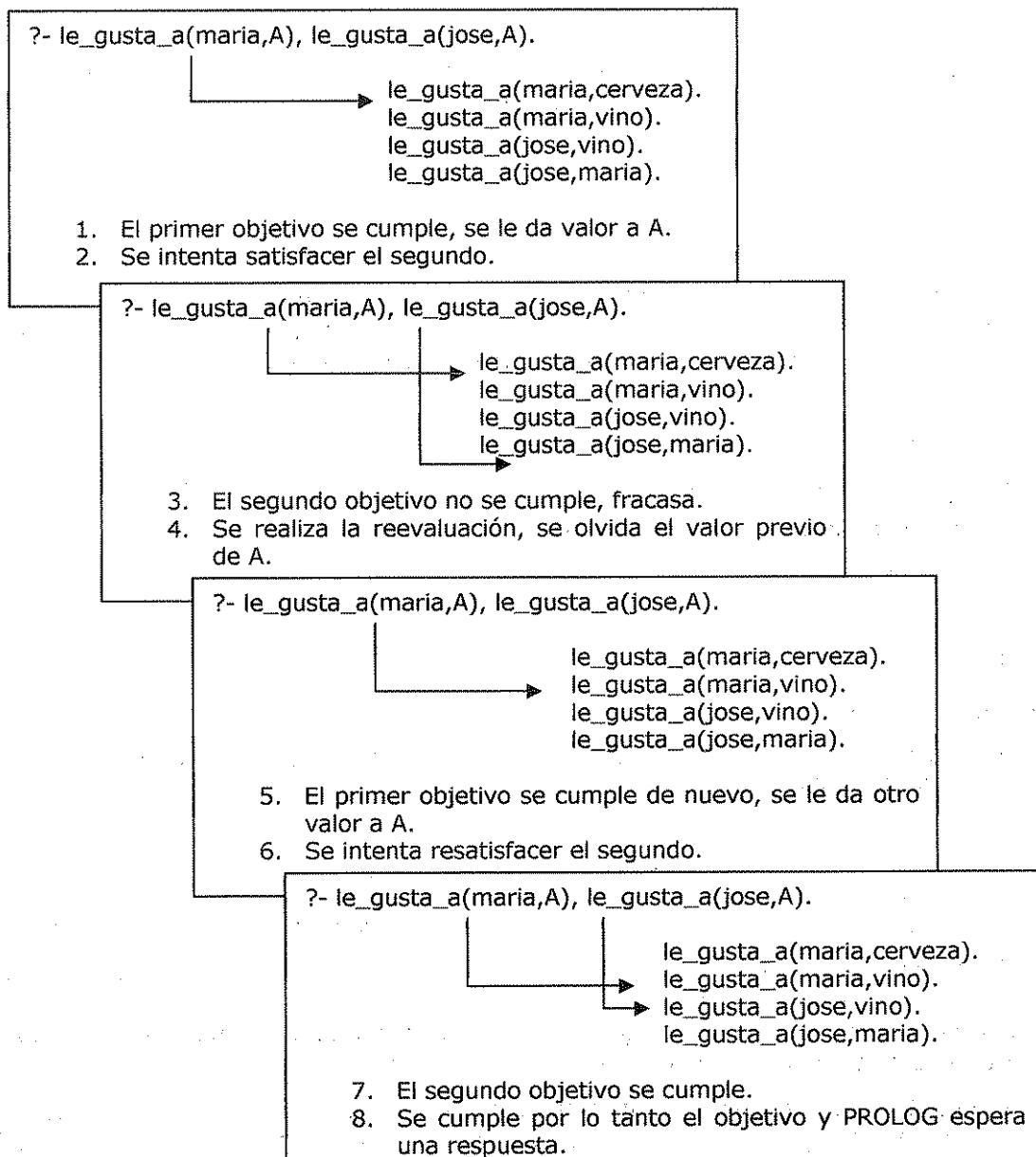
A este comportamiento del PROLOG de intentar repetidamente satisfacer o resatisfacer los objetivos de una conjunción, se lo llama **REEVALUACION** (*o backtracking*).

Ejemplo: supongamos la siguiente base de conocimientos\_

```
le_gusta_a(maria,cerveza).  
le_gusta_a(maria,vino).  
le_gusta_a(jose,vino).  
le_gusta_a(jose,maria).
```

y la pregunta `?- le_gusta_a(maria,A), le_gusta_a(jose,A).`

Ilustra el comportamiento de "reevaluación" de la siguiente forma:



En este punto los objetivos se han satisfecho y la variable A contiene el valor *vino*. El primer objetivo tiene una marca en la base en la posición del hecho *le\_gusta\_a(maria,vino)*, y el segundo en el hecho *le\_gusta\_a(jose,vino)*.

### 4.3. Estructura de un Programa en Prolog

#### 4.3.1. Composición de un programa. Cláusulas. Predicados. Dominios

La mayoría de los programas PROLOG están organizados en 4 etapas:

- Cláusulas
- Predicados
- Dominios
- Objetivos

Aunque no todas estas secciones necesitan estar presentes en todos los programas, debe familiarizarse con todas ellas.

#### Cláusulas

Los hechos que construyó con los objetivos y las relaciones se listan en la sección de cláusulas (*clauses*). La sección puede también contener reglas y otras construcciones que se estudiarán más adelante.

#### Predicados

Los *predicados* son las relaciones. El término "*predicado*" viene de la lógica formal, la cual es uno de los fundamentos iniciales de PROLOG. Siempre que su programa vaya a usar un predicado particular en las cláusulas, necesita declararlo formalmente en la sección "*predicates*". Una típica declaración de predicados puede ser como ésta:

**nombrepredicado (argumento1, argumento2)**

Se pueden tener múltiples declaraciones de predicados, en donde el mismo predicado esté definido más de una vez, cada vez con diferentes dominios de argumentos, como en:

nombrepredicado (argumento1, argumento2)

nombrepredicado (argumento3, argumento4, argumento5)

o

tiene\_dedos\_pie (persona, número)

tiene\_dedos\_pie (animal, número)

podrá utilizar los predicados de las dos formas. Los diferentes argumentos tampoco tienen que ser del mismo tipo de dominio.

#### Dominios

El PROLOG necesita un nivel más de explicación antes de que un programa esté completo.

Necesita decirle los argumentos que van a usar los predicados. Para usar una mínima cantidad de memoria y hacer a los programas más fáciles de depurar, el PROLOG tiene muchas *comprobaciones de tipo* con las que los programadores en Pascal están familiarizados. Necesita conocer de antemano que "*tipo*" de cosas puede ser un argumento.

Una vez que el PROLOG sabe que argumentos van con cada predicado, necesita saber con qué dominio (*que tipo de datos*) trabajarán los argumentos. Los mismos argumentos pueden usarse con varios predicados. Por ejemplo:

**tiene\_dedos\_pie (persona, número)**  
**tienden\_dedos\_mano (persona, número)**

### Tipos de dominios estándares

#### Symbol

Hay dos tipos de símbolos:

1. Un grupo de caracteres consecutivos (letras, números y signos de subrayado) que comienzan con un carácter en minúscula. Por ejemplo:

eduardo  
 eduardo\_coche  
 la\_clase\_de\_Eduardo

2. Un grupo de caracteres consecutivos (letras y números) que comienzan y terminan con una marca de dobles comillas ("")

Los símbolos y cadenas parecen iguales, pero el PROLOG los interpreta diferentes. Los símbolos necesitan más memoria que las cadenas, pero se identifican más rápidamente cuando se ejecuta un programa. Por ejemplo:

"Eduardo"  
 "choqué contra un árbol el coche de Eduardo"

#### String

Cualquier grupo de caracteres consecutivos (letras y números) que comienza y termina con una marca de doble comilla (""). Esto es lo mismo que el segundo tipo de símbolos descrito anteriormente. Sin embargo, recuerde que los símbolos y cadenas no son tratados de la misma forma por Prolog. Por ejemplo:

"eduardo"  
 "eduardo es un @"a&&"?i¿@"  
 "eduardo escucha Volcan"

#### Integer

Cualquier número comprendido entre (e incluyendo) -32768 y 32767. El límite viene impuesto porque los enteros se almacenan como valores de 16 bits. Quince de estos bits representan el número y el otro bit representa el signo. Por ejemplo:

9  
 -200  
 21444

#### Real

Cualquier número real en el rango +/- 1 E-307 a +/- 1 E+308. El formato incluye estas opciones: signo, número, punto decimal, fracción, E (para indicar el exponente), signo para el exponente, exponente. Los números reales pueden variar desde los muy sencillos hasta los muy complejos. Por ejemplo:

3

-31615926525E-218

(Ambos números son reales, pero en muchos casos sería mejor hacer que el primero fuera entero en vez de real. El entero necesita menos memoria)

### Char

Cualquier carácter de la lista ASCII estándar, posicionado entre dos marcas de comillas sencilla (' '). Por ejemplo:

't'

'x'

'&'

### File

Para declarar un dominio de archivo, se debe declarar los nombres de archivos simbólicos de los archivos que se usarán. Solo se puede tener una declaración de dominio de archivo en un programa pero dicha declaración puede contener varios nombres de archivos simbólicos.

Todos los nombres de archivos simbólicos deben comenzar con una letra minúscula. Hay dos tipos de archivos:

1. *Archivos predefinidos*: son los archivos que están siempre disponibles en PROLOG.

Por ejemplo:

printer

keyboard

screen

com 1

2. *Archivos definidos por el usuario*: son los archivos a los que uno da nombres. Puede usarse cualquier nombre que no esté reservado por PROLOG. Por ejemplo:

estearchivo

esearchivo

el\_otro\_archivo

### Objetivos (goal)

Esta es la sección que le dice al PROLOG lo que ha de encontrar o lo que desea que la computadora haga con la información que se le ha suministrado en las otras tres secciones.

Normalmente un programa tendrá al menos las secciones de predicados y cláusulas, pero no es posible tener un programa que tenga solo la sección objetivos.

Un programa PROLOG puede tener dos tipos diferentes de goal: interno o externo.

- Si un programa tiene un goal interno se ejecutará y automáticamente procederá a aprobar o desaprobar el objetivo y le informará de lo obtenido.
- Si en cambio el programa posee un goal externo comenzará a ejecutarse y luego esperará que se le introduzca un objetivo. Una vez que se verifica si se junta el objetivo, el programa le pedirá uno nuevo y así sucesivamente.

### 4.3.2. Ejemplos

predicates  
    hacerlo

goal  
    hacerlo.

Este programa incluye una regla sin parámetros, que imprime un cartel

clauses

    hacerlo :- write ("Aire, Fuego, tierra, Agua").

/\* Programa con goal externo \*/

domains  
    persona = symbol

predicates  
    observa (persona, persona)

clauses

    observa(eduardo, nestor).  
    observa(javier, tatiana).  
    observa(marco, veronica).

Este programa pedirá que se introduzca un goal en la ventana de diálogo para comenzar a ejecutarse.  
Por ejemplo:  
goal: observa(eduardo,tatiana)  
El resultado será False

/\* Programa con goal interno \*/

domains  
    persona = symbol

predicates  
    observa (persona, persona)

goal  
    observa (eduardo, tatiana).

Este programa posee un goal interno, se ejecutará en forma automática y el resultado será False

clauses

    observa (eduardo, nestor).  
    observa (javier, tatiana).  
    observa (marco, veronica).

/\* Es posible no definir ..domains..  
y luego colocar el tipo de las  
variables que las relaciones  
usan en ..predicates.. \*/

predicates  
    observa (symbol, symbol)

goal  
    observa (eduardo, tatiana).

clauses

    observa (eduardo, nestor).  
    observa (javier, tatiana).  
    observa (marco, veronica).

```
/* Programa que incluye una regla por la
   cual se define el concepto de hermana */
```

```
domains
    nombre = symbol

predicates
    hombre (nombre)
    mujer (nombre)
    padres (nombre, nombre, nombre)
    hermana (nombre, nombre)

clauses
    hombre (jorge).
    hombre (german).
    mujer (tatiana).
    mujer (verónica).
    padres (tatiana, verónica, jorge).
    padres (german, verónica, jorge).
    hermana (X,Y) :- mujer (X), padres (X,M,P), padres (Y,M,P), X<>Y.
```

La regla `hermana (tatiana,german)` determina si tatiana es hermana de german. La primer conjunción `mujer(X)`, buscará una mujer con el nombre tatiana. La segunda, `padres(X,M,P)`, devolverá los padres de tatiana si la/s conjunción/es anteriores fueron verdaderas, en este caso será M=verónica y P=jorge. La siguiente buscará si existe una cláusula `padres(Y,M,P)` con valores Y=german, M=verónica y P=jorge. Esto evaluará si los padres de tatiana son los mismos que los de german. Finalmente se evalúa si las personas X e Y no son la misma.

Cada vez que se evalúa una conjunción las conjunciones anteriores tuvieron que ser verdaderas. En caso contrario no se evaluarán las siguientes conjunciones.

#### 4.4. Ingeniería del Conocimiento

*La Ingeniería del Conocimiento es la disciplina que trata de la forma en que se organizan, construyen y verifican las bases de conocimiento.*

*El Ingeniero de Conocimiento es el encargado de entrevistar a los expertos reales, para aprender sobre lo que ellos saben, y poder así organizar la información obtenida para construir una Base de Conocimientos.*

##### 4.4.1. Organización de una Base de Conocimientos

Como creador de una base de conocimientos usted puede controlar la organización de la información dentro de la misma, lo que implica que *puede haber ordenaciones mejores o peores*.

- A menudo, una buena forma de organizar la información es **situar los objetos más probables lo más cerca del comienzo**. El problema es decidir qué objetos son los más y los menos probables. Usted puede crear algunas veces la base de conocimientos aleatoriamente y dejar que sea usada un período corto mientras registra el número de veces que se selecciona cada objeto. Usando estas frecuencias, puede entonces reordenar la base de conocimientos.
- Otra forma de organizar la información es **situar cerca del comienzo de la base de conocimientos aquellos atributos que causan la poda de la mayor parte del "árbol"**. Para bases de conocimientos grandes, de nuevo puede ser difícil determinar qué atributos causan el mayor efecto.

##### 4.4.2. Encontrando el Experto

La única hipótesis que se ha hecho hasta ahora es que se puede encontrar un experto humano quien puede extraer fácilmente el conocimiento del experto. Desafortunadamente, rara vez es éste el caso.

- Primero, a menudo **es difícil determinar quien es o no es un verdadero experto**. Este es un problema especialmente cuando se conoce poco sobre la materia.
- Segundo, **dos expertos diferentes en la misma materia tienen a menudo opciones contradictorias**, lo que hace difícil saber cual usar.

Cuando las opiniones de los expertos difieren, generalmente puede elegir entre 3 opciones útiles.

- **Primera**, puede **seleccionar un experto e ignorar al otro**. Esta es claramente la más fácil, pero puede significar que la base de conocimientos puede contener alguna información errónea. Pero la cantidad de información errónea no es más que la que el experto puede proporcionar.
- **Una segunda solución es promediar la información**. Esto es, intentar **usar solo información que es común entre los expertos**. Aunque esta opción no es tan fácil como la primera, usted puede hacer el proceso ocasionalmente sin mucho problema. La desventaja es que la base de conocimientos esté "revuelta" y no refleje el conocimiento pleno de ningún experto.
- **La solución final es incluir el conocimiento de ambos expertos en el sistema y dejar al usuario decidir**. Podría añadir un factor de probabilidad para cada elemento. Aunque esto puede ser aceptable en algunas situaciones, para muchas aplicaciones, querrá que el sistema experto sea la autoridad, y no querrá forzar al usuario a decidir.
- Otro punto problemático es que **la mayoría de los expertos humanos no saben lo que saben**. Los humanos no pueden hacer un volcado de memoria de la misma forma que una computadora. En consecuencia, **puede ser difícil extraer toda la información necesaria**.

- Además, **algunos expertos estarán simplemente menos inclinados a perder tiempo** diciéndole todo lo que saben sobre la materia.

#### 4.4.3. Verificaciones de la BC

Si supone que ha encontrado un experto que es cooperativo y que da una descripción completa de su materia de experto, todavía se enfrenta con el problema de **verificar que ha transcripto e introducido correctamente los conocimientos en la computadora**. En esencia, esto significa que debe **testear la base de conocimientos**.

La pregunta es: ¿Cómo se testea una base de conocimientos?

La búsqueda exhaustiva es imposible sobre algo que no sea un conjunto de datos extremadamente pequeño debido a la explosión combinatoria.

Esto significa que *para la mayoría de los sistemas expertos del mundo real no hay forma de verificar completamente que la base de conocimientos sea exacta*.

- En consecuencia, la mejor solución es hacer suficiente testeo de forma que pueda confiar bastante en la base de conocimientos. **Usando las técnicas de muestreo estadísticas, puede idear una serie de tests que producirán cualquier nivel de confianza que se desee.**
- En otro enfoque, tiene la **autocomprobación del sistema por consistencia y ver que toda la información de la base de conocimientos concuerda consigo misma**. Aunque esta no encontrará todos los problemas, encontrará algunos. Sin embargo, dependiendo de cómo esté implementada la autocomprobación, en algunas bases de datos grandes puede alcanzar la muralla de piedra combinatoria, haciendo este enfoque imposible.

De este modo, según crece el uso de los sistemas expertos, la verificación de las bases de conocimientos será una de las más importantes áreas de investigación.

#### 4.5. Definición de Sistema Experto

Asumiendo el riesgo que comporta toda simplificación, podríamos concretar en dos las *ideas claves que han conducido a la construcción de los llamados "sistemas expertos" y a su popularización*.

1. Por una parte, el **énfasis en la representación, adquisición y uso del conocimiento especializado**: lo que diferencia específicamente a una persona experta en un área del saber de otra que no lo sea radica más sus conocimientos concretos sobre el área en cuestión que en sus capacidades generales para resolver problemas (Elstein et al.; 1978).
2. Por otra, el **reconocimiento de que los sistemas, además de resolver problemas, deben poseer otros atributos que se suponen propios de los expertos humanos**:
  - Capacidad para adquirir nuevos conocimientos y para matizar o perfeccionar los que ya poseen.
  - Capacidad para justificar sus conclusiones.
  - Capacidad para explicar por qué hacen sus preguntas cuando están intentando resolver un problema.
  - Capacidad conversacional para todo ello.

#### 4.5.1. Definición funcional

*Para que un sistema informático pueda llamarse "experto", ha de satisfacer un criterio similar al conocido "test de Turing". Es decir, que visto como una "caja negra", sea indistinguible por su comportamiento de un experto humano.*

El "comité de Sistemas Expertos" de la British Computer Society da la siguiente definición (cf. Naylor, 1983):

" se considera que un sistema experto es:

- la incorporación en un ordenador de un componente basado en el conocimiento
- que se obtiene a partir de la habilidad de un experto,
- de forma tal que el sistema pueda dar consejos inteligentes o tomar decisiones inteligentes...
- Una característica adicional deseable, y que para muchos es fundamental, es que el sistema sea capaz, bajo demanda, de justificar su propia línea de razonamiento de una forma inmediatamente inteligible para el que lo usa".

Hayes Roth(1984a), sin dar exactamente una definición, concreta algo más, dando la siguiente relación de funciones que un sistema experto debe ser capaz de realizar:

- Resolver problemas muy difíciles tan bien o mejor que un experto humano.
- Razonar heurísticamente, utilizando reglas que los expertos humanos consideran eficaces.
- Interactuar eficazmente y en lenguaje natural con las personas.
- Manipular descripciones simbólicas y razonar sobre ellas.
- Funcionar con datos erróneos y reglas imprecisas.
- Contemplar simultáneamente múltiples hipótesis alternativas.
- Explicar por qué plantean sus preguntas.
- Justificar sus conclusiones.

#### 4.5.2. Definición estructural

Habitualmente, cuando diseñamos un sistema informático para resolver una clase de problemas, *mezclamos en el código*, de manera más o menos desordenada, *los conocimientos concretos para abordar problemas de esa clase y los procedimientos que, actuando sobre esos conocimientos, permiten resolver cada problema específico*.

Ahora bien, puesto a diseñar un sistema con unas especificaciones funcionales que satisfagan las características enunciadas anteriormente, es aconsejable **separar claramente los dos componentes: conocimientos y procedimientos**. De este modo será más fácil conseguir que el sistema posea dos características básicas:

- Flexibilidad para adquirir y modificar sus conocimientos.
- Transparencia en la explicación de sus razonamientos y conclusiones.

Ello se traduce en la aparición de un nivel estructural nuevo en el diseño de los sistemas, que, siguiendo a Sowa (1984), podemos ilustrar con las siguientes fases evolutivas:

**A.** En un “*sistema elemental*”, el *programa contiene no solo el conocimiento y los procedimientos que hacen uso de él, sino también las informaciones sobre la forma de comunicación con el usuario y las formas de acceder a los datos* (Figura 4.1). Esta concepción estructural plantea el doble problema de que las posibles decisiones de modificar las estructuras de datos, por una parte, y el modo de comunicación con el usuario, por otra, exigen cambios en el programa. Y son bien conocidas las consecuencias negativas que, por efectos laterales, pueden provocar los “pequeños cambios” en un programa complejo.

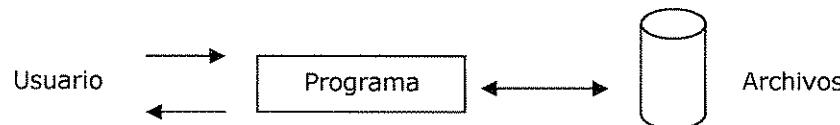


Figura 4.1  
Sistema “elemental”

**B.** Las soluciones a esos problemas son bien conocidas. Por una parte, *el programa se independiza de los formatos de comunicación con el usuario* gracias a la capa de entrada-salida del sistema operativo. Por otra, *se independiza de las estructuras de datos mediante el uso de un sistema de gestión de bases de datos*, como indica la Figura 4.2.

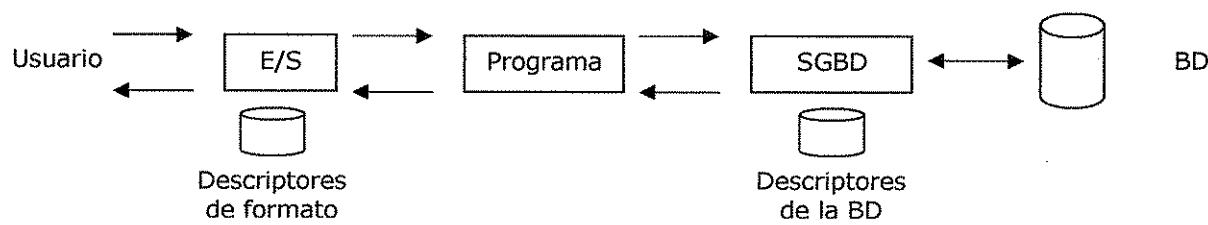


Figura 4.2  
Sistema con subsistema de E/S y BD

No obstante, persisten los problemas anteriores a otro nivel: todo intento de modificación del conocimiento exige cambios en el programa.

**C.** En un *sistema experto* se *independiza el conocimiento de los procedimientos que hacen uso de él*, por tanto, dos módulos diferenciados: *la base de conocimiento y el motor de inferencia*, como se observa en la Figura 4.3.

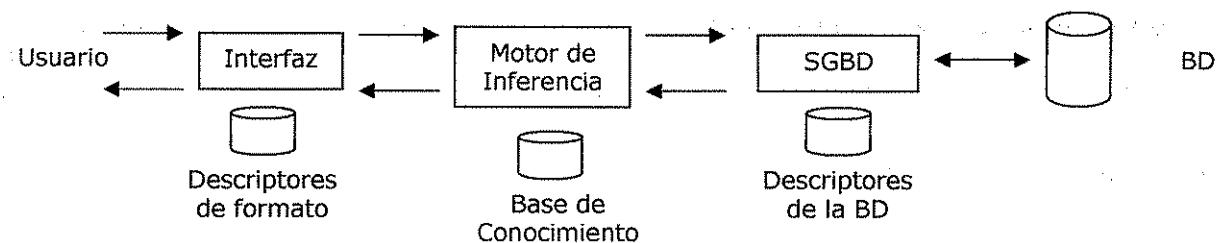


Figura 4.3  
Sistema con Base de Conocimientos

Así, podemos distinguir tres **componentes estructurales básicos** en un sistema experto:

- La **Base de hechos** (como se suele denominar a lo que en la figura aparece como Base de Datos), que contiene el conocimiento declarativo, a nivel de datos, sobre el problema particular que en un momento dado se intenta resolver y sobre el estado del sistema en cada instante.
- La **Base de Conocimiento**, formada por el conocimiento específico y procedimental acerca de la clase de problemas en los que el sistema es experto.
- El **Motor de inferencia**, que controla al resto del sistema en sus funciones deductivas.

Por otra parte, tres son los **tipos de personas que se relacionan con un sistema experto**:

- El **usuario final**, que dialoga con el sistema para resolver problemas o para aprender.
- El **experto humano**, que comunica su conocimiento para construir el sistema.
- El **ingeniero de conocimiento**, que diseña las estructuras de datos más adecuadas para la representación del conocimiento y que traduce a tales estructuras (bien directamente o construyendo herramientas adecuadas) los conocimientos del experto humano.

Hay que señalar que la idea de separar los conocimientos de los procedimientos de inferencia no es nueva: estaba ya presente en el GPS (General Problem Solver) de Newell et al. (1959).

Lo que diferencia específicamente a la línea de trabajo actual sobre sistemas expertos de los trabajos "clásicos" sobre resolución automática de problemas son los dos puntos que señalábamos al principio:

- concentrarse en áreas muy específicas del saber y no en "*problemas generales*" y
- dar importancia a que los sistemas puedan explicar sus razonamientos y justificar sus conclusiones al usuario.

Esto último obliga a considerar otro subsistema (Justificación y Explicación) para esas funciones, además de la interfaz para comunicación en lenguaje próximo al natural. Puede, asimismo, construirse otra interfaz para que el experto humano transfiera su conocimiento o perfeccione el que ya tiene el sistema. Cuanto más elaborado sea este módulo menos necesario será la intervención del ingeniero de conocimiento en esa fase de adquisición de conocimiento.

Si bien las expresiones "**Sistema Experto**" y "**Sistema basado en Conocimientos**" suelen utilizarse indistintamente y de manera intercambiable, parece que la primera podría asociarse de manera más natural con la "visión funcional", mientras que la segunda haría referencia a la "visión estructural".

#### 4.6. Armazones de Sistemas Expertos

En un principio, los sistemas expertos que se construían partieron de cero, generalmente en LISP. Pero tras haber creado varios sistemas de esta manera, se vio claro que, a menudo, *tenían bastantes puntos en común*. En concreto, desde que los sistemas se creaban como un conjunto de representaciones declarativas (reglas en su mayor parte) combinadas con un intérprete para esas declaraciones, *fue posible separar el intérprete y el conocimiento específico del dominio*, y así *se pudo crear un sistema capaz de construir nuevos sistemas expertos sin más que añadir nuevos conocimientos correspondientes al dominio del nuevo problema*.

Los intérpretes resultantes se llaman **armazones (shells)**.

Un ejemplo significativo de lo que es un armazón lo constituye EMYCIN (de Empty MYCIN) (Buchanan y Shortliffe, 1984), que se derivó de MYCIN.

En este momento hay disponibles comercialmente varios armazones que sirven como base para muchos de los sistemas expertos que actualmente se construyen. Estos armazones proporcionan mucha mayor flexibilidad a la hora de la representación del conocimiento y del razonamiento frente a la proporcionada por MYCIN. *Típicamente tienen como base reglas, marcos, sistemas de mantenimiento de la verdad y otros muchos mecanismos de razonamiento.*

- Los primeros armazones de sistemas expertos proporcionaban mecanismos para la **representación del conocimiento, el razonamiento y la explicación**.
- Más tarde, se les añadieron herramientas para la **adquisición del conocimiento**.
- Pero conforme aumenta la experiencia a la hora de utilizar estos sistemas para resolver problemas del mundo real, parece evidente que los armazones de los sistemas expertos necesitaban, además, hacer algo más: necesitaban facilitar las cosas para **integrar sistemas expertos con otros tipos de programas**.

Los sistemas expertos no pueden actuar aislados, algo que sus homólogos humanos si pueden hacer. Los sistemas expertos necesitan acceder a bases de datos colectivas, y acceder a ellas supone ser controlados igual que lo son otros sistemas. A menudo son imbuidos dentro de programas de aplicación más grandes que usan técnicas de programación convencionales principalmente.

*De modo que una de las características importantes que debe tener un armazón es un interfaz fácil-de-usar entre un sistema experto escrito con el armazón y un gran entorno de programación, que probablemente será más convencional.*

## 4.7. Aplicaciones de los Sistemas Expertos

### 4.7.1. Ventajas de la aplicación de Sistemas Expertos

La utilidad de un sistema experto se basa principalmente en la **eficacia y conveniencia**.

1. A diferencia de un experto humano que tiene que dormir, comer, descansar, tomarse vacaciones y otras cosas, el sistema experto está accesible para usarse veinticuatro horas al día, todos los días del año.
2. Además se pueden crear muchos sistemas expertos, mientras que el número de expertos humanos puede ser limitado, lo cual hace virtualmente imposible en muchas situaciones tener un experto disponible cuando se necesita.
3. Además, a diferencia de los humanos, los expertos informatizados nunca mueren, llevándose el conocimiento con ellos. El conocimiento de un sistema experto puede copiarse y almacenarse fácilmente siendo excepcional la pérdida permanente del conocimiento del experto.
4. Otra ventaja de un sistema experto sobre los expertos humanos es que el experto informatizado está siempre rindiendo al máximo. Cuando un experto humano se cansa, puede resentirse la fiabilidad del consejo del experto. Sin embargo el experto informatizado generará siempre la mejor opinión posible, dentro de las limitaciones de su conocimiento.
5. Una desventaja menos importante de un sistema experto es su falta de personalidad. Probablemente se habrá dado cuenta de que no todas las personalidades son compatibles. Si no se lleva bien con un experto, puede tener reservas para utilizar el conocimiento de un experto. La situación contraria también es cierta: un experto humano a quien no le guste usted puede no ser capaz de proporcionarle información fiable.
6. Una ventaja final de un sistema experto es que después de tener un experto informatizado, puede adquirir un nuevo experto simplemente copiando el programa de una máquina a otra. Un humano necesita largo tiempo para convertirse en un experto en ciertos campos, lo cual hace difícil adquirir nuevos expertos humanos.

#### 4.7.2. Fases de la "inserción social" de los Sistemas Expertos

Hayes-Roth (1983) esquematiza en la Figura 4.4 la secuencia de ideas, actividades y problemas que se suceden desde que se concibe la posibilidad de la construcción de un sistema experto en una determinada área de conocimiento hasta que el sistema se integra en las estructuras industriales y sociales y, eventualmente, contribuye a la transformación de éstas.

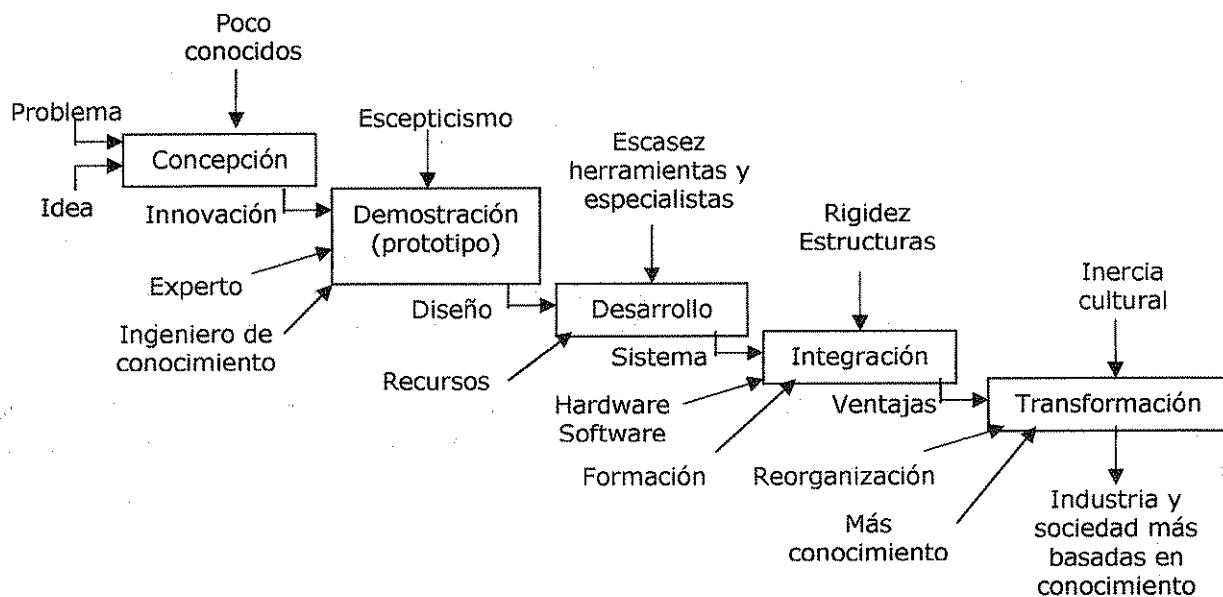


Figura 4.4

#### 4.8. Sistemas Expertos más conocidos

##### Evolución de los sistemas expertos.

De acuerdo con Cuenca (1984, 1985a,b), podemos distinguir cuatro etapas en la evolución histórica de los sistemas expertos:

**Etapa de invención (1965-1970):** en la que se construyen sistemas que hoy consideramos precursores, algunos de los cuales siguen utilizándose, por ejemplo:

**DENDRAL**, desarrollado en la Universidad de Stanford, deduce la estructura química molecular de un compuesto orgánico a partir de su fórmula y de datos espectrográficos y de resonancia magnética nuclear (Buchanan et al., 1969), y utilizado por varias empresas farmacéuticas americanas a través de la red de información médica Sumex- Aim.

**MACSYMA**, en el MIT, experto en cálculo diferencial e integral mediante manipulación simbólica de expresiones algebraicas (Martín y Fateman, 1972), y comercializado por Symbolics.

**Etapa de prototipos (1970-1977):** en la que se diseñan los sistemas expertos más conocidos actualmente, por ejemplo:

**INTERNIST**, en la Universidad de Pittsburgh, para diagnóstico en medicina interna (Pople et al., 1975; Myers y Pople, 1977). Este sistema es, por cierto, una excepción, en el sentido de que abarca un campo de saber relativamente amplio: alrededor del 80% de la medicina interna, según sus autores. Una de las principales críticas que ha recibido ha sido, precisamente, sobre la poca profundidad de sus conocimientos médicos; por ello, se han desarrollado versiones posteriores (la última se llama **CADUCEUS**, cf. Myers et al., 1982) que trata de incluir posibilidades de razonamiento basadas en conocimientos anatómicos y fisiológicos.

**MYCIN**, en la Universidad de Stanford, para diagnósticos y tratamientos de enfermedades infecciosas (Shortliffe, 1976).

**CASNET**, en la Universidad de Rutgers, para diagnóstico y tratamiento en oftalmología (Weiss et al., 1978).

**PROSPECTOR**, en el Stanford Research Institute, para exploración geológica (Duda et al., 1976). Especialmente conocido desde que con su ayuda se descubrió un importante depósito de molibdeno, valorado en 100 millones de dólares, en el estado de Washington.

**Etapa de experimentación (1977-1981):** en la que las ideas obtenidas sobre la representación del conocimiento en los prototipos se van asentando, y aparecen los "sistemas esenciales" o "armazones", con los que se desarrollan rápidamente otros muchos sistemas expertos.

*Un sistema esencial (también llamado "sistema vacío" o "shell") viene a ser el resultado de "eliminar" el conocimiento de la base de conocimiento y conservar todos los demás módulos.* ¶

Es así una herramienta potente para construir sistemas expertos en otras áreas del saber distintas de la original, si bien presenta la eliminación de la representación del conocimiento ha de hacerse necesariamente en estructuras prefijadas.

Así, con **EMYCIN** (Esencial MYCIN, cf. Van Melle, 1980), se construyeron **PUFF** (para enfermedades pulmonares, cf. Kuntz et al., 1978), **CLOT** (para problemas de coagulación cf. Bennet y Goldman, 1980), **HEAMED** (para psicofarmacología, cf. Heiser y Brooks, 1978), etc. Se dotó además, a EMYCIN de módulos de uso general para comunicación, **BAOBAB** (Bonnet 1984), para adquisición y modificación del conocimiento, **TEIRESIAS** (Davis, 1979) para explicación y funciones de tutoría, **GUIDON** (Clancey, 1979). Sin embargo otros sistemas diseñados en Stanford no pudieron hacer uso de EMYCIN debido a que el conocimiento exigía la consideración de atributos con valores cambiantes en el tiempo, **VM** (para cuidados intensivos, cf. Shortliffe et al., 1981).

De manera similar, de PROSPECTOR se derivó **KAS** (Knowledge Acquisition System, Duda et al., 1978) y de CASNET, **EXPERT** (Weiss y Kulikowski ,1979).

**Etapa de industrialización (desde 1981):** año en que se fundó la primera empresa dedicada a sistemas expertos y herramientas para su desarrollo, Teknowledge.

En esta etapa diversas empresas establecidas (Fairchild, Xerox, Texas, etc.) se interesan por este nuevo campo y se crean otras nuevas dedicadas exclusivamente a él (Teknowledge, Syntelligence, Cognitive Systems, etc.) y en la que aparecen los primeros sistemas con interés comercial. (En particular, IBM, que años atrás había prácticamente abandonado la inteligencia artificial, ahora adquiere y comercializa su primer producto: Intellec).

Dos ejemplos bien conocidos son **R1** (luego llamado XCON), diseñado en Digital Equipment Corporation para configurar sistemas con PDP y VAX (con el que la propia empresa estima que ahorra unos diez millones de dólares anuales y **Drilling Advisor**, desarrollado por Teknowledge a través de su filial francesa, Framentec, por encargo de Elf-Aquitaine para detectar averías en los equipos de sondeo geológico.

Y es también en 1981 cuando se presenta el conocido proyecto japonés de la "**quinta generación**", soportado por el MITI (Ministerio de Industria y Comercio Internacional) y 8 fabricantes, al que siguen otras iniciativas en los EEUU y en Europa. En todos ellos se concede un papel importante a la investigación sobre sistemas basados en conocimiento paralelamente a las tecnologías de base y las arquitecturas.



## **UNIDAD DIDACTICA 5**

**Redes**

**Neuronales**



**UNIDAD DIDACTICA 5****EJE CONCEPTUAL**

Redes Neuronales

**TEMAS**

<b>5.1. Modelos Conexionistas .....</b>	<b>4</b>
5.1.1. Origen del paradigma de computación conexionista .....	4
<b>5.2. Redes Neuronales Biológicas.....</b>	<b>5</b>
<b>5.3. Redes Neuronales Artificiales.....</b>	<b>6</b>
5.3.1. Definición de Red Neuronal .....	6
5.3.2. Estructura de las Redes Neuronales.....	6
5.3.3. Comparación entre RNB y RNA .....	10
5.3.4. Formas de interconexión de las RNA.....	10
5.3.5. Características de las RNA.....	12
<b>5.4. Ventajas y Desventajas de las Redes Neuronales.....</b>	<b>12</b>
5.4.1. Ventajas que ofrecen las RNA .....	12
5.4.2. Desventajas que ofrecen las RNA .....	14
<b>5.5. Mecanismos de Aprendizaje .....</b>	<b>14</b>
5.5.1. Aprendizaje Supervisado.....	15
5.5.2. Aprendizaje No Supervisado .....	16
<b>5.6. El Perceptrón.....</b>	<b>18</b>
5.6.1. Aprendizaje del perceptrón.....	20
5.6.2. La separación lineal y el problema del XOR.....	22
<b>5.7. Redes de Hopfield .....</b>	<b>24</b>
<b>5.8. Máquinas de Boltzman .....</b>	<b>27</b>
<b>5.9. Redes Recurrentes .....</b>	<b>28</b>

## 5.1. Modelos Conexionistas

El cerebro es un procesador de información con unas características muy notables: es capaz de procesar a gran velocidad grandes cantidades de información procedentes de los sentidos, combinarla o compararla con la información almacenada y dar respuestas adecuadas, incluso en situaciones nuevas. Logra discernir un susurro en una sala ruidosa, distinguir una cara en una calle mal iluminada o leer entre líneas en una declaración política; pero lo más impresionante de todo es su capacidad de aprender a representar la información necesaria para desarrollar tales habilidades sin instrucciones explícitas para ello.

Aunque todavía se ignora mucho sobre la forma en que el cerebro aprende a procesar la información, se han desarrollado modelos que tratan de mimetizar tales habilidades, denominados **Redes Neuronales Artificiales o modelos de computación conexionista**.

*La elaboración de estos modelos supone en primer lugar la deducción de los rasgos o características esenciales de las neuronas y sus conexiones, y en segundo lugar la implementación del modelo en una computadora de forma que se pueda simular.*

Es obvio decir que estos modelos son idealizaciones burdas de las auténticas redes neuronales, en muchos casos de dudosa plausibilidad neurofisiológica, pero que sin embargo resultan interesantes cuanto menos por sus capacidades de aprendizaje.

### 5.1.1. Origen del paradigma de computación conexionista

La Inteligencia Artificial, entendida muy ampliamente como el modelado y la simulación de las actividades cognitivas complejas (*percepción, memoria, solución de problemas, etc.*) que caracterizan a los organismos avanzados y en particular a los seres humanos, se separó casi desde su inicio en dos ramas bien diferenciadas:

- Por un lado se trató de modelar la actividad racional mediante **sistemas formales de reglas y manipulación simbólica** (generalmente mediante sistemas lógicos), constituyendo quizás la **rama** más conocida de la IA, que podríamos denominar **simbólico-deductiva** (se postulan una serie de reglas y el sistema resuelve los problemas realizando deducciones sobre las reglas existentes).
- Por otro lado se desarrollaron modelos computacionales inspirados en las **redes neuronales biológicas**, denominados **inductivos o subsimbólicos**.

Si bien es mucho más conocida la aproximación simbólico-deductiva y su principal aplicación: los **sistemas expertos** (*sistemas o agentes basados en conocimiento*), existe un considerable y renacido interés por los **modelos conexionistas**. El progreso de las neurociencias nos está conduciendo a una comprensión cada vez mayor de la estructura física y lógica del cerebro; los avances tecnológicos ofrecen recursos cada vez mayores para representar estructuras muy complejas, realizar cálculos a gran velocidad y en paralelo, apoyando y fomentando así la investigación en este campo.

Podríamos situar el origen de los modelos conexionistas con la definición de la neurona formal dada por McCulloch y Pitts en 1943 como un dispositivo binario con varias entradas y salidas.

Un Psicólogo, D.O. Hebb, introdujo en 1949 dos ideas fundamentales que han influido de manera decisiva en el campo de las redes neuronales:

- la idea de que una percepción o un concepto se representa en el cerebro por un conjunto de neuronas activas simultáneamente y
- la idea de que la memoria se localiza en las conexiones entre las neuronas (sinapsis).

Las hipótesis de Hebb presentan de manera intuitiva el modo en que las neuronas memorizan información, y se plasman sintéticamente en la famosa **Regla de aprendizaje de Hebb**.

**Esta regla indica que las conexiones entre dos neuronas se refuerzan si ambas son activadas.**

## 5.2. Redes Neuronales Biológicas

A grandes rasgos, recordemos que el cerebro humano se compone de decenas de billones de neuronas interconectadas entre sí formando circuitos o redes que desarrollan funciones específicas.

Una **neurona** típica recoge señales procedentes de otras neuronas a través de una pléyade de delicadas estructuras llamadas **dendritas**. La neurona emite impulsos de actividad eléctrica a lo largo de una fibra larga y delgada denominada **axón**, que se escinde en millares de ramificaciones.

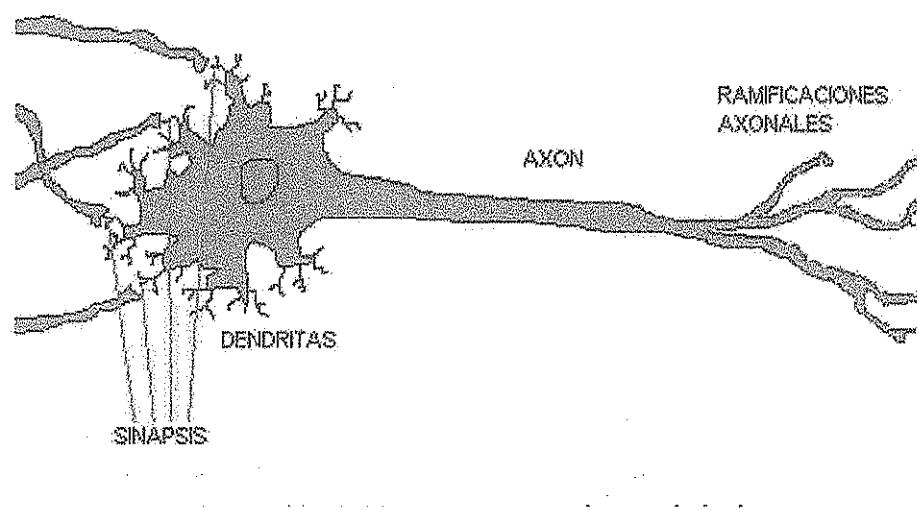


Figura 5.1

Las extremidades de estas ramificaciones llegan hasta las dendritas de otras neuronas y establecen unas conexiones llamadas **sinapsis**, en las cuales se produce una transformación del impulso eléctrico en un mensaje neuroquímico, mediante la liberación de unas sustancias llamadas **neurotransmisores**.

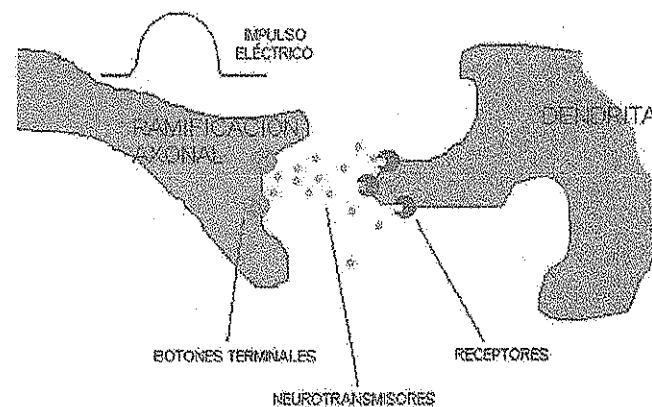


Figura 5.2

El **efecto de los neurotransmisores** sobre la neurona receptora puede ser **excitatorio o inhibitorio**, y es variable, de manera que podemos hablar de la fuerza o efectividad de una sinapsis.

Las señales excitatorias e inhibitorias recibidas por una neurona se combinan, y en función de la estimulación total recibida, la neurona toma un cierto nivel de activación, que se traduce en la **generación de breves impulsos nerviosos** con una determinada frecuencia o tasa de disparo, y su propagación a lo largo del axón hacia las neuronas con las cuales sinapta.

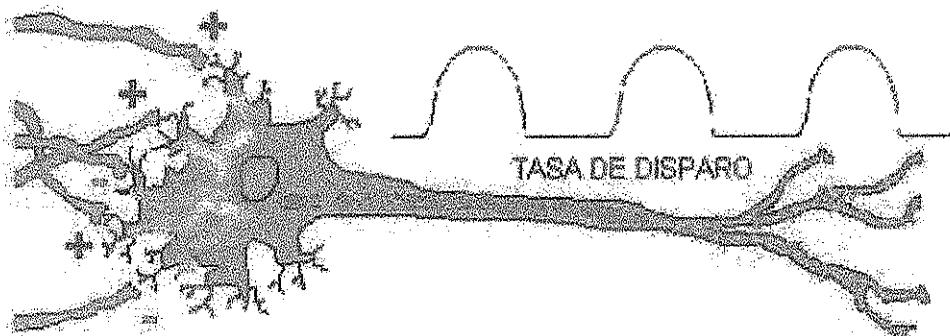


Figura 5.3

De esta manera **la información se transmite de unas neuronas** a otras y va siendo procesada a través de las conexiones sinápticas y las propias neuronas.

El aprendizaje de las redes neuronales se produce mediante la variación de la efectividad de las sinapsis, de esta manera cambia la influencia que unas neuronas ejercen sobre otras, de aquí se deduce que la arquitectura, el tipo y la efectividad de las conexiones en un momento dado, representan en cierto modo la memoria o estado de conocimiento de la red.

### 5.3. Redes Neuronales Artificiales

#### 5.3.1. Definición de Red Neuronal

Darpa (1988), define una **red neuronal** como:

**un sistema compuesto de muchos elementos simples de procesamiento los cuales operan en paralelo y cuya función es determinada por:**

- **la estructura de la red y**
- **el peso de las conexiones**

**realizándose el procesamiento en cada uno de los nodos o elementos de cómputo.**

Según Haykin (1994), una red neuronal es:

**un procesador paralelo masivamente distribuido que tiene una facilidad natural para el almacenamiento de conocimiento obtenido de la experiencia para luego hacerlo utilizable.**

#### 5.3.2. Estructura de las Redes Neuronales

Las neuronas se modelan mediante **unidades de proceso**.

Cada unidad de proceso se compone de:

- una red de conexiones de entrada
- una función de red (de propagación), encargada de computar la entrada total combinada de todas las conexiones
- un núcleo central de proceso, encargado de aplicar la función de activación
- la salida, por donde se transmite el valor de activación a otras unidades

La función de red es típicamente el sumatorio ponderado, mientras que la función de activación suele ser alguna función de umbral o una función sigmoidal.

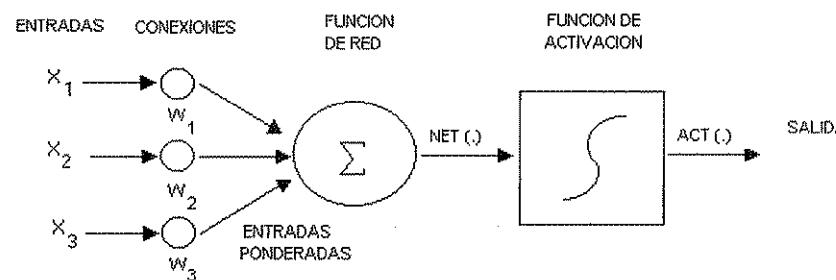


Figura 5.4

**Conexiones ponderadas:** hacen el papel de las *conexiones sinápticas*.

**Peso Sináptico ( $w_i$ ):** el peso de la conexión equivale a la fuerza o efectividad de la sinapsis.

La existencia de conexiones determina si es posible que una unidad influya sobre otra.

El valor de los pesos y el signo de los mismos define el tipo (excitatorio/inhibitorio) y la intensidad de la influencia.

**Función de Red o de Propagación:** calcula el valor de base o entrada total a la unidad, generalmente como simple *suma ponderada de todas las entradas recibidas*, es decir de las entradas multiplicadas por el peso o valor de las conexiones.

Equivale a la *combinación de las señales excitatorias e inhibidoras de las neuronas biológicas*.

**Función de Activación:** es quizás la característica principal o definitoria de las neuronas, la que mejor define el comportamiento de las mismas.

Se encarga de calcular el nivel o estado de activación de la neurona en función de la entrada total.

**Salida:** calcula la salida de la neurona en función de la activación de la misma, aunque normalmente no se aplica más que la función identidad, y se toma como salida el valor de activación.

El valor de salida cumpliría la función de la tasa de disparo en las neuronas biológicas.

### Función de Red o de Propagación

Como ya hemos comentado, se encarga de calcular la entrada total de la neurona como combinación de todas las entradas.

Podemos citar entre las más importantes:

#### A. Función lineal de base (FLB): es la más utilizada.

Consiste en la sumatoria ponderada de todas las entradas. Se trata de una función de tipo *hiperplano*, esto es, de *primer orden*.

Dado una unidad  $j$ , y  $n$  unidades conectadas a ésta, si llamamos  $X$  al vector de entradas (que coincide con las salidas de las unidades de la capa anterior) y  $W_j$  al vector de pesos de las conexiones correspondientes, esta función quedaría así:

$$net_j(X, W_j) = \sum_{i=1}^n x_i w_{ij}$$

Al representar los pesos utilizamos dos subíndices para indicar que conectan dos unidades,  $i$  y  $j$ , donde  $j$  se refiere a la unidad actual.

#### B. Función radial de base (FRB):

función de tipo *hiperesférico*, de *segundo orden*, no *lineal*. El valor de red representa la distancia a un determinado patrón de referencia.

$$net_j(X, W_j) = \sqrt{\sum_{i=1}^n (x_i - w_{ij})^2}$$

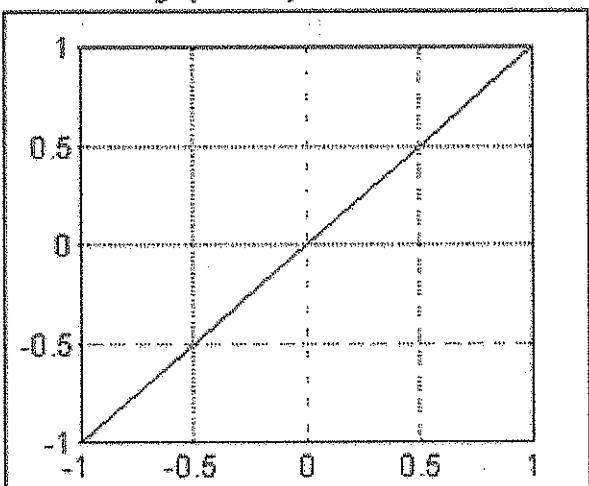
### Función de Activación

Podemos distinguir entre:

#### 1. Funciones lineales

En las que la salida es proporcional a la entrada

$$f(\text{neta}) = \text{neta}$$

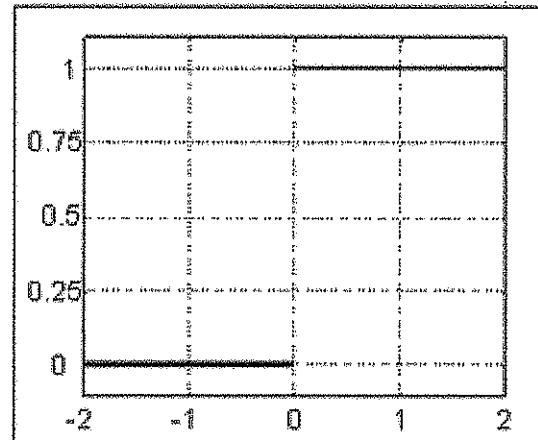


## 2. Funciones de umbral

En las cuales la salida es un valor discreto (típicamente binario 0 o 1) que depende de si la estimulación total supera o no un determinado valor de umbral.

En un principio se pensó que las neuronas usaban una función de umbral, es decir, que permanecían inactivas y se activaban solo si la estimulación total superaba cierto valor límite. Esto se puede modelar con una **función escalón**: la más típica es el **escalón unitario**: la función devuelve 0, por debajo del valor crítico (umbral) y 1, por encima.

$$f(\text{neta}) = \begin{cases} 1 & \text{neta} \geq 0 \\ 0 & \text{neta} < 0 \end{cases}$$



Después se comprobó que las neuronas emitían impulsos de actividad eléctrica con una frecuencia variable, dependiendo de la intensidad de la estimulación recibida, y que tenían cierta actividad hasta en reposo, con estimulación nula. Estos descubrimientos llevaron al uso de funciones no lineales con esas características, como la función sigmoidal, con un perfil parecido al escalón de una función de umbral, pero continua.

## 3. Funciones no lineales

No proporcionales a la entrada.

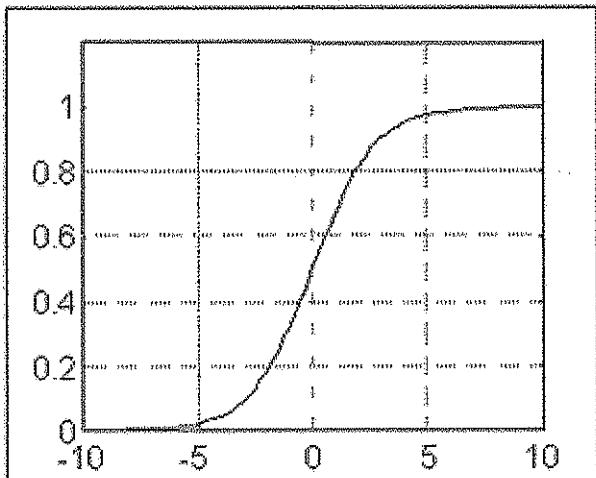
### Función sigmoidal o logística

Es probablemente la función de activación más empleada en la actualidad

$$\text{act}_j(\text{net}_j) = \frac{1}{(1+e^{-\text{net}_j})}$$

Se trata de una función continua no lineal con bastante plausibilidad fisiológica. La función sigmoidal posee un rango comprendido entre 0 y 1. Esto, aplicado a las unidades de proceso de una red neuronal artificial significa que, sea cual sea la entrada, la salida estará comprendida entre 0 y 1.

$$f(\text{neta}) = \frac{1}{1+e^{-\text{neta}}}$$



La salida de una unidad vale 0.5 cuando la entrada es nula, esto significa que la unidad tiene cierta actividad aún en ausencia de estimulación. Al aumentar la estimulación la unidad aumenta su activación, y la disminuye si la estimulación es inhibitoria, de forma parecida a como se comportan las neuronas reales.

Casi todos los avances recientes en conexionismo se atribuyen a arquitecturas multicapa que utilizan funciones de activación no lineales como una función de umbral, una gaussiana o en la mayoría de los casos una función sigmoidal.

El problema de trabajar con modelos no lineales radica en que son difíciles de describir en términos lógicos o matemáticos convencionales.

### 5.3.3. Comparación entre RNB y RNA

Redes Neuronales Biológicas	Redes Neuronales Artificiales
Neuronas	Unidades de proceso
Conexiones sinápticas	Conexiones ponderadas
Efectividad de las sinapsis	Peso de las conexiones
Efecto excitatorio o inhibitorio de una conexión	Signo del peso de una conexión
Efecto combinado de las sinapsis	Función de propagación o de red
Activación → tasa de disparo	Función de activación → salida

### 5.3.4. Formas de interconexión de las RNA

Para diseñar una red debemos establecer como estarán conectadas unas unidades con otras y determinar adecuadamente los pesos de las conexiones.

Lo más usual es disponer las unidades en forma de capas, pudiéndose hablar de redes de una, de dos o de más de dos capas, las llamadas redes multicapa.

Nota: en algunos manuales (EJ. Wassermann, 1989) se cuentan solo aquellas capas que poseen conexiones de entrada modificables, según este criterio la capa de entrada no contaría como tal).

Aunque inicialmente se desarrollaron redes de una sola capa, lo más usual es disponer tres o más **capas**:

- **Capa de entrada:** es la primera capa y actúa como buffer de entrada, almacenando la información bruta suministrada a la red o realizando un sencillo pre-proceso de la misma.
- **Capa de salida:** actúa como interfaz o buffer de salida, almacenando la respuesta de la red para que pueda ser leída.
- **Capas ocultas:** son las capas intermedias, principales encargadas de extraer, procesar y memorizar la información.

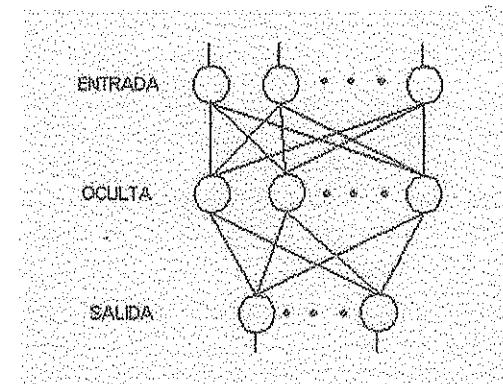


Figura 5.5

Podemos clasificar las RNA, además de por el número de capas de una red, en función de **cómo se interconectan unas capas con otras**:

- **Redes en cascada:** feed-forward. La información fluye unidireccionalmente de una capa a otra (desde la capa de entrada a las capas ocultas y de éstas a la capa de salida), y además, no se admiten conexiones intracapa.
- **Redes recurrentes:** feed-back. La información puede volver a lugares por los que ya había pasado, formando bucles, y se admiten las conexiones intracapa (laterales), incluso de una unidad consigo misma.

Las **conexiones** entre una capa y otra pueden ser:

- **Totales:** cada unidad se conecta con todas las unidades de la capa siguiente.
- **Parciales:** una unidad se conecta con solo algunas de las unidades de la unidad siguiente, generalmente siguiendo algún patrón aleatorio o pseudo-aleatorio (por ejemplo, mediante algoritmos genéticos).

Desde una **aproximación temporal** se puede distinguir entre:

- **Conexiones sin retardo.**
- **Conexiones con retardo.**

Esto permite modelar aspectos dinámicos, por ejemplo, para modelos psicofisiológicos de memoria.

### 5.3.5. Características de las RNA

1. **Aprendizaje inductivo:** no se le indican las reglas para dar una solución, sino que *extrae sus propias reglas a partir de los ejemplos de aprendizaje, modifican su comportamiento en función de la experiencia.* Esas reglas quedan almacenadas en las conexiones y no representadas explícitamente como en los sistemas basados en conocimiento (simbólico-deductivos).
2. **Generalización:** una vez entrenada, se le pueden presentar a la red datos distintos a los usados durante el aprendizaje. La respuesta obtenida dependerá del parecido de los datos con los ejemplos de entrenamiento.
3. **Abstracción o tolerancia al ruido:** las redes neuronales artificiales son capaces de extraer o abstraer las características esenciales de las entradas aprendidas, de esta manera *pueden procesar correctamente datos incompletos o distorsionados.*
4. **Procesamiento paralelo:** las neuronas reales trabajan en paralelo; en el caso de las redes artificiales es obvio que si usamos un solo procesador no podrá haber proceso paralelo real; sin embargo *hay un paralelismo inherente*, lo esencial es que la estructura y modo de operación de las redes neuronales las hace especialmente adecuadas para el *procesamiento paralelo real mediante multiprocesadores* (se están desarrollando máquinas específicas para la computación neuronal).
5. **Memoria distribuida:** el conocimiento acumulado por la red se halla distribuido en numerosas conexiones, esto tiene como consecuencia la tolerancia a fallos: *una red neuronal es capaz de seguir funcionando adecuadamente a pesar de sufrir lesiones con destrucción de neuronas o sus conexiones, ya que la información se halla distribuida por toda la red*, sin embargo en un programa tradicional un pequeño fallo en cualquier punto puede invalidarlo todo y dar un resultado absurdo o no dar ningún resultado.

## 5.4. Ventajas y Desventajas de las Redes Neuronales

### 5.4.1. Ventajas que ofrecen las RNA

1. **Aprendizaje adaptativo:** capacidad de aprender a realizar tareas basadas en un entrenamiento o en una experiencia inicial.
2. **Auto-organización:** una red neuronal puede crear su propia organización o representación de la información que recibe mediante una etapa de aprendizaje.
3. **Tolerancia a fallos:** la destrucción parcial de una red conduce a una degradación de su estructura, sin embargo, algunas capacidades de la red se pueden retener, incluso sufriendo un gran daño.
4. **Operación en tiempo real:** los cálculos neuronales pueden ser realizados en paralelo, para esto se diseñan y fabrican máquinas con hardware especial para obtener esta capacidad.
5. **Fácil inserción dentro de la tecnología existente:** se pueden obtener chips especializados para redes neuronales que mejoran su capacidad en ciertas tareas. Ello facilitará la integración modular en los sistemas existentes.

#### Aprendizaje adaptativo

La capacidad de aprendizaje adaptativo es una de las características más atractivas de las redes neuronales. Esto es, *aprenden a llevar a cabo ciertas tareas mediante un entrenamiento con ejemplos ilustrativos.*

Como las redes neuronales pueden aprender a diferenciar patrones mediante ejemplos y entrenamientos, no es necesario elaborar modelos a priori, ni especificar funciones de distribución de probabilidad.

Las redes neuronales son **sistemas dinámicos autoadaptativos.**

- Son adaptables debido a la capacidad de autoajuste de los elementos procesales (neuronas) que componen el sistema.
- Son dinámicos, pues son capaces de estar constantemente cambiando para adaptarse a las nuevas condiciones.

En el proceso de aprendizaje, los enlaces ponderados de las neuronas se ajustan de manera que se obtengan ciertos resultados específicos.

*Una red neuronal no necesita un algoritmo para resolver un problema, ya que ella puede generar su propia distribución de pesos en los enlaces mediante el aprendizaje. También existen redes que continúan aprendiendo a lo largo de su vida, después de completado su período de entrenamiento.*

La función del diseñador es únicamente la obtención de la arquitectura apropiada. No es problema del diseñador el como la red aprenderá a discriminar.

Sin embargo, si es necesario que desarrolle un buen algoritmo de aprendizaje que le proporcione a la red la capacidad de discriminar, mediante un entrenamiento con patrones.

### Auto-organización

Las redes neuronales emplean su capacidad de aprendizaje adaptativo para autoorganizar la información que reciben durante el aprendizaje y/o la operación.

*Mientras que el aprendizaje es la modificación de cada elemento procesal, la autoorganización consiste en la modificación de la red neuronal completa para llevar a cabo un objetivo específico.*

Cuando las redes neuronales se usan para reconocer ciertas clases de patrones, ellas autoorganizan la información usada. Por ejemplo, la red llamada backpropagation, creará su propia representación característica, mediante la cual puede reconocer ciertos patrones.

*Esta autoorganización provoca la generalización: facultad de las redes neuronales de responder apropiadamente cuando se les presentan datos o situaciones a las que no había sido expuesta anteriormente.*

El sistema puede generalizar la entrada para obtener una respuesta. Esta característica es muy importante cuando se tiene que solucionar problemas en los cuales la información de entrada no es muy clara. Además permite que el sistema de una solución, incluso cuando la información de entrada está especificada de forma incompleta.

### Tolerancia a fallos

Las redes neuronales fueron los primeros métodos computacionales con la capacidad inherente de tolerancia a fallos. Comparados con los sistemas computacionales tradicionales, los cuales pierden su funcionalidad cuando sufren un pequeño error de memoria,

*en las redes neuronales, si se produce un fallo en un número no muy grande de neuronas y aunque el comportamiento del sistema se ve influenciado, no sufre una caída repentina.*

Hay dos aspectos distintos respecto a la tolerancia a fallos:

- a) Las redes pueden aprender a reconocer patrones con ruido, distorsionados o incompletos. Esta es una tolerancia a fallos respecto a los datos.
- b) Las redes pueden seguir realizando su función (con cierta degradación) aunque se destruya parte de la red.

La razón por la que las redes neuronales son tolerantes a los fallos es que tienen su información distribuida en las conexiones entre neuronas, existiendo cierto grado de redundancia en este tipo de almacenamiento. La mayoría de los ordenadores algorítmicos y sistemas de recuperación de datos almacenan cada pieza de información en un espacio único, localizado y direccionable. En cambio, las redes neuronales almacenan información

no localizada. Por lo tanto, la mayoría de las interconexiones entre los nodos de la red tendrán sus valores en función de los estímulos recibidos, y se generará un patrón de salida que represente la información almacenada.

### Operación en tiempo real

Una de las mayores prioridades, casi en la totalidad de las áreas de aplicación, es la necesidad de realizar procesos con datos de forma muy rápida.

Las redes neuronales se adaptan bien a esto debido a su implementación paralela. Para que la mayoría de las redes puedan operar en un entorno de tiempo real, la necesidad de cambio en los pesos de las conexiones o entrenamiento es mínimo.

### Fácil inserción dentro de la tecnología existente

Una red individual puede ser entrenada para desarrollar una única y bien definida tarea (tareas complejas, que hagan múltiples selecciones de patrones requerirán sistemas de redes interconectadas). Con las herramientas computacionales existentes (no del tipo PC), una red puede ser rápidamente entrenada, comprobada, verificada y trasladada a una implementación hardware de bajo coste.

Por lo tanto, no se presentan dificultades para la inserción de redes neuronales en aplicaciones específicas, por ejemplo de control, dentro de los sistemas existentes. De esta manera, las redes neuronales se pueden utilizar para mejorar sistemas en forma incremental y cada paso puede ser evaluado antes de acometer un desarrollo más amplio.

#### 5.4.2. Desventajas que ofrecen las RNA

##### 1. Definición de muchos parámetros:

Una de las desventajas de las redes neuronales es que **requieren definición de muchos parámetros antes de poder aplicar la metodología.**

Por ejemplo hay que decidir la arquitectura más apropiada, el número de capas ocultas, el número de nodos por capa, las interconexiones, la función de transformación, etc. Las técnicas estadísticas convencionales, sin embargo, solo requieren la extracción y normalización de una muestra de datos. Es un argumento erróneo sostener que el tiempo de desarrollo para los modelos basados en una red neuronal sea más corto que el tiempo necesario para desarrollar, por ejemplo, una tabla de puntuación basada en una regresión múltiple. Los estudios donde se ha constatado un tiempo de desarrollo más corto no han tenido en cuenta la preparación de datos que requiere una red neuronal.

##### 2. Caja negra:

Otra desventaja es que **no ofrecen una interpretación fácil**, como hace, por ejemplo, un algoritmo de scoring. **En lugar de ser un sistema de apoyo a la decisión, la caja negra se puede convertir en el "tomador" de la decisión.** Puede ocurrir que un director de riesgo deniege un crédito solo porque lo dice la caja negra, sin que él pueda argumentar esta decisión ya que no entiende el funcionamiento de la red neuronal.

#### 5.5. Mecanismos de Aprendizaje

**El aprendizaje es el proceso por el cual una red neuronal modifica sus pesos en respuesta a una información de entrada.**

Los cambios que se producen durante el mismo se reducen a la destrucción, modificación y creación de conexiones entre las neuronas. En los sistemas biológicos existe una continua destrucción y creación de conexiones. En los modelos de redes neuronales artificiales, la

creación de una nueva conexión implica que el peso de la misma pasa a tener un valor distinto de cero. De la misma manera, una conexión se destruye cuando su peso pasa a ser cero.

**Durante el proceso de aprendizaje, los pesos de las conexiones de la red sufren modificaciones, por lo tanto, se puede afirmar que este proceso ha terminado (la red ha aprendido) cuando los valores de los pesos permanecen estables.**

Un aspecto importante respecto al aprendizaje de las redes neuronales es el conocer como se modifican los valores de los pesos, es decir, cuales son los criterios que se siguen para cambiar el valor asignado a las conexiones cuando se pretende que la red aprenda una nueva información.

Existen dos métodos de aprendizaje importantes que pueden distinguirse:

- Aprendizaje supervisado
- Aprendizaje no supervisado

Otro criterio que se puede utilizar para diferenciar las reglas de aprendizaje se basa en la siguiente consideración:

- Aprendizaje on-line

Si la red puede aprender durante su funcionamiento habitual.

- Aprendizaje off-line

Si el aprendizaje supone la desconexión de la red, es decir, su inhabilitación hasta que el proceso se termine.

Cuando el aprendizaje es off-line se distingue entre una fase de aprendizaje o entrenamiento y una fase de operación o funcionamiento, existiendo *un conjunto de datos de entrenamiento y un conjunto de datos de test o prueba*, que serán utilizados en la correspondiente fase. Además los pesos de las conexiones permanecen fijos después que termina la etapa de entrenamiento de la red.

*Debido, precisamente a su carácter estático, estos sistemas no presentan problemas de estabilidad en su funcionamiento.*

### 5.5.1. Aprendizaje Supervisado

**El aprendizaje supervisado se caracteriza porque el proceso de aprendizaje se realiza mediante un entrenamiento controlado por un agente externo (supervisor, maestro) que determina la respuesta que debería generar la red a partir de una entrada determinada.**

El supervisor controla la salida de la red y en caso de que ésta no coincida con la deseada, se procederá a modificar los pesos de las conexiones, con el fin de conseguir que la salida obtenida se aproxime a la deseada.

En este tipo de aprendizaje se suelen considerar, a su vez, tres formas de llevarlo a cabo, que dan lugar a los siguientes aprendizajes supervisados:

- Aprendizaje por corrección de error.
- Aprendizaje por refuerzo.
- Aprendizaje estocástico.

### A. Aprendizaje por corrección de error

Consiste en ajustar los pesos de las conexiones de la red en función de la diferencia entre los valores deseados y los obtenidos a la salida de la red, es decir, en función del error cometido en la salida.

$$\text{Error cometido} = \text{Valor deseado} - \text{Valor obtenido}$$

Un ejemplo de este tipo de algoritmos lo constituye la regla de aprendizaje del Perceptrón, utilizada en el entrenamiento de la red del mismo nombre que desarrolló Rosenblatt en 1958. Esta es una regla muy simple: para cada neurona, en la capa de salida se le calcula la desviación a la salida objetivo como el error,  $\delta$ , el cual luego se utiliza para cambiar los pesos sobre la conexión de la neurona precedente.

### B. Aprendizaje por refuerzo

Se trata de un aprendizaje supervisado, más lento que el anterior, que se basa en la idea de no disponer de un ejemplo completo del comportamiento deseado, es decir, de no indicar durante el entrenamiento exactamente la salida que se desea que proporcione la red ante una determinada entrada.

En el aprendizaje por refuerzo la función del supervisor se reduce a indicar mediante una señal de refuerzo si la salida obtenida en la red se ajusta a la deseada (éxito = +1 o fracaso = -1), y en función de ello se ajustan los pesos basándose en un mecanismo de probabilidades. Se podría decir que en este tipo de aprendizaje la función del supervisor se asemeja más a la de un **crítico** (que opina sobre la respuesta de la red) que a la de un maestro (que indica a la red la respuesta concreta que debe generar), como ocurriría en el caso de supervisión por corrección del error.

### C. Aprendizaje estocástico

Consiste básicamente en realizar cambios aleatorios en los valores de los pesos de las conexiones de la red y evaluar su efecto a partir del objetivo deseado y de distribuciones de probabilidad.

En el aprendizaje estocástico se suele hacer una analogía en términos termodinámicos, asociando a la red neuronal con un sólido físico que tiene cierto estado energético. En el caso de la red, la energía de la misma representaría el grado de estabilidad de la red, de tal forma que el estado de mínima energía correspondería a una situación en la que los pesos de las conexiones consiguen que su funcionamiento sea el que más se ajusta al objetivo deseado.

Según lo anterior, el aprendizaje consistiría en realizar un cambio aleatorio de los valores de los pesos y determinar la energía de la red. Si la energía es menor después del cambio, es decir, si el comportamiento de la red se acerca al deseado, se acepta el cambio. Si, por el contrario, la energía no es menor, se aceptaría el cambio en función de una determinada y preestablecida distribución de probabilidades.

#### 5.5.2. Aprendizaje No Supervisado

**Las redes con aprendizaje no supervisado (también conocido como autosupervisado) no requieren influencia externa para ajustar los pesos de las conexiones entre sus neuronas. La red no recibe ninguna información por parte del entorno que le indique si la salida generada en respuesta a una determinada entrada es o no correcta.**

Estas redes deben encontrar las características, regularidades, correlaciones o categorías que se puedan establecer entre los datos que se presenten en su entrada. Existen varias posibilidades en cuanto a la interpretación de la salida de estas redes, que dependen de su estructura y del algoritmo de aprendizaje empleado:

- En algunos casos, la salida representa el grado de familiaridad o similitud entre la información que se le está presentando en la entrada y las informaciones que se le han mostrado hasta entonces (en el pasado).
- En otro caso, podría realizar un establecimiento de categorías, indicando la red a la salida, a qué categoría pertenece la información presentada a la entrada, siendo la propia red quien debe encontrar las categorías apropiadas a partir de las correlaciones entre las informaciones presentadas.

En cuanto a los algoritmos de aprendizaje no supervisado, en general se suelen considerar dos tipos, que dan lugar a los siguientes aprendizajes:

- Aprendizaje hebbiano
- Aprendizaje competitivo y comparativo

#### A. Aprendizaje hebbiano

Los sistemas neuronales biológicos no nacen preprogramados con todo el conocimiento y las capacidades que llegarán a tener eventualmente.

**Un proceso de aprendizaje que tiene lugar a lo largo de un período de tiempo modifica de alguna forma la red para incluir la nueva información.**

La teoría básica procede de un libro de 1949 escrito por Hebb. La idea principal se expresaba en forma de suposición y es la siguiente:

Cuando un axón de la célula A está suficientemente próximo para excitar a una célula B o toma parte en su disparo de forma persistente, tiene lugar algún proceso de crecimiento o algún cambio metabólico en una de las células, o en las dos, de tal modo que la eficiencia de A, como una de las células que desencadena el disparo de B, se ve incrementada.

Para ilustrar la idea básica, consideramos el ejemplo clásico de condicionamiento, empleando el conocido experimento de Pavlov, reflejado en una idealización de tres neuronas que forman parte del proceso, como indica la Figura 5.6:

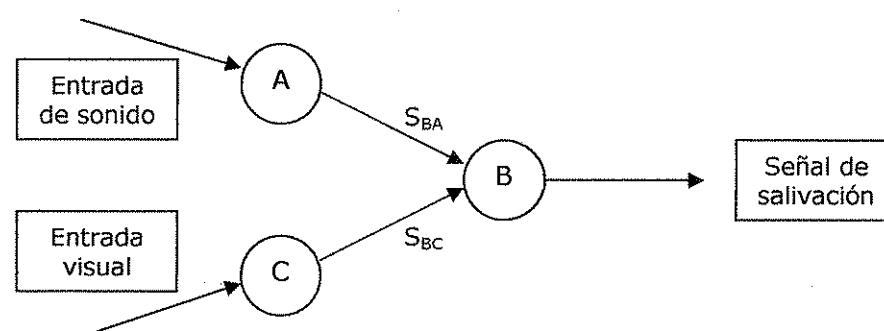


Figura 5.6

Supongamos que la excitación de C, causada por la visualización de la comida, es suficiente para excitar a B, que da lugar a la salivación. Además supongamos que, en ausencia de estimulación adicional, la excitación de A, que se debe a oír una campanilla, no basta para dar lugar al disparo de B.

Permitamos que C dé lugar a que B dispare mostrando comida al sujeto, y, mientras B sigue disparando, estimulemos a A haciendo sonar una campanilla. Dado que B sigue

disparando, A participa ahora en la excitación de B, aún cuando por sí sola A no sería suficiente para dar lugar a que B disparese.

*En esta situación, la suposición de Hebb determina que se produce algún cambio entre A y B, de tal modo que la influencia de A sobre B se ve incrementada. Si el experimento se repite con frecuencia suficiente, A será capaz de lograr, eventualmente, que se dispare B incluso en ausencia de la estimulación visual procedente de C.*

**Llevado esto al terreno de las RNA, significa que el peso de la conexión entre ambas neuronas se ve incrementado.**

### B. Aprendizaje competitivo y comparativo

**El aprendizaje competitivo se orienta a la clasificación de los datos de entrada. En este tipo de aprendizaje, las unidades de salida luchan por el control sobre porciones del espacio de entrada.**

Las unidades de entrada se conectan directamente a las unidades de salida, pero éstas también se conectan entre sí con conexiones precableadas negativas o inhibitorias.

La unidad de salida con la mayor activación en sus entradas tendrá más fuerza que sus competidores. Como resultado, los competidores se vuelven más débiles, perdiendo su poder de inhibición sobre las unidades más fuertes. Las unidades más fuertes se hacen cada vez más fuertes y su efecto inhibitorio sobre las demás unidades de salida se hace cada vez mayor. Pronto, el resto de las unidades de salida estará completamente inactivo. Este tipo de inhibición mutua se denomina: comportamiento de el ganador se lleva todo.

### 5.6. El Perceptrón

El Perceptrón, inventado por Rosenblatt (1962), fue uno de los primeros modelos de redes neuronales.

Consiste en pesos entrenables multiplicativos, un sumador y una función umbral.

Un perceptrón imita una neurona tomando la suma ponderada de sus entradas y enviando a la salida un 1 si la suma es más grande que algún valor umbral ajustable, y enviando un 0 en caso contrario.

**Un perceptrón es una representación, o sea, una red neuronal en la que:**

- Solo hay una neurona.
- Las entradas son binarias: solo se permiten 0 y 1.
- Las cajas lógicas pueden interponerse entre las entradas y los pesos del perceptrón. Cada caja lógica puede verse como una tabla que produce un valor de salida de 0 o 1 para cada combinación de 0 y 1 que pueda aparecer en sus entradas.
- La salida del perceptrón es 0 o 1, dependiendo de si la suma ponderada de las salidas de las cajas lógicas es mayor que el umbral.

En la Figura 5.7 se puede observar un modelo de un perceptrón.

Las entradas son 0 o 1, al igual que las salidas de las cajas lógicas. Si la suma de las salidas ponderadas de las cajas lógicas es mayor que 0, la salida del perceptrón es 1 y se dice que el perceptrón dice SI, se ha reconocido una clase. En cualquier otro caso, se dice que el perceptrón dice NO, no se ha reconocido una clase.

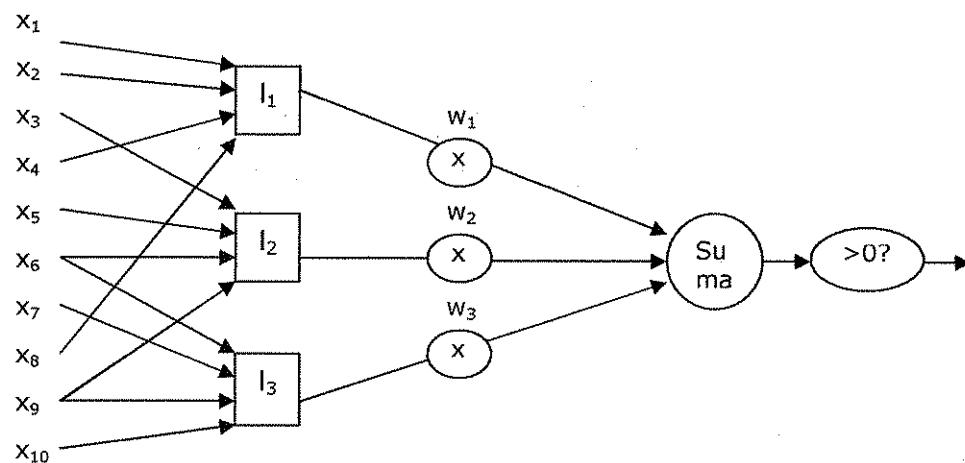


Figura 5.7

Para expresar la situación en notación matemática, suponga que la salida de la  $i$ -ésima caja lógica es  $I_i$ , el  $i$ -ésimo peso es  $w_i$ , y que finalmente, el umbral es  $T$ . Entonces la salida del perceptrón completo,  $P$ , está dada por la siguiente fórmula:

$$P = \begin{cases} 1 & \text{si } \sum_i w_i \times I_i > T \\ 0 & \text{en otro caso} \end{cases}$$

Note que si cualesquiera de las cajas lógicas del perceptrón pudieran atender todas las entradas del perceptrón, entonces no habría necesidad de otras cajas lógicas ni tampoco de pesos o de una función de umbral. En lugar de ello, todas las combinaciones de 0 y 1 de entrada se registrarían en la tabla interna de la caja lógica, junto con la salida apropiada de 0 o 1.

Por supuesto que el número de entradas de la tabla sería de  $2^n$  si hubiera  $n$  entradas y se tuvieran que manejar todas las posibles combinaciones de 0 y 1. Esta relación exponencial entre el número de entradas de la tabla y el número de entradas del perceptrón hace que no sea práctico el que cualesquiera de las cajas lógicas atiendan a todas las entradas, e incluso si atendiera una fracción sustancial de todas las entradas, si hubiera muchas.

En consecuencia, se supone que cada caja lógica de un perceptrón debe atender solo un pequeño número de entradas, por ejemplo, las cajas lógicas del perceptrón de la Figura 5.7 atienden un máximo de 4 entradas.

En un **perceptrón limitado por el orden**, de orden  $n$ , cada caja lógica atiende  $n$  o menos entradas.

Otra limitación más consiste en reducir las cajas lógicas a una sola entrada, que se pasa por el correspondiente multiplicador:

En un **perceptrón directo**, cada caja lógica tiene solo una entrada, y la salida es siempre la misma que la entrada.

De manera alternativa, un perceptrón directo puede verse como un perceptrón sin cajas lógicas, como se ilustra en la Figura 5.8.

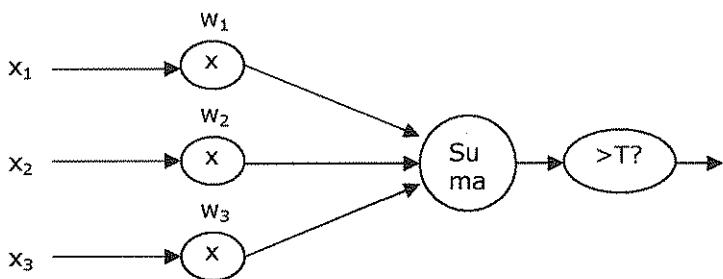


Figura 5.8

### 5.6.1. Aprendizaje del preceptrón

Es importante hacer notar que existe un procedimiento que descubre un buen conjunto de pesos para un perceptrón, dado que tal conjunto exista.

Además, el procedimiento es increíblemente directo.

Básicamente, se empieza con todos los pesos en 0. Despues se intenta el perceptrón con todas las muestras, una a la vez. Siempre que el perceptrón cometa un error, se cambian los pesos de modo que el error se haga menos probable, en cualquier otro caso, no se hace nada.

- Por el momento suponga que el perceptrón dice no, al producir un 0, cuando debería decir sí, produciendo un 1. Se puede aumentar la probabilidad de que diga sí la próxima vez si se incrementan los pesos asignados a aquellas cajas lógicas que producen 1 en el instante en que se comete el error. Una forma de asegurar el posible descubrimiento de un conjunto completamente eficaz de pesos consiste en aumentar cada uno de tales pesos en 1.
- De manera simétrica, siempre que el perceptrón dice sí cuando debería decir no, se disminuyen los pesos asignados a las cajas lógicas que producen 1, con la disminución igual a 1.
- En ningún caso se alteran los pesos asignados a las cajas que producen 0. Despues de todo, tales pesos están multiplicados por 0, de modo que no es posible que al jugar con ellos se pueda cambiar el resultado actual y sí podría hacerse erróneo algún otro resultado correcto.

Finalmente, para conseguir el efecto de un umbral entrenable, se añade una entrada virtual extra, cuyo valor siempre se supone es 1, a la caja lógica que acaba de pasar el valor de entrada. **Con esta añadidura, el perceptrón se puede ver como si tuviera un umbral de 0, y dice sí siempre que la suma ponderada de las salidas de la caja lógica sea mayor que 0.**

Observe que se puede hablar del procedimiento de entrenamiento del perceptrón de forma más concisa mediante el lenguaje de vectores. Las salidas de caja lógica y los pesos, expresados en notación vectorial, son  $(l_1, l_2, \dots, l_n)$  y  $(w_1, w_2, \dots, w_n)$ . Sumar uno a cada uno de los pesos asignados a cajas lógicas que producen 1 es lo mismo que sumar el vector de salida de caja lógica al vector peso, según la siguiente descripción:

#### Para entrenar a un perceptrón:

- Hasta que el perceptrón produzca el resultado correcto para cada muestra de entrenamiento, para cada muestra:
  - Si el preceptrón produce una respuesta errónea:
    - Si el perceptrón dice no cuando debería decir sí, añada el vector de salida de caja lógica al vector peso.
    - En cualquier otro caso, reste el vector de salida de caja lógica del vector peso.
  - En cualquier otro caso, no haga nada.

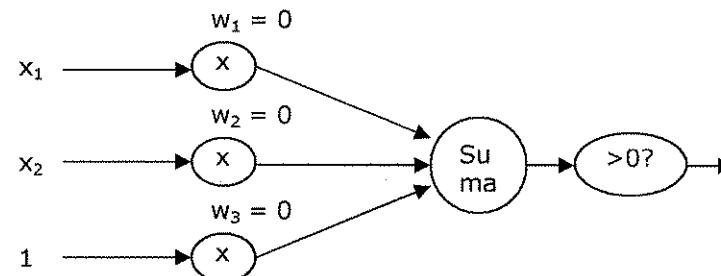


Figura 5.9

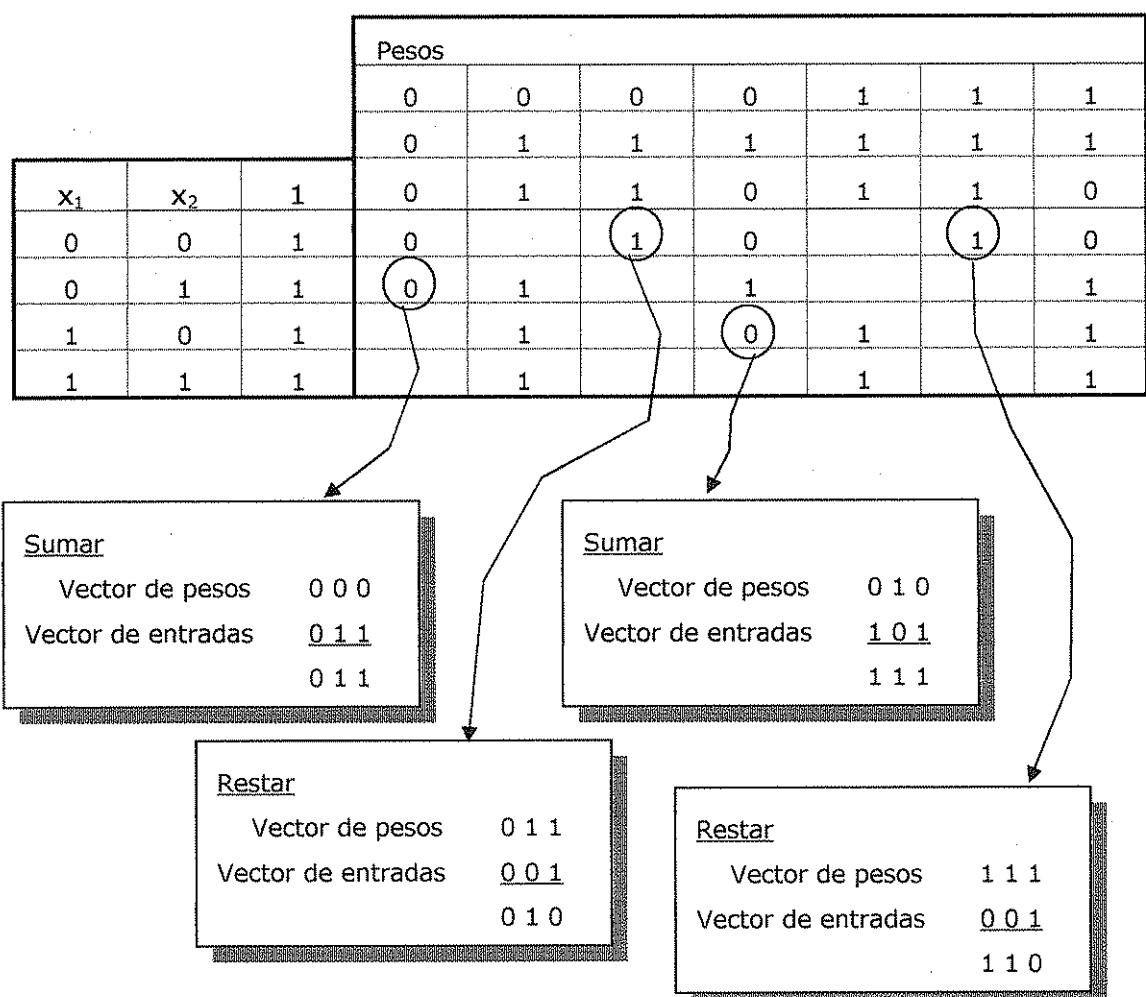
Por ejemplo, considere el **perceptrón directo** que se muestra en la Figura 5.9 y suponga que desea entrenarlo para que efectúe una **función lógica OR**, comenzando desde la línea de partida.

Como el perceptrón es directo, el vector de salida de caja lógica,  $(l_1, l_2, l_3)$ , es el mismo que el vector de entrada,  $(x_1, x_2, x_3)$ . Por tanto, las muestras de entrada y las correspondientes salidas de caja lógica, para una OR lógica, son las siguientes:

Muestra	$x_1 = (l_1)$	$x_2 = (l_2)$	$x_3 = (l_3) = 1$	Salida deseada
1	0	0	1	0
2	0	1	1	1
3	1	0	1	1
4	1	1	1	1

Procediendo en ciclos a través de estas muestras, finalmente el perceptrón aprende el OR lógico, después de cuatro cambios:

1. El primero ocurre después de un error en la segunda muestra durante el primer paso. En ese momento, el vector peso es  $(0 \ 0 \ 0)$ , de modo que la salida es 0 cuando debería ser 1. En consecuencia, cuando el vector de entrada,  $(0 \ 1 \ 1)$ , se suma al vector peso, el nuevo vector peso es  $(0 \ 1 \ 1)$ . Las siguientes dos muestras encontradas durante el primer paso producen 1, como deberían, de modo que no hay que hacer más cambios durante el primer paso.
2. Durante el segundo paso, la primera muestra produce un 1, pero debería producir un 0. En consecuencia, el vector de entrada  $(0 \ 0 \ 1)$  se resta del vector peso  $(0 \ 1 \ 1)$ , lo que produce un nuevo vector peso  $(0 \ 1 \ 0)$ .
3. Sin embargo, con este cambio, la tercera muestra produce un 0 cuando debería producir un 1. Por tanto, el vector de entrada,  $(1 \ 0 \ 1)$ , se suma al vector peso,  $(0 \ 1 \ 0)$ , lo que produce un nuevo vector peso,  $(1 \ 1 \ 1)$ .
4. A continuación, la primera muestra produce de nuevo un error durante el tercer paso. El resultado debe ser 0, pero es 1. Al restar el vector de entrada,  $(0 \ 0 \ 1)$ , del vector peso,  $(1 \ 1 \ 1)$ , se produce un nuevo vector peso,  $(1 \ 1 \ 0)$ , que posteriormente funciona para todas las muestras.



### 5.6.2. La separación lineal y el problema del XOR

El **teorema de convergencia del perceptrón** de Rosenblatt (1962), garantiza que el perceptrón encontrará un estado solución, es decir, aprenderá a clasificar cualquier conjunto de entradas linealmente separables.

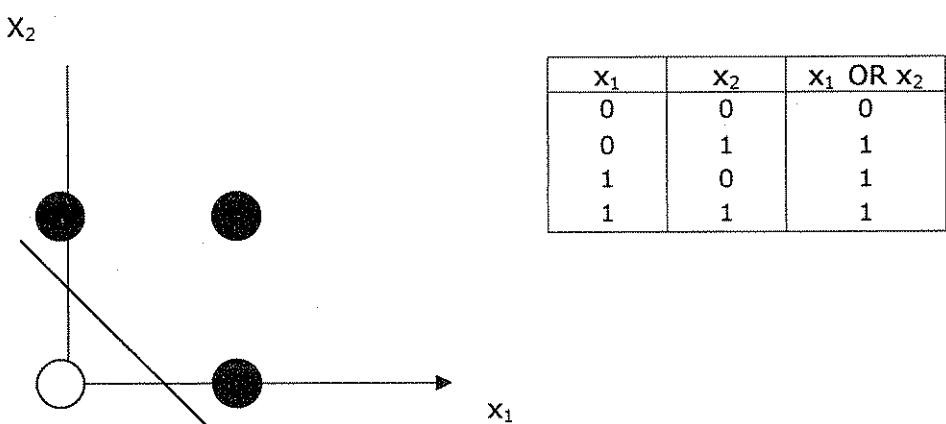


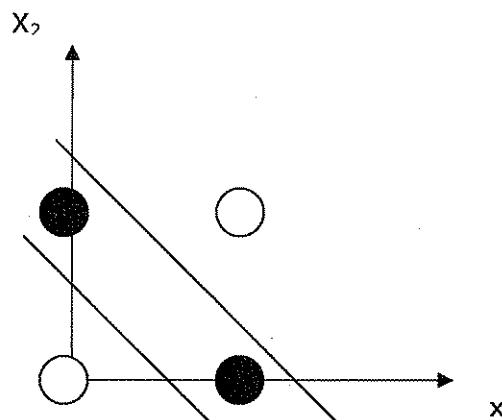
Figura 5.10

La introducción de los perceptrones a finales de la década de los cincuenta causó un gran entusiasmo. Suponía un dispositivo que se asemejaba mucho a una neurona y para el que había disponibles algoritmos de aprendizaje bien definidos. Hubo mucha especulación acerca de cómo podrían formarse sistemas inteligentes a partir de bloques construidos con perceptrones.

En su libro Perceptrones, Minsky y Papert (1969) pusieron fin a dicha especulación analizando las capacidades de cálculo de dicho sistema. Ellos advirtieron que mientras el teorema de convergencia garantizaba una correcta clasificación de la información linealmente independiente, la mayoría de los problemas no proporcionaban este tipo de datos tan bellos.

En realidad, el perceptrón es incapaz de aprender a resolver algunos problemas verdaderamente sencillos. Un ejemplo dado por Minsky y Papert es el problema **OR-Exclusiva (XOR)**:

Dadas dos entradas binarias, dará salida 1 si solamente una de las entradas está conectada, y dará salida 0 si ocurre de modo contrario. Se puede considerar a XOR como un problema de clasificación de patrones en el que existen 4 patrones y 2 salidas posibles (ver Figura 5.11).



$x_1$	$x_2$	$x_1 \text{ XOR } x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Figura 5.11

El perceptrón no puede aprender a crear una superficie lineal de decisión que separe estas diferentes salidas, porque dicha superficie de decisión no existe. **No hay una única recta que pueda separar las salidas 1 de las salidas 0.** Minsky y Papert proporcionaron varios problemas con estas propiedades.

Démonos cuenta que aquí *la deficiencia* no está en el algoritmo de aprendizaje del perceptrón, sino en *el modo en el que el perceptrón representa el conocimiento*.

- Si se pudiera dibujar una **superficie de decisión elíptica** se podrían englobar las dos salidas "1" en el espacio XOR. **Sin embargo, los perceptrones son incapaces de modelar dichas superficies.**
- Otra idea puede ser la de emplear dos escenarios diferentes para el dibujo de líneas. Se podría dibujar una línea para aislar el punto ( $x_1 = 1, x_2 = 1$ ) y después otra línea para dividir los tres puntos que quedan en dos categorías. Utilizando esta idea se podría **construir un perceptrón "multicapa"** para solucionar el problema.

### 5.7. Redes de Hopfield

La historia de la IA es curiosa. Los primeros problemas con los que se enfrentaron los investigadores de la IA fueron problemas como el *ajedrez* o la demostración de teoremas, porque pensaban que en la solución de estos problemas se hallaba la *esencia de la inteligencia*.

*La visión y la comprensión del lenguaje* (tareas que un niño de cinco años ya realiza correctamente) *no eran considerados problemas difíciles*. Hoy en día ya se tienen programas expertos de ajedrez, así como programas expertos en realizar diagnósticos médicos, pero *ningún programa puede llevar a cabo las características básicas de percepción que tiene un niño*.

Los investigadores de redes neuronales reconocen que hay una falta de coincidencia evidente entre la tecnología empleada en el procesamiento de la información en una computadora estándar y la tecnología usada por el cerebro.

Además de estas tareas de percepción, la IA está empezando a tratar ahora los principales problemas referentes a la *memoria* y *al razonamiento de sentido común*. Ya es conocida la falta de sentido común de las computadoras. Mucha gente cree que el sentido común es una consecuencia del almacenamiento masivo de conocimiento, y aún más importante, de nuestra capacidad de acceder a conocimientos relevantes rápidamente, sin esfuerzos y a su debido tiempo.

Por ejemplo, cuando se lee la descripción "gris, grande, mamífero" automáticamente se piensa en elefantes y en sus características asociadas.

#### Accedemos a la memoria mediante contenidos.

*En las implementaciones tradicionales, el acceso por contenido da lugar a complicados procedimientos de búsqueda y emparejamiento. Las redes masivamente paralelas sugieren un método más eficaz.*

Hopfield (1982), introdujo una red neuronal que propuso como una nueva teoría de la memoria. Una **red de Hopfield** tiene las siguientes importantes características:

1. **Representación distribuida:** una memoria se almacena como un patrón de activación a través de un conjunto de elementos de proceso. Las memorias pueden estar superpuestas una sobre otra. Las diferentes memorias se representan por diferentes patrones sobre el mismo conjunto de elementos de proceso.
2. **Control asíncrono y distribuido:** cada elemento de proceso toma decisiones basadas únicamente en su propia situación local. Todas estas situaciones locales se unen para alcanzar una solución global.
3. **Memoria direccionable por contenido:** se puede almacenar un determinado número de patrones en una red. Para recuperar un patrón únicamente se necesita una parte específica de él. La red encuentra automáticamente el emparejamiento más próximo.
4. **Tolerancia a fallos:** aunque algunos de los elementos procesadores de la red fallasen, ésta todavía funcionará adecuadamente.

#### ¿Cómo se alcanzan estas características?

En la Figura 5.12 se muestra una sencilla red de Hopfield. Los elementos de proceso, también llamados **unidades**, siempre se encuentran en uno de dos posibles estados, activos o inactivos.

En la figura **las unidades en negro están activas, y las unidades en blanco están inactivas**. Las unidades están conectadas unas con otras por conexiones simétricas y con pesos.

Una **conexión con peso positivo** indica que las dos unidades tienden a activarse la una a la otra. Una **conexión con peso negativo** permite que una unidad activa desactive a su unidad vecina.

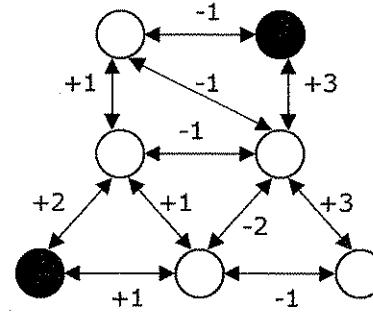


Figura 5.12

La red funciona del siguiente modo.

Se elige una unidad **aleatoriamente**. Si alguna de sus vecinas está activada, la unidad calcula la suma de los pesos en las conexiones de esas unidades. Si la suma es positiva, la unidad se activa y si ocurre de modo contrario, se desactiva.

Entonces se elige otra unidad aleatoriamente y se repite el proceso hasta que la red alcanza un **estado estable**, es decir, hasta que no quede ninguna unidad que pueda cambiar de estado. Este proceso se denomina **relajación paralela**.

Si la red comienza a trabajar en el estado que se muestra en la Figura 5.12, la unidad de la esquina inferior izquierda tenderá a activar la unidad que se encuentra por encima de ella. Esta unidad, por otro lado, tenderá a activar la unidad que se encuentra por encima de ella, pero la conexión inhibidora que procede de la unidad superior derecha aborta este intento, y así sucesivamente.

Esta red tiene únicamente cuatro estados estables distintos, que son los que se muestran en la Figura 5.13. Dado un estado inicial, la red necesariamente se asentará en una de estas cuatro configuraciones. La red puede verse como un "almacenador" de los patrones de la Figura 5.13.

**La mayor contribución de las redes de Hopfield es la de mostrar que dado un conjunto de pesos y un estado inicial, su algoritmo de relajación paralela en algún momento llevará a la red hacia un estado estable. No puede no existir divergencia u oscilación.**

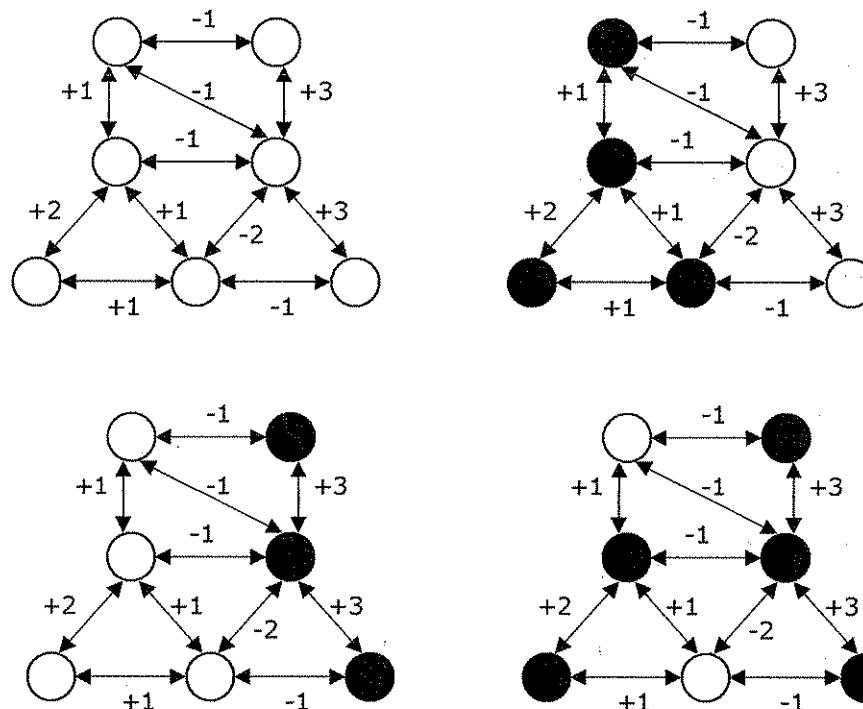


Figura 5.13

La red se puede utilizar como una memoria direccional por contenido haciendo que las actividades de las unidades se correspondan con un patrón parcial. Para recuperar un patrón, únicamente se necesita una parte de él. Entonces la red se asentará en el estado estable que mejor se empareje con el patrón parcial. En la Figura 5.14 se muestra un ejemplo de lo anterior.

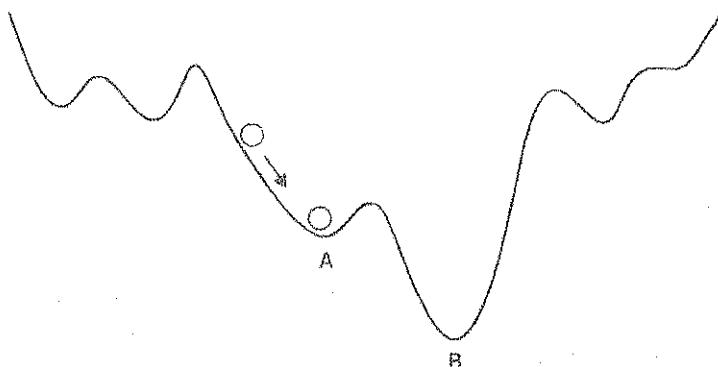


Figura 5.14

*La relajación paralela no consiste más que en una búsqueda.*

Resulta útil pensar en los diferentes estados de una red como formando un espacio de búsqueda como se muestra en la Figura 5.15.

Un estado que se haya elegido aleatoriamente se transformará a sí mismo en uno del tipo de mínimo local que es el estado estable más cercano. Esta es la forma de obtener un comportamiento direccional por contenido.

También tenemos un comportamiento corrector de errores. Supóngase que se tiene la descripción "gris, grande, pez, que come plancton". Automáticamente imaginaremos una ballena, aún sabiendo que ésta es un mamífero y no un pez. Por tanto aún cuando en el estado inicial existan inconsistencias, la red de Hopfield se asentará en la solución que viole el menor número posible de restricciones que ofrecen las entradas. Los procedimientos tradicionales de emparejamiento-y-recuperación son menos prohibitivos.

Ahora supóngase que una unidad falla de repente haciéndose activa o inactiva cuando no debe. Esto no representaría mayor problema, ya que las unidades que le rodean rápidamente volverían a ponerla en el camino correcto.

Sería necesario el esfuerzo de muchas unidades erróneas para hacer que la red se saliera de un estado estable. En las redes compuestas por miles de unidades altamente interconectadas, dicha tolerancia al fallo es todavía más patente, ya que las unidades y las conexiones pueden desaparecer completamente sin afectar de un modo adverso el comportamiento general de la red.

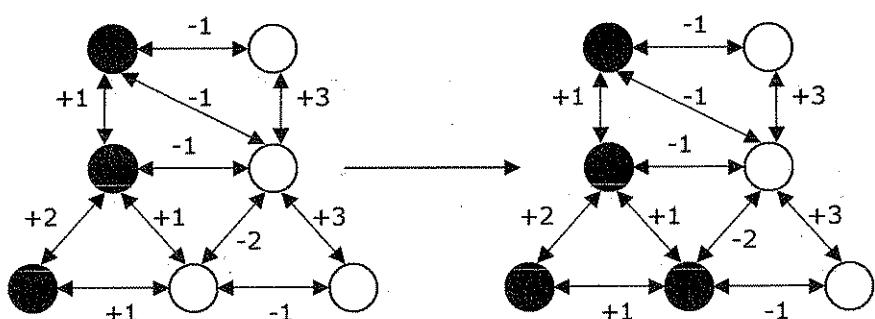


Figura 5.15

## 5.8. Máquinas de Boltzman

Una **máquina de Boltzmann** es una variación de una red de Hopfield.

Recuerde que en una red de Hopfield las unidades se conectan entre sí con pesos simétricos. Las unidades actualizan sus valores de forma asíncrona observando sus conexiones locales con otras unidades.

Las redes de Hopfield, además de servir como memorias direccionables por contenido, pueden resolver varios *problemas de verificación de restricciones*. La idea es contemplar cada unidad como una "hipótesis", y situar pesos positivos entre las unidades que representen hipótesis compatibles o que se apoyan mutuamente, y pesos negativos en las conexiones entre unidades incompatibles. Como una red de Hopfield converge hacia un estado estable, intenta asignar la certeza o falsedad de las distintas hipótesis al violar el menor número de restricciones posibles.

**El principal problema de las redes de Hopfield es que convergen hacia mínimos locales.**

Tener muchos mínimos locales es adecuado para construir una memoria direccionable por contenido, pero para las tareas de verificación de restricciones es necesario encontrar el estado globalmente óptimo de la red. Este estado se corresponde con una interpretación que satisfaga tantas restricciones como sea posible.

Desafortunadamente, las redes de Hopfield no pueden encontrar soluciones globales porque se sitúan en los estados estables por medio de un algoritmo completamente distribuido.

Si una red alcanza un estado estable como el de la Figura 5.15, ninguna unidad querrá cambiar su estado para moverse hacia arriba. Así, la red nunca alcanzará el estado óptimo B. Si varias unidades deciden cambiar su estado simultáneamente, la red podría escalar la subida y descansar sobre el estado B. Es necesaria una forma de situar las redes en los estados globalmente óptimos si mantenemos nuestro enfoque de algoritmo distribuido.

A la vez que se desarrollaron las redes de Hopfield, apareció en la literatura una nueva técnica de búsqueda denominada **enfriamiento simulado**.

**El enfriamiento simulado, es una técnica para encontrar soluciones globalmente óptimas en problemas combinatorios.**

Hinton y Sejnowsky (1986) combinaron las redes de Hopfield y el enfriamiento simulado para producir redes denominadas **máquinas de Boltzmann**.

Para entender como se aplica el enfriamiento, vuelva a la Figura 5.15 e imagínela como una caja negra.

Imagine una bola rodando por la caja. Si no puede verse dentro de la caja negra, ¿cómo podríamos convencer a la bola para que se vaya al valle más profundo? Sacudiendo la caja, por supuesto.

- Si se sacude demasiado violentamente, la bola podrá saltar de un valle a otro aleatorio. Esto es, si la bola estaba en el valle A, podría saltar al valle B; pero si la bola estaba en el valle B, ésta podría saltar al valle A.
- Sin embargo, si se sacude con demasiada suavidad, la bola podría saltar dentro del valle A, pero nunca saltar a otro valle.
- La solución que sugiere el enfriamiento es sacudir la caja violentamente al principio, y gradualmente ir haciéndolo más suavemente. En algún punto, la probabilidad de saltar de A a B es mayor que la de saltar de B a A. Será muy probable que de esta forma la bola se encuentre en el valle B, y como cada vez la sacudida es más suave, será incapaz de escapar de él. Esto es justo lo que queremos.

**¿Cómo se implementa esta idea en una red neuronal?**

*Las unidades de las máquinas de Boltzmann actualizan sus estados binarios individuales mediante una regla estocástica, en lugar de una determinista.*

La probabilidad de que una determinada unidad se active viene dada por p:

$$p = \frac{1}{1 + e^{-\Delta E / T}}$$

donde  $\Delta E$  es la suma de las líneas de entrada activas de las unidades, y T es la "temperatura" de la red.

La actualización estocástica de las redes es muy similar a la actualización de las redes de Hopfield, excepto en el factor que introduce la temperatura.

*A altas temperaturas, las unidades presentan un comportamiento aleatorio, mientras que a temperaturas bajas, las unidades se comportan como redes de Hopfield.*

**El enfriamiento es el proceso de ir gradualmente de altas a bajas temperaturas. La aleatoriedad que añade la temperatura ayuda a la red a escapar de los mínimos locales.**

Se trata de un procedimiento de aprendizaje para las máquinas de Boltzmann, es decir, un procedimiento que asigna pasos a las conexiones entre las unidades dando un conjunto de estados iniciales y finales de entrenamiento.

*Si el enfriamiento se produce de forma adecuada, las máquinas de Boltzmann podrían evitar los mínimos locales y aprender a calcular cualquier función calculable de entradas y salidas de tamaño fijo.*

## 5.9. Redes Recurrentes

Una clara deficiencia de los modelos de redes neuronales en comparación con los modelos simbólicos, es la *dificultad en conseguir modelos de redes neuronales que traten tareas de IA temporales como la planificación y el análisis del lenguaje natural.*

**Las redes recurrentes, o redes con lazos, constituyen un intento de remediar esta situación.**

Considere que se intenta enseñar a una red como lanzar un balón de básquet para que enceste.

A la red se le presenta una situación de entrada (distancia y altura del aro, posición inicial de los músculos), pero es necesario más de un vector de salida.

Se necesita una serie de vectores de salida: el primero mueve los músculos de este modo, luego de este modo, luego de este modo, etc.

Jordan (1986) inventó una red que hacía algo similar a esto. Se muestra en la Figura 5.16. Las unidades de salida simultáneamente dan órdenes (por ejemplo, mover el brazo x a la posición y) y actualizan las unidades de estado.

**La red nunca converge hacia un estado estable. En lugar de esto, cambia en cada paso de tiempo.**

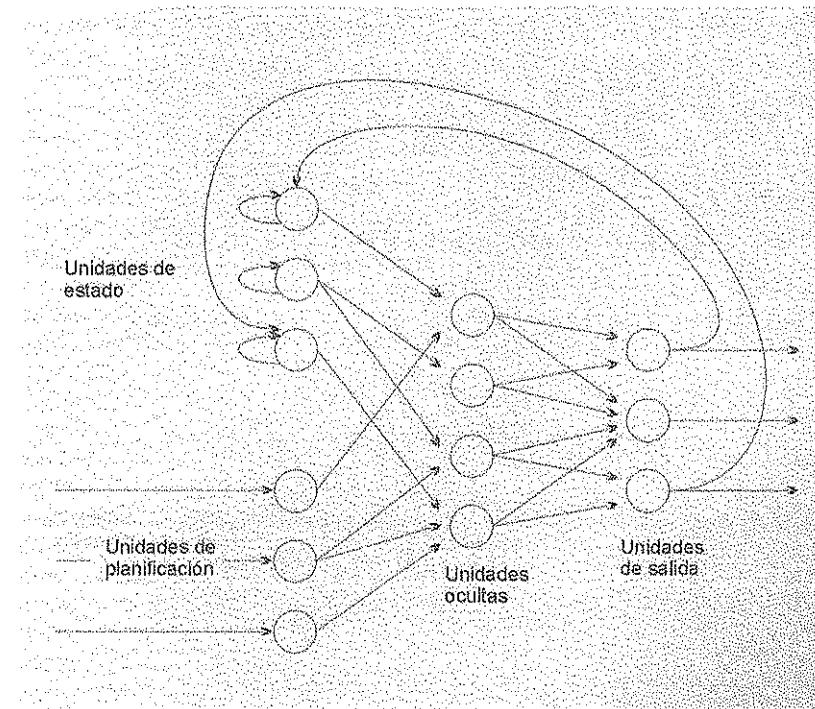


Figura 5.16  
Una red de Jordan

Las redes recurrentes se pueden entrenar mediante el algoritmo de propagación hacia atrás.

**En cada paso se comparan las activaciones de las unidades de salida con las activaciones deseadas y se propagan los errores hacia atrás por la red.**

Cuando se completa el entrenamiento, la red podrá llevar a cabo una secuencia de acciones.



