Asignatura: Introducción a la Inteligencia Artificial
Curso 2º de Ingeniería Técnica Informática (Sistemas).
Pedro Pérez Ostiz
Tudela.- Mayo 2002.

Inteligencia Artificial

Índice

Estos apuntes han sido realizados para el estudio de la asignatura Introducción a la Inteligencia Artificial del curso 2º de Ingeniería Técnica informática en la rama de Sistemas de la UNED.
He tomado base de otros apuntes realizados por *Ferran Gómez y José M. Godoy*, y de los libros de texto recomendados por la UNED:
Aspectos básicos de la Inteligencia Artificial.- J. Mira, A.E. Delgado, J.G. Boticario, F.J. Díez
Problemas resueltos de Inteligencia Artificial Aplicada.- Fdez. Galán, Gzlez Boticario, J. Mira.

Tudela Mayo 2002.
Pedro Pérez Ostiz

1 PERSPECTIVA HISTÓRICA Y CONCEPTUAL

1.1 Concepto de Inteligencia Artificial.

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

Para conseguir una visión razonablemente amplia del contenido de la inteligencia artificial usaremos criterios ***extensionales*** (porporcionando una relación lo más completa posible de los temas que estudia la inteligencia artificial), junto a otros criterios ***intensionales*** (que establecen las reglas de clasificación), de forma que al enfrentarnos con un problema computacional (o con un método de solución) específico podamos concluir si pertenece o no al campo de la inteligencia artificial en función.

Así, en la inteligencia artificial existen dos perspectivas básicas:

a) La inteligencia artificial como ciencia de lo natural o análisis

El procedimiento teórico busca una *explicación* de esa correlación en términos de un conjunto de leyes generales de un nivel superior que permiten predecir lo que ocurriría en otros casos no observados. Realmente lo que hace es buscar un modelo del conocimiento humano, generalmente organizado en varios niveles (estático, dinámico y estratégico) para poder usarlo en predicción. Esta técnica se basa en experimentos para conseguir una teoría del conocimiento computable con capacidad predictiva (como una ley física)

b) Inteligencia artificial como ciencia de lo artificial o ingeniería de síntesis

Aspira a convertirse en una ingeniería en sentido estricto. Ahora se parte de un conjunto de especificaciones funcionales y se busca la síntesis de un sistema (programa más máquina) que las satisfaga.

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

- ☞ Computación ***simbólica***, de grano grueso y programable.
- ☞ Computación ***conexionista***, de grano pequeño y autoprogramable por aprendizaje.

En inteligencia artificial trabajamos con información y conocimiento, y ambos son pura forma, totalmente independiente del sistema físico que las soporta. El resultado es un programa de ordenador sobre una máquina específica desarrollado a partir de un modelo del conocimiento que supuestamente usa el operador humano que realiza esa tarea.

Las tareas que aborda la inteligencia artificial de síntesis son tareas de alto nivel, y pueden clasificarse en tres grandes grupos ordenados en grado de dificultad creciente:

1. **Dominios formales**.- Las tareas toman la forma genérica de *solucionadores de problemas* mediante búsquedas en un espacio de estados de conocimiento y pueden ser juegos o problemas *lógico-matemáticos.* Son tareas precisas en el conocimiento, con pocos elementos y su comportamiento se puede describir de forma completa e inequívoca. Son *micromundos formales*. Este apartado forma parte de la etapa inicial de la inteligencia artificial y aportó los procedimientos de búsqueda como tarea genérica

2. **Dominios técnicos**.- Tienen que ver con el *diagnóstico médico,* la *detección de fallos,* la *planificación* de trayectorias de robots, etc. Aquí. La tarea a sintetizar admite una representación dentro de una jerarquía de tareas genéricas de análisis, de modificación o de síntesis que son válidas en muchas aplicaciones con sólo modificar la parte del conocimiento que hace referencia a entidades específicas del dominio de la aplicación. La característica de estas tareas es el carácter limitado del conocimiento que manejan (dominios estrechos) y la posibilidad de formalizar ese conocimiento con las técnicas disponibles. Ha dado lugar a la *Ingeniería del conocimiento* y busca procedimientos de síntesis de sistemas con las siguientes facetas:

a) se parte de la descripción de la tarea a nivel de conocimiento (Allen Newell). Para ello es necesario realizar un proceso de obtención de ese conocimiento a partir del experto humano que lo posee.
b) se busca una representación de ese conocimiento separándolo de los mecanismos de aplicación del mismo (inferencia) de forma que pueda acumularse por procedimientos incrementales (donde no es deseable una separación total)
c) se seleccionan las técnicas adecuadas para su implementación y se desarrolla un primer prototipo
d) se hace énfasis en el carácter de *ingeniería* buscando procedimientos (explícitos, reproducibles y parcialmente independientes del dominio) sistemáticos de implentación, evaluación y refinamiento de esos prototipos
e) se usan lenguajes y entornos de programación que facilitan el desarrollo rápido y eficiente de aplicaciones

3. **Funciones básicas y genuinas del comportamiento humano**.- Realmente es lo que hacemos a todas horas sin darnos cuenta; ver, oír, caminar, pensar, hablar, etc. Por su importancia se le va a dedicar el siguiente apartado.

1.2 Idea intuitiva del comportamiento artificial

El concepto de inteligencia está sacado de la psicología cognoscitiva y pertenece a una familia más amplia de construcciones teóricas para ayudar en la descripción del comportamiento observable de sistemas complejos en interacción con su medio. Hablar de la inteligencia artificial en ese sentido supone querer comprender y duplicar las funciones del comportamiento humano. Algunas de sus características son:

a) Su aparente simplicidad en el ser humano.
b) Lo complejo que son los procesos cognoscitivos elementales a la hora de sintetizarlos.
c) El uso masivo de conocimientos, suponiendo que los mecanismos de inferencia por encadenamiento de reglas son insuficientes para modelar tareas cognoscitivas (hace falta un lenguaje de representación con la capacidad y robustez del lenguaje natural).
d) El estilo peculiar de computación que usa el ser vivo. Toda computación artificial es extensional, por el contrario hay evidencias sobre el carácter intensional (por propiedades) e intencional (por propósitos) de la computación biológica, donde la inferencia es inmediata (refleja).
e) El reconocimiento (según *Maturana* y *Varela*) de que todo conocer depende de la estructura que conoce (lo cognoscitivo es propio de lo vivo).
f) La hipótesis fuerte de la IA es hacer computacional este conocimiento propio de lo vivo (reducir los procesos cognoscitivos a un nivel simbólico).

En un diagrama funcional de un ***agente inteligente*** (Newell y Simon), el agente interacciona con su medio a través de un conjunto de sensores. Posteriormente se realiza un procesamiento multisensorial de alta semántica con referencia a contenidos de memoria a la que se llama ***percepción***. El objetivo de ambos procesos es identificar al medio de acuerdo con un modelo de representación interna que permite comprender el significado de imágenes y palabras. El agente realiza también tareas motoras que inciden en el medio, suelen ser acciones de semántica baja o media. El grado de semántica de una señal es proporcional a la capacidad operacional del símbolo que transporta.

Entre estas dos familias de tareas (*perceptuales* y *motoras*) existe un conjunto intermedio de tareas de decisión que trabajan entre espacios de representación, sin conexión directa con el medio. Incluyen los procesos cognoscitivos asociados al pensamiento y al lenguaje. Para la realización de estas tareas el agente posee un modelo del medio y un conjunto de propósitos y para alcanzar sus metas usa el *'principio de racionalidad'* (según Newell).

El principio de racionalidad no es operacional, es decir, nos dice ***qué hacer*** pero no ***cómo hacerlo***. Cada vez que se encuentra un procedimiento efectivo para identificar, capturar y representar de forma computable el conocimiento necesario para la síntesis no trivial de esas tareas de percepción, decisión o planificación motoras tenemos un sistema de IA. El problema de la IA aplicada es pasar del nivel de especificaciones al nivel de componentes. Toda la computación termina en el nivel físico de la electrónica digital. La clave de la Inteligencia Artificial es conseguir programas traductores intermedios que conecten las primitivas de bajo nivel con las del lenguaje de representación cada vez más próximo al ***lenguaje natural***.

Los criterios intensionales (por propiedades) permiten distinguir las fronteras de la Inteligencia Artificial con las otras ramas de la computación. Tenemos un problema de Inteligencia Artificial siempre que:

- ☞ No exista una solución analítica o algorítmica conocida.
- ☞ Cuando existiendo esa solución, la explosión combinatoria la haga ineficiente.
- ☞ Cuando el conocimiento necesario es masivo, incompleto, complejo y difícil de representar.
- ☞ Cuando es necesario el aprendizaje y la inyección de conocimiento del dominio.
- ☞ Siempre que abordemos tareas cognoscitivas que usen conocimiento de sentido común.

Y tendremos una solución de problemas propia de la inteligencia artificial cuando:

- ☞ Utiliza una estructura de tareas genéricas que permite capturar los aspectos generales del problema y de sus procedimientos de solución de forma que las situaciones individuales se tratan por los métodos asociados a las clases a las que pertenecen.
- ☞ Usa heurísticas que intentan capturar el conocimiento accesible (incompleto e impreciso) del dominio.
- ☞ Separa el conocimiento de su uso en inferencia y hace énfasis en el primero
- ☞ Permite manejar el razonamiento impreciso y temporal
- ☞ Incluye algún tipo de aprendizaje: simbólico o conexionista. Sin aprendizaje no hay inteligencia artificial.

Para entender el paso de la computación anatítico-algorítmica a la inteligencia artificial es preciso dar un salto desde el conocimiento que es necesario inyectar desde el exterior del sistema para entender el proceso (pasar del nivel bajo al nivel alto).

En el proceso de **bajo nivel** se pueden describir tres *pasos*. En el procesado de bajo nivel prácticamente toda la información está en la imagen (para el reconocimiento de imágenes, por ejemplo). El segundo paso es el *preproceso* que extrae características locales o integrales de naturaleza analítica que no exigen conocimiento complementario para ser entendidos por un observador (descripción complementaria). La etapa final del procesado de bajo nivel es el reconocimiento de formas basado en la definición analítica de distancia entre el valor que toman las propiedades usadas para describir la imagen y los valores correspondientes a esas variables en un conjunto de patrones.

En el procesado de **alto nivel** (percepción), nos hace falta recurrir a la inyección de conocimiento externo del dominio para dar significado a las estructuras de datos y a los procesos (porque su sentido sólo quede claro para quien posee ese conocimiento y en ningún caso es evidente a partir de las entidades del nivel simbólico o del nivel fisico en el caso del conexionismo). Ese segundo nivel, de comprensión de imágenes, es responsabilidad de la inteligencia artificial.

Conocemos ya algo sobre los límites de la inteligencia artificial por debajo (en su frontera con la analítica y la computación numérica). Veamos ahora su frontera con lo humano. Esos límites pueden encontrarse, al menos, en los siguientes puntos:

a) Desconocimiento del operador humano.
b) Falta de teoría (principios organizacionales y estructuras).
c) Diferencias fundamentales entre el nivel fisico de la computación (cristales semiconductores) y el propio de los seres vivos (tejidos biológicos).

No se puede afirmar que la mente es reducible a computación si no se conoce como funcionan las neuronas biológicas, ni cómo está organizado el cerebro. La metáfora computacional supone que todos los procesos pueden moldearse mediante dos espacios de representación (el de entradas y el de salidas), y un conjunto de reglas de transformación que proyectan configuraciones del espacio de entradas en las configuraciones correspondientes del espacio de salidas.

Por otro lado, la inteligencia artificial de síntesis no tiene por qué depender de la comprensión de lo vivo. Es decir, una forma alternativa y eficiente de extender los límites de la inteligencia artificial es desarrollarla como ciencia y tecnología de lo artificial, sin referencia directa con la biología. Entonces deben resolverse, al menos, las siguientes cuestiones:

1. Modelado y representación de tipos de inferencia y de entidades y relaciones propias del dominio usando lenguajes más próximos al lenguaje natural (con su robustez, flexibilidad y capacidad representacional).
2. Búsqueda de nuevos principios de autoorganización capaces de generar estructuras simbólicas robustas y de amplio uso a partir de entradas de información más desorganizadas y fragmentadas.
3. Desarrollo de nuevos lenguajes de programación, tan funcionales como los actuales y, además, que permitan un cierto nivel de autoprogramación y ayuda a la edición de conocimiento.
4. Énfasis en las teorías computacionales del aprendizaje tanto simbólico como conexionista o híbrido.

1.3 Perspectiva histórica de la inteligencia artificial

1.3.1 Etapa neurocibernética

La Inteligencia Artificial comenzó a ser computacional cuando Warren S. McCulloch y Walter Pitts introducen el primer modelo formal, formado por una función lógica seguida de un retardo y donde se sustituye la programación por el aprendizaje. Una red de neuronas formales es equivalente a una máquina de Turing de cinta finita. Si la red es programable, entonces es equivalente a una máquina universal.

Las ideas básicas de esta época se basan en considerar que los seres vivos y las máquinas pueden ser comprendidas usando los mismos principio de organización y las mismas herramientas formales. La aproximación entre ambos debe realizarse a nivel de procesador, de forma que para estudiar como computa el cerebro, hay que estudiar las neuronas biológicas, modelarlas formalmente, construir redes y ver como surge el comportamiento a partir de procesos locales de autoorganización, memoria asociativa y aprendizaje. Tres trabajos fundacionales del movimiento son:

- en ***Conducta, propósito y teleología*** (de Rosemblueth, Wiener y Bigelow) se introducen tres conceptos importantes en inteligencia artificial: la *realimentación* como principio organizacional, la *computación* por propósitos y la idea de *información* como pura forma, separable de la señal física que la transporta.
- en ***Un cálculo lógico de las ideas inmanentes en la actividad nerviosa*** (de Warren S. McCulloch y Walter Pitts), sobre redes neuronales formales, se inicia la "*Teoría Neuronal Del Conocimiento*"; se buscan las redes de procesadores capaces de reconocer, recordar, cooperar, aprender o autoorganizarse. En esta primera etapa de la inteligencia artificial se busca la solución de los problemas a nivel físico (donde estructura y función coinciden). Se inicia la teoría modular de autómatas y se usa la lógica (determinista y probabilística) para representar el conocimiento.
- en ***La naturaleza de la explicación*** (de K. Craik) se interpreta la actividad del sistema nervioso en términos de un conjunto de procesos encaminados a construir una representación interna del medio (modelo) y usarla para *predecir*. Craik contribuyó a la moderna inteligencia artificial con dos aportaciones clave: *razonamiento abductivo* y *espacios de representación.*

La inferencia en inteligencia artificial está asociada al uso individual o combinado de tres tipos de razonamiento:

1. La ***deducción lógica***: Se parte de un conjunto de fórmulas (axiomas o validez general) y sobre ellas se aplican un conjunto de reglas o procedimientos de demostración que nos permiten obtener nuevas fórmulas válidas. Permite pasar de lo general a lo particular con certeza del resultado obtenido. Su punto débil es la necesidad de conocer con precisión el conjunto de axiomas que define el dominio.
2. La ***inferencia inductiva:*** Usa pistas (heurísticas) con el conocimiento del dominio para pasar de lo particular a lo general. Nunca podemos garantizar la completitud y certeza de la inferencia en este caso.
3. En el ***razonamiento abductivo:*** Trata situaciones en las que se conoce una relación causa efecto y un suceso, y, se debe lanzar una hipótesis sobre su causa más probable. (como en los diagnósticos médicos).

Según Craik: *"Nuestra pregunta no es qué es implicación o causalidad, sino cuál es la estructura y los procesos necesarios para que un sistema mecánico sea capaz de imitarlos correctamente y predecir sucesos del mundo externo, a la vez que crear nuevas entidades".* El segundo punto de la obra de Craik es la propuesta de un mecanismo de razonamiento por analogía en el modelo del medio donde la implicación formal es el equivalente a la causalidad en el mundo físico. Distinguía Craik tres procesos:

A. Traslación de los procesos externos a símbolos en un espacio de representación.
B. Obtención de otros símbolos mediante inferencia en el modelo del medio que paraleliza la causalidad externa.
C. Retraslación de esos símbolos transformados al dominio de sus referentes externos (predicción).

La característica fundamental de un **Sistema Basado en el Conocimiento** (SBC) es su poder para modelar sucesos basados en dos tipos de símbolos: números y palabras. La propuesta de Craik es que la codificación de estos símbolos no es arbitraria, sino que mantiene su identidad desde la entrada sensorial a la salida motora. La Inteligencia Artificial en su primera etapa neurocibernética aborda los siguientes temas:

- Redes de neuronas formales.
- Algoritmos de análisis y síntesis.
- Aprendizaje por ajuste de parámetros.
- Tolerancia a fallos.
- Modelos de memoria asociativa.
- Reconocimiento de caracteres.
- Aprendizaje asociativo y por refuerzo.
- Desarrollos en Teoría Modular de Autómatas.
- Codificación y Teoría de la Información.
- Principios Organizacionales:
  - Autoprogramación.
  - Autorreproducción.
- Programas programadores.
- Traducción automática.
- Comprensión del Lenguaje Natural

1.3.2 computación: de Platón a Turing

Dreyfus sugiere que la inteligencia artificial comenzó alrededor del año 450 a. C. cuando, de acuerdo con Platón, Sócrates pregunta a Euthyphro por un conjunto de *reglas de decisión* definidas de forma tan precisa que en cada momento pudiéramos calcular la respuesta del sistema aplicando esas reglas a la entrada.

Hobbes (1.588-1679) afirma que el pensamiento (razocinio) consiste en ejecutar operaciones simbólicas siguiendo de forma metódica un conjunto de reglas de decisión.

Descartes (1.566-1650) descubre el carácter invariante de la matemática para representar el conocimiento. Intenta formalizar el razonamiento usando tres procesos: *enumeración*, *deducción* e *intuición*, siendo este último el más difícil de mecanizar.

Leibniz (1.646-1716) introduce el concepto de programa y de programación.

Boole (1815-1860) introduce el modelo lógico haciendo énfasis en la separación de símbolos y números y en la posibilidad de usar los primeros como objetos de cálculo.

Un aspecto importante de los desarrollos en Inteligencia Artificial es el de intentar mantener las ventajas del modelo lógico y disminuir los inconvenientes que supone su carácter binario e invariante de sus valores. La lógica borrosa, la probabilística y las extensiones del razonamiento no monótono y temporal son muestra de ello.

Con von Neumann se avanza en *arquitectura de computadores, teoría modular de autómatas* y *redes neuronales y teoría del cerebro*. Su contribución a la teoría de autómatas y a la inteligencia artificial conexionista no fue sólo a nivel formal, sino que planteó cuestiones fundamentales como son:

- *Reformulando la máquina de Turing en términos de autómatas celulares*
- *Autoprogramación* (autómatas que diseñan otros autómatas)
- *Autoreproducción* y *evolución* (constructores universales que se reproducen)
- *Tolerancia a fallos* y *establildad lógica* ante cambios de función local

Finalmente, llegamos a Turing y sus dos contribuciones básicas:

A. Un *Modelo Computacional Universal* (la máquina de Turing)
B. Un *Procedimiento Experimental de Medir la Inteligencia Artificial de un Programa* (test de Turing)

La Máquina de Turing es una máquina matemática, todo proceso que se describe de forma clara, completa, precisa e inequívoca en lenguaje natural puede representarse mediante un autómata. A partir de esta representación se puede encontrar una secuencia de instrucciones que al operar sobre símbolos codificados genera un programa ejecutable en cualquier computador. Desde este punto de vista todas las arquitecturas son equivalentes, y sus diferencias son los factores de complejidad, tamaño, tiempo, eficacia o recursos.

El test de Turing parte de un operador humano, un programa de ordenador y un interrogador que realiza un conjunto de preguntas, y compara las respuestas de ambos. Si de la comparación de las respuestas el observador no es capaz de distinguir que información corresponde al operador humano y cual al programa de Inteligencia Artificial, se dice que para esa tarea en ese dominio se supone el mismo nivel de inteligencia a ambos operadores.

Moov, modificó el esquema en términos de acumulación de evidencia inductiva, a partir de una serie de críticas, la más fuerte es sobre el carácter conductista del test de Turing. La contestación adecuada a esta crítica es la de considerar el test más que una medida de inteligencia como un procedimiento de recoger y acumular evidencias inductivas acerca de la eficacia del programa para resolver un problema.

Esta interpretación inductiva elimina parcialmente la crítica de Searle sobre la diferencia entre el conocimiento superficial (diccionario) y el profundo. Este segundo caso supone que el agente debe comprender el significado del mensaje a nivel semántico generando otro mensaje en otro lenguaje, con el mismo significado, sin referencia a la sintaxis del mensaje de partida.

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

La capacidad de aprendizaje de un programa es la siguiente medida importante para evaluar el nivel de Inteligencia Artificial.

1.3.3 búsqueda heurística y dominios formales

El primer trabajo fue el programa *Logic Theorist* que da origen a toda la rama del razonamiento automático que persiste en la inteligencia artificial, reforzado por el *principio de resolución* de Robinson y sus refinamientos posteriores, incluyendo la creación del lenguaje **Prolog**.

Comienza la preocupación por *lenguajes para procesar información* más orientados al manejo de símbolos que al cálculo numérico, iniciándose el camino hacia el **Lisp**.

Otro trabajo representativo es el programa **GPS** (*Solucionador General de Programas*) con un intento de dotarlo de capacidad de aprendizaje y autoorganización; es un programa que incorpora medios heurísticos para resolver problemas seleccionando una secuencia de operadores que haga disminuir la diferencia entre el estado inicial y el estado meta hasta anularla.

Todos los trabajos de esta primera época se centraron en problemas propios de dominios formales, demostración de teoremas, estrategias heurísticas y problemas de juegos, planificación de acciones, etc. En todos los casos la Inteligencia Artificial se entiende como búsqueda en un espacio de estados sin conocimiento del dominio, haciendo énfasis en los métodos de control de la búsqueda. Actualmente se intenta recuperar las propuestas iniciales del GPS, el propósito es separar los aspectos estructurales comunes al procedimiento de solución de una clase general de problemas, del conocimiento específico de una tarea particular.

A mediados de los sesenta aparecen cambios en la orientación de la Inteligencia Artificial dando cada vez más importancia al conocimiento del dominio y por consiguiente a los problemas asociados a su representación y posterior uso en inferencia, donde se separan conocimiento y uso del mismo.

En esta época se inician los estudios del lenguaje natural, limitando sus dominios, para conseguir un modelo completo de las estructuras y procesos de ese dominio. Aparece la llamada "falacia del primer paso" (Dreyfus) ya que aunque se suponía que todos los programas eran un primer paso para la comprensión del lenguaje natural, nunca llegó el segundo paso, que lo hizo vía los sistemas basados en el conocimiento.

McCarthy realizó un trabajo sobre *programas con sentido común*, donde afirmaba: "podemos decir que un programa tiene sentido común si deduce de forma automática y por si mismo una clase suficientemente amplia de consecuencias inmediatas de cualquier cosa que se le dice y él ya conoce.

1.3.4 énfasis en el conocimiento (197x-198x)

La complejidad de los problemas que aborda la Inteligencia Artificial aconseja combinar todas las técnicas de representación, integrando el simbolismo con las redes neuronales. La segunda idea que caracteriza esta segunda etapa de la Inteligencia Artificial es que no basta con el énfasis del conocimiento sino que además la Inteligencia Artificial de síntesis (ingeniería) debe centrarse en tareas científico - técnicas en dominios estrechos donde se pueda abarcar el conocimiento de forma completa y sin influencia del sentido común. Se habla así de motivación inteligente, clasificación basada en el conocimiento, consejeros de terapia médica, sistemas de supervisión y detección de alarmas, etc.

El nacimiento de los Sistemas Basados en el Conocimiento y los Sistemas Expertos ocurre en una época dominada por dos preocupaciones:

(a) Énfasis en la representación computacional del conocimiento para tareas del mundo real. El conocimiento específico del dominio es *poder*.
(b) Selección de tareas técnicas en dominios estrechos (Sistemas Expertos), donde se separa el conocimiento de sus mecanismos de aplicación (inferencia).

1.3.4.1 representación del conocimiento

El problema de la representación del conocimiento sigue abierto tanto en la Inteligencia Artificial teórica, (*¿qué es?, relación entre el conocer y la estructura de lo que se conoce*) como en la Inteligencia Artificial de síntesis (*¿qué representación es mejor?, ¿cuál permite un uso eficiente del conocimiento?, ¿cómo abordar la semántica?, ¿cómo profundizar el razonamiento?*). La conclusión de esta etapa es una propuesta de representación modular e híbrida que incluye aspectos de los cuatro procedimiento básicos: **lógica**, **reglas**, **redes asociativas** y **marcos** (objetos estructurados).

En esta época se considera al conocimiento como 'algo' transportable desde la cabeza del experto humano a la memoria del computador. La visión actual considera el problema de la representación formal del conocimiento como un proceso de modelado o de reconstrucción de lo que se supone que sabe el experto humano, estructurado en varios niveles.

El uso de la lógica para representar el conocimiento tiene como ventajas la semántica, expresividad y la eficacia de sus procesos deductivos. Sus inconvenientes provienen de la falta de estructuración del problema de la completitud y de la existencia de problemas en los que la conclusión se obtiene por defecto, sin garantizar que los resultados son ciertos. Esta ruptura de la monotonía (suposición usual en lógica) ha dado origen a la búsqueda de nuevos formalismos para el razonamiento no monótono, que junto al aproximado, el temporal y el cualitativo han dado lugar la estado actual de las investigaciones.

Para intentar solucionar el problema de la ineficiencia de las representaciones lógicas surgen las representaciones basadas en reglas, sacrificando expresividad por eficiencia. Estos sistemas aportan un nuevo paradigma de programación.

Parten de la idea de que todo el conocimiento relevante para realizar una tarea puede representarse mediante un conjunto de condicionales. El resultado de aplicar la regla produce nuevos hechos y activa otras reglas, de forma que el razonamiento está asociado a ese encadenamiento de reglas.

Las redes semánticas (Ros Quillian) son un modelo de memoria asociativa adecuado para la representación computacional del conocimiento necesario para la comprensión y traducción del lenguaje natural. Selts (1.922) en su teoría de la anticipación esquemática propuso una organización relacional del conocimiento en términos de redes de conceptos (nodos) enlazados por relaciones (arcos) y con herencia de propiedades. Quillian retomó esta línea; la base de conocimientos en una gran red de nodos (significado de palabras) interconectadas por diferentes tipos de arcos asociativos. Las redes asociativas engloban a las redes semánticas, siendo estas últimas un caso particular que hace referencia a su origen, como forma de representar el significado (semántica) de las palabras en lenguajes naturales.

Las tres características que distinguen los distintos tipos de redes son:

(I) • Tipo de conocimiento representado.
(II) • Método de inferencia.
(III) • Forma de abordar el problema del aprendizaje.

Los marcos (Minsky) como forma de representación del conocimiento suponen que el sistema nervioso organiza su experiencia perceptiva y su memoria en términos de un conjunto de esquemas que describen de forma estructurada nuestro conocimiento sobre un conjunto de situaciones reales, de forma que al usarlos para razonar se acepta gran cantidad de conocimiento implícito con el que se rellenan los campos que quedan sin especificar.

Una característica importante de los marcos es su ordenación jerárquica, los *marcos-nodo* inferiores no sólo heredan las propiedades de sus superiores sino que pueden heredar también valores y métodos asociados a estas propiedades.

Los guiones y planes son análogos a los marcos, sólo que hacen referencia a estructuras que describen secuencias temporales de sucesos. El ser humano organiza su conocimiento en términos de millones de marcos, guiones y planos que usa constantemente en todas sus tareas(Schank, Minsky).

Los avances desde 1.980 hasta nuestros días se resumen en los siguientes puntos:

- ☞ Mayor rigor en los desarrollos teóricos.
- ☞ Énfasis en las representaciones declarativas frente a las procedimentales.
- ☞ Intento de formalización e integración de los distintos tipos de razonamiento. Se camina hacia una teoría unificada del conocimiento.
- ☞ Renacimiento del conexionismo y surgimiento de las redes bayesianas.
- ☞ Manejo de grandes bases de conocimiento, lo que exige el desarrollo de métodos automáticos de adquisición y actualización (aprendizaje).

1.3.4.2 sistemas basados en el conocimiento (SBC) y sistemas expertos (SE)

Cuando en un sistema se hace uso intensivo del conocimento del dominio y se separa de los mecanismos que controlan su uso en inferencia, decimos que tenemos un SBC. Dentro de un SBC hay un grupo de sistemas en los que el conocimiento procede de un experto humano especialista en una tarea concreta y un dominio técnico; decimos entonces que tenemos un SE. El desarrollo de los métodos básicos de representación del conocimiento ha caminado paralelo al desarrollo de los sistemas expertos. Cada forma de representación está asociada históricamente al primer sistema que la usó. Las características fundamentales de un SE son:

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

En la primera época los módulos básicos de un S.E. eran la **base de hechos ciertos**, la **base de conocimientos**, el **mecanismo de inferencia**, una **base de datos convencional** y un **módulo de explicación y diálogo** (esquema que corresponde a un sistema basado en reglas). Si la representación del conocimiento se hace usando marcos o redes semánticas, la estructura no se corresponde claramente con estos módulos. Una conclusión importante de esta etapa de la Inteligencia Artificial es que los sistemas basados en reglas constituyen un procedimiento efectivo de representación y uso del conocimiento distinto de la programación convencional. La computación se realiza identificando y aplicando la regla adecuada a un hecho. Después, se estudia si como consecuencia de la aplicación, se ha resuelto el problema, y en caso contrario, buscar una nueva regla cuyo campo de condición se adapta al campo de acción de la anterior. Como consecuencia se altera el contenido de la memoria dinámica del sistema. Esta forma de programar supone dos cambios esenciales:

- Todo conocimiento debe representarse de forma modular y declarativa.
- En el modelado se debe cuidar el contenido asignado a los campos de condición y acción de cada regla porque ahí se construye el conocimiento que guiará el encadenamiento de las reglas.

En mitad de los sesenta, los resultados de los trabajos de Stanford mostraron las características básicas en todos los SE:

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

Y a partir de otro trabajo:

5. Validez del razonamiento por encadenamiento de reglas.
6. Tratamiento del problema de la incertidumbre mediante mecanismos sencillos y eficientes que combinan distintos factores de certeza.
7. Capacidad de *explicación* del razonamiento seguido para alcanzar la meta que propone.
8. Mención de conceptos que se consolidaron más tarde, tales como la *metarreglas* (reglas que explican como utilizar otras reglas) y la *adquisición de conocimiento* como tarea genérica en inteligencia artificial. Las metarreglas se introducen con tres fines: construir la base de conocimientos mediante un diálogo con el experto humano, guiar el razonamiento mediante objetivos explícitos y mejorar la capacidad de explicación a partir de dichos objetivos.

En la actualidad las tendencias en el campo de los SE son:

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

1.3.5 aprendizaje y renacimiento del conexionismo

1.3.5.1 aprendizaje

La idea más general de aprendizaje es la acumulación de conocimiento. Por consiguiente, un programa aprende cuando es capaz de acumular conocimiento sobre una tarea. Bajo el nombre de aprendizaje, se engloban procesos muy diversos que podemos clasificar en términos de la tarea genérica a la que se refiere y del mecanismo de razonamiento en el que se basa. Hay tres familias de tareas:

1. tareas perceptivas:
Incluye aspectos tales como el reconocimiento de caracteres y la formación de conceptos (selección de características para describir los objetos de entrada, creación de un lenguaje de descripción de conceptos y representación simbólica de todo el conocimiento previo).
2. tareas de planificación:
Incluye los aspectos complementarios de la percepción. Ahora en vez de clasificar, se parte de conceptos centrales en un modelo del medio y se programa un generador de acciones elementales.
3. tareas de organización central:
Incluye aspectos tales como la adquisición automática de nuevo conocimiento declarativo y su integración en una organización interna.

Si consideramos ahora el aprendizaje como el cambio en los mecanismos de razonamiento, hay tres paradigmas básicos:

1. el aprendizaje inductivo:
En él, los cambios en las estructuras de datos y en los algoritmos van encaminados a la generalización del conocimiento extraible de los ejemplos usados en el entrenamiento. Ese proceso está guiado por criterios heurísticos. Si se dispone de mucho conocimiento del dominio no es necesario mucho entrenamiento, se tiene una estrategia de aprendizaje basado en casos (CBL).

2. el aprendizaje deductivo:
Está asociado a situaciones en las que se dispone de un conocimiento general bastante completo y unas reglas de inferencia para obtener casos bajo la ley y explicar el proceso deductivo. Al tener que encontrar teorías completas se debe complementar con otras estrategias inductivas o abductivas.

3. el aprendizaje abductivo:
Busca un procedimiento para proponer y seleccionar las hipótesis que mejor expliquen las conclusiones conocidas.

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

- ☞ Desarrollo de sistemas multiestrategia, basados en la comprensión de las funcionalidades y las condiciones de aplicación de las distintas estrategias individuales y en la cooperación de estas en función del tipo de problema.
- ☞ Desarrollos teóricos y metodológicos distinguiendo entre las funcionalidades en el dominio propio y en el del observador y proponiendo un nivel de procesos comunes en términos de los que las distintas estrategias son equivalentes.

1.3.5.2 conexionismo

El salto cualitativo en la perspectiva conexionista de la inteligencia artificial aparece al comienzo de los años 80 con las propuestas de Rumlhart y colaboradores [1986] y Barto [1983]. En la inteligencia artificial conexionista se está intentando resolver los mismos problemas que han preocupado a la inteligencia artificial simbólica. El contenido en extenso de la inteligencia artificial conexionista aborda los siguientes temas:

1. Conexiones con la neurociencia.
2. Modelos de computación distribuida y autoprogramable.
3. Búsqueda de arquitecturas multicapa e incrementales y genéticas que faciliten la cooperación y la autoorganización.
4. Estudio del aprendizaje supervisado y no supervisado y su conexión con el aprendizaje simbólico.
5. Desarrollo de neurosimuladores (entornos de programación, evaluación y desarrollo de redes neuronales a nivel software).
6. Implementación de redes como arquitecturas paralelas de propósito especial (preprocesadores, coprocesadores o neuro-circuitos completos).
7. Un gran capítulo de aplicaciones, en percepción y control, y problemas genéricos de clasificación en tiempo real.

Como conclusión decir que ambas perspectivas (simbólica y conexionista) han reconocido sus limitaciones y han decidido cooperar buscando formulaciones híbridas de tareas genéricas y soluciones cooperativas que usan ambos métodos tomando de cada uno lo más adecuado para la solución del problema.

2 ASPECTOS METODOLÓGICOS EN IA

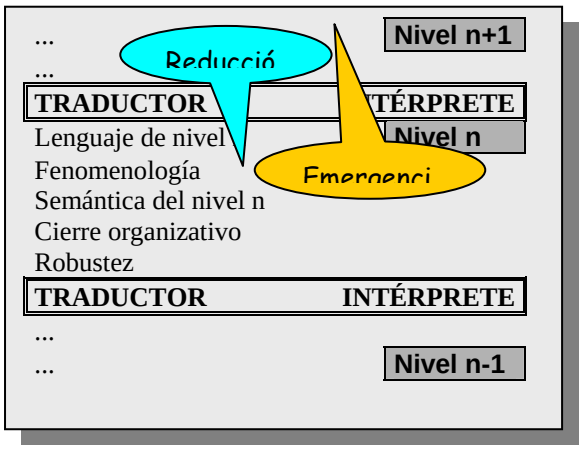
2.1 Niveles de computación.

Cuando nos enfrentamos a un sistema complejo, tanto en tareas de *análisis* como de *síntesis*, es útil usar una **jerarquía de niveles**, que nos permitan segmentar esa complejidad.

Cada nivel de descripción, está caracterizado por una fenomenología, un conjunto de entidades y relaciones y unos principios organizativos y estructurales propios. (Por ejemplo, en biología, se puede pasar del nivel *protoplásmico* <subcelular con procesos fisicoquímicos>, al *celular* <bioquímico-eléctrico>, al *orgánico* y al de comportamiento *global* <animal y humano>).

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

Cada nivel enlaza con el inferior mediante un proceso de **reducción** de acuerdo con unas leyes (programas *traductores*, *intérpretes*, *compiladores*), y con el superior mediante procesos de **emergencia**, que necesitan la inyección de *conocimiento* por parte de un *observador externo*. Por ejemplo: del conocimiento detallado del comportamiento de los inversores y biestables, no se deduce el algoritmo ni las estructuras de datos del programa que está corriendo. Análogamente, no se deduce el funcionamiento de los biestables conociendo el algoritmo utilizado. Hay varios algoritmos posibles para una misma computación y hay múltiples implementaciones posibles para un algoritmo.



La teoría de niveles en la computación fue introducida por D. Marr y A. Newell, previamente Chomsky introdujo los conceptos de competencia y ejecución para distinguir los lenguajes naturales de los sistemas arbitrarios y formales de manipulación de símbolos.

Todo nivel de computación admite una representación general en términos de un espacio de entradas, un espacio de salidas y un conjunto de reglas de transformación que enlazan las representaciones en ambos espacios de representación. El espacio de entradas en un espacio multidimensional de las características del proceso que se consideran relevante. El espacio de salidas también es multidimensional, en el que se seleccionan los resultados de la computación en función del tiempo, las entradas y reglas de transformación del proceso de este nivel. La computación de un nivel se completa con las descripciones de las transformaciones que producen nuevos valores en el espacio de salidas a partir de la secuencia de valores previos en ambos sentidos.

Los tres niveles de computación que propone Marr comienzan a tener una clara comprensión de lo que se debe calcular y como se debe hacer. No basta pues, el conocimiento de la computación a nivel de procesador, ni el nivel de algoritmos y estructura de datos, debe existir un nivel adicional de comprensión que analice y comprenda los mecanismos y estructuras particulares que se implementan en nuestros cerebros. Sustituyendo cerebro por computador se tiene cómo analizar o sintetizar una tarea computacional. Veamos los tres niveles de computación, tal como los propuso David Marr en 1982:

1. En el primer nivel tenemos los fundamentos teóricos de la computación, el planteamiento del problema en lenguaje natural y un posible esquema de solución en términos del conocimiento del dominio.
2. El segundo nivel de análisis o síntesis de un proceso es la elección de un lenguaje de representación para los espacios de entrada y salida y de un algoritmo que haga efectivas las transformaciones que enlazan esas representaciones.
3. El tercer nivel tiene que ver con todo el proceso de implementación que nos lleva del algoritmo a los procesadores físicos. Incluye la selección de un lenguaje de programación y la construcción del programa.

Para una tarea concreta, los tres niveles de descripción están relacionados. Cuando se baja de nivel siempre se pierde información, ya que no hay una representación única de ese nivel. La hipótesis fuerte de la Inteligencia Artificial es que a pesar de estas pérdidas de semántica es posible hacer computacional la inteligencia humana.

2.2 El Nivel de Conocimiento de Allen Newell.

Newell introdujo en 1981 el nivel de conocimiento como un nuevo nivel de descripción por encima del nivel simbólico. La clave del nivel es su carácter abstracto, genérico e independiente tanto del dominio como de los lenguajes usados para representar el conocimiento (marcos, reglas, lógica o redes) y para usarlo (inducción, deducción o abducción).

Pylyshyn ha asociado el nivel de conocimiento al cálculo intencional y Clancey señaló el carácter relativista de las descripciones en el nivel de conocimiento.

Una computación intencional (a nivel de conocimiento) es invariante ante cambios en la representación a nivel simbólico, de la misma forma que una computación a nivel simbólico es *invariante* ante cambios en el nivel de implementación.. En el trabajo de Newell se plantean tres objetivos:

1. Razonar acerca de la naturaleza del conocimiento. El esfuerzo de Newell se centra en distinguir el conocimiento de sus posibles representaciones, y dotar al primero de entidad propia.
2. Proponer la existencia de un nivel específico del conocimiento cuyas entidades básicas son las creencias, objetivos, planes e intenciones y el principio de racionalidad, que conecta causalmente las intenciones con las acciones.
3. La descripción de este nivel y sus conexiones con los niveles inferiores (simbólico y de implementación).

Así, Newell propone un nuevo nivel (el nivel de conocimiento) situado sobre el nivel simbólico y caracterizado por el conocimiento como medio y el principio de racionalidad como ley general de comportamiento.

Para poner este nivel en relación con los otros (simbólico y de implementación) se introducen cinco aspectos para caracterizar los niveles: sistema, medio, componentes, leyes de composición y leyes de comportamiento.

	SISTEMA	MEDIO	COMPONENTES	OPERADORES	LEYES DE COMPORTAMIENTO
NIVEL de CONOCIMIENTO	Agente Inteligente	Conocimiento	Metas Acciones Intenciones	Los usados por el lenguaje natural	Principio de racionalidad.
NIVEL SIMBOLICO	Visión del programador de ordenador	Símbolos y expresiones	Operadores y memorias (Primitivas del lenguaje)	Designación Asociación	Interpretación fija
NIVEL DE IMPLEMENTACIÓN	E. Digital + Arquitectura de Computadores	Vectores Booleanos y Estados lógicos de puertas	Registros ALU's Decodificadores de Instrucciones	Algebra de Boole	Teoría de Autómatas

El *principio de racionalidad* o principio de causalidad semántica dice:

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

Desde la perspectiva de la Inteligencia Artificial aplicada, el problema es encontrar un procedimiento para reducir el nivel de conocimiento al nivel simbólico. Es decir, pasar de las descripciones en lenguaje natural a un lenguaje accesible al nivel simbólico, donde el principio de racionalidad, las metas, las creencias y las acciones se proyectan en estructuras de datos y algoritmos. Para facilitar el proceso de reducción se están buscando entidades de nivel intermedio como la teoría de agentes cooperativos (descomposición, segmentación y especialización), las estructura de las tareas genéricas y la metodología KADS entre otras.

2.3 El agente observador y los dos dominios de descripción.

La introducción de la figura del observador proviene de la física, al reconocer la existencia del mismo en la computación se introduce la idea de distintos sistemas de referencia. Las descripciones de las observaciones de una computación a nivel físico o simbólico deben de hacerse en dos sistemas de referencia, uno para cada nivel. Al dominio que los engloba se le denomina *dominio propio* (**DP**) o autocontenido. El otro dominio es el *dominio del observador* (**DO**) que usa el lenguaje natural para describir y dotar el significado de los procesos del dominio propio. En las descripciones de dominio propio todo lo que ocurre el *causal*, y las relaciones son de *necesidad*, es decir, lo que ocurre es porque la estructura y función coinciden y las conexiones entre las magnitudes siguen sus leyes propias.

Por otro lado, el observador siempre actúa a nivel de conocimiento. Su lenguaje es el lenguaje natural y su función de diseño e interpretación de la computación (niveles físico y simbólico) está caracterizada por su modelo del conocimiento, por las limitaciones de los otros dos niveles y por la inyección de conocimiento necesaria para recuperar en la interpretación todo lo que se perdió en el proceso de reducción de niveles. En el DO siempre hay más conocimiento del que se puede deducir de la sola consideración del contenido de los otros dos niveles.

Cuando la computación es realizada por un ordenador a través de un programa, el problema está en mezclar entidades y relaciones de DP con otras del sistema operativo (con las que no tienen ninguna relación causal). El problema importante en inteligencia artificial aparece cuando se mezclan las tablas de semántica de ambos dominios y se pretende dar significados de entidades propias del nivel de conocimiento en DO a las entidades de los otros dos niveles (DP).

En el DP se pueden distinguir tres capas. La más profunda está relacionada con la estructura de tareas genéricas y con la arquitectura global del sistema e incluye el conocimiento estratégico. En la capa intermedia se encuentran las herramientas y los métodos (conjunto de formalismos de representación y mecanismos de inferencia). La tercera capa, la más externa en el DO, incluye la arquitectura global de la tarea, los aspectos organizativos del modelo, los mecanismos de adquisición del conocimiento del dominio, la segmentación en distintos agentes cooperantes y la asignación de tareas a esos agentes y, finalmente, la construcción de las tablas de semántica que se volverán a usar en la interpretación del nivel simbólico.

2.4 Estructura de Tareas Genéricas para Modelar Conocimiento en el DO.

Para facilitar el proceso de adquisición de conocimiento y su posterior reducción al nivel simbólico (inferencia y representación) se han intentado desarrollar métodos abstractos de modelar conocimiento en términos de un conjunto de tareas genéricas (TG) y métodos (M) para desarrollarlas. Al ser estas TG de validez general nos permiten modelar el conocimiento de forma análoga a como lo hace la teoría de sistemas continuos. En cada caso, los métodos, las formas de representar el conocimiento y los mecanismos de inferencia variarán, pero la organización de la tarea permanece. Su esquema conceptual es el mismo.

Así, las tareas genéricas son bloques funcionales que participan en la solución de una familia amplia de problemas. En un sistema de control las TG son: comparación, amplificación, actuación y medida y la estructura de TG es la realimentación negativa. En inteligencia artificial veremos que las tareas genéricas son la clasificación jerárquica y heurística, el diagnóstico, la monitorización, el diseño, etc, y los métodos son heurísticos.

En el análisis modelamos un segmento del razonamiento humano especificando lo que debe hacer el sistema. En síntesis, proponemos una solución estructurada diciendo cómo debe resolverse el problema usando un conjunto muy limitado de módulos genéricos (TG) cada uno de los cuales se concentra en un aspecto particular del proceso de solución.

2.4.1 La clasificación como Tarea Genérica.

Muchos de los programas de inteligencia artificial, están basados en la estructura genérica de un clasificador que asocia configuraciones de entrada a configuraciones de salida. Las entradas son datos mientras que las salidas son configuraciones que representan decisiones, posibles fallos del sistema o conceptos seleccionados. Ambos espacios deben estar especificados en extenso y de forma completa porque la clasificación ordena las categorías de salida y/o selecciona una de ellas. Los métodos usados dependen del tipo de clasificación. Así, en la clasificación jerárquica el método suele ser establecer y refinar. En la clasificación heurística, el método es la comparación a nivel abstracto y en la clasificación conexionista, el método es el ajuste de parámetros mediante la minimización de una función de coste.

Clancey introdujo la clasificación heurística como tarea genérica subyacente a muchos sistemas expertos y sugirió su descomposición en tres subtareas:

(1) La **abstracción** de los datos permite separar el valor numérico de una variable del módulo de conocimiento asociado de forma implícita. Por ejemplo que un valor de temperatura es alto.
(2) La **equiparación heurística** permite de nuevo buscar asociaciones entre tipos generales de síntomas y tipos de patologías. Así, la alta temperatura indica fiebre y está asociada a infección.
(3) El **refinamiento** está asociado al resto del conocimiento necesario para especificar el diagnóstico.

La clasificación conexionista usa en general el conocimiento del problema para definir en una estructura con cuatro tareas genéricas, distribuidas por capas:

(1) *Extracción de propiedades*: comienza con la construcción de un espacio de características (analíticas, algorítmicas o puramente simbólicas) con una parte fija y otra autoprogramable por aprendizaje.

(2) *Métricas*: incorpora las métricas asociadas a las clases de equivalencia (calcular la proximidad de cada punto del espacio de características a cada una de las clases).
(3) *Selección de máximos*: es la selección de la clase a la que la distancia es mínima o el cálculo de la función de pertenencia de ese punto del espacio de medida a las distintas clases en las formulaciones borrosas.
(4) *Refuerzo*: es la última TG del clasificador y es el aprendizaje.

2.4.2 La Metodología KADS.

La metodología KADS para el desarrollo de sistemas basados en el conocimiento se apoya en el nivel de las tareas genéricas y en la distinción de cuatro capas: ***estrategia***, ***tareas***, ***inferencia*** y ***conocimiento estático del dominio***, para modelar el conocimiento. El proceso comienza con el análisis a nivel global, que permite una descomposición del problema en tres módulos: el módulo basado en el conocimiento, la interfaz y el resto de la aplicación. La siguiente etapa modela la inferencia, está relacionada con las técnicas de representación del conocimiento estático.

La tercera etapa contiene la estructura de tarea genérica. La capa más externa se usa para modelar los aspectos estratégicos del conocimiento.

El uso de una estructura de tarea genérica para modelar conocimiento tiene la ventaja de ser modular, planteando la existencia de primitivas de computación, a partir de las cuales se puede modelar y sintetizar el conocimiento.

2.5 IA simbólico VERSUS IA conexionista

Hay que poner de manifiesto su complementariedad y su significado en la teoría de los dos dominios (DP y DO). La inteligencia artificial que nació conexionista, fue después dominantemente simbólica y tras el renacimiento de la computación neuronal hemos llegado a una etapa de cooperación alternativa entre ambos, dependiendo de la naturaleza del problema, como dos técnicas complementarias y en algunos casos simbióticas.

El simbolismo nace en el dominio del observador durante el proceso de reducción del nivel de conocimiento al nivel simbólico y en el proceso inverso de interpretación. Cuando la reducción se hace a través del nivel simbólico, terminando en las primitivas de un lenguaje, la inteligencia artificial es simbólica. En cambio, cuando la reducción se hace directamente del nivel de conocimiento a la red de procesadores la inteligencia artificial es conexionista.

Una primera distinción entre IA simbólica e IA conexionista está en el salto directo que las redes neuronales dan desde la formulación del problema a nivel de conocimiento hasta el nivel físico, lo que supone un análisis pormenorizado de la aplicación. Como alternativa a este paso directo, la IA simbólica utiliza un conjunto de entornos y lenguajes de alto nivel que facilitan la reducción al nivel simbólico de un modelo de conocimiento.

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

NIVEL 3: Implementación en un soporte físico del nivel 2 (entornos software y nivel hardware)

NIVEL 2: Representación y algoritmo. ¿Cómo puede implementarse esta teoría de cálculo? (Espacios simbólicos de representación de las entradas y salidas y algoritmos que los enlazan).

NIVEL 1: Teoría computacional de la tarea a nivel genérico incluyendo cuál es el objetivo de la computación, por qué es apropiado y cuál es la lógica de la estrategia adecuada para implementarlo.

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

3 FUNDAMENTOS Y TÉCNICAS BÁSICAS DE BÚSQUEDA

3.1 Planteamiento del problema

En la formulación de cualquier problema, se parte de la declaración en lenguaje natural del mismo, que responde a su concepción en el *nivel de conocimiento*. Se afirma que el sistema actúa siguiendo el *principio de racionalidad* (si un sistema tiene el conocimiento de que una de sus acciones conduce a una de sus metas, entonces realiza dicha acción).

La inteligencia artificial estudia espacialmente métodos que permiten resolver problemas en los que no existe el conocimiento sistemático para plantear una solución analítica (métodos considerados *débiles* por realizar procedimientos de búsqueda *no informada*).

Ante cualquier problema se presentan dos situaciones posibles: tener el conocimiento de lo que hay que hacer o indagar qué podría hacerse para resolverlo; los problemas complejos requieren una formulación combinada. Las técnicas de representación que permiten describir los aspectos conocidos del problema se denominan **declarativos** (declaran *el qué*). Sus ventajas son su claridad, su predisposición al planteamiento modular y que se pueden añadir nuevos hechos a los ya existentes.

Las técnicas de representación que se centran en especificar el proceso que hay que realizar para encontrar las soluciones buscadas se denominan **procedimentales** (se centran en *el cómo*), fijan el conocimiento del dominio que no se ajusta a esquemas declarativos y ese mismo conocimiento sirve para guiar las líneas de conocimiento del proceso de búsqueda.

Ningún sistema es completamente declarativo o exclusivamente procedimental. En principio son intercambiables (cuando exista el proceso de interpretación adecuado para las declarativas). En general se puede afirmar que el carácter procedimental conlleva un tratamiento algoritmo y el declarativo uno heurístico. El planteamiento general de un problema consiste en encontrar o construir una solución (o varias) de dicho problema. Este problema requiere:

- ☞ Un agente (sistema) con una serie de objetivos que se quieren alcanzar
- ☞ Un conjunto de acciones que permiten obtener los objetivos o metas
- ☞ Un procedimiento de elección entre diferentes formas de llegar a las metas (cada solución constituye una *secuencia de acciones*). El módulo que construye la solución suele llamarse *solucionador*.

Así, lo preferible es seguir ***el esquema de los problemas de búsqueda***:

DADOS:

- *Conjunto de estados*: todas las configuraciones de las situaciones posibles en el dominio.
- *Uno o más estados* (iniciales) desde los cuales el solucionador comienza a actuar.
- *Uno o más estados* (finales) que serán aceptados como soluciones.
- *Un conjunto de operadores* o reglas que describen los acciones posibles.

ENCONTRAR:

- Una secuencia de operadores que permitan pasar del estado inicial al final (o a alguno si existen varios).

La mejor forma de resolver este esquema es utilizar un grafo para representar el proceso de búsqueda realizado.

3.2 Espacios de representación

3.2.1 Nociones sobre grafos

Es un conjunto de *nodos* unidos mediante *arcos*. Un **camino** es una sucesión de nodos conexos. Si el último nodo del camino coincide con el primero es un **camino cerrado**; si no es un *camino abierto*. Cuando entre dos nodos cualesquiera existe un camino estamos ante un **grafo conexo**. Si este camino es único el grafo es **simplemente conexo**; si son dos do más (caminos cerrados) el grafo es **múltiplemente conexo**.

En un grafo no dirigido si podemos recorrer el camino siguiendo la dirección de los arcos y volvemos al punto de partida tenemos un **ciclo**, si no, tenemos un **bucle**.

Un *grafo dirigido acíclico o* **GDA** es aquel que no tiene ciclos. Los GDA conexos que no contienen bucles se llaman *poliárboles.* Un árbol (dirigido) es un GDA conexo en el cual cada nodo tiene como máximo un padre, así:

- Un árbol no puede contener ciclos ni bucles
- En todo árbol existe un único nodo (raíz) que no tiene antepasados y es antepasados de todos los demás
- Cada nodo, salvo la raíz, tiene un solo padre
- Para cada nodo sólo existe un camino que lo conecta a la raíz

Para el caso concreto de los problemas de búsqueda tenemos las siguientes nociones:

- **Nodo**: elemento en el espacio de estados que representan situaciones válidas en el dominio. Un nodo terminal que satisface las condiciones del objetivo recibe el nombre de *meta*.
- **Expansión de un nodo**: obtener todos sus posibles sucesores
- **Nodo cerrado**: aquel que ya ha sido expandido
- **Nodo abierto**: aquel en el que queda por aplicar algún operador
- **Coste de un arco**: valor numérico que refleja el tiempo requerido para aplicar un operador a un estado en el proceso de búsqueda (por defecto puede tomar el valor 1)
- **Coste de un nodo**: Tiempo en alcanzar el nodo desde la raíz a lo largo del mejor camino encontrado hasta el momento.
- **Factor de ramificación**: número medio de descendientes de un nodo (número medio de operadores que pueden aplicarse a un estado)
- **Longitud de la trayectoria**: número de nodos generados en un camino (número de operadores aplicados en un camino)
- **Profundidad**: longitud del camino más corto desde el estado inicial a una meta. La profundidad del nodo raíz es 0 y la de cualquiera otro nodo es la de su antecesor (*menos profundo*) +1.

3.2.2 Representación de los problemas de búsqueda

Hay dos esquemas generales de funcionamiento:
- el *esquema de producción o método de búsqueda en el espacio de estados*
- el *esquema de resolución* (basado en la *descomposición del problema*)

Definiciones:

- **Espacio del problema**: entorno en le cual se desarrolla el proceso de búsqueda. Está formado por un conjunto de estados y un conjunto de operadores.
- **Instancia del problema**: espacio del problema donde se señala cual es el estado inicial y cual el final.
- **Espacio de reducción**: espacio en el que realiza el proceso de búsqueda el esquema de reducción.
- **Espacio de estados**: espacio en el que realiza el proceso de búsqueda el esquema de producción. En el cada estado representa una situación que se debe considerar como candidata a pertenecer a la solución del problema.
- **Equiparación**: proceso de comprobación de la aplicabilidad de los operadores.

Una vez aplicado el primer operador se obtiene una nueva descripción derivada de la situación inicial a la que también se le considera estado. Si todavía no se ha obtenido la solución caben dos opciones:

- seguir intentando aplicar operadores al estado inicial
- centrarse en el nuevo estado y aplicar alguno de los operadores disponibles sobre él.

La **estrategia de control** o **EC** es la que optimiza el proceso de búsqueda. Partiendo de las condiciones iniciales se realiza un proceso continuo de selección de estados y de generación de sucesores. Bajo este esquema los estados también se denominan *pasos* en la búsqueda. Ahora bien, *la búsqueda surge cuando hay más de un operador aplicable o existe más de una instanciación para un operador concreto*. Entonces, una decisión local (aplicación de un operador a un determinado estado) puede no ser

acertada, pero el proceso en su conjunto debe ser un método sistemático (ajustado a un procedimiento) que explora las distintas alternativas que permitan acercarse a la meta. Para ello debe:

- Seleccionar los nodos
- Determinar las reglas aplicables al nodo elegido (pueden existir diferentes formas de aplicación de alguna de ellas)
- Seleccionar una determinada regla y aplicarla
- Comprobar que el estado generado cumple las condiciones asociadas al estado meta

El conjunto de estados obtenidos en el proceso de búsqueda se denomina **grafo de estados** (en el caso más sencillo es un árbol). El **espacio de estados** está formado por todos aquellos estados que podrían obtenerse si se aplicarán todos los operadores posibles a todos los estados que se fueran generando. Por lo tanto, el grafo de estados es una medida *explícita* (refleja los caminos explorados) del proceso realizado y el espacio de estados es una medida *implícita* que da una idea de la complejidad del problema. El mejor método será aquél que necesite hacer explícita la menor parte del grafo implícito (o sea la menos costosa) para obtener una solución del problema.

La complejidad de un cálculo es la cantidad de recursos necesarios para efectuarlo. Para evitar ambigüedades consideraremos la **complejidad temporal** como una medida del número de operaciones realizadas y la **complejidad espacial** como el conjunto de datos que es necesario recordar en un momento dado.

Definiciones:

Encadenamiento hacia delante o guiado por los datos: La dirección del proceso de búsqueda parte de los datos suministrados y avanza en el camino de búsqueda hacia un estado meta. El estado inicial y todos los que vayan obteniéndose sucesivamente a partir de este formarán parte la base de datos de trabajo (**BD**).

Encadenamiento hacia atrás o guiado por las metas: La dirección del proceso de búsqueda parte de un estado solución y avanza en el camino de búsqueda hacia el estado inicial.

En estos planteamientos, se supone que en cada paso del proceso de búsqueda, se considera la aplicación de un único operador a un único estado.

3.2.3 Cómo limitar el espacio de búsqueda

El problema de hallar un estado que satisfaga una condición puede formularse como la búsqueda de un grafo que posea un nodo cuya descripción asociada coincida con el estado objetivo. La búsqueda consiste en obtener un grafo explícito suficientemente grande para que contenga la solución del problema inicial. Pero puede aparecer la *explosión combinatoria* (cuando el número de nodos crece exponencialmente convirtiéndose en un problema intratable mediante técnicas de búsqueda exhaustivas). Así, la búsqueda heurística no consiste en excluir parte de ese espacio *por definición*, sino en añadir información, basándose en el espacio estudiado hasta ese momento, de forma que se restrinja drásticamente dicha búsqueda.

3.3 Búsqueda en integración simbólica

Vamos a seguir un ejemplo a partir del trabajo [Boticario, 1988] en el que partiendo de integrales indefinidas especificadas mediante un lenguaje concreto de expresiones matemáticas. El proceso queda definido en el siguiente cuadro:

Dados:

- **Estado inicial**: según el ejemplo del libro $\int$ **x sin(x) dx**
- **Estado final o meta**: un estado que no contenga el símbolo de integración $\int$
- **Espacio de estados**: todas las expresiones que resulten de aplicar distintas secuencias de operaciones junto con todos los operandos disponibles.
- **Operadores**: ejemplos
  - Métodos generales de integración
  - Integrales inmediatas
  - Relaciones trigonométricas
  - Transformaciones entre expresiones algebraicas equivalentes

Encontrar:

Una secuencia de transformaciones (aplicación de operadores) que permita resolver la integral planteada.

El estado meta es aquél que se quiere alcanzar sin considerar *como alcanzarlo*.

3.3.1 Descripción general del sistema.

Para resolver el problema de búsqueda es necesario tener en cuenta los siguientes elementos:

☞ Representar adecuadamente los elementos que participan en el proceso de búsqueda.

- ☞ Utilizar heurísticas para que la estrategia de búsqueda sea más eficiente.
- ☞ Restringir los recursos (tiempo y espacio) asignados a la solución.
- ☞ Eliminar:
  - Los nodos ya examinados.
  - Los nodos que no pueden aportar nueva información.
  - Los subárboles que no pueden contener la solución.

En cuanto a los operadores, cada uno consta de:
- Un *dominio de estados* a los cuales se les puede aplicar dicho operador.
- Una *regla de reescritura*.
- Un rango de estados que pueden producirse por aplicación del operador.

Por consiguiente los operadores describen *subdominios* conocidos del problema a los que se les puede aplicar una transformación que permite avanzar en el proceso de búsqueda de la solución.

3.3.2 Lenguajes de descripciones.

No se trata de comprobar si un estado pertenece al conjunto de estados a los que se puede aplicar un operador, sino que realmente tratamos de verificar si la expresión de un estado tiene una forma semejante a la del operador que se le puede aplicar. Para ello son necesarios los siguientes elementos:

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

En el sistema LEX se utiliza un lenguaje generado mediante una gramática que sirve para describir los estados, los operadores y las reglas utilizadas por la estrategia de control para guiar la búsqueda (las sentencias terminales y no terminales sirven como descripciones).

3.3.3 Lenguajes de operadores.

Los diferentes operadores son los que permiten ir modificando la situación inicial del proceso (problema planteado) hasta alcanzar el objetivo. El espacio de estados representaría todas las transformaciones posibles implícitas en la descripción de los operadores del sistema. Los operadores poseen una estructura propia que los caracteriza:

Parte izquierda (PI o LHS):	Descripción del dominio del operador.
Parte derecha (PD o RHS):	Representa el rango del operador.
Parte avanzar (PA):	Para poder sustituir por PD es necesario asignar algunos valores a aquellos símbolos de PI que no estén en PD.
Parte comentario (PC):	Describe en lenguaje matemático la acción asociada al operador.

La forma en que un operador se aplica a un estado consiste en:
Extraer la descripción del dominio del operador (PI).
Comprobar la semejanza (equiparar) de este con alguna porción del estado.
Realizar la transformación asociada al operador (indicada por PD y apoyada por PA), reescribiendo la parte del estado afectada (el resto de la descripción del estado permanecerá invariable).

3.3.4 Equiparación de descripciones.

El lenguaje generado permite definir una relación de orden parcial entre las expresiones válidas en el lenguaje. Cuando la precondición asociada a un operador es más general que la descripción de un estado, entonces se le puede aplicar el operador al estado (la relación más general establece un ordenamiento parcial en el espacio de descripciones de aplicabilidad de cada operador).

3.3.5 Solucionador.

Es la parte del sistema que realiza la búsqueda de la solución del problema planteado. La estrategia seguida para alcanzar el objetivo es el encadenamiento hacia adelante. El problema se representa mediante un grafo de búsqueda que contiene todos los nodos que ya se han alcanzado partiendo de la raíz (el problema que se quiere solucionar) por sucesivas aplicaciones de operadores y heurísticas disponibles.
El procedimiento **SOLUCIONADOR** recibe como argumentos un problema (expresado según el lenguaje) junto con unos **recursos asignados** para su solución (tiempo y espacio que se le permite emplear). Se ha introducido esa limitación (de recursos) para evitar que se llegue a intentar establecer soluciones con

caminos excesivamente largos. Entonces los enlaces que constituyen los caminos que conducen a la solución del problema llevan asociados implícitamente un coste de cálculo. Se denomina **coste real** de un nodo a la suma de los tiempos de cálculos consumidos en cada paso a lo largo del camino que conduce de la raíz al nodo considerado.

La estrategia de control (EC) debe tener como principal objetivo la eficiencia, además de la eficacia (*EFICACIA* ES LA CAPACIDAD DE REALIZAR UNA TAREA, MIENTRAS QUE *EFICIENCIA* CONSISTE EN REALIZAR DICHA TAREA CON EL MENOR COSTE POSIBLE). El coste es un factor necesario para conducir la búsqueda. También existen otros parámetros (independientes del dominio) que determinan la eficiencia deseada y son la profundidad y el factor de ramificación. Dos claves de EC para ser eficiente son:

- ☞ Que ayude a avanzar en la obtención de la meta.
- ☞ Que siga un procedimiento organizado de recorrer el espacio de búsqueda.

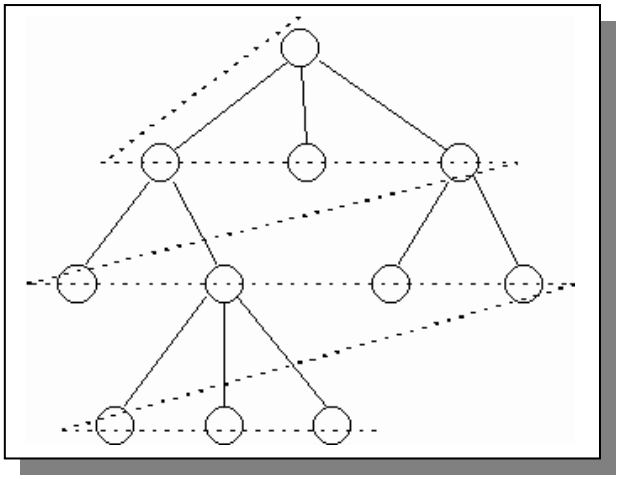
3.4 Búsqueda sin información del dominio.

Estudiaremos las formas básicas de control que permiten obtener, a través de un proceso exhaustivo (búsqueda sistemática y objetiva, también llamada **búsqueda ciega**), un estado.

Partimos de la descripción del problema planteado que no cumple la condición de ser meta. Se intenta encontrar el camino óptimo entre la descripción del problema y la meta.

Los principios que deben cumplir las estrategias sistemáticas de control son: No dejar a priori ningún nodo sin explorar, y no explorar más de una vez el mismo nodo.

3.4.1 Búsqueda en amplitud.



Descripción:

Es aquel procedimiento de control en el que se revisan todas las trayectorias de una determinada longitud antes de crear una trayectoria más larga.

La búsqueda puede estar orientada a recorrer el árbol o grafo correspondiente por niveles. Para conseguirlo se utiliza una estructura tipo **cola** (FIFO) en la que se van introduciendo los nodos que son generados. Este tipo de exploración recibe el nombre de *búsqueda en amplitud* y garantiza la obtención de la solución de *menor coste* (óptima), si es que ésta existe

Procedimiento:

1. Crear una lista de nodos llamada **ABIERTA** e *inicializarla* con un único nodo raíz, al que se le asigna el estado inicial del problema planteado.
2. Hasta que **ABIERTA** esté vacía o se encuentre una meta realizar las siguientes acciones:
   2.1 Extraer el primer nodo de **ABIERTA**, llamar a este nodo *m*.
   2.2 Expandir *m* (generar todos sus sucesores)
   - Para cada operador aplicable y cada forma de aplicación:
     (1) aplicar el operador *m*, obtener un nuevo estado y crear un puntero que permita saber que su antecesor inmediato es *m*.
     (2) si el nuevo estado generado es *meta* salir del proceso iterativo inicializado en 2.2 y devolver dicho estado.
     (3) incluir el nuevo estado al final de **ABIERTA**. (una vez completado este proceso para todos los sucesores de *m* -cuando no se haya encontrado antes una meta- se continua el proceso iterativo en el paso 2).

Observaciones:

Si el algoritmo termina con una *meta*, el camino de la solución puede obtenerse recorriendo los punteros creados desde el nodo *meta* al nodo raíz. En caso contrario el proceso terminará sin haber encontrado la solución.

Complejidad:

Temporal:

Depende en gran medida del factor de ramificación y de la profundidad de la solución. Si el número medio de sucesores es *n* y la solución se alcanza en el nivel *p* el tiempo empleado es $1 + n + n^2 + \ldots + n^p$ que es del orden de $\mathbf{O(n^p)}$.

Espacial:

Dado que antes de abandonar la generación de todos los sucesores de un nivel se deben almacenar en **ABIERTA** todos los nodos de dicho nivel, es también del orden de $\mathbf{O(n^p)}$.

Análisis:

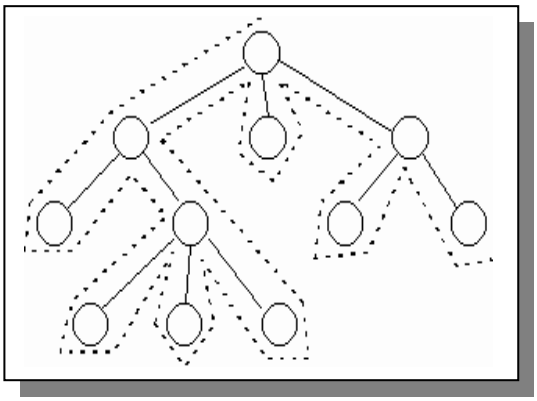
Ventajas:

Si el problema tiene una solución garantiza el encontrarla y si existen varias se obtiene la de menor coste.

Desventajas:

Si el nivel de profundidad asociado a la solución es significativamente menor que el factor de ramificación se expandirán demasiados nodos inútilmente. La principal desventaja es el espacio de almacenamiento requerido, lo que le hace inviable en muchos problemas del mundo real.

3.4.2 Búsqueda en profundidad.



Descripción:

En cada nivel se selecciona un único nodo para ser expandido (al contrario que en amplitud). En el caso de que el camino pueda tener una longitud infinita o demasiado larga se establece un *límite de profundidad (**lp**)*. También, en otros casos, se puede establecer una prueba de viabilidad para evitar caer en una *vía muerta*.

Procedimiento:

1. Crear una lista de nodos llamada **ABIERTA** e *inicializarla* con un único nodo raíz, al que se le asigna el estado inicial del problema planteado.
2. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:
   - (1) Si **ABIERTA** está vacía, terminar con *fallo;* en caso contrario continuar.
   - (2) Extraer el primer nodo de **ABIERTA** y denominarlo *m*.
   - (3) Si la profundidad de *m* es igual a *lp*, regresar a 2; en caso contrario, continuar.
   - (4) Expandir *m* creando punteros hacia *m* en todos sus sucesores, de forma que pueda saberse cuál es su antecesor. Introducir dichos sucesores al principio de **ABIERTA** siguiendo un *orden arbitrario*. (La falta de orden refleja el carácter no informado de este procedimiento).
     - (4.1) Si algún sucesor de *m* es *meta*, abandonar el proceso iterativo señalado en 2 devolviendo el camino de la solución, que se obtiene recorriendo los punteros de sus antepasados.
     - (4.2) Si algún sucesor de *m* se encuentra en un *callejón sin salida* eliminarlo de **ABIERTA** (se continúa el proceso iterativo en el paso 2)

Complejidad:

Temporal:

Tiene la misma complejidad que en amplitud ya que se generan los mismos nodos aunque en distinto orden.

Espacial:

A lo largo del grafo de búsqueda sólo es necesario guardar constancia del camino construido hasta el momento, luego la complejidad para un camino de longitud *p* será dicho valor más el factor de ramificación *r*, ya que cada vez es necesario guardar constancia de todos los sucesores del nodo que se está expandiendo, es decir O(p + r).

Análisis:

Ventajas:

El reducido valor de su complejidad espacial (con múltiples soluciones posibles, la eficiencia aumenta).

Desventajas:

El tiempo requerido (se puede recorrer un camino demasiado largo que no llegue a ninguna solución) y determinar *lp* (si hacemos *lp* inferior a la longitud real de la solución, ésta nunca se encontraría). Por esta razón, *lp* se debería llamar **límite de exploración**.

3.4.3 Búsqueda con retroceso.

Descripción:

En cada iteración sólo considera un sucesor, igualmente se eliminan los nodos que sean una vía muerta. Cuanto mejor sea el criterio utilizado para limitar el número de estados considerados más eficiente será el proceso de búsqueda.

Procedimiento:

1. Crear una lista de nodos llamada **ABIERTA** e *inicializarla* con un único nodo raíz, al que se le asigna el estado inicial del problema planteado.
2. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:
   (1) Si **ABIERTA** está vacía, terminar con *fallo;* en caso contrario continuar.
   (2) Seleccionar el primer nodo de **ABIERTA** y denominarlo *m*.
   (3) Si la profundidad de *m* es igual a *lp* o si *m* ya no tiene más sucesores posibles, eliminarlo de **ABIERTA** y regresar a 2; en caso contrario, continuar.
   (4) Generar un nuevo sucesor *m'* de *m* e introducirlo al principio de **ABIERTA**, creando un puntero a *m*, y señalar que dicha rama ya ha sido considerada.
      (4.1) Si *m'* es *meta*, abandonar el proceso iterativo iniciado en 2 devolviendo el camino de la solución, que se obtiene recorriendo los punteros de sus antepasados.
      (4.2) Si *m'* se encuentra en un *callejón sin salida* eliminarlo de **ABIERTA** y se continúa el proceso iterativo en el paso 2.

Complejidad:

Temporal:
El número de operaciones para un factor de ramificación *n* y para una solución situada en el nivel *p* es del orden de $\mathbf{O(n^p)}$.
Espacial:
Basta con recordar el camino recorrido hasta un momento dado, entonces del orden de **O(p).**

Análisis:

Ventajas:
Necesita menos espacio de almacenamiento y no se generan las ramas del árbol de búsqueda que se encuentren después del camino de la solución.
Desventajas:
Saber cuál es *lp* y no establecer ningún orden previo de recorrido de los sucesores de un nodo (la eficiencia temporal del método se queda limitada por su carácter fortuito).

3.4.4 Otros métodos derivados

Búsqueda en profundidad progresiva:

Es una combinación de los métodos anteriores y consiste en realizar una búsqueda en profundidad por niveles, de forma que el límite de exploración aumenta una unidad por cada nivel analizado. Primero se hace una búsqueda en profundidad con *lp*=1, si no se encuentra la solución, se hace con *lp*=2... y así sucesivamente. Es la estrategia más eficiente de las *no informadas* que son capaces de encontrar la solución de menor coste existente (óptima). Su complejidad temporal es del orden de $\mathbf{O(n^p)}$, y la espacial es sólo del orden de **O(p).**

Búsqueda bidireccional:

Descripción:

Además de la descripción del problema, se parte de una meta explícita por lo que se realiza simultáneamente encadenamiento hacia delante y hacia atrás hasta que ambos procesos confluyan en algún estado. Para este tipo de búsqueda se necesita que los operadores sean invertibles y se debe establecer expresamente cuál es la meta que se quiere alcanzar.

Se parte de un estado inicial desde el que se busca la meta y de un estado meta desde donde se busca el estado inicial; con la única condición de que una de las dos búsquedas anteriores sea en **amplitud**, se garantiza que si existe algún camino solución, en algún momento las dos búsquedas pasarán por un estado común desde el que se podrá reconstruir dicho camino.

Procedimiento:

1. Inicializar dos grafos de búsqueda. En el primero, que llamamos (A.1), el nodo raíz será el estado inicial del problema planteado y en el segundo (A.2) tendrá como raíz la meta de dicho problema.
2. Inicializar el límite de exploración *lp = l*.
3. Continuar con la realización de la búsqueda (A.1) hasta el nivel *lp*.
4. Continuar con la realización de la búsqueda (A.2) hasta el nivel *lp*.
5. Comprobar si alguno de los nuevos estados generados en 3 y en 4 coinciden. En tal caso, devolver la trayectoria de la solución formada por la concatenación de los caminos obtenidos en (A.1) y en (A.2) hasta el nodo encontrado. En caso contrario, volver al paso 2.

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

En la práctica, la estrategia más eficiente consiste en cambiar el sentido de exploración en cada nuevo nivel, en vez de cambiarlo después de un cierto número de iteraciones.

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

Complejidad:

Temporal:

Para un factor de ramificación *n* y la solución situada en un cierto nivel *p* tenemos $\mathbf{O(n^{p/2})}$.

Espacial:

Por la condición impuesta para garantizar la convergencia es del orden de $\mathbf{O(n^{p/2})}$.

Análisis:

Ventajas:

Se exploran menos nodos que en un proceso de un solo sentido y es un método con gran eficiencia.

Desventajas:

Dar el valor a *lp* y no disponer de algún criterio adicional para poder ordenar la selección de nodos meta a lo largo del proceso.

El número de iteraciones (*l*) antes de cambiar el sentido de la búsqueda es un parámetro crítico para la eficiencia de este método.

A continuación figuran las complejidades temporal y espacial de todos los algoritmos vistos:

COMPLEJIDADES		
	TEMPORAL	**ESPACIAL**
Búsqueda en amplitud	$O(n^p)$	$O(n^p)$
Búsqueda en profundidad	$O(n^p)$	$O(n \cdot p)$
Búsqueda con retroceso	$O(n^p)$	$O(p)$
Búsqueda en profundidad progresiva	$O(n^p)$	$O(p)$
Búsqueda bidireccional en amplitud	$O(n^{p/2})$	$O(n^{p/2})$

Tabla 1-1 Complejidades de los algoritmos.

n: "factor de ramificación" (número medio de sucesores de todos los nodos)

p: "profundidad de la solución"

3.5 Reducción del problema.

Dentro del *razonamiento dirigido por las metas*, se distinguen dos situaciones: Que cada operador produce un único estado (como hemos visto hasta ahora), o que al aplicar un operador, se generan una serie de subproblemas (cada uno de los cuales es más sencillo que el problema original); estos problemas se representan mediante el esquema conocido como ***reducción del problema***.

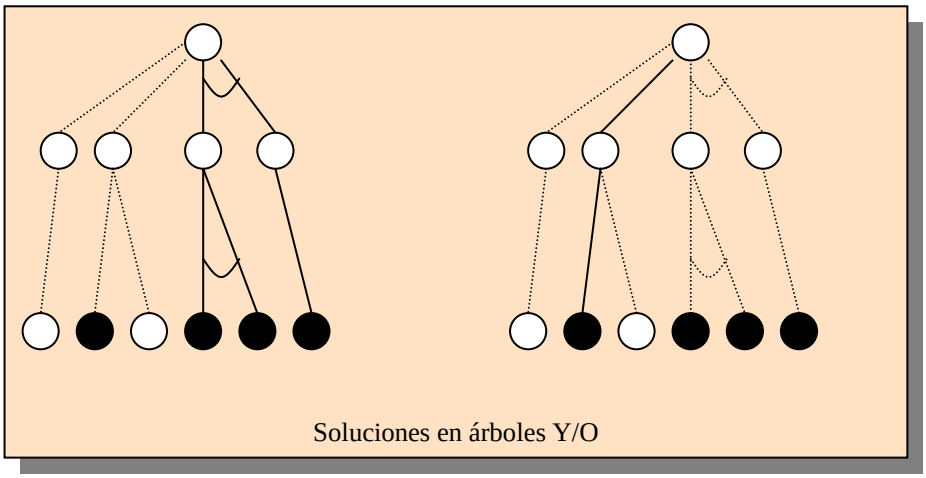
Los problemas pueden descomponerse en dos tipos de operadores aplicables:
- Los que realizan la descomposición
- El resto (por ejemplo, reconocer los estados meta).

La descomposición de un problema permite tratar los subproblemas en los cuales se divide de una forma independiente. Cada movimiento genera elementos en la base de datos que pueden tratarse a parte, como si cada uno de ellos fuera una nueva base de datos. La forma de representación más adecuada son los llamados **grafos Y/O** ó **árboles Y/O** (según el planteamiento del problema).

El árbol Y/O tiene como raíz el problema completo que se intenta resolver. Los nodos serán los subproblemas o las submetas. Los nodos se conectan entre sí por medio de dos tipos de enlaces:

- **Los enlaces O**: representan problemas que pueden resolverse de varias maneras. Establecen una relación entre tan sólo dos nodos.
- **Los enlaces Y**: reflejan situaciones que pueden descomponerse en una serie de etapas o subproblemas. Se representan por medio de una línea arqueada que conecta todos los nodos implicados (puede conectar a más de dos nodos). El problema inicial quede resuelto cuando se ha obtenido la solución de todas sus partes.

Los árboles vistos en los apartados anteriores, eran todos del tipo O, ya que se trataba de encontrar un único camino hasta la meta.



Soluciones en árboles Y/O

La solución encontrada en los grafos Y/O no es un simple camino sino otro grafo llamado ***grafo solución***. Dado que en ese tipo de grafos hay nodos que requieren la terminación de todos sus hijos, la solución puede estar formada por varias hojas. En la gráfica las ramas discontinuas representan aquellas que no pertenecen al grafo solución y los nodos terminales aparecen rellenos.

Como ejemplos podemos citar:

-Los juegos donde los adversarios van alternando sus jugadas; los movimientos de un jugador se pueden representar mediante enlaces O (las diferentes alternativas posibles) y los del adversario en enlaces Y (hay que considerar todos los posibles contra ataques del adversario).

-Los problemas en los que la solución es un conjunto no ordenado de acciones.

-Los problemas de razonamiento lógico y los de planificación por etapas (los que se pueda establecer una secuencia de pasos determinados por una serie de condiciones, no por el orden en que éstas se vayan cumpliendo).

3.6 Algoritmo general de búsqueda en grafos.

En cualquier proceso de búsqueda es importante tener la posibilidad de determinar si un nuevo estado ya había sido generado y expandido previamente, pues con ello se evitaría repetir la exploración de determinados caminos. El *algoritmo general de búsqueda en grafos* se basa en la idea anterior. Este método, aunque se puede aplicar al caso particular de los árboles, está diseñado para el caso más general de grafos, donde cada nodo puede tener varios antecesores.

Lo más útil es mantener, a lo largo del proceso, dos listas, llamadas **ABIERTA** y **CERRADA**, respectivamente. **ABIERTA** permite reanudar en cualquier momento un camino abandonado previamente por existir un camino alternativo que entonces se consideró más favorable. **CERRADA** sirve como registro de los nodos elegidos a lo largo de todo el grafo de búsqueda.

Procedimiento:

1. Crear un grafo de exploración G que consista en un único nodo que contiene la descripción del problema. Crear una lista de nodos llamada **ABIERTA** e inicializarla con dicho nodo.
2. Crear una lista de nodos llamada **CERRADA** que inicialmente estará vacía.
3. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:
   (1) Si **ABIERTA** está vacía terminar con *fallo* en caso contrario, continuar.
   (2) Eliminar el primer nodo de **ABIERTA**, llamar a ese nodo *m* e introducirlo en la lista **CERRADA**.
   (3) Expandir *m*:
      (3.1) Generar el conjunto M de todos sus sucesores que no sean antecesores e introducirlos como sucesores de *m* en G.
      (3.2) Si algún miembro de M es *meta*, abandonar el proceso iterativo señalado en 3 devolviendo el camino de la solución, que se obtiene recorriendo los punteros de sus antepasados.
      (3.3) Poner un puntero a *m* desde los nuevos nodos generados (aquellos que no pertenecen a G). Incluir dichos elementos en **ABIERTA**.
      (3.4) Para cada nodo de M que ya estuviese en **ABIERTA** o en **CERRADA**, decidir si se redirige o no su puntero a *m*.
      (3.5) Para cada nodo de M que ya estuviese en **CERRADA**, decidir para cada uno de sus descendientes, si se redirigen o no sus punteros.
   (4) Reordenar la lista abierta aplicando algún criterio previamente establecido.

Observaciones:

1. Se crean dos grafos de exploración, uno explícito que coincide con el grafo de búsqueda y otro, implícito, que se obtiene recorriendo los punteros que conducen desde la meta al estado inicial. Este último mantiene la estructura de árbol, ya que cada nodo tiene un puntero a un único antecesor inmediato. Este árbol va cambiando progresivamente, ya que en el proceso de generación del grafo parcial mencionado puede aparecer un nuevo camino más corto desde la raíz a un determinado nodo. Esto produce una reorientación del puntero del árbol que parte de dicho nodo
2. El paso 3.1 garantiza que ningún nodo es antecesor de sí mismo (lo que supone un coste elevado).
3. Los nodos de **ABIERTA** son los nodos frontera no expandidos y los de **CERRADA** son todos los expandidos, independientemente de que hayan tenido sucesores o no.
4. Los punteros se reasignan en función del criterio aplicado en 3.4 y 3.5 (debe garantizar que señalen el camino mejor encontrado hasta el momento).
5. Si un nodo de **CERRADA** cambia el puntero de su antecesor 3.5 el camino *menos costoso* puede ser parte del camino *menos costoso* de alguno de sus sucesores.
6. La realización eficiente de este procedimiento requiere establecer en el paso 4 el *criterio de ordenación* de los nodos de **ABIERTA**.

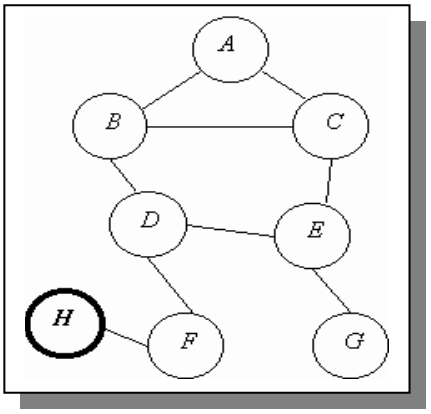
Hay que considerar tres situaciones diferentes a la hora de realizar la expansión de un nodo:

a) Que el nodo generado no esté ni en ABIERTA ni en CERRADA. En este caso basta con enlazar el nodo generado con su padre.

b) Que el nodo generado esté en ABIERTA. Ahora habrá un nuevo camino desde el nodo raíz hasta el nodo generado. Por tanto, debe determinarse si este nuevo camino es menos costoso que el anterior. En caso afirmativo hay que redirigir el puntero del árbol que parte del nodo generado.

c) Que el nodo generado ya estuviera en CERRADA. Si el nuevo camino es menos costoso, además de llevar a cabo los pasos del caso anterior, hay que hacer un estudio de los descendientes del nodo generado por si hubiera que redirigir sus enlaces actuales.

PROBLEMAS



• PROBLEMA 3.1:

Dado el siguiente grafo, donde A es el nodo inicial y H el nodo meta, explorarlo mediante los siguientes métodos:

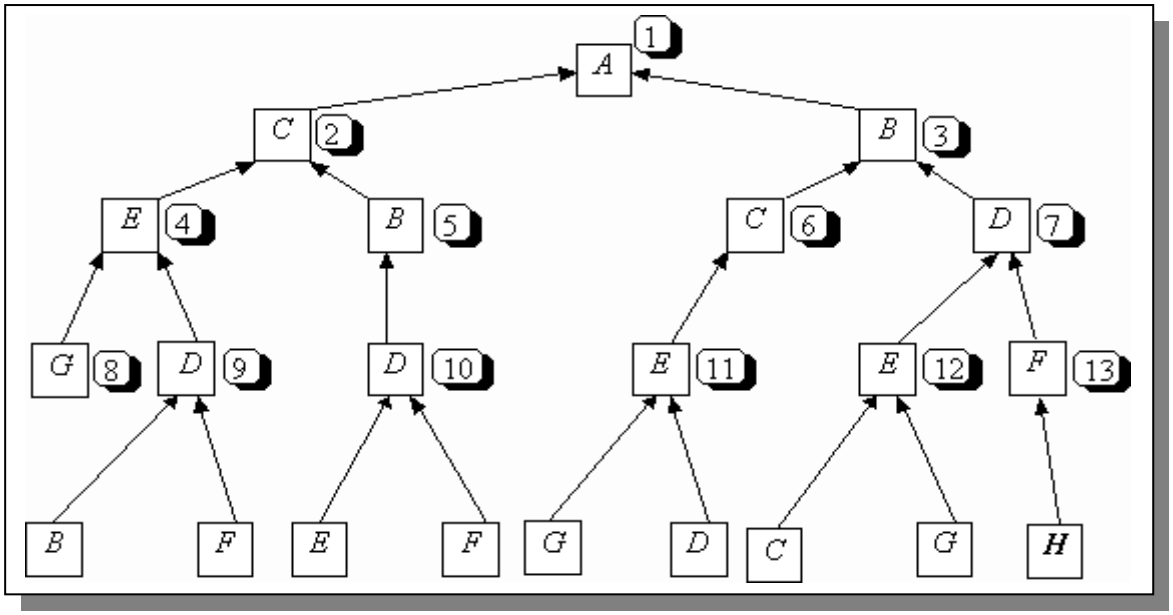
🕮 a) Búsqueda en amplitud

🕮 b) Búsqueda con retroceso

SOLUCIÓN:

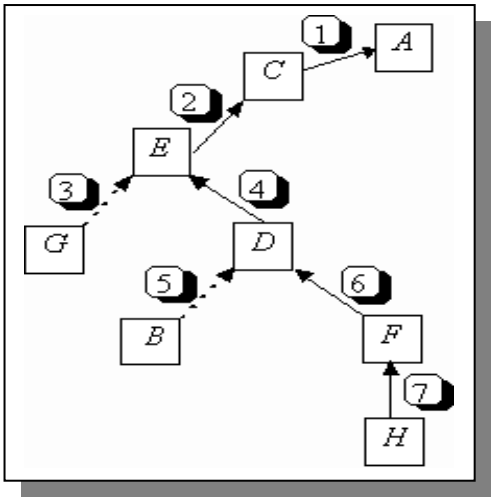
a) Búsqueda en amplitud

En el proceso de generación del árbol de búsqueda, al expandir un nodo se van a generar todos sus sucesores que no sean antecesores del mismo en dicho árbol. Los números que acompañan a los nodos de la figura muestran el orden de expansión de los mismos.



b) Búsqueda con retroceso

Aquellos enlaces que llevan a "*callejones sin salida*" se representarán mediante líneas a trazos. Etiquetando los arcos por el orden de generación de los mismos, se obtiene el resultado que aparece en la siguiente figura. Obsérvese cómo en esta estrategia de búsqueda sólo es necesario almacenar en cada momento un camino hasta el nodo raíz, al contrario que en el método de búsqueda en amplitud.



• PROBLEMA 3.2:

Se dispone de dos cántaros de agua, uno de 4 litros y otro de 3 litros de capacidad, siendo ésta la única información que se tiene de los mismos. Existe una bomba de agua con la que se pueden llenar los cántaros. Se desea que el cántaro de 4 ls. de capacidad quede lleno por la mitad y el de 3 ls. vacío. Abordar esta cuestión como un problema de búsqueda en un espacio de estados.

SOLUCIÓN:

¿Qué pretende este problema? Se presenta un ejemplo donde es necesario definir un espacio de estados a partir de un problema concreto real. Además, se muestra la conveniencia de realizar en dicho espacio una búsqueda derivada de los tipos básicos.

El espacio de estados para este problema consistirá en el conjunto de pares de enteros (x, y), tal que x = 0, 1, 2, 3 o 4 e y = 0, 1, 2 o 3, donde x e y representan el número de litros de agua que hay en los cántaros de 4 y 3 litros respectivamente. Se considerará que el estado inicial es (0, 0) y el estado meta (2, 0). En cuanto a los operadores que se pueden aplicar a los estados descritos con anterioridad, se tendrán los siguientes:

1. $(x, y) \longrightarrow (4, y)$	*Llenar el cántaro de 4 ls.* si $x < 4$
2. $(x, y) \longrightarrow (x, 3)$	*Llenar el cántaro de 3 ls.* si $y < 3$
3. $(x, y) \longrightarrow (0, y)$	*Vaciar en el suelo el cántaro de 4 ls.* si $x > 0$
4. $(x, y) \longrightarrow (x, 0)$	*Vaciar en el suelo el cántaro de 3 ls.* si $y > 0$
5. $(x, y) \longrightarrow (4, y - (4 - x))$	*Verter agua del cántaro de 3 ls. al de 4 hasta llenarlo.* si $x + y \geq 4$, $y > 0$ y $x < 4$
6. $(x, y) \longrightarrow (x - (3 - y), 3)$	*Verter agua del cántaro de 4 ls. al de 3 hasta llenarlo.* si $x + y \geq 3$, $x > 0$ e $y < 3$
7. $(x, y) \longrightarrow (x + y, 0)$	*Verter todo el agua del cántaro de 3 ls. al de 4.* si $x + y \leq 4$ e $y > 0$
8. $(x, y) \longrightarrow (0, x + y)$	*Verter todo el agua del cántaro de 4 ls. al de 3.* si $x + y \leq 3$ y $x > 0$

Respecto al proceso de búsqueda en sí que se va a emplear, se puede aprovechar el hecho de que se conoce con exactitud cuáles son el estado inicial y meta. Como consecuencia de ello, el método de búsqueda **bidireccional en amplitud** (ver apartado teórico) puede ser apropiado para abordar este problema.

El proceso de búsqueda tendrá las siguientes características:

a) Por un lado, se parte del estado inicial y se realiza un encadenamiento hacia adelante de los operadores. En cada paso, para cada estado se seleccionan aquellos operadores que puedan ser aplicables al mismo (comparando el estado con el antecedente de cada operador), generándose una serie de estados nuevos y un árbol de búsqueda que será recorrido en amplitud.

b) Por otro lado, se parte del estado final y se realiza un encadenamiento hacia atrás de los operadores. Ahora, en cada paso, para cada estado se seleccionan aquellos operadores que, una vez ejecutados, pueden llegar a producir el estado mencionado (comparando dicho estado con el consecuente de cada operador), generándose en este caso también una serie de estados nuevos y un árbol de búsqueda que será recorrido en amplitud. A diferencia del apartado a), en éste hay que ir propagando una serie de ligaduras o restricciones entre las variables que definen cada estado a lo largo del árbol (ver siguiente figura).

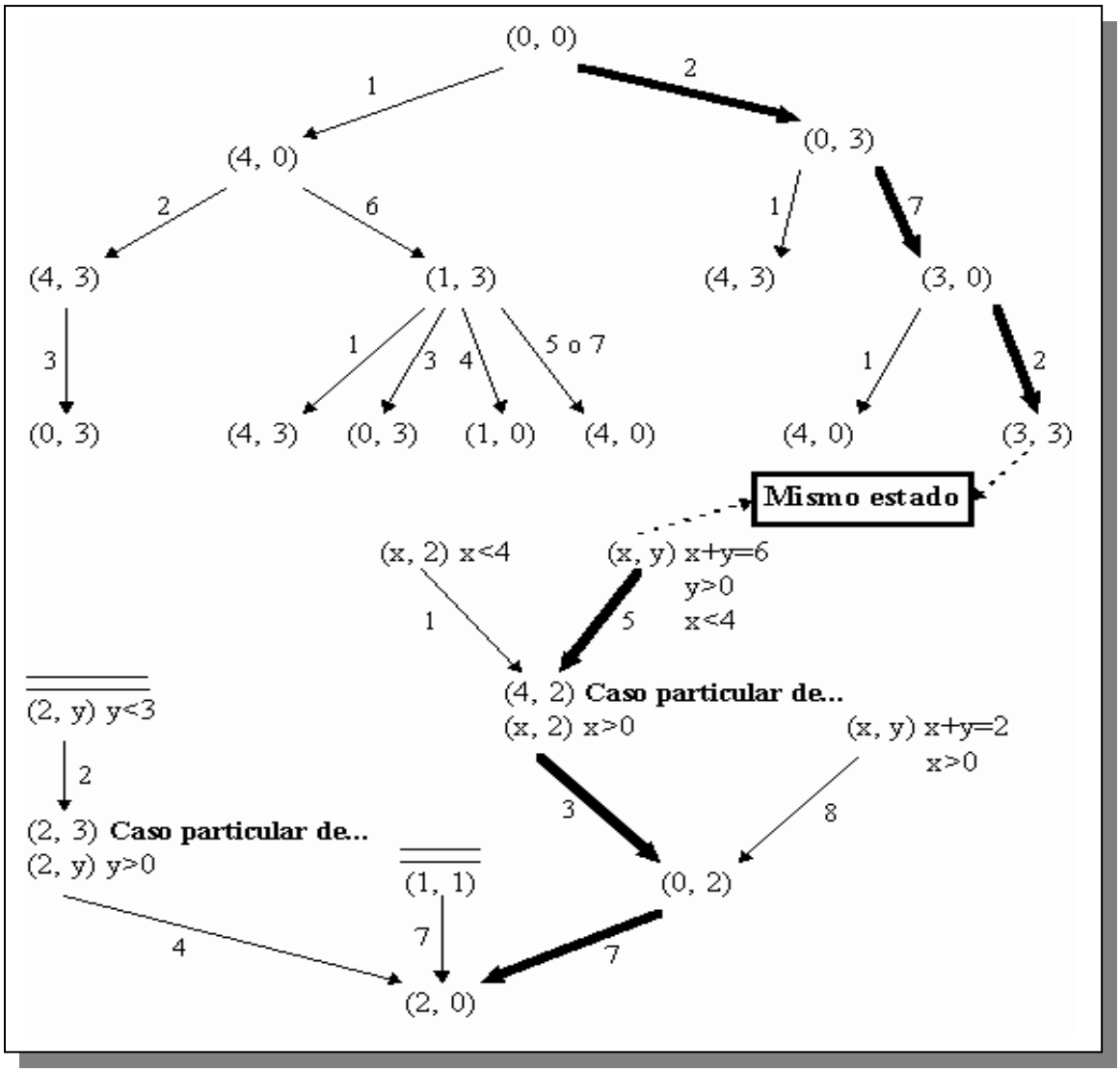
c) En cuanto a cada una de las dos búsquedas en amplitud y los dos árboles que se van creando, reseñar lo siguiente:

1) Puede ocurrir que la aplicación de un operador conduzca a un estado que coincida con un antecesor suyo en el árbol (ciclo). En este caso no se aplicará este operador.

2) Si no ocurre lo mencionado en el apartado c.1), pero el estado generado ya figuraba en el árbol de búsqueda, no tiene sentido seguir la búsqueda por este estado, puesto que la misma ya se está realizando en otra parte del árbol. Cuando esto ocurra se subrayará el estado donde se pare la búsqueda y no se colgará ningún subárbol de él.

3) Puede ocurrir que no se pueda aplicar ningún operador a un determinado estado. En este caso se subrayaría doblemente dicho estado (ver figura).

4) La generación de niveles será alternativa en los dos árboles existentes. Primero se expandirá un nivel de un árbol, luego otro nivel del otro... En cada generación de un nuevo nodo o estado se comprobará si éste figura en el otro árbol de búsqueda. En caso afirmativo, la búsqueda finaliza, ya que se ha encontrado un punto de unión entre los estados inicial y meta y, por tanto, se ha hallado un camino solución tal como se puede apreciar en la figura.



Por tanto, el camino solución encontrado es:

$$(0,0) \xrightarrow[2]{} (0,3) \xrightarrow[7]{} (3,0) \xrightarrow[2]{} (3,3) \xrightarrow[5]{} (4,2) \xrightarrow[3]{} (0,2) \xrightarrow[7]{} \mathbf{(2,0)}$$

4 Búsqueda heurística

El conocimiento dependiente del dominio puede ayudar a dirigir el proceso de búsqueda, de manera que sean exploradas en primer lugar aquellas trayectorias que parecen más prometedoras a la hora de conducir a un estado solución. Las **búsquedas heurísticas** son las que usan este conocimiento.

Básicamente se trata de asociar, a cada nodo, un valor estimativo de la distancia que le separa del *nodo meta*. Con este valor se guía la búsqueda. Para establecer estos valores se usan las **funciones de evaluación heurística** (*fev*).

4.1 Elementos implicados

Se puede distinguir dos marcos de referencia diferentes que ayudan a concretar los problemas y las soluciones aportadas. Es conveniente diferenciar las atribuciones que le corresponden a cada uno de estos:

Dominio del observador:

Los métodos heurístricos tratan de resolver problemas difíciles eficientemente. No garantizan encontrar la solución óptima, pero sí buenas aproximaciones con menos esfuerzo. El origen de estos métodos está en el análisis del comportamiento humano.

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

1. Problemas de *solución parcialmente conocida*, como por ejemplo los casos de diagnostico en medicina
2. problemas intratables de solución conocida. Son los que no pueden resolverse completamente por la complejidad implicada, pero que sí se conoce el método para resolverlos, como por ejemplo el ajedrez.

Los métodos heurísticos son adecuados para problemas en los que para alcanzar un nivel de principiante no es necesario realizar un proceso excesivamente complejo. Es objetivo del modelo no tiene por que ser exactamente obtener una determinada meta, también puede ser reducir el coste o el tamaño de la solución.

Dominio propio:

Los métodos heurísticos se utilizan para obtener soluciones útiles en problemas sujetos al fenómeno de la *explosión combinatoria*. Estos problemas difíciles pueden ser de dos tipos:

Tipo P: Los que pueden ser resueltos mediante un procedimiento de complejidad polinómica (no suele ser elevado).

Tipo PN: En los que todos los algoritmos conocidos requieren un tiempo exponencial en el peor caso, aunque se conocen algoritmos eficientes no deterministas.

Los ejemplos clásicos son el *ajedrez* (el número de operaciones necesarias es 10^{120}), *el viajante de comercio* (la solución de la mejor ruta depende en forma factorial del número de ciudades implicadas) y el juego del *8-Puzzle* (para el caso de 4×4 el espacio de estados lo forman 16! Nodos distintos).

Los objetivos declarados en el dominio del observador suelen traducirse en establecer las llamadas *funciones de evaluación heurística o fev* que ayudan a seleccionar el estado más prometedor.

4.1.1 Conocimiento de control

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

Reglas de control heurístico

En función del conocimiento reflejado, pueden ser de dos tipos:

- Dependiente del dominio: para seleccionar algunas de las situaciones alcanzadas y también para seleccionar la siguiente acción aplicable sobre aquella.
- Conocimiento general disponible sobre el funcionamiento del solucionador.

Funciones de evaluación heurística (fev)

Son una aplicación del espacio de estados sobre un espacio numérico.

$$\bullet(estado_j) = n_j$$

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

El conocimiento reflejado por las *fev* se obtiene a través del establecimiento de *modelos simplificados del modelo original*. Suelen ser *funciones de evaluación estáticas* que miden factores que determinan la proximidad al objetivo. Por ejemplo, en el caso del 8-Puzzle (8 fichas que se mueven en un tablero de 3x3 gracias a que una posición está vacía), un método heurístico consiste en comprobar cuál es el número de piezas correctamente situadas.

Si simplificamos el modelo, podemos establecer una medida heurística que se utiliza para determinar la distancia de un nodo cualquiera a la meta, llamada **distancia de Manhattan** *o distancia entre los bloques de edificios*. El grafo de búsqueda se recorre en función de un objetivo que consiste en maximizar o minimizar el valor indicado por dichas funciones, teniendo en cuenta que dicho valor debe reflejar la cercanía relativa con respecto a la meta buscada.

Evidentemente, cuanto mejor indique la *fev* la cercanía real a la meta, menor será el grafo de búsqueda expandida. Una condición que deben cumplir las *fev* es que su valor máximo (o mínimo) debe alcanzarse al ser aplicado a un estado meta. A la hora de buscar las *fev* hay que tener en cuenta el coste que eso conlleva. O sea, que si compensa el coste al hecho de realizar el proceso sin dicho conocimiento o con otras funciones menos exactas pero más sencillas. El equilibrio suele esta en obtener funciones en modelos simplificados del problema inicial.

Las técnicas que analizaremos, proporcionan un marco general independiente del dominio concreto de aplicación. Sin embargo, su efectividad depende fundamentalmente de la identificación correcta de los aspectos más relevantes en el problema que se está intentando resolver.

4.1.2 Planteamiento del problema

Se realiza con el esquema de la tarea genérica de búsqueda. La diferencia radica en la utilización del conocimiento del dominio para guiar la selección de los operadores disponibles. Una forma inmediata de aprovechar las *fev* es aplicarlas sobre los nodos generados a lo largo del proceso de búsqueda, de manera que se ordene los nodos en **ABIERTA** según el valor de la *fev* utilizada.

Así, la *fev* mide la cercanía estimada al nodo meta. Como la información disponible hasta el momento en el que se toma la decisión no es completa, la *fev* es un valor inexacto, y el control realizado no determina con certeza cuál es el mejor nodo pero sí que ayuda a seleccionar un buen nodo teniendo en cuenta dicha información. Sirve para reducir el número de nodos considerados *(complejidad espacial)*, y el tiempo empleado *(complejidad temporal)*

4.2 Una estrategia irrevocable

Hasta ahora sólo hemos presentado estrategias de búsqueda *tentativas* (en las que, en cualquier momento a lo largo del problema, puede abandonarse un determinado camino de exploración para seguir estudiando otros más prometedores).

También existen problemas para los que una estrategia *irrevocable* es adecuada; la dificultad está en la dependencia absoluta existente con respecto al criterio de ordenación seguido en el espacio de búsqueda ya que nunca se podrá abandonar el camino de exploración elegido.

4.2.1. Método del gradiente o búsqueda en escalada

Se sigue el recorrido a través de los nodos en los que el valor de la pendiente sea máximo. El parámetro que define el sentido de la máxima pendiente es el valor de la *fev* aplicada, que llamaremos •. Es decir, se sigue el recorrido a través de los nodos en los que el valor de dicha función sea máximo. No existe la posibilidad de retomar caminos abandonados, es decir, no se pueden reconsiderar las decisiones tomadas. Para que este algoritmo pueda ser aplicado, *f* debe cumplir la condición de que, para todo nodo visitado, debe existir al menos un sucesor suyo con un valor mayor de *f* — o menor si *f* alcanza su valor mínimo en los nodos meta—

Procedimiento:

1. Denominar *m* al estado inicial del problema planteado y asignar *m* a una variable llamada *elegido*.
2. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:
  2.1. Expandir *m* creando el conjunto de todos sus sucesores.
    - Para cada operador aplicable y cada forma de aplicación:
      (1) Aplicar el operador *m* generando un estado *nuevo*.
      (2) Si *nuevo* es *meta*, salir del proceso iterativo iniciado en el paso **2** y devolver dicho estado.
      (3) Si *f*(*nuevo*) es mejor que *f*(*elegido*), cambiar el valor de la variable *elegido* asignándole el valor *nuevo*.
  2.2 Si *f*(*elegido*) ≠ f(m) asignar *m* = *elegido*; en caso contrario, devolver *fallo*.

Campo de aplicación:

Se usa en todos los casos en que se pueda identificar una función que tenga la propiedad de ir creciendo (o decreciendo) hasta el valor que tenga la función en el nodo meta:

-La teoría de juegos

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

-Las situaciones en las que en cualquier estado es posible aplicar cualquiera de los operadores disponibles, lo que se denomina *conmutatividad*.

```
fun gradiente (m: nodo)
        elegido ← m
        fallo ← falso
        mientras (m#meta) y (fallo#cierto) hacer
                para cada sucesor 'n' de 'm' hacer
                        si n= meta entonces
                                devolver (n)
                        sino
                                si fev(n) mejor que fev (elegido) entonces
                                        elegido ← n
                                fsi
                        fsi
                fpara
                si fev(elegido)#fev(m) entonces
                        m ← elegido
                sino
                        fallo ← cierto
                fsi
        fmientras
ffun
```

Análisis:

Ventajas: su sencillez y que la única memoria necesaria consiste en guardar el estado alcanzado (*elegido*). En cada iteración, *elegido* será el que tenga un mejor valor de *f*. No se necesita recordar cuál es el camino recorrido.

Desventajas: la dependencia con respecto de la definición correcta de la función • (que sea lo más informativa posible). Es decir, tiene que tener un valor máximo en un nodo *meta*, debe ser relativamente sencilla de calcular, y no debe incurrir en máximos o mínimos locales.

4.3 Estrategias de exploración de alternativas

Es complicado encontrar funciones tan informadas como las requeridas por el método del gradiente, sin embargo, sí es posible identificar criterios que ayuden a dirigir adecuadamente el proceso de búsqueda, aunque no sea de una forma infalible (precisamente por la información parcial de la que parte el criterio utilizado). Las estrategias de búsqueda heurística más conocidas y utilizadas son las que realizan una exploración de alternativas en el espacio de búsqueda.

4.3.1 Búsqueda Primero el Mejor

Descripción:

Consiste en recorrer el grafo de búsqueda eligiendo en cada momento el nodo que tenga un mejor valor para una determinada función heurística •. A diferencia del *método del gradiente*, se pueden retomar caminos de exploración abandonados anteriormente (lo que resuelve la existencia de *mesetas* o puntos de *estancamiento*). La finalidad del proceso es encontrar el camino de menor coste, por lo que la función • debe medir la distancia a la meta.

Procedimiento:

1. Crear un grafo de exploración **G** que consista en un único nodo que contiene la descripción del problema. Crear una lista de nodos llamada **ABIERTA** e inicializarla con dicho nodo.
2. Crear una lista de nodos llamada **CERRADA** que inicialmente estará vacía.
3. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:
  (1) Si **ABIERTA** está vacía terminar con *fallo* en caso contrario, continuar.
  (2) Eliminar el primer nodo de **ABIERTA**, llamar a ese nodo *m* e introducirlo en la lista **CERRADA**.

(3) Expandir *m* creando punteros a *m* en todos sus sucesores, de forma que pueda saberse que su antecesor inmediato es *m*.
(4) Si algún sucesor de *m* es *meta*, abandonar el proceso iterativo establecido en el paso **2**, devolviendo el camino de la solución, que se obtiene recorriendo los punteros de sus antepasados.
(5) Para cada sucesor *n* de *m* calcular *f(n)* teniendo en cuenta las siguientes consideraciones:
(5.1) Si *n* es nuevo (no estaba en **ABIERTA** ni en **CERRADA**) asociar a *n* el valor *f(n)* e introducirlo en **ABIERTA.**
(5.2) En caso contrario (ya existía en el grafo):
(a) Si *f(n)* es mayor que el que ya tenía asociado *n*, descartar el nuevo nodo.
(b) Si no (el nuevo camino es mejor), sustituir el valor que tuviera asociado el viejo nodo por el nuevo *f(n)*, y cambiar el puntero a *m*; si el nodo estaba en **CERRADA**, *actualizar el coste de sus descendientes que ahora podrían ver reducido su valor.*

Análisis:

Ventajas: permite evitar el problema de los *mínimos locales*, tiene la garantía que, tarde o temprano, encontrará el mínimo global.

Desventajas: no se considera el camino recorrido hasta el momento, es decir, si un nodo se encuentra cerca de la meta pero se avanza por otro, que a priori tiene una *f* mejor, en una cierta profundidad se tendrá un menor coste hasta la meta, aunque en realidad el camino total será peor.

```
fun primero_el_mejor (m: nodo)
        crear (ABIERTA)
        crear (CERRADA)
        meter (ABIERTA,m)
        fallo ← falso
        mientras (m#meta) y (fallo #cierto) hacer
                sacar (ABIERTA, m); meter (CERRADA, m)
                para cada sucesor 'n' de 'm' y no antecesor de 'n' hacer
                        si n= meta entonces
                                Devolver_camino_solución (n)
                        si no_está (ABIERTA, n) y no_está (CERRADA, n) entonces
                                n.valor ← fev (n)
                                n.padre ← m
                                meter (ABIERTA, n)
                        sino
                                si n.valor es pero que fev(n) entonces
                                        n.valor ← fev (n)
                                        n.padre ← m
                                        si está (CERRADA,n) entonces
                                          Recalcular_Descendientes (n)
                                        fsi
                        fsi
                fpara
                si Abierta = Ø entonces
                        fallo ← cierto
                fsi
        fmientras
ffun
```

Existe una variante de este método llamada ***búsqueda en haz*** que pretende acelerar el proceso de búsqueda ampliando el rango de estados que son considerados simultáneamente como mejores. En cada momento se mantienen dos estructuras de datos, una contiene los estados que pertenecen al haz de búsqueda actual y otra agrupa a los posibles sucesores. Cuanto más informada sea la función que determina la validez de un nodo, más estrecho será el haz y más eficiente será el proceso de búsqueda realizado. En general este método requiere menos recursos.

4.3.2 algoritmo A*

Es el mejor de tipo *Primero el Mejor* que sirve para resolver el problema que conlleva el no considerar el camino recorrido hasta un momento dado. La clave de este método está en ampliar la definición de • para que tenga en cuenta el coste del camino recorrido, resultando entonces la expresión siguiente:

$$\bullet(n) = g(n) + h(n)$$

Siendo g(n) el coste real del camino de menor coste encontrado hasta el momento, h(n) es el coste del camino óptimo desde el nodo *n* hasta una meta y •(n) es el coste total del camino óptimo de una solución que pase por *n*.

Procedimiento:

1. Crear una lista de nodos llamada **ABIERTA** e inicializarla con un único nodo raíz que contiene el estado inicial del problema planteado. Llamar a este elemento *r* y asignarle $g(r) = 0$.
2. Crear una lista de nodos llamada **CERRADA** que inicialmente estará vacía.
3. Hasta que se encuentre una *meta* o se devuelva *fallo* realizar las siguientes acciones:
3.1. Si **ABIERTA** está vacía terminar con *fallo*; en caso contrario, continuar.
3.2. Eliminar el nodo de **ABIERTA** que tenga un valor mínimo de *f*, llamar a este nodo *m* e introducirlo en la lista **CERRADA**.
3.3. Si *m* es *meta*, abandonar el proceso iterativo señalado en **3** devolviendo el comino de la solución, que se obtiene recorriendo los punteros de sus antepasados (creados en **3.5**).
3.4. En caso contrario expandir *m* generando todos sus sucesores.
3.5. Para cada sucesor *n'* de *m*:
(1) Crear un puntero de *n'* a *m*.
(2) Calcular *g(n') = g(m) + c(m, n')| c(a, b)*: coste de pasar de *a* a *b*.

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

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

(4) si *n'* no cumple (**3**), comprobar si está en **CERRADA**, llamar *n* al nodo encontrado en dicha lista y realizar lo siguiente:

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

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

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

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

```
fun Procedimiento_A* (raíz: nodo)
        crear (ABIERTA)
        crear (CERRADA)
        meter (ABIERTA, raíz)
        fallo ← falso
        raíz.g ← 0
        mientras (m # meta) y (fallo # cierto) hacer
                sacar (ABIERTA, m)
                meter (CERRADA, m)
                si m = meta entonces
                        Devolver_Camino_Solución (n)
                fsi
        para cada sucesor 'n' de 'm' hacer
                g(n) ← g(m) + coste (m,n)
                si está (ABIERTA,n) entonces
                        si n.g es peor que g(n) entonces
                                n.g ← g(n)
                                n.f ← g(n)+h(n)
                                n.padre ← m
                        fsi
                sino
                        si está (CERRADA, n) entonces
                                si n.g es peor que g(n) entonces
                                        Recalcular_Descendientes (n)
                                fsi
                        fsi
        fpara
        si ABIERTA=Ø entonces
                fallo ← cierto
        fsi
    fmientras
ffun
```

Relaciones con otros métodos de búsqueda:

1. Si ∀n ∀m (h(n) = 0) ∧ (c(m,n) = 1) entonces A* se convierte en un proceso de *búsqueda en amplitud*.
2. Si ∀n h(n) = 0 entonces A* se convierte en el método denominado de *coste uniforme* (es una variación de la búsqueda en amplitud donde lo que se busca no son caminos de la menor longitud, sino menor coste).

Propiedades formales

En ocasiones se han criticado los métodos de búsqueda heurística por que no son precisos. Sin embargo, el método A* posee una serie de propiedades que pueden confirmarse de antemano:

1. Es un *método completo*: incluso para grafos infinitos encuentra la solución, sí ésta existe.
2. Es *admisible*: la solución que encuentra es la óptima.
3. Si la fev h es *monótona y consistente,* es decir, si el valor de la función nunca decrece a lo largo de un camino desde la raíz, entonces el algoritmo encuentra un camino óptimo para todos los nodos expandidos.
4. Entre todos los algoritmos que utilizan una fev consistente h(n) y que encuentra una solución óptima, A* expande el menor número de nodos.

Análisis:

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

Desventajas: La propiedad de admisibilidad hace que el algoritmo gaste mucho más esfuerzo en discriminar entre caminos prácticamente iguales, por lo que es más realista buscar soluciones dentro de unos márgenes acotados de error. Un inconveniente es el espacio requerido.

Una opción para reducir el espacio (requerido por mantener **ABIERTA**) es realizar un planteamiento semejante al método de búsqueda en haz, llamado **A* en profundidad progresiva** *(A*-P).*

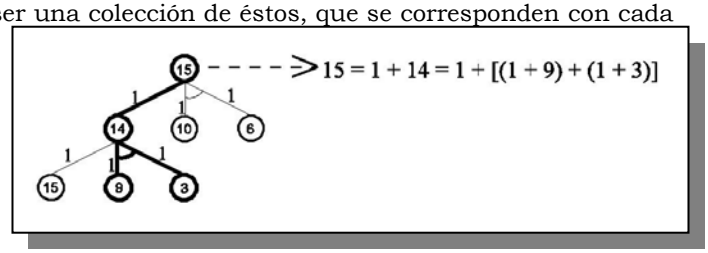
4.3.3 Búsqueda heurística en integración simbólica

** Leer este ejemplo en el libro (página 158 y siguientes)**

4.3.4 Método AO*

Descripción:

Este procedimiento es una generalización del algoritmo A* para el caso de grafos Y/O. Encuentra una solución óptima siempre y cuando la función heurística *h* de estimación de la distancia al objetivo siga un criterio *optimista* de valoración (devuelva un valor menor o igual que la distancia real existente). Aquí, la solución ya no es un único nodo meta; puede ser una colección de éstos, que se corresponden con cada uno de los subproblemas en los que puede haberse dividido el problema inicial. Para calcularlo es necesario distinguir los enlaces O de los enlaces Y, ya que en estos últimos hay que sumar el coste de la obtención de cada uno de los nodos conectados por el enlace.



Esto implica que, ahora, ya no es suficiente con encontrar un camino solución, sino que, en muchos casos, se obtendrá un árbol (subárbol) solución, o un grafo (subgrafo) solución. Pues bien, el método AO* intenta que es grafo sea óptimo (de menos coste).

Procedimiento:

1. Crear un grafo G que en un principio está formado exclusivamente por el nodo raíz *m*. Estimar la distancia heurística a la meta de Esta mediante *h(m)*.
2. Realizar hasta que *m* se considere *resuelto* o hasta que su coste supere un cierto valor *coste-máximo* (incluye el caso que no queden más sucesores por obtener; ver paso **(2)**.
(1) Seguir los enlaces marcados que señalan el mejor grafo parcial de la solución obtenido hasta el momento hasta alcanzar los extremos de dicho grafo. Escoger alguno de dichos extremos que llamaremos *n*.
(2) Expandir *n* generando todos sus sucesores *s*. Si no tiene ninguno, asignarle el valor *coste-máximo.*
(2.1) ∀*s*: si *s* es un nodo terminal marcarlo como *resuelto* asignarle *h(s) = 0*; en caso contrario asignarle *h(s)*.
(3) Crear un conjunto S solo con los nodos s.
(4) Realizar hasta que S se vacíe:
(4.1) Sacar de S algún nodo *s'* que no tenga descendientes en S.
(4.2) Actualizar el coste de *s'*:
- Calcular el coste de todos los enlaces que parten de *s'* y asignarle como *h(s')* el menor de éstos marcando dicho enlace (y borrando si existiera una marca de otro enlace previamente elegido saliente de *s'*). Si todos los nodos del enlace marcado son terminales, marcar también *s'* como *resuelto*.

(4.3) Si **(4.2)** ha hecho que s' sea resuelto o si su *h(s')* ha cambiado, añadir todos sus antecesores a S y continuar el proceso recursivo de actualización de valores de los nodos (que pueden llegar hasta la raíz del grafo G).

Observación: la clave del algoritmo AO* está en la búsqueda (hacia delante) de formas de expandir el mejor grafo parcial de la solución encontrado hasta el momento.

Propiedades:

Si se cumplen:
1. ∀n h(n) ≤ h*(n)
2. Cada vez que se actualiza el coste de h(n) en función de sus sucesores el valor previo de h(s) nunca supera al calculado en función de aquellos.

Entonces el algoritmo es *admisible y encuentra el camino óptimo.*

4.4 Búsqueda con Adversarios

Los juegos son un área especialmente adecuada para abstraer los elementos dependientes del dominio de aplicación y poder centrarse en los mecanismos que guían el comportamiento. En inteligencia artificial se han estudiado aquellas situaciones en las que participan varios jugadores (generalmente dos)

perfectamente informados (ambos conocen las reglas de juego y la disposición de todas las piezas del juego).

En los problemas de juegos, los *enlaces O* se utilizan para representar los distintos movimientos realizables en un instante dado; y los *enlaces Y* para señalar los movimientos del adversario. Los dos jugadores participan por turnos en el juego. El resultado de cada movimiento sitúa a cada jugador en un nuevo *conjunto de estados*. Esta forma de interpretar el árbol que representa el juego, permite considerar a la vez todas las respuestas posibles ante una jugada seleccionada (cada nodo, además de ser un estado del juego, representa una clase de respuestas alternadas a las acciones del contrario). Los *nodos hojas* representan las tres situaciones posibles: ganar, perder o empatar la partida representada por la rama que conduce de la raíz a dicho nodo. El número de movimientos posibles que deben considerarse para garantizar una estrategia ganadora es generalmente intratable. Surge por lo tanto el problema de la *explosión combinatoria* en la mayoría de los problemas (se plantea la necesidad de aplicar *fev' s*). Por ser problemas intratables se simplifica el proceso determinando el estado de los nodos hasta un cierto *límite de profundidad* en el que el valor del nodo es el valor devuelto por la aplicación de la *fev*. Llamamos al nivel más profundo analizado *frontera de explosión o frontera*.

4.4.1 Estrategia MiniMax

Descripción:

Es un esquema genérico de búsqueda para situaciones en las que el objetivo es ganar una partida en la que participan dos adversarios que realizan movimientos alternos en el juego. La efectividad de esta estrategia reside en la *fev* aplicada.

El método consiste en prever el mayor número de jugadas posibles (tanto propias como del adversario), y actuar en consecuencia. Se recorre un árbol Y/O llevando a cabo una política conservadora. Se considera siempre que el adversario va a realizar la mejor de las opciones posibles cuando a él le toque mover. Los nodos del árbol son de dos tipos: nodos **MAX** y nodos **MIN**, cada uno de éstos toma las decisiones uno de los dos contrincantes. El objetivo del jugador MAX es maximizar el valor de la función *f* de evaluación que mide la proximidad estimada a la meta de una situación dada y, así mismo, los nodos MIN reflejan el modo de actuar del adversario con respecto al objetivo del nodo MAX; es decir, realizar aquel movimiento que haga más pequeño el valor de *f*.

Para poder etiquetar los nodos con un valor *fev*, se recorre el árbol hasta los nodos hojas (siempre y cuando el árbol sea tratable) o hasta un determinado límite de profundidad. Llegados a este límite, se aplica una *fev* heurística al estado de la partida representado por el nodo (si es un nodo hoja se asigna: ganador=1, perdedor=-1 y empate=0). Los valores de *fev* obtenidos se van subiendo por el árbol aplicando a cada nodo padre el valor máximo o mínimo del valor de sus hijos (dependiendo si es un nodo a jugar por MAX o por MIN), hasta llegar al nodo actual.

El estado de un nodo MAX será:
☺ **ganador** si alguno de sus sucesores conduce a un terminal ganador (carácter disyuntivo).
☹ **perdedor** si ningún sucesor conduce a uno ganador (carácter conjuntivo).
😐 **empate** si ningún sucesor conduce a la meta y al menos uno conduce al empate.

El estado de un nodo MIN será:
☺ **ganador** si todos sus sucesores llegan a un final ganador.
☹ **perdedor** si alguno conduce a un final perdedor.
😐 **empate** si ningún sucesor es perdedor y alguno conduce a un empate.

Por lo tanto una estrategia ganadora para MAX sería un subárbol en el que todos sus nodos terminales son ganadores y una estrategia ganadora para MIN será el subárbol en el que todos sus nodos terminales son perdedores.

El método de MINIMAX se comporta como un procedimiento de *búsqueda con retroceso* con una frontera de exploración calculada.

Existen dos formas de etiquetar los nodos del árbol de búsqueda o, dicho de otro modo, de evaluarlos (ver figura siguiente):

a) Etiquetado según MAX: a cada *nodo* se le asocia un valor que representa lo prometedora que es dicha situación de la partida **para el jugador MAX**, independientemente de si dicho nodo se corresponde con una situación de la partida en la que le toca "mover" a MAX o a MIN. Como consecuencia de ello, desde un nodo padre MAX se elegirá aquella jugada que conduzca al hijo con mayor valor asociado. Por otra parte, desde un nodo padre MIN se elegirá la jugada que lleve al hijo con menor valor asociado. Por tanto,

$$MAX(m)=\begin{cases} \bullet\ fev(m) \text{ si } m \text{ no tiene sucesores.} \\ \bullet\ \max_n\{MAX(n)\}, \text{ siendo } n \text{ sucesor de } m \text{ y } m \text{ un nodo MAX.} \\ \bullet\ \min_n\{MAX(n)\}, \text{ siendo } n \text{ sucesor de } m \text{ y } m \text{ un nodo MIN.} \end{cases}$$

b) Método *MMvalor*: este tipo de etiquetado es menos intuitivo que el anterior y se explica aquí únicamente debido a que los algoritmos que lo emplean suelen ser más eficientes. Se intenta que desaparezca la distinción entre nodos MAX y MIN. Asigna un valor a cada nodo que representa lo prometedora que es dicha situación de la partida **para el jugador que posee el turno de movimiento en la misma**. Se sigue el convenio de que si una situación de la partida es prometedora para el jugador MAX en un valor f_0, lo será para MIN con un valor $-f_0$. En estas condiciones, independientemente de si nos encontramos en un nodo MAX o MIN, la labor que hay que realizar es siempre la misma: desde cualquier nodo padre se elegirá aquella jugada que conduzca al hijo con un valor asociado menor, al que llamaremos v_0. Como desde el nodo padre se ha elegido acceder a una situación de la partida que es prometedora para su contrincante en un valor v_0, a dicho nodo padre se le asociará un valor $-v_0$, ya que se ha supuesto en este método que el criterio para asignar valores a nodos depende del tipo de nodo considerado. Como consecuencia de todo lo anterior, en cada actualización del valor de un nodo hay que realizar primeramente una operación de cálculo de un mínimo y posteriormente un cambio de signo. En la literatura de búsqueda, sin embargo, normalmente aparece otro conjunto de operaciones equivalentes a la anterior: primero se cambian de signo los valores asociados a los nodos hijo y a continuación se elige el máximo de los mismos como valor asociado al nodo padre. Por tanto,

$$MMvalor(m)=\begin{cases} \bullet\ fev(m) \text{ si } m \text{ no tiene sucesores y es un nodo MAX.} \\ \bullet\ -fev(m) \text{ si } m \text{ no tiene sucesores y es un nodo MIN.} \\ \bullet\ \max_n\{-MMvalor(n)\}, \text{ siendo } n \text{ sucesor de } m. \end{cases}$$

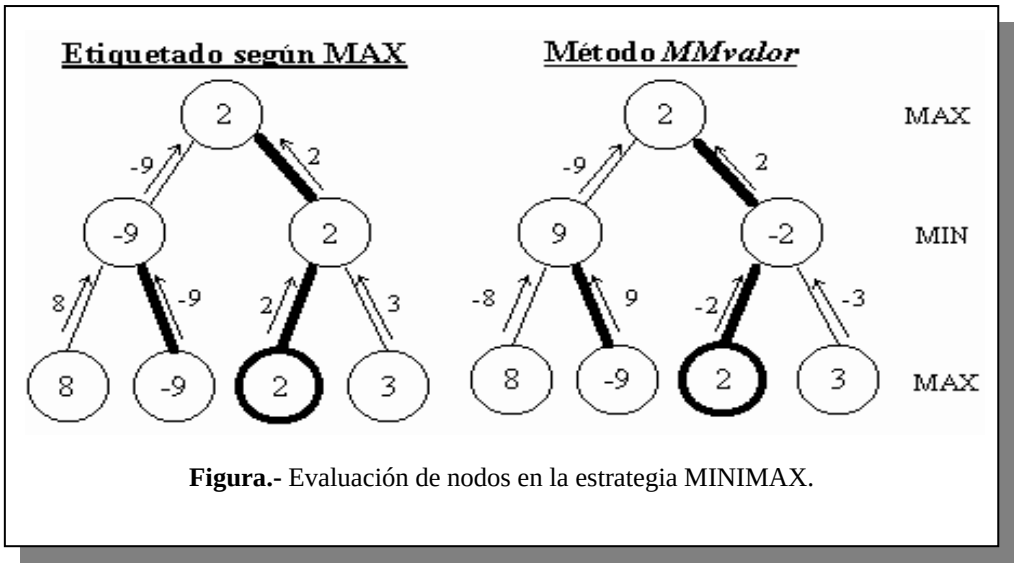


Figura.- Evaluación de nodos en la estrategia MINIMAX.

El algoritmo es un proceso recursivo de exploración del árbol que refleja el etiquetado realizado por *MMvalor(n).*

Procedimiento:

MINIMAX*(m, profundidad, jugador)*

1.si *m* no tiene sucesores o si se considera que *m* ha alcanzado el límite de profundidad en *profundidad*, devolver: *mmv(m) = f(m)*

2. generar los sucesores de *m*:

2.1 inicializar la variable *mejor* con el valor mínimo que esta pueda tener (puede ser un valor de referencia previamente establecido)

mejor = min_j {*f(j)* ∀*j*}

2.2 para cada sucesor *n* de *m*:

(1) *mmv(n) = MINIMAX(n, profundidad + 1, C(jugador));*
siendo *C(jugador)* una función que cambia de jugador.

(2) *mejor =max[-mmv(n), mejor]*

3. una vez se han analizado recursivamente todos los sucesores de un determinado nivel (indicado por la variable *profundidad*), se devuelve el que ha obtenido un mejor valor

mmv = mejor

observación: a la vez que se devuelven los valores mejores se podría considerar necesario el haber devuelto el camino que pasa por ellos pero en la versión más sencilla de este procedimiento basta con saber cual es la siguiente mejor jugada. La primera llamada al procedimiento sería MINIMAX(*nodo-inicial*, 0, MAX).

Complejidad:

Un árbol de coste uniforme de una profundidad *p* y con factor de ramificación *n* contiene n^p nodos terminales que serán analizados por el procedimiento MINIMAX. Pero hay que encontrar al menos dos estrategias, una para MAX y otra para MIN, que sean compatibles. Dado que una estrategia se expande en niveles alternativos del árbol su número de nodos es alrededor de $n^{p/2}$, desgraciadamente hay que realizar un número indeterminado de intentos antes de encontrar dos estrategias compatibles, por lo que dicho límite rara vez es alcanzable.

Análisis:

Ventajas:
Es un método general y sencillo que permite representar y buscar estrategias ganadoras en los problemas de juegos en los que hay dos adversarios. El etiquetado de los nodos permite identificar las distintas estrategias posibles.

Desventajas:
Peca por ser demasiado exhaustivo (poco eficiente debido a su complejidad).

4.4.2 Estrategia de poda α–β.

Descripción:

Es una modificación del procedimiento MINIMAX que permite ahorrarse el recorrido de caminos inútiles del árbol de búsqueda. Dada su efectividad es el método más común en los problemas de juegos.

Esta estructura abandona el estudio de subárboles que no pueden conducir a soluciones mejores que las ya encontradas en un cierto nodo. Para que el recorrido de un cierto subárbol no sea rechazado de antemano se tiene en cuenta las siguientes consideraciones:

- ☞ Si se está bajo un nodo MAX, se tiene que garantizar por lo menos el valor máximo ya alcanzado.
- ☞ Si se está bajo un nodo MIN se tiene que cumplir que se está buscando por debajo de valor mínimo encontrado hasta el momento.

Los valores de umbral inferior para nodos MAX y de cotas superiores nodos MIN se propagan hacia abajo en el proceso recursivo de exploración del árbol. Así, **α** es el valor mínimo que debe tener un nodo MIN para que éste siga siendo explorado (igual al mayor valor encontrado hasta el momento de todos sus antecesores MAX) y **β** es el valor máximo que debe tener un nodo MAX para que este siga siendo explorado (igual al valor más pequeño encontrado de todos sus antecesores MIN).

Procedimiento de poda α–β.:

Alfabeta(*m, profundidad, jugador, límite-actual, límite-siguientes*)

1. si *m* no tiene sucesores o si se considera que *m* ha alcanzado el límite de profundidad en *profundidad*, devolver: **αβ***v(m) = f(m)*

2. generar los sucesores de *m*:

2.1 para cada sucesor de *m*:

(1) **αβ***v(n) = alfabeta* (*n, profundidad* + 1, C(*jugador*), -*límite-siguientes*, -*límite-actual*);
siendo C(*jugador*) una función que cambia de jugador.

(2) *límite-siguientes = max*[-**αβ***v(n), límite-siguientes*]

(3) si *límite-siguientes* ≥ *límite-actual* entonces abandonar este nivel y devolver *límite-siguientes*.

3. una vez se han analizado recursivamente todos los sucesores, se devuelve el que ha obtenido un mejor valor:

$\alpha\beta v(n)$ = *límite-siguientes*;

observación: la primera llamada al procedimiento se realiza tomando como α el valor máximo que podría tener *f(n)* y como valor β el valor mínimo de la misma. Límite-actual sirve para comprobar que en el nodo actual no se viola la restricción impuesta por un límite de un nivel superior y límite-siguientes para determinar cuál es el mejor valor que deben considerar los descendientes en el árbol. Así, dentro de un nivel, límite-siguientes refleja el mejor valor encontrado hasta el momento en dicho nivel y si dicho valor no supera el valor ya alcanzado por niveles superiores reflejado en la variable límite-actual.

Complejidad:

Si el árbol de búsqueda está perfectamente ordenado (es decir, si en las ramas de más a la izquierda se obtienen los mejores valores de α y de β, entonces el número de nodos que son evaluados es (con longitud *p* de árbol y factor de ramificación *n*):

1. si *p* es un número par: $2n^{p/2} - 1$
2. si *p* es un número impar: $n^{(p+1/2)} + n^{(p-1)/2} - 1$

En el peor caso, habría que examinar todos los nodos terminales y tendría la misma complejidad que ya se ha comentado en el procedimiento MINIMAX.

Análisis:

Ventajas:
Es un método eficiente que evita el recorrido exhaustivo y reduce el número de nodos expandidos y así, permite afrontar problemas supuestamente intratables.

Desventajas:
La dependencia excesiva del ordenamiento realizado de los nodos del árbol.

4.5 Análisis de medios-fines.

Este método se definió inicialmente en el contexto llamado *Solucionador General de Problemas* (GPS). Consiste en distinguir cuáles son los *medios* disponibles (los operadores) y los *fines* (los estados meta). Así, analiza las diferencias que hay entre la descripción inicial del problema planteado y la meta deseada. Una vez detectadas. Se accede a una tabla en la que aparecen ordenadas en función de su importancia en el dominio. En cada fila se indica cuáles son los operadores disponibles para reducir la diferencia correspondiente a dicha fila. Puede ocurrir que un operador no se pueda aplicar directamente en la situación inicial; en cuyo caso, se establece como *submeta* el alcanzar un estado en que dicho operador sea aplicable. La aplicación de dicho operador garantiza la reducción de algunas diferencias pero no todas.

Podemos sacar una serie de consecuencias:

1. Es necesario establecer claramente cuales son las condiciones previas (*precondiciones*) que permiten aplicar cada uno de los operadores del sistema.
2. Deben ser fácilmente accesibles los resultados causados por la utilización de cada uno de los operadores.
3. La tabla de diferencias depende fuertemente del dominio de aplicación.
4. El recorrido realizado del grafo de búsqueda de la solución es una combinación de *búsqueda con retroceso* (resolver las diferencias, y para encontrar estados en los que los operadores puedan ser aplicados) y de *encadenamiento hacia delante* (cada vez que se aplica un operador y se alcanza un nuevo estado).

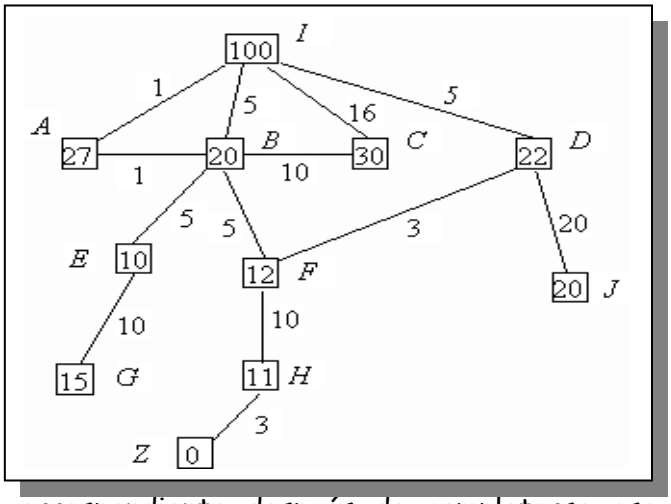
La solución consiste en tener dos tablas: una de operadores, con dos columnas claramente diferenciadas (la de *precondiciones* y la de *resultados*) y otra, la de las diferencias ya mencionadas.

Una variación de este método denominada STRIPPS se ha aplicado en la resolución de problemas en el campo de la robótica. Cada operador esta formado por dos listas: *adiciones* (elementos introducidos como resultado de la aplicación del operador) y *eliminaciones* (con el conjunto de condiciones que dejan de cumplirse una vez se haya aplicado dicho operador).

PROBLEMAS

- **PROBLEMA 4.1:**

Aplicar el algoritmo A* al siguiente grafo. El nodo inicial es I y hay un solo nodo meta que en este caso es Z. A cada arco se le ha asociado su coste y a cada nodo la estimación de la menor distancia desde ese nodo al nodo meta.



SOLUCIÓN :

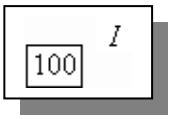
¿Qué pretende este problema? Este ejercicio hace hincapié en los procesos de reorientación de enlaces que pueden ocurrir durante la ejecución del algoritmo A*.

Junto con la explicación de lo que se hace en cada etapa del algoritmo, figura cómo quedarían ABIERTA, CERRADA y el grafo correspondiente después de completarse esa etapa. Al lado de cada nodo en la lista ABIERTA figura el valor de la función heurística en ese nodo.

• **Inicialmente**:

ABIERTA: *I*(0+100)

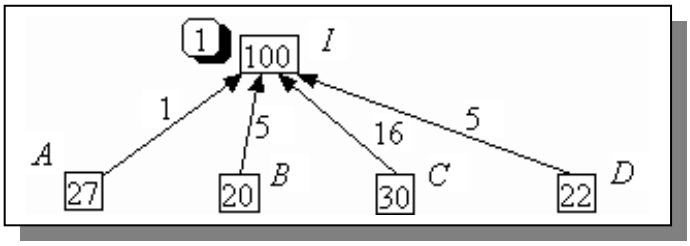
CERRADA: ∅



• **La única posibilidad ahora es expandir *I*** :

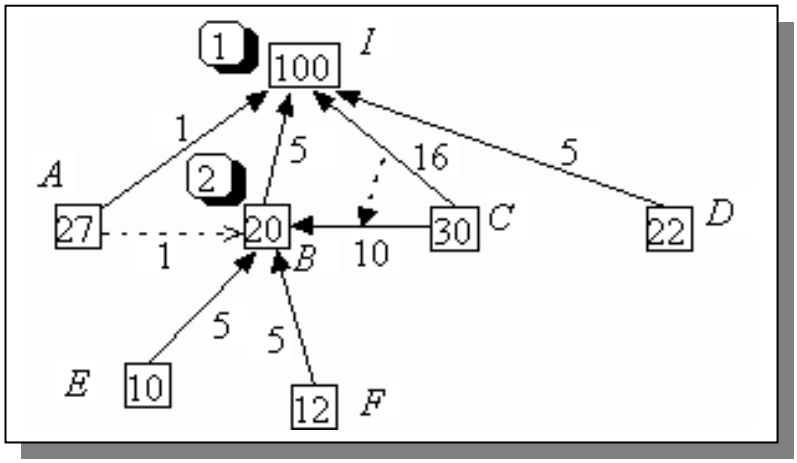
ABIERTA: *A*(1+27=28), *B*(5+20=25), *C*(16+30=46), *D*(5+22=27)

CERRADA: *I*



• **Ahora el nodo más prometedor de ABIERTA es *B*, que es seleccionado para su expansión**. *C* y *A*, dos de los sucesores de B, ya estaban en ABIERTA. El nuevo camino encontrado hasta *C* es menos costoso que el anterior, con lo que hay que efectuar una reorientación. Esto no ocurre con *A*, ya que el coste del nuevo camino encontrado desde este nodo hasta la raíz es 6, mientras que el coste inicial era 1.

ABIERTA: *A*(28), *C*(45), *D*(27), *E*(20), *F*(22)



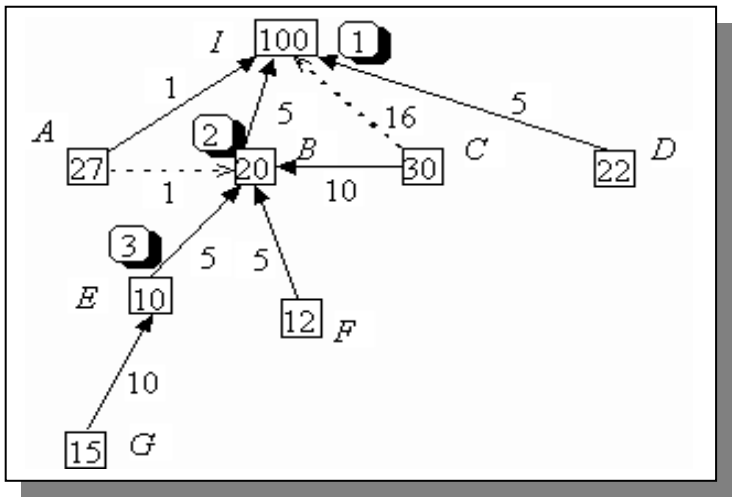
CERRADA: *I*, *B*

Como consecuencia de la reorientación hecha, se pasa de *C*(46) a *C*(45). Este cambio queda resaltado subrayando el nodo *C* en al lista ABIERTA.

• **El siguiente nodo a expandir de entre los que están en ABIERTA es *E*** :

ABIERTA: *A*(28), *C*(45), *D*(27), *F*(22), *G*(35)

CERRADA: *I*, *B*, *E*



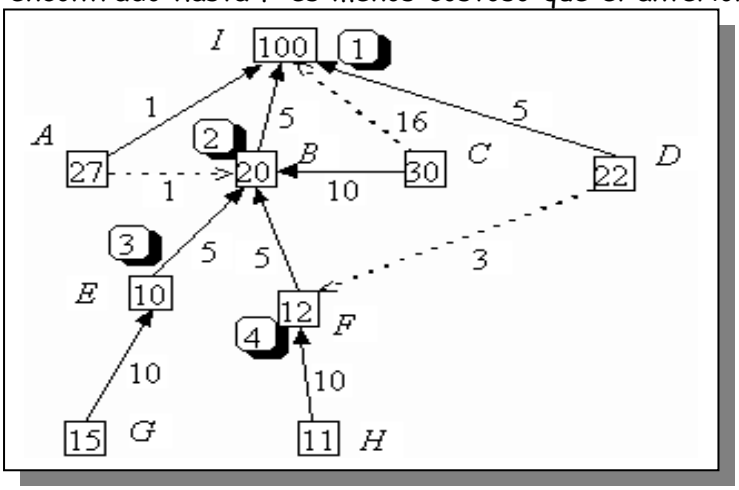
• **Expandiendo ahora *F*** :

ABIERTA: *A*(28), *C*(45), *D*(27), *G*(35), *H*(31)

CERRADA: *I*, *B*, *E*, *F*

Obsérvese que no hay reorientación desde el nodo D.

• **Al expandir en esta ocasión el nodo *D***, su sucesor F ya estaba en CERRADA. Como el nuevo camino encontrado hasta F es menos costoso que el anterior, hay que redirigir el enlace del árbol que parte de F. Como consecuencia de ello, habrá que estudiar si los caminos de sus descendientes también pueden verse afectados. La situación final es la siguiente :

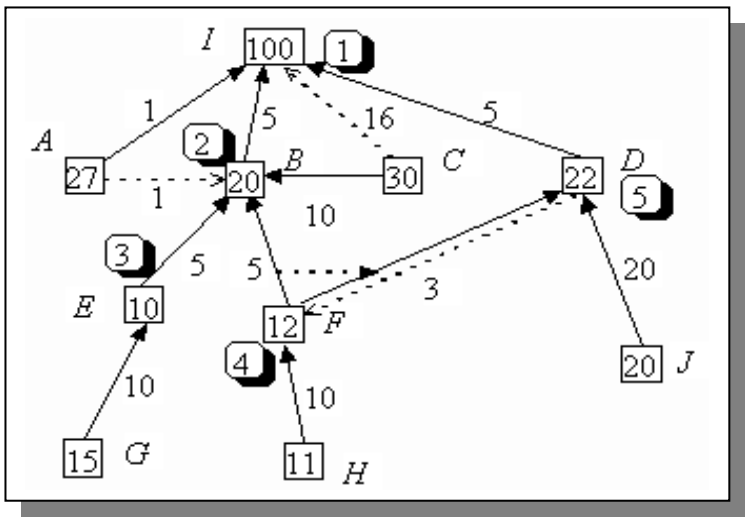


ABIERTA: *A*(28), *C*(45), *G*(35), *H*(29), *J*(45)

CERRADA: *I*, *B*, *E*, *F*, *D*

H pasa de tener un coste estimado de 31 a 29 (ver figura) como consecuencia de que el camino de menos coste entre la raíz y H ya no pasa por el nodo B, sino por D.

• **El nodo a expandir ahora es *A***. Al generarse un nuevo camino desde el nodo raíz hasta B, estamos en el mismo caso de la última expansión. Ahora habrá que redirigir el enlace que partía de B y, además, habrá que estudiar lo que pasa con sus descendientes: E, C y G verán modificado el valor que la función de evaluación heurística toma sobre ellos; por otra parte hay que redirigir

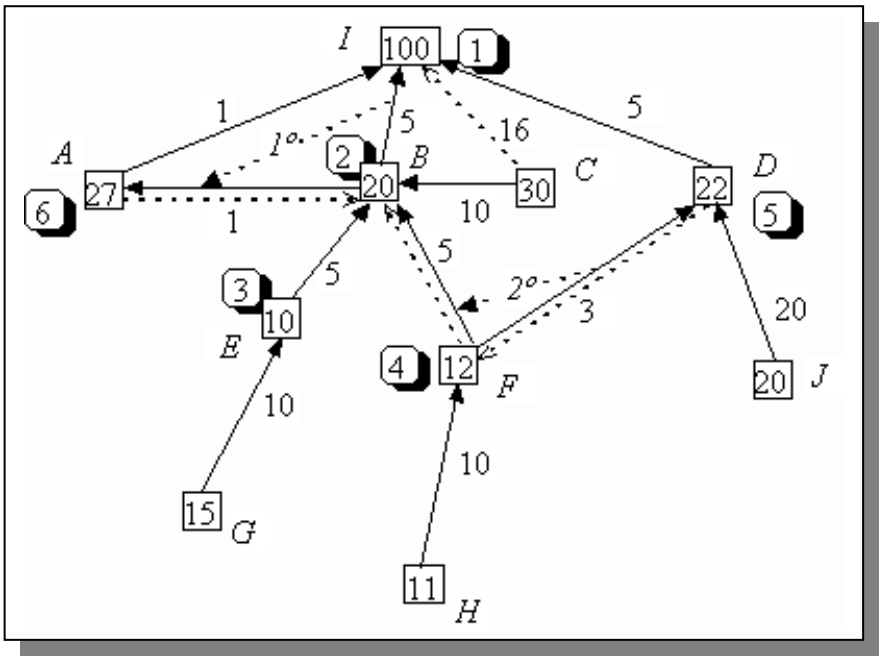


nuevamente el enlace que parte de F y recalcular, como consecuencia de ello, el valor que la función heurística asocia al nodo H.

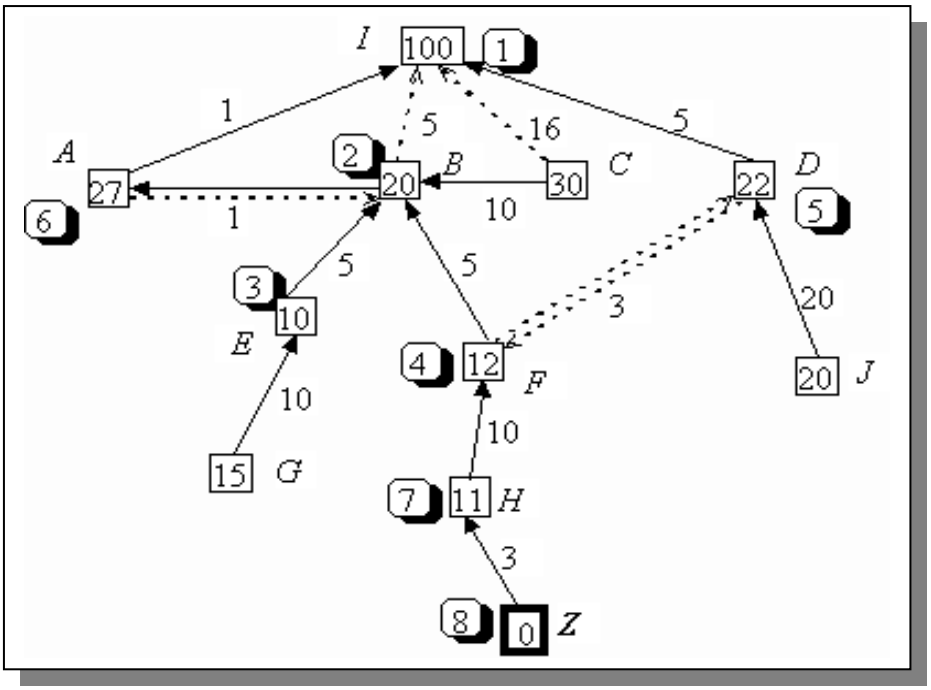
ABIERTA: *C*(42), *G*(32), *H*(28), *J*(45)

CERRADA: *I*, *B*, *E*, *F*, *D*, *A*

Obsérvese en la figura la segunda reorientación que hay que efectuar como consecuencia de la primera (1º ⇒ 2º).



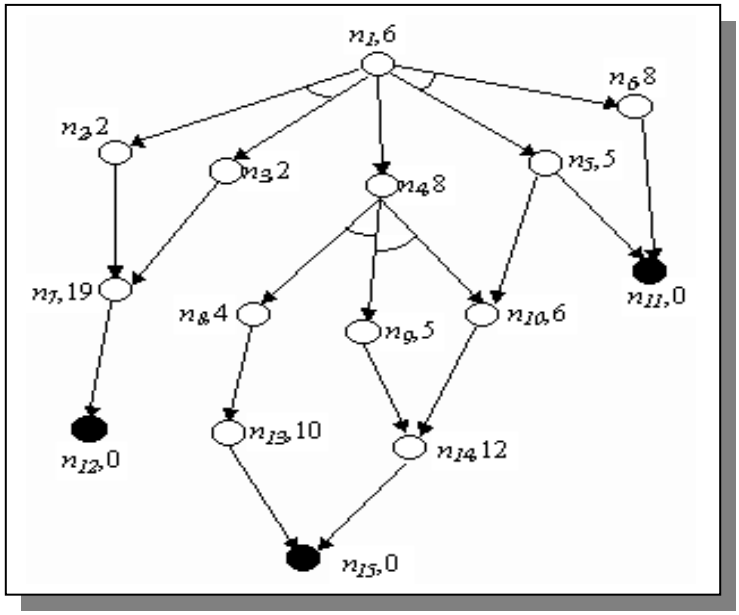
• **Finalmente, el algoritmo expandirá el nodo *H* y posteriormente el nodo meta *Z***, con lo que termina todo el proceso.



El camino solución será (ver figura):

I, A, B, F, H, Z

- **PROBLEMA 4.2:**



Dado el grafo Y/O de la figura, explorarlo mediante el algoritmo YO*.

n1 es el nodo inicial y n11, n12 y n15 los nodos terminales. Junto a cada nodo figura el valor de la función heurística h en el mismo. Si hubiera habido en el grafo algún nodo sin sucesores que no fuera terminal, se le habría asignado un valor de h de magnitud coste-máximo (ver algoritmo YO* del apartado teórico).

SOLUCIÓN:

- **Inicialmente**:

A continuación se describirá el comportamiento del algoritmo a lo largo de cada ciclo. En cada uno de ellos, primero se expande un nodo hoja cualquiera del grafo solución parcial que cuelga de n1, y luego se realiza, si procede, una actualización hacia arriba de los costes de los grafos solución parciales que cuelgan de los nodos antepasados del nodo expandido.

- **Ciclo 1**:

- Se expande n_1.

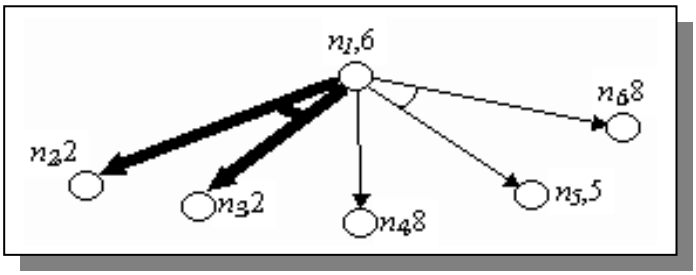
- La evolución del conjunto S es la siguiente:

1) $S = \{n_1\}$

2) n_1 es sacado de S. Su nuevo coste es 1+1+2+2=6 (igual que el anterior).

3) $S = \{\}$

- La situación final es la siguiente:



- **Ciclo 2**:

- Se expande n_2, aunque también podría haberse expandido n_3 (ver figura).

- Evolución del conjunto S:

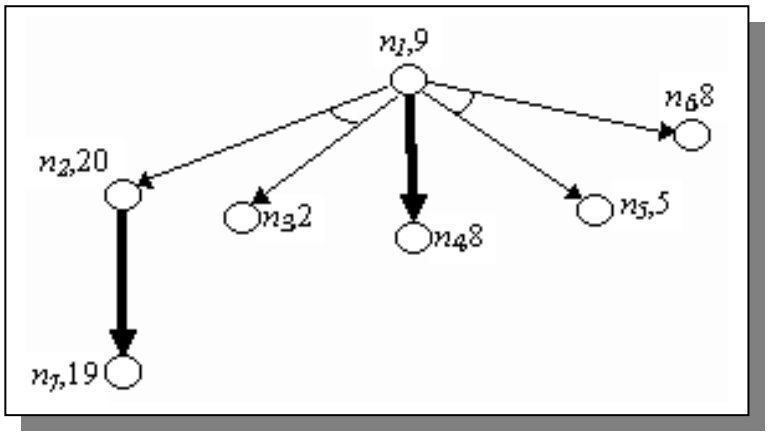
1) $S = \{n_2\}$

2) Se saca n_2 de S. Su nuevo coste es: 1+19=20. Al haber cambiado este coste respecto al antiguo, se introducen los predecesores de n_2 en S.

3) $S = \{n_1\}$

4) n_1 es sacado de S. Su nuevo coste es 9 por el enlace central.

5) $S = \{\}$



- Finalmente,

- **Ciclo 3**:

- Se expande n_4.

- Evolución de S:

1) $S = \{n_4\}$

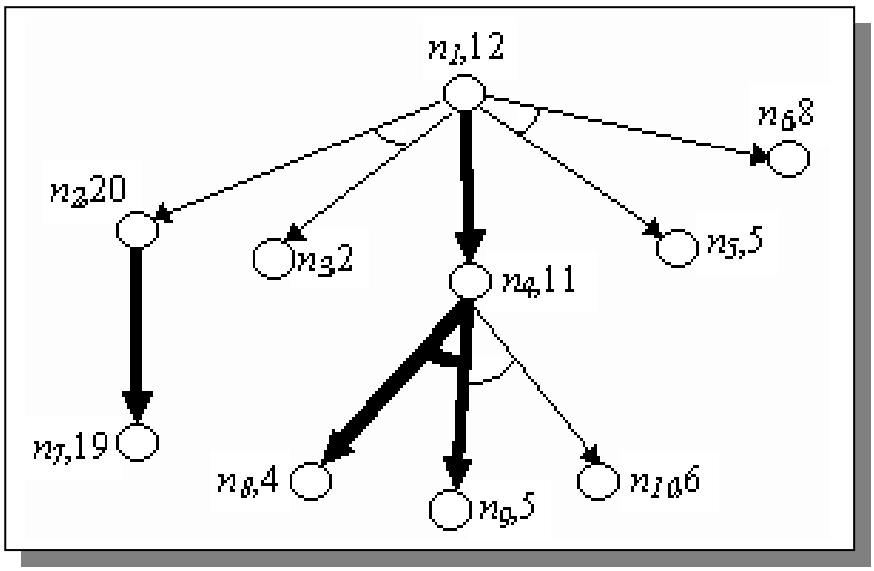
2) Nuevo coste de n_4: 1+1+4+5=11

3) $S = \{n_1\}$

4) Nuevo coste de n_1: 1+11=12

5) $S = \{\}$

- Al final de este ciclo:



• **Ciclo 4**:

- Se expande n_8, aunque también se podría haber expandido n_9.

- Evolución de S:

1) $S = \{n_8\}$

2) Nuevo coste de n_8: 1+10=11

3) $S = \{n_4\}$

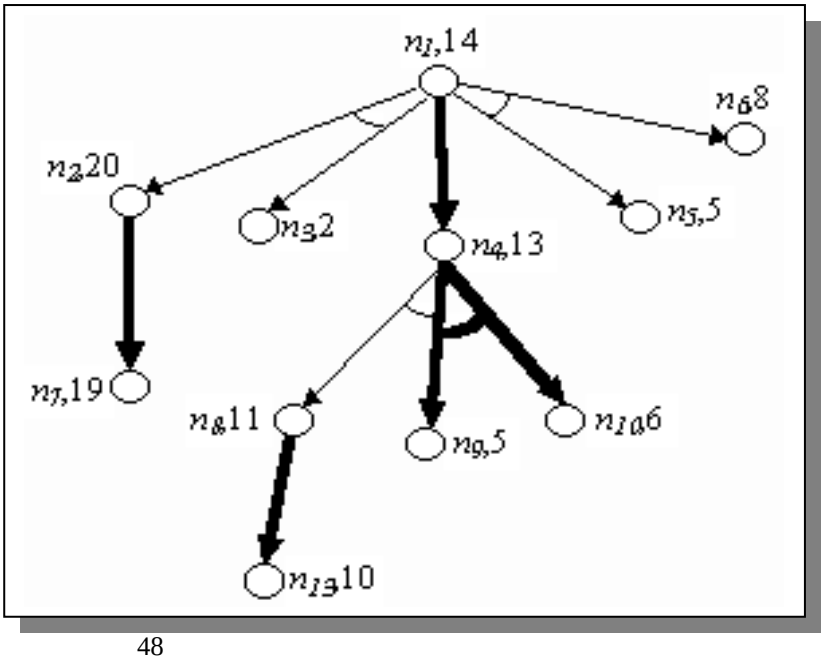
4) Nuevo coste de n_4: 1+1+5+6=13 por el enlace Y de la derecha.

5) $S = \{n_1\}$

6) Nuevo coste de n_1: 1+13=14

7) $S = \{\}$

- Finalmente,



• **Ciclo 5**:

- Se expande n_{10}, aunque también se podría haber expandido n_9.

- Evolución de S:

1) $S = \{n_{10}\}$

2) Nuevo coste de n_{10}: 1+12=13

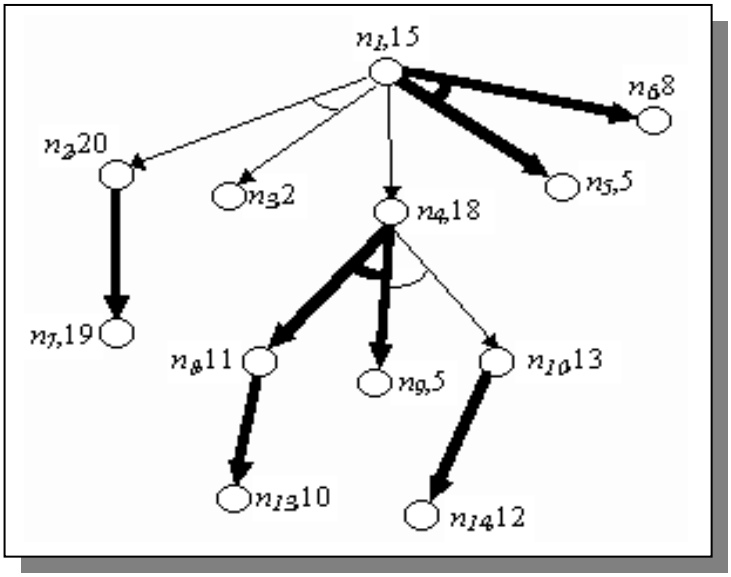
3) $S = \{n_4\}$

4) Nuevo coste de n_4: 1+1+11+5=18 por el enlace Y de la izquierda.

5) $S = \{n_1\}$

6) Nuevo coste de n_1: 1+1+5+8=15 por el enlace Y de la derecha.

7) $S = \{\}$



- Finalmente,

- **Ciclo 6**:

  - Se expande n_5, aunque n_6 podría haber sido el nodo elegido para su expansión.

  - Evolución de S a lo largo del ciclo:

    1) $S = \{n_5\}$

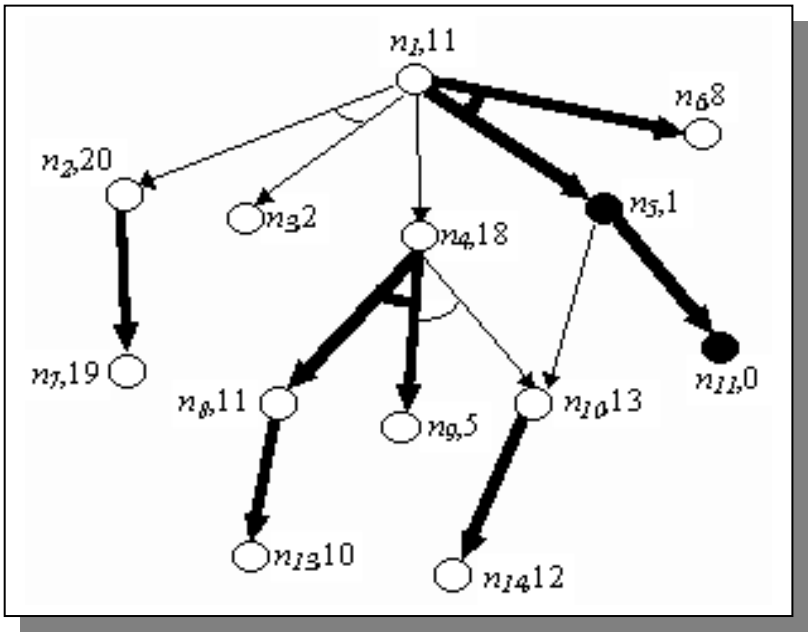
    2) Nuevo coste de n_5: 0+1=1, quedando resuelto este nodo.

    3) $S = \{n_1\}$

    4) Nuevo coste de n_1: 1+1+1+8=11.

    5) S = {}

- Finalmente,



• **Ciclo 7**:

- En esta ocasión se expande n_6.

- Evolución de S a lo largo del ciclo:

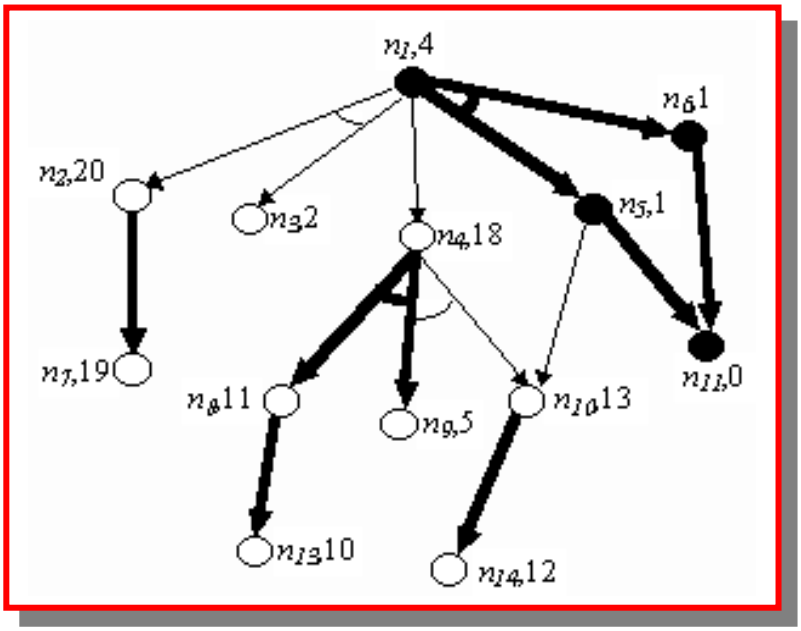
1) $S = \{n_6\}$

2) Nuevo coste de n_6: 0+1=1, quedando este nodo resuelto.

3) $S = \{n_1\}$

4) Nuevo coste de n_1: 1+1+1+1=4, quedando n_1 resuelto.

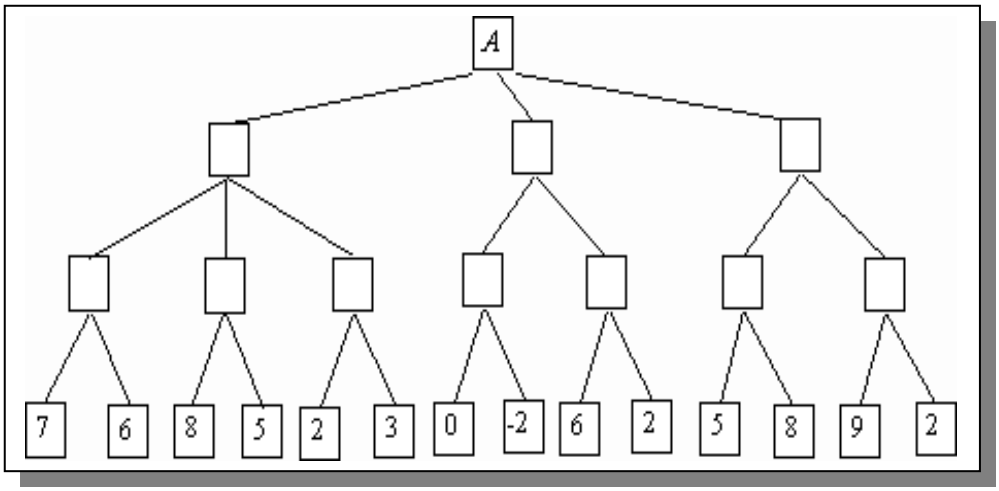
- Finalmente,



con lo que el nodo inicial queda resuelto y el algoritmo finaliza. El subárbol solución encontrado es el indicado por las líneas gruesas a partir de n_1.

• PROBLEMA 4.3:

Considere el siguiente árbol:



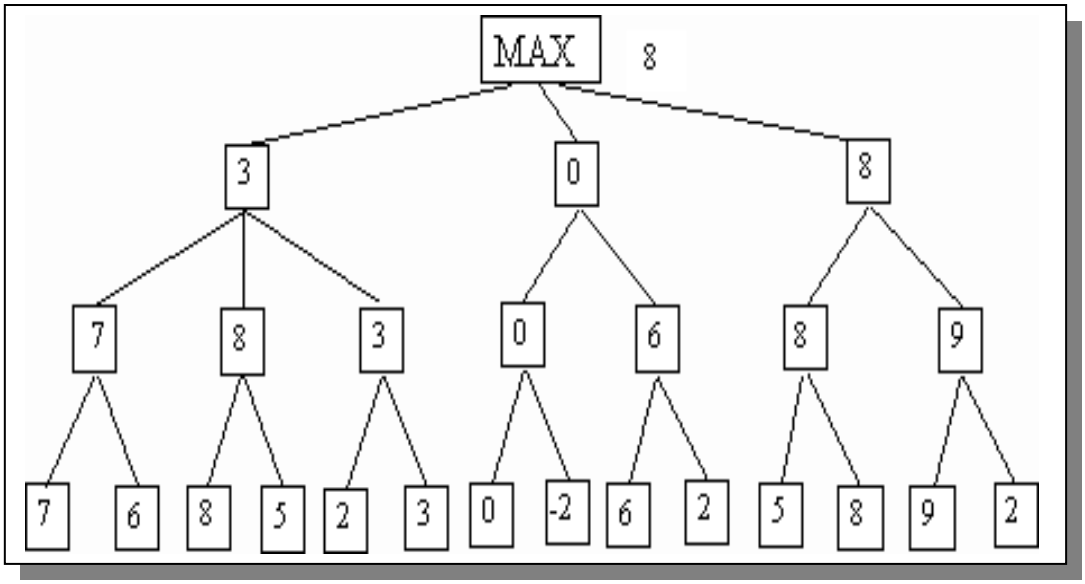
Suponiendo que A es el jugador MAX:

🕮 **a) ¿Qué movimiento debería escoger?**

🕮 **b) En el árbol anterior, ¿qué nodos tendrían que ser examinados usando la estra-tegia de poda α –β ?**

SOLUCIÓN:

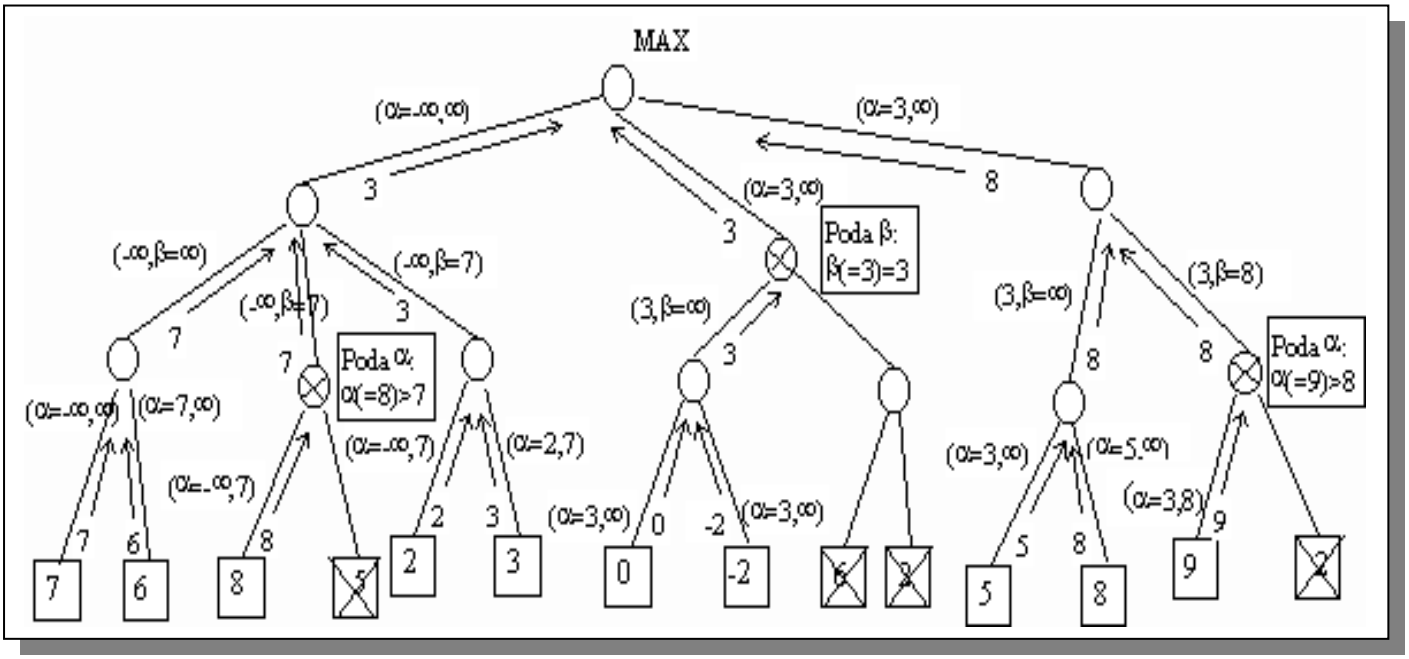
a) El método MINIMAX realizaría un recorrido en profundidad del árbol, seleccionando en cada momento los siguientes valores:



Se debería escoger, por tanto, el camino de la derecha para poder llegar al nodo terminal de valor 8.

-o-o-o-o-o-o-

b)



Tras haber recorrido el árbol de izquierda a derecha, de nuevo se comprueba que el mejor camino es el de la derecha. Por este camino se podrá acceder a un nodo terminal de valor 8.

5 LÓGICA

La lógica tiene como objeto de estudio los procesos de razonamiento expresados a través del lenguaje.

5.1 El uso de la lógica en la representación del conocimiento.

El formalismo elegido para la representación de los problemas es una de las principales áreas de investigación en la IA. La riqueza expresiva de los formalismos de representación del conocimiento y su realización computacional determinan la *eficiencia* y la *corrección* de las soluciones obtenidas.

Las tres formas más generales de representar el conocimiento en inteligencia artificial son:

(1) las *estructuras orientadas a la representación de los objetos/situaciones* mediante sus características. (*marcos*)
(2) las *fórmulas lógicas.*
(3) las *reglas de producción.*

La combinación de estas técnicas en lugar de la utilización exclusiva de una de ellas, es la opción más efectiva en la solución de problemas en inteligencia artificial. Es evidente, porque por ejemplo, no todo el razonamiento humano es de tipo lógico.

Para valorar la eficiencia de un método se debe de tener en cuenta dos capacidades del mismo: la de representar todo el conocimiento requerido y la de disponer de un método de obtención de las conclusiones requeridas por el conocimiento (capacidad de *razonamiento* o de *inferencia*). La utilización de la lógica como formalismo de representación cubre ambos aspectos. La utilización de la lógica en la representación del conocimiento (**RC**) tiene como ventajas: su riqueza expresiva y su semántica perfectamente definida.

5.2 Lógica de proposiciones.

La lógica de enunciados, de *proposiciones o cálculo proposicional* permite estudiar la validez formal de los razonamientos realizados mediante frases formadas por enunciados simples (**proposiciones**) que tiene un significado sobre el que es posible pronunciarse y que puede ser verdadero (*V*) o falso (*F*), pero no ambos a la vez.

Los símbolos básicos utilizados para representar los enunciados simples son las llamadas *variables proposiciones* (p, q, r, etc.).

Los enunciados en los que aparecen varias proposiciones (enunciados compuestos) se representan enlazando las variables llamadas conectivas (ver tabla). Cada conectiva representa varias expresiones del lenguaje natural:

- **p ∨ q** indica p o q o ambos a la vez.
- **p ⊕ q** indica p o q pero no ambos a la vez.
- **p ∧ q** indica p y q, p sin embargo q, p a pesar de q, etc.
- **p → q** se puede asociar a si p entonces q, p sólo si q, q si p, p es suficiente para q, q es necesario para p, ect.
- **p ↔ q** indica p si y sólo si q.

Las expresiones así formadas se conocen como *fórmulas correctamente formadas (fcf).* Una fórmula cualquiera puede ser:

- ***Válida***: cierta en todas las interpretaciones posibles (**tautologías**).
- ***Satisfactible***: cierta en al menos una interpretación.
- ***Insatisfactible***: falsa en todas las interpretaciones (**contradicciones**).

p	q	¬p	p ∧ q	p ∨ q	p ⊕ q	p → q	p ↔ q
V	V	F	V	V	F	V	V
V	F	F	F	V	V	F	F
F	V	V	F	V	V	V	F
F	F	V	F	F	F	V	V

Una *demostración en el sistema axiomático* consiste en, dadas una serie de fórmulas válidas de las cuales se parte (llamadas *axiomas*), obtener un conjunto de nuevas fórmulas igualmente válidas aplicando para ello las denominadas *reglas de transformación* (sucesivamente hasta que se genere la conclusión buscada).

La regla de transformación básica de los procesos deductivos de la lógica clásica es el ***modus ponens***, "*si S y S → R son fórmulas válidas, entonces R también el una fórmula válida*".

Además del *modus ponens*, existen otra serie de reglas que muestran los procesos de razonamiento del lenguaje ordinario (*deducción natural*). La diferencia con el *método axiomático* es que las inferencias aplicadas para poder avanzar en las secuencias deductivas de éste requerían que tanto las fórmulas de las que se partía, como aquéllas que se generaban, fueran enunciados formalmente válidos (verdaderos en todos los casos). Sin embargo, en deducción natural los enunciados están indeterminados en su valor de verdad.

Ambos métodos son equivalentes en el sentido que en ambos son correctas las mismas estructuras deductivas.

En *deducción natural* existe una regla básica (equivalente al modus ponens) que se denomina ***regla de eliminación del condicional*** (RE →) y la ***regla de introducción del condicional*** (RI →).

Existe una definición semántica de *estructura deductiva correcta* (cuando no existe ninguna interpretación que satisfaga todas las premisas y no satisfaga la conclusión, por la tanto, la fórmula: $\mathbf{P_1 \wedge P_2 \wedge ...P_n \rightarrow C}$ es una tautología).

Para tener criterios que simplifiquen el proceso de construcción de la secuencia deductiva se puede realizar el planteamiento de los llamados ***métodos de deducción automática*** (aplicación de procedimientos que detectan si la negación de una fórmula es insatisfactible o lo que es lo mismo, si la fórmula es válida).

Para verificar si una fórmula es deducible, habrá que comprobar que si las premisas P_i son ciertas, entonces la conclusión C también lo sea, o por ***refutación***, demostrando que la fórmula ($P_1 \wedge P_2 \wedge ...P_n \rightarrow \neg C$) es una contradicción.

Ahora bien, para aplicar la regla de resolución a un conjunto de premisas, éstas tiene primero que ponerse en ***forma clausulada*** (conjunción de cláusulas, las cuales a su vez son disyunciones de variables, negadas o sin negar).

Procedimiento:

- Eliminar los condicionales y bicondicionales:
  $p \rightarrow q \equiv \neg p \vee q$
  $p \leftrightarrow q \equiv ((p \rightarrow q) \wedge (q \rightarrow p))$
- Introducir las negaciones, para que sólo afecten a las variables proposicionales.
- Pasar a forma clausulada aplicando la distributiva para la disyunción.

La sentencia obtenida está en forma clausulada y se puede simplificar:

- Si hay dos o más cláusulas idénticas se pueden eliminar, dejando sólo una.
- Si aparece una cláusula con un literal y su negación se puede eliminar.
- Si en una cláusula aparece el mismo literal varias veces, sólo hay que escribirlo una vez.

La regla de resolución se puede aplicar a dos premisas en forma de cláusulas denominadas generatrices, que contienen una variable proposicional común sin negar en una y negada en otra. La resolución consiste en construir otra cláusula, denominada resolvente, formada por la disyunción de todas las variables proposicionales de las generatrices menos la común. Una estrategia posible para validar un proceso deductivo es realizar un proceso exhaustivo de aplicación de la regla de resolución entre todas las cláusulas hasta encontrar la cláusula vacía.

El cálculo de proposiciones basado en la regla de resolución como única regla de inferencia cumple tres propiedades básicas de los formalismos lógicos: es *consistente*, *completo* y *decidible*. Pero, para elegir el formalismo de representación se nos presenta el problema de *granularidad* o *adecuación representacional* (que se pueda expresar todos los elementos relevantes para la solución del problema).

5.3 Lógica de predicados.

La lógica de predicados abarca a la de proposiciones. En el cálculo de predicados, las *propiedades* y los nombres de *relaciones* entre individuos se denominan *predicados*. Sí estos tienen un único argumento, se denominan *predicados monádicos*, y si tienen más *poliádicos* (*diádicos, triádicos, ...*).

En cuanto a los **cuantificadores** (∀ y ∃) realmente se puede utilizar sólo uno:

$\neg \forall x\ P(x) \leftrightarrow \exists x\ \neg P(x)$ $\qquad$ $\forall x\ \neg P(x) \leftrightarrow \neg \exists x\ P(x)$

$\neg \forall x\ \neg P(x) \leftrightarrow \exists x\ P(x)$ $\qquad$ $\neg \exists x\ \neg P(x) \leftrightarrow \forall x\ P(x)$

Para poder interpretar el cálculo de predicados tenemos que hacer referencia a individuos concretos, por lo tanto, para validar sentencias en el cálculo de predicados habrá que referirse a un determinado *universo de discurso*. Si consideramos que el universo de discurso es el conjunto {a, b, c} se tienen en cuenta las equivalencias siguientes:

$\forall x\ P(x) \leftrightarrow P(a) \wedge P(b) \wedge P(c)$ $\qquad$ $\exists x\ P(x) \leftrightarrow P(a) \vee P(b) \vee P(c)$

En el *cálculo de predicados* es necesario introducir nuevas *reglas de inferencia* para tratar convenientemente las expresiones cuantificadas. En todas las reglas de inferencia se presupone que:

- P es una variable metalingüística que representa cualquier predicado.
- $a_1, a_2, \ldots, a_n$ son símbolos cualesquiera de nombres de individuo (llamados *parámetros*).

- Regla de eliminación del cuantificador universal (RE ∀),

$$\frac{\forall x1, x2,\ldots,xn\ \ P(x1, x2,\ldots,xn)}{P(a1, a2,\ldots, an)}$$

Lo que se cumple para todos los individuos se cumple para un conjunto cualquiera de estos

- Regla de introducción del cuantificador universal (RI ∀)

$$\frac{P(a1, a2,\ldots, an)}{\forall x1, x2,\ldots,xn\ \ P(x1, x2,\ldots,xn)}$$

Los parámetros se consideran constantes cualesquiera y que no han surgido como resultado de aplicar (RE ∃).

- Regla de eliminación del cuantificador existencial (RE ∃)

$$\frac{\exists\, x1, x2,\ldots,xn\ \ P(x1, x2,\ldots,xn)}{P(a1, a2,\ldots, an)}$$

Si algún individuo cumple una determinada propiedad no nos da derecho a representar tal individuo por la constante a_i que nosotros elijamos por conveniencia.

- Regla de introducción del cuantificador existencial (RI ∃)

$$\frac{P(a1, a2,\ldots, an)}{\exists\, x1, x2,\ldots,xn\ \ P(x1, x2,\ldots,xn)}$$

si un individuo cumple una propiedad entonces se puede afirmar que hay alguno que cumple dicha propiedad.

Vamos a resumir algunas cuestiones relevantes en cuanto a la representación realizada en dicho formalismo:

1. El cuantificador universal no implica necesariamente la noción de existencia.
2. El símbolo " $\vee$ " tiene carácter inclusivo, por consiguiente, el sentido exclusivo de una expresión se tendría que representar explícitamente.
3. Un problema muy importante es la elección de la *granularidad* de los enunciados, es decir su nivel de detalle o su capacidad expresiva.
4. Puede existir la necesidad de cuantificar los predicados. Para ello se han definido *lógicas de predicados de* ***orden superior*** que permiten cuantificar las funciones.
5. Un problema inherente a la lógica de predicados es que la robustez del razonamiento realizado se basa en la suposición que se ha representado todo el conocimiento necesario en el dominio.
6. Un predicado determina el conjunto de elementos que cumplen la propiedad especificada, sin embargo, la imprecisión es un elemento intrínseco al lenguaje natural. Para poder representar expresiones imprecisas se utiliza la ***lógica difusa***.

5.3.1 Ampliaciones de la Lógica de predicados.

LÓGICA DE PREDICADOS DE ORDEN SUPERIOR:

Es necesario introducir la posibilidad de representar dominios de referencia tanto para predicados como para funciones, entonces, los nombres de predicados y de función tienen las mismas atribuciones que los nombres de variable, es decir que pueden ser cuantificados. Este tipo de lógica recibe el nombre de *cálculo de predicados de segundo orden*, así como de tercer orden y todas se agrupan bajo el nombre de *lógicas de orden superior*.

Ejemplo: "todos los mamíferos tienen una Propiedad común"

$$\exists P\ \forall x\ (Mamífero(x) \rightarrow P(x))$$

Hay que tener en cuenta que el cálculo de predicados sólo es *decidible* si los predicados son monádicos y bajo ciertas restricciones en el caso de los predicados poliádicos.

LÓGICA DE PREDICADOS CON IDENTIDAD:

Supone la ampliación de la lógica de predicados tradicional con el predicado diádico " **=** " , de manera que si se escribe **x=y**, lo que se quiere expresar es que "x" e "y" se refieren al mismo elemento.

Algunos ejemplos de la utilización de este nuevo predicado podrían ser:

1. "Alguien que no es Roberto es rico."

$\exists\ x\ (x \neq r \wedge Rx)$

donde "r" representa "Roberto", "R" representa "ser rico" y ≠ equivale a la negación lógica de la identidad.

2. "Sólo Roberto es rico."

$Rr \wedge \neg\ \exists\ x\ (x \neq r \wedge Rx)$

3. "Por lo menos dos personas son ricas."

$\exists\ x\ \exists\ y\ (x \neq y \wedge Rx \wedge Ry)$

4. "Exactamente dos personas son ricas."

$\exists\ x\ \exists\ y\ ((x \neq y \wedge Rx \wedge Ry) \wedge \neg\ \exists\ z\ (z \neq x \wedge z \neq y \wedge Rz))$

Se necesitan dos reglas nuevas cuando se amplía la lógica de predicados con la identidad:

1. **regla a):** a = a siempre es cierto, sea cual sea la constante "a".
2. **regla b):** Fa, a = b → Fb para cualesquiera dos constantes "a" y "b".

Por ejemplo:

Cj "Juan es ciego."
j = a "Juan es tu mejor amigo."
∴ Ca Por tanto, "Tu mejor amigo es ciego.", donde
∴ representa "se infiere" o "se deduce".

Las expresiones de igualdad o de identidad tienen una serie de peculiaridades dentro del cálculo de predicados que estamos analizando. Por lo tanto, la l*ógica de predicados con identidad* es una extensión de la lógica de predicados. Por ejemplo, la ley de *indiscernibilidad* de los idénticos es:

$$\forall x\ \forall y\ ((x = y) \rightarrow (P(x) \leftrightarrow P(y))$$

5.4 Deducción automática: Resolución

La resolución es un método sistemático de validación de fórmulas que pertenecen al cálculo de predicados de primer orden. Este procedimiento puede no terminar, pero a pesar de ello, existe un número importante de fórmulas *decidibles*, lo que permite enmarcarlo dentro de los métodos de *deducción automática*. Estas técnicas son aplicables a la verificación formal de programas y sirven de base para la formulación de intérpretes de sistemas de programación.

5.4.1 Forma clausulada.

Una condición básica para poder aplicar el método de resolución (*resolución binaria*) es que las premisas y la conclusión estén en **forma clausulada** (*forma normal de Skolem o forma normal conjuntiva*) que se traduce con las siguientes condiciones:

- ☞ Todos los cuantificadores están a la cabeza de la fórmula.
- ☞ Sólo existen cuantificadores universales.
- ☞ El resto de la fórmula (todo lo que no es *cabeza*) se denomina *matriz* y está formado por una conjunción de fórmulas, cada una de las cuales está constituida a su vez por una disyunción de fórmulas atómicas, negadas o sin negar.

PROCEDIMIENTO DE CONVERSIÓN DE SENTENCIAS EN FORMA CLAUSULADA:

1. Eliminación de $\rightarrow$ y $\leftrightarrow$:

$(P \rightarrow S) \leftrightarrow (\neg P \vee S)$

$(P \leftrightarrow S) \leftrightarrow (P \rightarrow S) \wedge (S \rightarrow P)$

2. Eliminación de $\neg$ de las fórmulas compuestas:

2.1 Utilizar las leyes de De Morgan:

$\neg(P \vee S) \leftrightarrow \neg P \wedge \neg S$

$\neg(P \vee S) \leftrightarrow \neg P \vee \neg S$

2.2 Aplicar la ley de la doble negación:

$\neg\neg P \leftrightarrow P$

2.3 Supresión de las negaciones en las fórmulas con cuantificadores:

$\neg \forall x\ P(x) \leftrightarrow \exists x\ \neg P(x)$ $\qquad$ $\forall x\ \neg P(x) \leftrightarrow \neg \exists x\ P(x)$

$\neg \forall x\ \neg P(x) \leftrightarrow \exists x\ P(x)$ $\qquad$ $\neg \exists x\ \neg P(x) \leftrightarrow \forall x\ P(x)$

3. Cambio de variable en las fórmulas cuantificadas para que cada una tenga un nombre de variable distinto:

$\forall x\ P(x) \leftrightarrow \forall y\ P(y)$ $\qquad$ $\exists x\ P(x) \leftrightarrow \exists y\ P(y)$

En lugar de cambiar siempre el nombre de las variables iguales se pueden aplicar también las siguientes reglas:

$\forall x\ (R(x) \wedge P(x)) \leftrightarrow \forall x\ R(x) \wedge \forall x\ P(x)$ $\qquad$ *ley de distribución del $\forall$ por la $\wedge$*

$\exists x\ (R(x) \vee P(x)) \leftrightarrow \exists x\ R(x) \vee \exists x\ P(x)$ $\qquad$ *ley de distribución del $\exists$ por la $\vee$*

NO SON VÁLIDAS:

$\forall x\ (R(x) \vee P(x)) \rightarrow \forall x\ R(x) \vee \forall x\ P(x)$ $\qquad$ *ley de distribución del $\forall$ por la $\vee$*

$\exists x\ (R(x) \wedge P(x)) \rightarrow \exists x\ R(x) \wedge \exists x\ P(x)$ $\qquad$ *ley de distribución del $\exists$ por la $\wedge$*

4. Colocar todos los cuantificadores a la *cabeza* de la fórmula. Para ello se aplican las siguientes reglas:

$R \vee \forall x\ P(x) \leftrightarrow \forall x\ (R \vee P(x))$ $\qquad$ $R \wedge \forall x\ P(x) \leftrightarrow \forall x\ (R \wedge P(x))$

$R(x) \vee \forall y\ P(y) \leftrightarrow \forall y\ (R(x) \vee P(y))$ $\qquad$ $R(x) \wedge \forall y\ P(y) \leftrightarrow \forall y\ (R(x) \wedge P(y))$

$R \vee \exists x\ P(x) \leftrightarrow \exists x\ (R \vee P(x))$ $\qquad$ $R \wedge \exists x\ P(x) \leftrightarrow \exists x\ (R \wedge P(x))$

$R(x) \vee \exists y\ P(y) \leftrightarrow \exists y\ (R(x) \vee P(y))$ $\qquad$ $R(x) \wedge \exists y\ P(y) \leftrightarrow \exists y\ (R(x) \wedge P(y))$

5. Supresión de los cuantificadores existenciales, actuando según uno de los dos casos siguientes posibles:

5.1 Si el $\exists$ no tiene $\forall$ a su izquierda (no se encuentra bajo su alcance) se elimina el $\exists$ y se introduce una *constante*. Por ejemplo:

$\exists x\ \forall y\ \forall z\ \forall u\ (((R(x) \vee P(x, u)) \wedge S(x, y, z))$

se transformaría en:

$\forall y\ \forall z\ \forall u\ (((R(a) \vee P(a, u)) \wedge S(a, y, z))$ estas constantes se denominan ***constantes de Skolem***.

5.2 si el ∃ tiene uno o varios ∀ a su izquierda, se elimina la variable afectada por el ∃ y se introduce en su lugar una ***función de Skolem***, por ejemplo:

$\forall y\ \forall z\ \exists x\ (R(x) \vee P(x, y, z))$

se transformaría en:

$\forall y\ \forall z\ (R(g(y, z)) \vee P(g(y, z), y, z))$

6. Eliminar todos los ∀.

7. Convertir la fórmula en una conjunción de disyunciones con ayuda de la propiedad distributiva:

$R \vee (P \wedge S) \leftrightarrow (R \vee P) \wedge (R \vee S)$

8. Considerar cada una de las disyunciones del punto 7 como una cláusula aparte; algunas de ellas puede estar formada por un único literal.

9. Diferenciar el nombre de las variables entre las distintas cláusulas surgidas en el punto 8, de tal forma que no haya dos que hagan referencia al mismo nombre de variable. Para ello se cambian los nombres de dichas variables.

5.4.2 Unificación.

Uno de los procesos básicos más comunes en los problemas de inteligencia artificial es la denominada ***equiparación*** *o equiparación de patrones* (poner en relación de igualdad dos cosas). En la lógica de predicados de primer orden consiste en comparar predicados con el mismo nombre y en comprobar si sus términos (argumentos) se pueden *unificar* (sustituirlos para hacerlos idénticos). Por ejemplo:

$$\left.\begin{matrix} P(u, b) \\ P(f(x), v) \end{matrix}\right\rangle s_1 : \{f(x)/u, b/v\} \Rightarrow P(u, b)s_1 = P(f(x), v)s_1 = P(f(x), b)$$

La sustitución S_1, se puede entender como: donde haya una *u* poner *f(x)'* y ' donde haya *v* una poner b';

Las dos expresiones del predicado P han podido unificarse teniendo en cuenta las consideraciones siguientes:

1. *u* y *f(x)* ocupan el primer lugar o plaza de los argumentos del predicado P, además en *f(x)* no aparece *u*, por tanto puede crearse el par *{f(x)/u}*.
2. *b* y *v* ocupan el segundo lugar, siendo *b* una constante, por tanto puede crearse el par *{b/v}*.

Se ha definido implícitamente la operación de *composición de sustituciones o producto*. Otro método de resolución requiere que la unificación entre los predicados de las cláusulas se realice utilizando el denominado *unificador mínimo o de máxima generalidad (UMG)*. El UMG se define como aquél que cumple la propiedad de que cualquier otro unificador puede obtenerse a partir de él por aplicación de alguna otra sustitución. Este unificador contiene el menor número de pares y por tanto deja un mayor número de variables sin sustituir. Para crear un UMG se resuelven sucesivamente las discordancias entre los términos que ocupan las mismas plazas entre los predicados que están siendo unificados.

5.4.3 Método general de resolución.

Establece un procedimiento sistemático (computable) de búsqueda de la demostración de un teorema, la demostración se realiza siguiendo el método lógico de refutación, es decir, demostrar que si el teorema es falso se llega a una contradicción.

La regla de inferencia consiste en poner en forma clausulada las premisas y la negación de la conclusión, seleccionar parejas de literales en cláusulas distintas *(generatrice*s) con el mismo nombre, uno negado y otro sin negar, compararlas (mediante su *unificador mínimo*) y si existe la unificación aplicar la regla básica de resolución que consiste en crear una nueva cláusula *(resolvent*e) formada por la disyunción del resto de los literales de las *generatrice*s. Dicha cláusula se añade al conjunto de cláusulas existente y, tomando aquél como referencia, se intenta aplicar de nuevo el proceso. La búsqueda termina cuando la cláusula está vacía.

PROCEDIMIENTO GENERAL DE RESOLUCIÓN:

1. Convertir todas las premisas P_i en su forma clausulada (FNC) e *inicializar* un conjunto de cláusulas P con dichas expresiones.

2. Convertir la expresión resultante de negar la conclusión ¬C en forma clausulada y añadirla al conjunto de cláusulas P.

3. Realizar los puntos 3.1 y 3.2 hasta encontrar la cláusula vacía o hasta que no se pueda seguir avanzando (no se pueda generar ninguna resolvente nueva en 3.2) o hasta que se hayan agotado los recursos de cálculo (en tiempo o en memoria fijados inicialmente).

3.1 Inicializar, asignar un valor de referencia, por ejemplo *nulo*, a la variable que contendrá el unificador mínimo encontrado *s* (ver 3.2).

3.2 Seleccionar parejas nuevas (no elegidas en etapas anteriores) de *átomos* (literales negados o sin negar) con el mismo nombre de predicado en cláusulas distintas. Denominar dichas cláusulas G_1 y G_2 (se analizará si pueden ser *generatrices*) y a la pareja de átomos seleccionada A_1 y A_2 respectivamente.

(1) Buscar el *unificador de máxima generalidad de* A_1 y A_2. Si se encuentra, guardar su valor en *s*.

(2) Si *s* tiene un nuevo valor aplicar dicha sustitución a G_1 y G_2 formando su *resolvente* de la siguiente forma:

(2.1) Si existe en G_1s un elemento de A_1 y en G_2s un elemento de $\neg A_2$; o en G_1s un elemento de $\neg A_1$ y en G_2s un elemento de A_2; crear una nueva cláusula (*resolvente*) formada por la disyunción del resto de los elementos de G_1s y G_2s (no se deben eliminar nuevas ocurrencias de la contradicción encontrada entre el resto de elementos; por ejemplo, si hay algún otro A_1 y $\neg A_2$, éstos sí aparecerán en la *resolvente*.

(2.2) Si la nueva *resolvente* está vacía (las generatrices sólo tenían el literal seleccionado) salir del bucle en el paso 3 (se ha encontrado la contradicción buscada). En caso contrario añadir la nueva *resolvente* al conjunto P.

Observación: La clave de la eficiencia del proceso de búsqueda realizado está en cuál sea el criterio para seleccionar las cláusulas *generatrices* en el paso 3.2 De las diferentes formas de hacerlo surgen las llamadas *estrategias de resolución*. Cuando el número de cláusulas es elevado la búsqueda ciega en el espacio de cláusulas puede llegar a ser intratable.

ESTRATEGIAS DE RESOLUCIÓN:

Es útil analizar el proceso de resolución como una búsqueda en un espacio de cláusulas donde en cada momento se tiene un grafo de cláusulas explícito, en el que el objetivo es encontrar la cláusula vacía. Las estrategias básicas que permiten acelerar el procedimiento de resolución son:

1. La estrategia que se propuso inicialmente consiste en seleccionar aquellas cláusulas que tengan un único literal. La cláusula vacía λ se obtiene cuando se resuelven dos cláusulas unitarias contradictoras.

2. Considerar alguna cláusula para guiar el proceso de búsqueda de modo que se use ella o alguna de sus descendientes en la generación de nuevas resolventes. Un planteamiento efectivo consiste en hacer que la cláusula, o conjunto de cláusulas elegido contenga hipótesis relativas al teorema a resolver, y de las obtenidas de negar la conclusión (*encadenamiento hacia atrás*, o *dirigido por las metas*).

3.Las estrategias llamadas de *simplificación* reducen el número y la forma de las cláusulas consideradas. Pueden distinguirse las siguientes:

(1) eliminar las tautologías, ya que el objetivo es llegar a obtener λ.
(2) eliminar átomos repetidos en las cláusulas, dejando sólo uno de ellos.
(3) eliminar cláusulas que contienen a otras.

5.5 Extensiones de la lógica clásica.

Las *lógicas clásicas* responden a una serie de supuestos utilizados en su construcción. Algunos de éstos son:

- ☞ no considerar enunciados de los que no se pueda afirmar su verdad o falsedad.
- ☞ no admitir más que dos valores de verdad.
- ☞ operar *extensionalmente* en las expresiones consideradas.

La operación en extenso (individuos abarcados en una idea) o en intenso (serie de reglas para comprobar si un individuo está abarcado o no por un concepto) se refiere a si se considera exclusivamente los elementos que forman un determinado conjunto y su valor de verdad (*extensión*) o si se tiene el concepto que designa (*intensión*: en un predicado sería la propiedad significada por el nombre del predicado), es decir, desde el punto de vista intensional dos predicados pueden ser diferentes aun cuando

estén definidos sobre los mismos elementos del dominio. Por lo tanto, la teoría intensional de tipos estará basada en la distinción entre predicados y clases. La distinción entre *intensión* y *extensión* es fundamental a la hora de valorar la capacidad de cualquier sistema computacional.

5.5.1 Lógica modal.

Surge para satisfacer la necesidad de manipular los conceptos de *necesidad* y *posibilidad*. Se introducen los siguientes símbolos:

■ P	es necesario que P
◆ P	es posible que P

estos símbolos tienen el mismo nivel que el operador ¬. Así, se cumple la siguiente ley de equivalencia:

◆ P ↔ ¬ ■ ¬ P	es posible que P ≡ no es necesario que no-P
■ ¬ P ↔ ¬ ◆ P	es necesario que no-P ≡ es imposible que P
R ⇒ S ≡ ¬ ◆ (R ∧ ¬S)	implicación en sentido estricto

Un objetivo de la lógica modal es llegar a expresar la relación de necesidad entre premisas y conclusiones en el razonamiento deductivo.

"■A" es un concepto más fuerte que "A es cierto" y significa "A *tiene que ser* cierto". Por otro lado, "◆A" es más débil que "A es cierto" y significa: "A *podría ser* cierto". Para poder realizar la interpretación de fórmulas modales se introduce la idea de **mundos** (*dominios parciales*). Los mundos son subdivisiones del universo del discurso, de forma que en un determinado mundo un predicado sea verdad y en otro falso. Se debe de considerar entonces una relación de accesibilidad entre los mundos. Un enunciado *necesario* sería aquél que es cierto en todos los mundos posibles. Un enunciado *verdadero* es aquél que es cierto en el *mundo real*. Finalmente, un enunciado *posible* es aquél que es cierto en algún posible mundo. Un enunciado *posible* puede o no ser cierto en el mundo real. Con cualquier fórmula válida (A), podemos considerar estas reglas:

■ A →A : Si A es necesario, entonces es cierto en cualquier posible mundo M que escojamos.

◆ A →A: Si A es posible, entonces es cierto en un mundo no especificado con anterioridad.

La mejor estrategia para probar un razonamiento en lógica modal consiste en intentar sacar los operadores modales fuera en cada enunciado y posteriormente hacerlos desaparecer especificando un mundo en el que el enunciado se cumple. Por otra parte, sólo se podrá usar una regla de inferencia dentro de un mundo dado. Si se tuviera "(A⊃ B)" y "A" en el mismo mundo, entonces se podría inferir "B", pero sólo en ese mundo.

5.5.2 Lógica difusa.

La lógica clásica no ofrece un marco adecuado para la representación de conocimiento de tipo impreciso y subjetivo. Además, no permite la realización de razonamientos donde la incertidumbre deba ser tenida en cuenta. Considérese, por ejemplo, la frase:

"Las personas altas generalmente son bastante pesadas."

Si nos encontramos con una persona que mida 1 metro y 70 centímetros, deberíamos poder decir algo a partir de este dato y el conocimiento representado en la frase anterior. Sin embargo, surgen una serie de preguntas:

- ¿Es alta una persona de 170 cm. de altura?
- ¿Cuál es el rango de pesos donde entran las personas *bastante pesadas*?
- Dada una persona considerada como *alta*, ¿cuándo se podrá decir que es bastante pesada? Es decir, ¿cuál es el efecto real del adverbio *generalmente* sobre el resto de la frase?

La lógica difusa intenta resolver las deficiencias que aparecen en la lógica clásica al abordar problemas de características similares a las mencionadas con anterioridad.

Esta lógica permite afirmar que los enunciados son más o menos verdaderos en un contexto determinado y más o menos falsos en otro contexto diferente. Para definir formalmente un conjunto difuso se define el valor de la llamada *función de pertenencia* que determina el grado de pertenencia de cada elemento del universo de discurso a dicho conjunto. La definición matemática de un conjunto difuso P es la siguiente:

$$P = \left\{ x \mid \mu_p(x), \forall x \in U \wedge \mu_p(x) \neq 0 \right\}$$

Los valores de verdad de los conjuntos difusos se toman como referencia en el intervalo [0, 1], la suma de los valores de verdad no está sujeta a ningún tipo de restricciones (no debe confundirse con el cálculo probabilístico).

Ejemplo Considérese el universo de edades *U*={0, 10, 20, 30, 40, 50, 60, 70, 80}. Se puede definir el conjunto borroso *C*, que representa el concepto "viejo", del siguiente modo:

C={0|0, 10|0, 20|0, 30|0'1, 40|0'2, 50|0'5, 60|0'9, 70|1, 80|1}

donde al lado de cada elemento (edad en años) aparece el grado de pertenencia del mismo al conjunto representado. A la función que asocia a cada elemento su grado de pertenencia se la llama *función de pertenencia*. Esta función da idea de la compatibilidad de cada elemento con el concepto representado.

• Operaciones definidas sobre conjuntos borrosos

Dados los conjuntos borrosos F y G cuyos elementos pertenecen a un universo U, se definen las siguientes operaciones:

a) INCLUSIÓN: $F \subset G$ si $\mu_F(x) \leq \mu_G(x)\ \forall\ x \in U$

b) IGUALDAD: $F = G$ si $\mu_F(x) = \mu_G(x)\ \forall\ x \in U$

c) DESIGUALDAD: $F \neq G$ si $\mu_F(x) \neq \mu_G(x)\ \forall\ x \in U$

d) COMPLEMENTO: $c(F) = \{x | \mu_{c(F)}(x) = 1 - \mu_F(x)\ \forall\ x \in U\}$

e) UNIÓN: $F \cup G = \{x | \mu_{F \cup G}(x) = \max(\mu_F(x), \mu_G(x))\ \forall\ x \in U\}$

f) INTERSECCIÓN: $F \cap G = \{x | \mu_{F \cap G}(x) = \min(\mu_F(x), \mu_G(x))\ \forall\ x \in U\}$

Dado un universo de discurso U y los conjuntos no borrosos X, Y y Z definidos en él, éstos cumplen las siguientes propiedades a partir de la teoría de conjuntos:

Involución $c(\ c(\ X\ )\ ) = X$

Conmutatividad $X \cup Y = Y \cup X,\ X \cap Y = Y \cap X$

Asociatividad $(X \cup Y) \cup Z = X \cup (Y \cup Z),\ (X \cap Y) \cap Z = X \cap (Y \cap Z)$

Distributividad $X \cap (Y \cup Z) = (X \cap Y) \cup (X \cap Z),\ X \cup (Y \cap Z) = (X \cup Y) \cap (X \cup Z)$

Idempotencia $X \cup X = X,\ X \cap X = X$

Absorción $X \cup (X \cap Y) = X,\ X \cap (X \cup Y) = X$

Absorción por U y ∅ $X \cup U = U,\ X \cap \varnothing = \varnothing$

Identidad $X \cup \varnothing = X,\ X \cap U = X$

Ley de contradicción $X \cap c(X) = \varnothing$

Ley del tercio excluso $X \cup c(X) = U$

Leyes de De Morgan $c(X \cap Y) = c(X) \cup c(Y),\ c(X \cup Y) = c(X) \cap c(Y)$

Para conjuntos difusos todas estas propiedades siguen cumpliéndose, excepto la de **contradicción** y la del **tercio excluso**. Por tanto, los conjuntos borrosos dotados de las operaciones de unión, intersección y complemento no pueden formar un álgebra de Boole.

• Modificadores lingüísticos

Los modificadores lingüísticos ("casi", "muy"...) se modelan en lógica borrosa a partir de determinadas operaciones sobre la función de pertenencia asociada al predicado que se está modificando. Algunas de las operaciones sugeridas para este fin son:

Tipo de operación	Representación	Cálculos realizados
Negación	NEG(μ (x))	$1-\mu(x)$
Concentración	CON(μ (x))	$\mu^2(x)$
Dilatación	DIL(μ (x))	$2\mu(x)-\mu^2(x)$
Intensificación	INT(μ (x))	$2\mu^2(x)$ si $0\leq\mu(x)\leq 0'5$ $1-2(1-\mu(x))^2$ si $\mu(x)>0'5$

de manera que se pueden definir modificadores difusos de la siguiente forma:

Modificador difuso	Operación asociada
"no"	NEG(μ (x))
"muy"	CON(μ (x))
"algo"	DIL(μ (x))
"casi"	DIL(DIL(μ (x)))
"bastante"	INT(CON(μ (x)))
...	...

• Inferencia en lógica difusa

Algunas de las reglas básicas de inferencia que se emplean en lógica difusa son las siguientes:

a) Principio de herencia: dado un conjunto borroso *A* asociado a un predicado, que es un subconjunto borroso de *B*, cualquier elemento con la propiedad *A* hereda la propiedad *B*.

$A(x)$
$\underline{A \subset B}$
$B(x)\ \forall\ x \in U, A \subset U, B \subset U$

Se dice que un conjunto borroso *A* es un *subconjunto* de otro conjunto borroso *B*, definidos sobre un mismo universo *U*, si el grado de pertenencia de cada elemento del universo al conjunto *A* es menor o igual que su grado de pertenencia al conjunto *B*:

$$A \subset B \text{ si } \mu_A(x) \leq \mu_B(x)\ \forall\ x \in U$$

b) Regla de composición: dado un elemento que cumple la propiedad representada por el conjunto borroso *A* y que está relacionado a través de una relación binaria *R* con otro elemento de un universo diferente, se le puede asociar a este segundo elemento la propiedad resultante de componer *A* y *R*.

$A(x)$
$\underline{R(x, y)}$
$(A \circ R)(y)\ \forall\ x \in U, \forall\ y \in V, A \subset U, R \subset U \times V$

c) *Modus Ponens* generalizado:

$A(x)$
$\underline{B(x) \rightarrow C(y)}$
$(A \circ (B \times C))(y)\ \forall\ x \in U, \forall\ y \in V, A \subset U, B \subset U, C \subset V$

siendo

$\mu_{A \circ (B \times C)}(y)=\max_{x \in U}[\min(\mu_A(x), \mu_{B \times C}(x, y))]=\max_{x \in U}[\min(\mu_A(x), \min(\mu_B(x), \mu_C(y)))]$

(Si "a" es el número de elementos del universo donde está definido el conjunto *A* y "b" es lo mismo, pero referente al conjunto *B*, la matriz correspondiente a $A \times B$ tendrá "a" filas y "b" columnas).

d) *Modus Tollens* generalizado:

$D(y)$
$\underline{B(x) \rightarrow C(y)}$
$(D \circ (C \times B))(x) \; \forall x \in U, \forall y \in V, D \subset V, B \subset U, C \subset V$

(Aunque aquí se ha incluido esta definición del *Modus Tollens* generalizado, hoy en día existe gran controversia sobre cómo debería ser definida esta regla. No existe un consenso general a tal respecto.)

e) Silogismo hipotético:

$A(x) \rightarrow B(y)$
$\underline{B(y) \rightarrow C(z)}$
$((A \times B) \circ (B \times C))(x, z) \; \forall x \in U, \forall y \in V, \forall z \in W, A \subset U, B \subset V, C \subset W$

5.5.3 Lógica no monótonas.

Una de las dificultades del cálculo de predicados es la excesiva rigidez a la que están sujetos los razonamientos realizados. Para que el conocimiento del dominio pueda ser efectivo y eficientemente formulado en forma de una teoría es necesario que ésta sea *completa*, *consistente* y *tratable*. Para garantizar la completitud toda fórmula válida debe ser demostrable sin omisiones. La consistencia supone que todo aquello que se añada al sistema no debe contradecir a las leyes existentes. Quizás el requerimiento de consistencia sea la limitación que más contrasta con las situaciones reales ya que la percepción de los problemas varían con el tiempo y pueden llevarnos a reconsiderar alguno de los principios iniciales. Esto también ocurre con el *razonamiento basado en el sentido común*, donde se aplican principios que se suponen válidos en el dominio, que pueden dejar de serlo en situaciones concretas.

Para solucionar estos problemas se plantea una formulación menos rígida, exigiéndose que las conclusiones sean consistentes con las condiciones de los problemas tratados y con las leyes aplicables a dichos problemas, entrando a considerar la temporalidad implícita de los razonamientos lógicos. La raíz del problema está en la falta de conocimiento inicial del dominio, en estas situaciones se crean reglas o leyes que permiten realizar un tipo de razonamiento denominado *razonamiento por defecto*. Una de las posibles soluciones a este tipo de razonamiento son las lógicas no monótonas, que tienen en cuenta lo que no se conoce.

Por ejemplo, si alguien nos dice que tiene un pájaro, nosotros enseguida pensaremos que ese pájaro vuela. Si posteriormente se nos dice que el pájaro es en realidad un pingüino, tendremos que dejar de pensar que puede volar y que seguramente su amo lo tenía encerrado en una jaula.

A lo largo de los últimos años han aparecido diversos formalismos cuyo objetivo es intentar servir de base al razonamiento no monótono. Entre otros podemos citar los siguientes:

- La *lógica por defecto* de Reiter.
- Los mecanismos de *circunscripción* de McCarthy.
- La *lógica modal no monótona* de McDermott y Doyle.
- La *lógica autoepistémica* de Moore.
- Las *redes de herencia* de Touretzky.

5.6 Conclusiones.

Podemos extraer las siguientes conclusiones generales:

- La lógica surge como intento de abstraer, formalizar y hacer calculables las propiedades inferenciales implícitas en el razonamiento expresado a través del lenguaje.
- Los diferentes cálculos lógicos intentan dar satisfacción a los distintos grados de complejidad expresiva e inferencial requerida en los problemas del mundo real.
- Existen métodos de inferencia sistemáticos para un número significativo de problemas.
- La efectividad de los métodos de inferencia depende del grado de conocimiento del dominio y la tarea.

Finalmente, señalaremos que, la complejidad inherente a muchos de los sistemas de inteligencia artificial hace aconsejable combinar las técnicas vistas con otras que veremos más adelante.

PROBLEMAS

PROBLEMA 5.1:

🕮 **Probar:**

Fa
$\therefore (\neg Fb \supset b \neq a)$

SOLUCIÓN:

1. $Fa[\therefore (\neg Fb \supset b \neq a)$
2. supóngase: $\neg (\neg Fb \supset b \neq a)$ es decir, negar la conclusión para demostrar por reducción al absurdo.
3. $\therefore \neg Fb$ (a partir de **2**), ya que $\neg (\neg Fb \supset b \neq a)$ es igual a $\neg (\neg\neg Fb \vee b \neq a)$, y esto, por De Demorgan, pasa a $(Fb \wedge b = a)$.
4. $\therefore b = a$ (a partir de **2**) Por lo mismo que antes.
5. $\therefore Fb$ (a partir de **1** y **4**, contradiciendo a **3**)
6. $\therefore$ **$(\neg Fb \supset b \neq a)$** (a partir de **2**, **3** y **5**)

• PROBLEMA 5.2:

Dado el siguiente razonamiento:
"**Necesariamente, si Juan es español, es europeo.**"
"**Es posible que Juan no sea europeo.**"
Por tanto, "Es posible que Juan no sea español."

🕮 **dar una prueba del mismo.**

SOLUCIÓN:

¿Qué pretende este problema? El presente problema presenta una demostración por reducción al absurdo de razonamientos donde figuran los operadores de necesidad y posibilidad. Se hace uso constante de las leyes que permiten ampliar la lógica de predicados tradicional a la lógica modal.

Considérese:

$A \equiv$ "Juan es español."
$B \equiv$ "Juan es europeo."

El problema, entonces, quedaría reducido a probar:

$\blacksquare (A \supset B)$
$\blacklozenge \neg B$
$\therefore \blacklozenge \neg A$

La prueba pedida sería:

1. $\blacksquare (A \supset B)$
2. $\blacklozenge \neg B$
$[\therefore \blacklozenge \neg A$
3. supóngase: $\neg \blacklozenge \neg A$

4. $\therefore \blacksquare A$ (a partir de **3** y de la *regla dada en apartado 5.5.1*)
5. $W \therefore \neg B$ (a partir de **2** y de la *regla dada en apartado 5.5.1*)
6. $W \therefore (A \supset B)$ (a partir de **1** y de la *regla dada en apartado 5.5.1*)
7. $W \therefore A$ (a partir de **4** y de la *regla dada en apartado 5.5.1*)
8. $W \therefore B$ (a partir de **6** y **7**, contradiciendo a **5**)
9. $\therefore \blacklozenge \neg A$ (a partir de **3**, **5** y **8**)

• PROBLEMA 5.3:

Sea U el universo de las posibles cantidades de dinero (en millones de pesetas) en que están valorados los bienes de una persona cualquiera. Sobre este universo se definen los siguientes conjuntos borrosos:

R ≡ "ser rico" = {0|0, 1|0, 5|0'1, 10|0'2, 50|0'5, 100|0'9, 500|1}
A ≡ "ser adinerado" = {0|0, 1|0'1, 5|0'5, 10|0'8, 50|1, 100|1, 500|1}

Por otra parte, sea V el universo de los posibles precios (en millones de pesetas) de un coche, en el que se considera el siguiente conjunto borroso:
C ≡ "ser un coche caro" = {0'5|0, 1|0, 2|0'5, 5|0'9, 10|1}

🕮 **Comprobar si a partir de las premisas:**
"Jorge es rico."
"Si Jorge es adinerado, se comprará un coche caro."

se puede concluir que
"Jorge se comprará un coche caro."

SOLUCIÓN:

¿Qué pretende este problema? En este problema se aplican a casos concretos los conceptos vistos en el apartado teórico sobre inferencia difusa.

Suponiendo que x representa los "bienes en millones de pesetas de Jorge" e y el "coste en millones de pesetas del coche que Jorge se va a comprar", habrá que comprobar si el siguiente razonamiento es correcto:

$$\begin{array}{l} R(x) \\ \underline{A(x) \rightarrow C(y)} \\ C(y) \;\forall\, x \in U,\ \forall y \in V \end{array}$$

Considerando la regla de Modus Ponens generalizado, a partir de las dos premisas del enunciado se puede deducir $(R \circ (A \times C))(y)$.

Por tanto, habrá que verificar que $R \circ (A \times C) \subset C$, ya que si esta condición se cumple, el principio de herencia garantiza la validez de C(y). Operando sobre A, R y C se obtienen los siguientes resultados:

A×C	*0'5*	*1*	*2*	*5*	*10*
0	0	0	0	0	0
1	0	0	0'1	0'1	0'1
5	0	0	0'5	0'5	0'5
10	0	0	0'5	0'8	0'8
50	0	0	0'5	0'9	1

100	0	0	0’5	0’9	1
500	0	0	0’5	0’9	1

De manera que para R o (*A* × *C*) se tiene:

$$(0 \quad 0 \quad 0'1 \quad 0'2 \quad 0'5 \quad 0'9 \quad 1) \circ \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0'1 & 0'1 & 0'1 \\ 0 & 0 & 0'5 & 0'5 & 0'5 \\ 0 & 0 & 0'5 & 0'8 & 0'8 \\ 0 & 0 & 0'5 & 0'9 & 1 \\ 0 & 0 & 0'5 & 0'9 & 1 \\ 0 & 0 & 0'5 & 0'9 & 1 \end{pmatrix} = (0 \quad 0 \quad 0'5 \quad 0'9 \quad 1)$$

Por tanto, el predicado conclusión resultante es el conjunto borroso:

$R \circ (A \times C) = \{0'5|0, 1|0, 2|0'5, 5|0'9, 10|1\}$

que coincide con el conjunto *C*, de manera que la conclusión:

"Jorge se comprará un coche caro."

es válida.

6 Reglas

Las reglas se utilizan para examinar un conjunto de datos y solicitar nueva información hasta llegar a un diagnóstico. Hoy en día las reglas, ampliadas con el uso de marcos y con algunos conceptos propios de las redes semánticas, constituyen el método estándar para la construcción de sistemas expertos.

6.1 Componentes básicos de los sistemas basados en reglas (SBRs).

Un sistema basado en reglas consta de cinco elementos básicos:

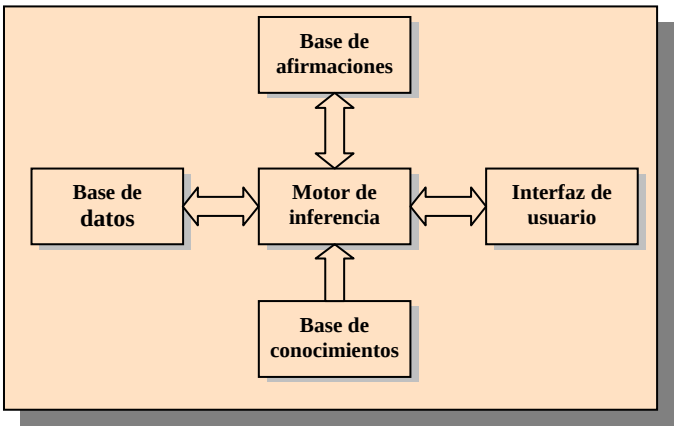
Motor de inferencia o intérprete de reglas:
Es el elemento central del sistema; se encarga de coordinar la información procedente de todos los demás y de enviar los resultados de la inferencia al lugar oportuno.

Base de conocimiento:
Contiene las reglas y, a veces, también algunas afirmaciones iniciales. Es específica del dominio en que el sistema puede considerarse experto (medicina, geología, diseño de circuitos...etc)

Base de afirmaciones o memoria de trabajo:
Contiene las afirmaciones iniciales, las que se extraen de la base de datos, las que el usuario introduce como datos y todas las que el motor de inferencia infiere a partir de las anteriores aplicando las reglas.

Interfaz de usuario:
Se encarga de solicitar al usuario la información necesaria y de mostrarle los resultados de la inferencia.

Base de datos:
Es necesaria para tener acceso eficiente a toda la información relativa a los casos anteriores.



En los sistemas convencionales, el motor de inferencia no modifica la base de conocimientos, por eso, en el esquema, dicha flecha es de un solo sentido. Si no fuera así, dicha flecha sería en ambos sentidos

Es importante señalar que el motor de inferencia es un programa de ordenador independiente del dominio de aplicación. Por tanto, es posible sustituir una base de conocimiento por otra diferente con la única condición de que tenga la misma sintaxis que la anterior.

6.2 Estructura de las reglas.

6.2.1 Antecedente y Consecuente

Una regla consta de dos partes:

1. ***Antecedente*** *o parte izquierda*: contiene las cláusulas que deben cumplirse para que la regla pueda evaluarse o ejecutarse.
2. ***Consecuente*** *o parte derecha*: indica las *conclusiones* que se deducen de las premisas (interpretación declarativa) o las *acciones* que el sistema debe realizar cuando ejecuta la regla (interpretación imperativa).

Los elementos que pueden intervenir en una regla son los siguientes:

- ***Hipótesis***: En un momento dado de una consulta, cada hipótesis tiene asociado un valor de verdad. Por ejemplo: *hielo-en-la-carretera*, *avería-eléctrica*, etc...
- ***Dato***: puede tomar cierto tipo de valores. En el caso de utilizar marcos, los datos pueden corresponder a los campos de las instancias. Por ejemplo: *nivel-de-gasolina*, *sexo-del-paciente*, etc...
- ***Relación de comparación***: se establece entre dos datos o entre un dato y un valor. Cada relación es cierta o falsa en un momento de la consulta, según la interpretación obvia del signo que determina la relación. Por ejemplo: *nivel-de-gasolina = 8*, *Temperatura-exterior < 0*, etc...
- ***Relación de pertenencia***: se establece entre una instancia y un marco. Una relación de pertenencia "*x* ES *y*" se considera cierta cuando *x* es una instancia del marco *y*. Por ejemplo: *Pedro-García* ES *paciente*, etc...
- ***Cláusula***: consiste en una hipótesis o una relación o bien en la negación, la conjunción o la disyunción de otras cláusulas.

Con respecto a las *conclusiones* que forman el *consecuente*, podemos considerar tres tipos de acciones:

1. **Afirmar**: Se establece una nueva relación de comparación o de pertenencia.
2. **Retractar**: Se invalida una relación que anteriormente fue afirmada.
3. **Actuar**: El sistema enviará una orden a los actuadores con los que está conectado.

Entre las distintas condiciones se considera implícita la conectiva Y. Además la palabra clave "AFIRMAR" puede omitirse, de modo que si en una de las conclusiones de la regla no aparece ninguna de las tres acciones mencionadas, se supone que hay "AFIRMAR" implícito.

Ejemplo de una regla:

SI *temperatura_exterior* > *temperatura_deseada* Y *temperatura_deseada* > *temperatura_exterior*
ENTONCES ACTUAR *encender_ventilador* Y AFIRMAR *ventilador_encendido*

6.2.2 Uso de variables en las reglas.

Nos interesa poder representar mediante reglas afirmaciones equivalentes a las que aparecen en la lógica de predicados, y para ello se hace necesario introducir *variables* en las reglas.

Una afirmación como "*Todos los catedráticos son doctores*", que en lógica de predicados se expresa como:

$$\forall x, \text{CATEDRÁTICO}(x) \Rightarrow \text{DOCTOR}(x)$$

dará lugar a la siguiente regla:

SI ?x ES catedrático ENTONCES ?x ES doctor

o bien a esta otra

SI ?x.categoría = catedrático ENTONCES ?x.titulación = doctor

En la primera, se interpreta cada predicado como la pertenencia a una clase (marco)
En la segunda, se interpreta cada predicado como una propiedad (o campo).
Conviene no confundir las *variables* (?x, ?y, ...) con los *datos* (Juan-López.categoría).

6.2.3 Comentarios

Faltaría determinar si las variables van a representar solamente *instancias* o si pueden representar también *clases* y *propiedades* (con lo que nos aproximaríamos a lógicas de orden superior). En este caso, el motor de inferencia que interpretase estas reglas debería ser mucho mayor.

Los sistemas expertos y las herramientas comerciales suelen utilizar una sintaxis rígida, con la función de tener motores de inferencia más eficientes. Aunque, introducen otros operadores para asignar variables dentro de la regla, para mostrar y solicitar información, para calcular expresiones matemáticas, para introducir comentarios, etc., que constituyen una ayuda valiosa en la práctica pero que alejan las reglas de la lógica de primer orden.

Una de las formas que más diferencia entre sí los sistemas basados en reglas es la forma de indicar la falsedad de una afirmación:

- NEXPERT y otros, indican explícitamente si una determinada hipótesis es cierta, falsa, no investigada o desconocida se ha tratado de averiguar su valor de verdad sin conseguirlo.
- PROLOG y otros, sólo almacenan las hipótesis confirmadas, lo que se conoce como *Axioma del mundo cerrado*: el considerar falsa toda proposición que no se encuentre en la base de afirmaciones ni pueda deducirse de la información disponible.
- GoldWorks y otros, son menos explícitos y el diseñador del programa tiene libertad para determinar si las va a considerar como desconocidas o como falsas.

Algunos sistemas (GoldWorks, p.e.) permiten indicar si los datos son **univaluados** o **multivaluados**:

- *Datos univaluados*: La asignación de un valor al dato elimina el que pudiera estar asignado anteriormente. Se evita información contradictoria. P.e. Pedro-García.edad = 63.
- *Datos multivaluados*: Hay casos en que no interesa que se pierda la información previa (Pedro-García.última-enfermedad). De este modo, la asignación de un valor al dato *no* elimina el que pudiera estar asignado anteriormente.

Algunas herramientas comerciales, al hablar de *variables* se refieren a ciertos registros de la base de afirmaciones, nosotros los llamamos *datos* para evitar confusiones.

6.3 Inferencia

6.3.1 Comparación de Patrones.

Se entiende por *patrón* un modelo que puede representar diferentes elementos. En el contexto de las reglas, este término se utiliza de forma restringida para denominar una *cláusula con variables* (equivale a un conjunto de afirmaciones. Por ejemplo, el patrón: "*?x ES profesor*" representa las afirmaciones "*Pedro-García ES profesor*", "*Antonio-Pérez ES profesor*"...etc). Una regla se ejecuta (se dispara) cuando se cumple su antecedente, y para ello es necesario que se cumplan cada una de las cláusulas que lo componen. Podemos tener tres situaciones:

1. ***Cláusulas sin variables****:* si consta de una hipótesis, se satisface cuando en la base de afirmaciones (**BA**) hay una afirmación que coincide con ella. Si se trata de una *relación*, se satisface cuando los valores de los datos se ajustan a ella.

```
SI hielo-en-la-carretera
   Velocidad >70
ENTONCES recomendación = reducir-la-velocidad.
```

2. Cuando ***una variable aparece por primera vez*** en una regla, la cláusula correspondiente se satisfará si y sólo si hay una asignación de valor a dicha variable tal que el patrón coincida con uno de los elementos existentes en la base de afirmaciones.

```
SI ?x ES catedrático
ENTONCES el-director-de-Dto-ha-de-ser-catedrático.
```

   En este caso, la regla se satisface si en la BA hay entre otras, asignaciones como:

```
Antonio-Pérez ES catedrático
Pedro-García ES catedrático
```

3. Cuando ***una variable aparece por segunda vez, o más***, en una regla, necesariamente tiene ya un valor que le fue asignado la primera vez que apareció. La cláusula se satisfará solamente si, al sustituir dicha variable por el valor asignado, la cláusula coincide con una de las afirmaciones de la base de afirmaciones.

La regla se ejecutará una vez por cada asignación de variables capaz de satisfacer todas sus cláusulas. En este caso se habla de diferentes *instanciaciones de una regla*.

6.3.2 Tipos de Encadenamientos.

Una conclusión obtenida a partir de una primera regla puede servir para ejecutar una segunda. Es decir, cuando una conclusión obtenida de una regla sirve para ejecutar otra regla. Sólo es aplicable cuando el consecuente de la primera regla es del tipo "AFIRMAR" y la variable tome el mismo valor en ambas reglas. (La nueva afirmación que se almacenará en la BA).

Por ejemplo:

```
SI director-del-dpto = ?x ENTONCES ?x.categoría = catedrático
SI ?x.categoría = catedrático ENTONCES ?x.titulación = doctor
```

Esto es lo que se conoce como *encadenamiento de reglas*, y puede ser básicamente de dos tipos:

- ***Encadenamiento hacia delante*** *o basado en datos:* cuando la información introducida en el sistema hace que se ejecute una regla, y la conclusión obtenida permite que se ejecuten otras reglas. Utiliza solamente los datos disponibles y es menos específico que el encadenamiento hacia atrás ya que ejecutará todas las reglas posibles en función de la información introducida.
- ***Encadenamiento hacia atrás*** *o basado en objetivos:* consiste en buscar una regla que permita establecer cierta conclusión y suele solicitar al usuario la información que no ha podido deducir. Si la afirmación existe en la BA se podrá realizar la inferencia. Si no, se puede buscar una regla que contenga en su consecuente la anterior afirmación y ver si sí existe en la BA su antecedente, y así sucesivamente. Lleva implícito un proceso de búsqueda, por lo que es más específico que el encadenamiento hacia delante y por tanto, más eficaz.

La dirección del encadenamiento no tiene nada que ver con la dirección en que se ejecuta una regla; estas siempre se ejecutan *hacia delante*, es decir, ejecutando el consecuente cuando se confirma el antecedente. Cuando hablamos de encadenamiento hacia atrás nos referimos solamente a un proceso de búsqueda y selección de reglas.

6.3.3 Dependencia Reversible e Irreversible.

Una de las ventajas de los sistemas basados en reglas frente a la lógica clásica es la capacidad de retractar afirmaciones anteriores como consecuencia de nuevas inferencias. Ahora bien, ¿qué ocurre si la información que ahora se retracta había sido utilizada para obtener conclusiones?

Las herramientas más avanzadas permiten que sea el diseñador del sistema quien tome esta decisión. Para ello, al definir una regla hay que indicar el *tipo de dependencia*. Comparemos las dos siguientes reglas:

```
SI bombilla-encendida
ENTONCES habitación-iluminada
```

En este caso, si en un momento dado se cumple la premisa, el consecuente pasará a la base de afirmaciones, y si se retracta la primera afirmación deberá retractarse también la segunda. En este caso hablaremos de ***dependencia reversible***.

```
SI bombilla-encendida
ENTONCES película-velada
```

La segunda regla plantea una situación muy distinta. Aunque luego la bombilla se apague la película va a seguir estando velada. Nosotros hablaremos de ***dependencia irreversible*** y en este caso la afirmación no debe retractarse.

Por lo tanto, el SBR debe registrar junto con cada afirmación la forma en que ha sido deducida. Los SBR actuales permiten que sea el diseñador quien decida el tipo de dependencia.

6.4 Control del Razonamiento

Vamos a hablar de *cómo seleccionar qué regla ejecutar primero* cuando hay varias disponibles. El control del razonamiento es importante por tres motivos:

1. Por el ***contenido*** de la inferencia: las conclusiones pueden depender del orden en que se apliquen las reglas; las más específicas, y en particular las que tratan las excepciones, deben activarse primero para que no se apliquen otras reglas más generales y para evitar retractaciones posteriores.

2. Por ***eficiencia***: el utilizar la regla adecuada llevará rápidamente a una conclusión.

3. Por el ***diálogo*** que genera: es importante que es sistema no pregunte al usuario la información que pueda deducir por sí mismo y, además, que el orden en que solicita la información sea razonable.

6.4.1 Mecanismos sencillos.

Para controlar el razonamiento, la solución más simple consiste en *ordenar las reglas* en la base de conocimiento, colocando en primer lugar las que deseamos que sean examinadas antes. Es el método más simple y es poco “elegante” y de difícil mantenimiento. Sólo aplicable a sistemas simples donde las reglas se hallan en una lista que se consulta cíclicamente siguiendo el orden en que fueron introducidas por el usuario.

Frente a esta evaluación *estilo intérprete*, otros sistemas realizan una especie de compilación de las reglas para formar una red en que un mismo elemento presente en dos o más reglas viene representado por un solo nodo.

Otra forma sencilla de control, aplicable en sistemas que utilizan razonamiento hacia atrás, consiste en *ordenar* con cuidado *las cláusulas* dentro de cada regla, para que unas sean examinadas antes que otras. Se colocan primero las que tienen más posibilidades de fallar y así se optimiza la búsqueda.

Una solución más robusta (robustez consiste en que el orden de ejecución de las reglas no varíe demasiado conforme se incorporen nuevas reglas), válida en ambas direcciones del encadenamiento, se basa en añadir *nuevas cláusulas* relacionadas con el punto de inferencia en que nos encontramos, con el fin de controlar en cada momento *qué reglas deben aplicarse y cuáles no*.

6.4.2 Control de las Agendas.

Las herramientas más avanzadas realizan primero una búsqueda de todas las reglas que en un momento dado están listas para ser ejecutadas y las reúnen en una agenda (puede ser una pila o una cola). Más exactamente cada instanciación (asignación de variables que satisface una regla) da lugar a un elemento distinto en la agenda. En algunas herramientas es posible tener varias agendas y el usuario puede escoger un tipo de estructura para cada una de ellas.

El orden para cada agenda puede alterarse mediante una *prioridad* para cada regla (consiste en un valor numérico dentro de un cierto rango). En las herramientas más avanzadas es posible incluso modificar de forma dinámica la prioridad de una regla en función del proceso de inferencia.

Otro mecanismo para controlar el encadenamiento hacia delante es tener varias agendas, controladas por sus respectivos *patrocinadores o sponsor*. Cada regla está asignada a un patrocinador, en cuya agenda se almacenarán las instanciaciones de la regla. Los recursos disponibles (número de elementos ejecutados en cada ciclo) se distribuyen entre todos los patrocinadores, equitativamente o del modo que el diseñador del sistema haya determinado. Es posible, incluso, tener una jerarquía de patrocinadores para distribuir los recursos.

Otro mecanismo de control, independiente de las agendas (no incompatible), consiste en agrupar las reglas en *conjuntos de reglas o rule sets* permite activar o desactivar algunas de ellas en bloque, evitando así la necesidad de añadir en varias etapas premisas adicionales. Estos conjuntos estarán desactivados cuando se supone que no van a aportar ninguna información relevante, y se activarán solamente cuando sea necesario.

Estos dos mecanismos ayudan a estructurar de algún modo la base de conocimiento, facilitando así su construcción y su mantenimiento y mejorando la eficiencia.

6.4.3 Metarreglas.

Utilizadas por primera vez en el sistema TEIRESIAS - extensión de MYCIN- juntamente con los modelos de reglas (una forma estructurada de indicar los atributos comunes a un grupo de reglas, que se usaban para orientar la adquisición de conocimiento). Los *modelos de reglas* y las *metarreglas* son las dos formas de introducir el metaconocimiento (conocimiento sobre el conocimiento) en TEIRESIAS. Una metarregla es una regla que se refiere a otras reglas, y nos lleva a examinar primero aquellas reglas que parecen más relevantes para el problema en cuestión. De hecho hace afirmaciones sobre la utilidad estimada (la posibilidad de que nos lleve a una conclusión).

Comentarios:

El metaconocimiento es un paso más en la explicitación del conocimiento. Las capacidades de un sistema basado en metaconocimiento son:

- ☞ Tiene la posibilidad de modificar la estrategia de control sin tener que reprogramar el motor de inferencia.
- ☞ Capacidad de explicar la estrategia que he utilizado, por qué ha aplicado unas reglas antes que o en vez de otras, etc. y de ahí puede surgir más adelante la posibilidad de *reorganizar el conocimient*o, de *aprender* a partir de la experiencia o mediante el diálogo con el usuario.

Estas capacidades se basan en que el metaconocimiento, de algún modo, puede considerarse como una forma rudimentaria de consciencia (el sistema conoce que conoce): “no puede hablarse de inteligencia sin que haya consciencia”...

6.4.4 Otros Mecanismos.

Al abordar el problema del control del razonamiento mediante la selección de reglas se deben tener en cuenta dos características que debe poseer el sistema:

1. La *sensibilidad* que indica la capacidad de responder a los estímulos (el sistema debe procesar con prontitud la nueva información que recibe).
2. La *estabilidad* exige que la llegada de información poco relevante no cambie excesivamente la línea de razonamiento.

Los mecanismos para lograr un compromiso entre ambos objetivos (que de alguna manera son autoexcluyentes) son los siguientes:

1. ***Refractariedad***: después de haber respondido a un estímulo, existe un tiempo en que el sistema es incapaz de volver a responder (evita respuestas repetitivas al mismo estímulo).
2. ***Actualidad***: examinar primero las reglas que se apoyan en la *información más reciente* (línea de razonamiento estable y sensible a la nueva información).
3. ***Especificidad***: aplicar las reglas más específicas (contienen más información) antes que las más generales. En el siguiente ejemplo:

```
R1: Si a entonces b
R2: Si a y d entonces e

Base de afirmaciones inicial: {a, d}
```

R2 se ejecutaría antes que **R1** por ser más específica.

6.5 Explicación del razonamiento.

El primer sistema capaz de *explicar su razonamiento* fue MYCIN, que hace una inferencia mediante encadenamiento hacia atrás de reglas. Una regla sólo se ejecuta cuando se ha comprobado que se cumplen todas sus premisas, teniendo en cuenta que ante la imposibilidad de deducir una conclusión "interesante" el sistema interrogará al usuario. En usuario puede, entonces, responder, o bien interrogar al sistema:

- POR_QUÉ (WHY): se solicita del sistema la razón por la cual solicita nueva información.
- CÓMO (HOW): se solicita del sistema la forma en que ha llegado a una determinada conclusión.

Posteriormente se incluyeron en MYCIN otras posibilidades de explicación para que el programa pudiera responder a preguntas en lenguaje natural.

6.6 Tratamiento de la incertidumbre.

La incertidumbre es uno de los puntos clave a la hora de construir sistemas expertos debido a:

- ☞ La información puede ser imprecisa, incompleta e incluso errónea.
- ☞ El modelo de que disponemos puede ser incompleto e inexacto.
- ☞ El dominio de aplicación suele ser no determinista.

6.6.1 Factores de certeza de MYCIN.

MYCIN fue el primer sistema basado en reglas que se ocupó del tratamiento de la incertidumbre; sus creadores desarrollaron un modelo que, aunque inspirado en la teoría de la probabilidad, se apartaba notablemente de ella.

El caso más sencillo se plantea cuando tenemos una regla que relaciona un solo hallazgo, que llamaremos *e* (evidencia), con una sola hipótesis *h*:

SI e ENTONCES h

En este caso, puede definirse un *factor de certeza*, *FC(h, e)*, que indica la fiabilidad con que podemos aceptar la hipótesis *h* en el caso de tener la evidencia *e*. Estos "factores de certeza" toman en MYCIN valores pertenecientes al intervalo [–1,1].

En la práctica los factores de certeza se obtiene directamente a partir de estimaciones subjetivas de los médicos participantes en el proyecto. (Si Vd. observara *e* en un paciente, ¿con qué certeza pensaría que el diagnóstico es *h*?).

Cuando en una regla hay varias premisas, se utiliza una combinación de máximos y mínimos, dependiendo de si se trata de conjunciones o disyunciones, con el fin de obtener un factor de certeza $FC(h,e_1, e_2, ..., e_n)$. El problema es la combinación de los factores de certeza, ya que el sistema está basado más en intuiciones que en una teoría consistente. Si hay varias reglas que aportan evidencia a favor o en

contra de una hipótesis, hace falta una fórmula que indique cómo combinar los factores de certeza correspondientes. Cuando tenemos dos reglas cuyos factores de certeza son *x* e *y*, su FC combinado viene dado por:

$$FC_{comb}(x,y) = \begin{cases} x + y \cdot (1 - x) & si \;\; x > 0 \;\; e \;\; y > 0 \\ \dfrac{x + y}{1 - min(|x|, |y|)} & si \;\; x \cdot y < 0 \\ -FC_{comb}(-x, -y) & si \;\; x < 0 \;\; e \;\; y < 0 \end{cases}$$

Fórmula de Van Melle

6.6.2 Lógica difusa en sistemas basados en reglas.

La lógica difusa se puede aplicar mediante reglas difusas (que contienen predicados difusos). Por ejemplo:

```
SI curva ES MUY cerrada
Y velocidad ES alta
ENTONCES ACCIÓN frenar moderadamente
```

La estructura general de una regla difusa es:

SI X_1 es $A_1 \wedge ... \wedge X_n$ es A_n ENTONCES Y es B

En el caso de que haya una sola premisa *"X* es A", la regla puede interpretarse mediante una distribución de posibilidad condicional, $\pi_{B|A}(x,y)$, que es una relación difusa binaria del tipo R(x,y).

La inferencia se realiza a partir del *modus ponens difuso*, definido así:

$$\frac{X \; es \; A^* \qquad X \; es \; A \;\; \rightarrow \;\; Y \; es \; B}{Y \; es \; B^*}$$

donde *B** es un conjunto difuso que viene dado por:

$$\mu_{B^*}(y) = \max_{x \in U} \; T(\mu_{A^*}(x), \pi_{B|A}(x,y))$$

La función que se suele utilizar como *T* es el mínimo.

En caso de que tengamos una regla con varias premisas, habrá que combinar las $\mu_{Ai}(x_i)$ mediante los operadores mínimo o máximo según se trate de una conjunción o de una disyunción. También es posible combinar las reglas en serie (encadenando las conclusiones de una de las premisas con las premisas de otra) o en paralelo (cuando varias reglas conducen a una misma conclusión) utilizando las propiedades de unión e intersección de conjuntos.

6.7 Valoración

6.7.1 Comparación con los Programas Basados en Comandos

Nos enfrentamos a una forma de programar diferente de la tradicional; aquí no se le dice al ordenador lo que debe hacer (*programación imperativa*) sino cuál es el conocimiento que debe aplicar (*programación declarativa*). Las diferencias existentes entre estos dos planteamientos son:

1. Los comandos tienen un *carácter secuencial*: el motor de inferencia es el que determinará en cada momento cuáles de las reglas debe examinar y ejecutar, en función de la información disponible. Los sistemas basados en reglas tienen mecanismos de control del razonamiento muy distintos del orden rígido que caracteriza la ejecución secuencial de comandos.
2. La aplicación de las reglas se basa en la *comparación de patrones*. El uso de *variables* está relacionado con el carácter modular (hace posible que una misma regla pueda adaptarse a distintas situaciones). Este uso de las variables es específico de las reglas y es completamente diferente del uso de variables en los lenguajes de programación tradicionales.
3. La distinción entre *dependencia reversible e irreversible*: es otro rasgo distintivo de los sistemas basados en reglas, que permite retractar una conclusión cuando deja de ser cierta la información que

permitió deducirla. Claramente, un comando IF-THEN nunca podrá deshacer la acción aunque deje de cumplirse la condición que contenía.

4. El hecho de que el *conocimiento* contenido en un sistema basado en reglas sea *explícito* permite posibilidades que escapan por completo a los programas basados en comandos (capacidad de *explicar* su razonamiento, de *reorganizar* el conocimiento que posee y de *aprender* de sus errores).

5. El *tratamiento de la incertidumbre* es otro de los rasgos más típicos de los sistemas basados en reglas. En cambio, los comandos nunca admiten grados de verdad intermedios.

Entonces, ¿Porqué no se usa siempre sistemas basados en reglas?. A parte de razones que para lenguajes imperativos tradicionales, existen infinidad de rutinas ya disponibles, las principales razones son eficiencia y economía de recursos. Si un problema tiene una solución que no precise de una búsqueda heurística, o sea, resoluble mediante un algoritmo, la solución será más rápida haciendo uso de sistemas imperativos que con otros sistemas. Las tareas de bajo nivel (lectura y escritura de archivos, gestión de bases de datos, acceso a dispositivos, etc.) son más fáciles de realizar en lenguajes tradicionales.

6.7.2 Comparación con la Lógica de Predicados.

En su formulación más simple, las reglas pueden considerarse como una versión reducida de la lógica de predicados.

Por otra parte, la principal diferencia entre ambos métodos es que las reglas reducen la expresividad y la capacidad de inferencia con el fin de lograr una mayor eficiencia. La limitación en la *expresividad* consiste en la imposibilidad, o al menos en las restricciones, a la hora de utilizar cuantificadores existenciales, la reducción en su capacidad de inferencia: sólo afirman el consecuente a partir de la afirmación del antecedente (*modus ponens*); en cambio no hacen uso del *modus tollens*. Por ejemplo, en lógica, de la regla $\mathbf{p \rightarrow q}$ y de la afirmación $\mathbf{\neg q}$, puede deducirse $\mathbf{\neg p}$, cosa que no puede hacerse en un SBR.

Una ventaja de las reglas, además de la eficiencia, es la capacidad de tratar la incertidumbre (las reglas superan una de las limitaciones de la lógica clásica y se acercan a las lógicas modales, al razonamiento no monótono y al razonamiento aproximado).

6.7.3 Crítica de los Sistemas Basados en Reglas.

Si observamos cualquier sistema experto *real*, encontramos muchos rasgos que apartan a las reglas del carácter puramente declarativo (es frecuente encontrar cláusulas que no contienen información relativa al dominio, sino que se han introducido para controlar la inferencia). En general es mucho más difícil predecir cómo va a comportarse un sistema basado en reglas que un método algorítmico, por lo que la labor de programación es mucho más compleja. Cuando un sistema experto crece por encima de cierto límite se hace muy costoso o prácticamente imposible comprobar la consistencia del conocimiento que posee. En particular, la adición de una nueva regla puede modificar el comportamiento de forma imprevisible.

El otro punto por el que el uso de reglas es más criticado se refiere al tratamiento de la incertidumbre, que no es, realmente, todo lo adecuado que debiera.

6.8 Apéndice: Expresividad y tratabilidad.

En principio, debemos escoger el mecanismo de representación del conocimiento que nos proporcione la mayor *expresividad* (Capacidad de representar el conocimiento, necesaria para poder plantear el problema) posible. Ahora bien, para construir un programa inteligente no basta con representar el conocimiento: *es necesario realizar inferencias*. Se observa así que una mayor expresividad puede llevar a la imposibilidad de demostrar algunos teoremas y a la imposibilidad de realizar ciertas inferencias.

Más aún, en la práctica no basta saber que un problema tiene solución computacional: hace falta que pueda resolverse dentro del tiempo y del espacio de memoria disponibles.

En resumen, para que un problema sea *tratable* hace falta que exista un método para resolverlo (la *decidibilidad* vista en "lógica") y que el método sea suficientemente rápido cuando se ejecuta sobre un computador (*eficiencia*). Generalmente, al aumentar la expresividad suele disminuir la tratabilidad, y viceversa.

En la representación computacional del conocimiento no se puede renunciar a ninguno de los dos factores:

- la *expresividad* es necesaria para poder *plantear el problema,*
- la *tratabilidad* es necesaria para llegar a *resolverlo*.

PROBLEMAS

• PROBLEMA 6.1:

Considérese el conjunto de reglas siguiente, así como la siguiente base de hechos inicial y concepto meta:

R_1: **Si** h_8 **y** h_6 **y** h_5 **entonces** h_4

R_2: **Si** h_6 **y** h_3 **entonces** h_9

R_3: **Si** h_7 **y** h_4 **entonces** h_9

R_4: **Si** h_8 **entonces** h_1

R_5: **Si** h_6 **entonces** h_5

R_6: **Si** h_9 **y** h_1 **entonces** h_2

R_7: **Si** h_7 **entonces** h_6

R_8: **Si** h_1 **y** h_7 **entonces** h_9

R_9: **Si** h_1 **y** h_8 **entonces** h_6

Concepto meta u objetivo: h_2

Base de hechos inicial:

BH_0
H_7
H_8

Aplicar encadenamiento hacia atrás y describir el proceso de inferencia resultante. Para la resolución de conflictos utilizar la estrategia consistente en seleccionar primero aquella regla con un subíndice menor.

SOLUCIÓN:

Cuando en un proceso de inferencia con reglas se emplea encadenamiento hacia atrás para intentar demostrar un hecho o concepto objetivo, se llevan a cabo los siguientes pasos:

• Dado el hecho que se pretende demostrar, se averigua en primer lugar si dicho hecho figura en la base de hechos actual, con lo que quedaría demostrado.

• Si lo anterior no se cumple, se crea un conjunto conflicto con aquellas reglas en cuyo consecuente figure el hecho que se pretende demostrar y se llevarían a cabo los siguientes pasos hasta que no queden más reglas por aplicar o el objetivo haya sido demostrado:

a) Aplicando la estrategia de resolución de conflictos que se considere conveniente, se selecciona una regla.

b) En un proceso recursivo, se intentarían demostrar cada una de las condiciones que figuran en el antecedente de la regla seleccionada.

El proceso de encadenamiento hacia atrás con reglas descrito anteriormente conduce de forma natural a una estructura de árbol en la que los nodos representan conceptos particulares, habiendo dos posibles tipos de enlaces:

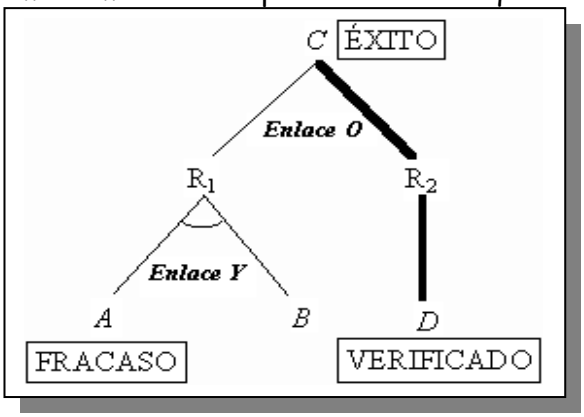
• **Enlaces O**: se dan en el caso de que exista una determinada submeta y la base de hechos contenga varias reglas con dicha submeta en sus consecuentes. Basta con que una de esas reglas produzca una demostración de la submeta mencionada para considerar que la misma ha sido verificada. Por ejemplo, dado:

R_1: A **y** $B \longrightarrow C$

R_2: $D \longrightarrow C$

Submeta: C; BH=$\{D\}$

se representaría como en la figura



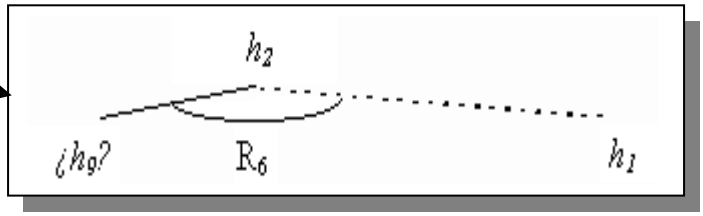
• **Enlaces Y**: sirven para representar el hecho de que, dada una submeta y una regla con dicha submeta en su consecuente, habrá que pasar a demostrar el antecedente de dicha regla. En dicho antecedente pueden figurar conjunciones de condiciones (o también disyunciones que originarían enlaces O), dando lugar a la aparición de enlaces Y en el árbol. Si el antecedente no puede ser demostrado, tampoco lo podrá ser la submeta que estaba siendo considerada, a no ser que haya otras reglas que lo permitan (en la figura puede observarse un enlace Y).

El proceso de encadenamiento hacia atrás de reglas conduce a una exploración en profundidad del árbol que se forma a partir de la base de conocimientos. En el caso de este problema se tendría:

1) El hecho objetivo es h2, con lo que habrá que averiguar si se encuentra en BH0. Como no es así, será necesario crear un **conjunto conflicto** con aquellas reglas que puedan llevarnos a demostrar h2:

• Conjunto conflicto: R6. Se tiene:

• Nuevo subobjetivo marcado: h_9.

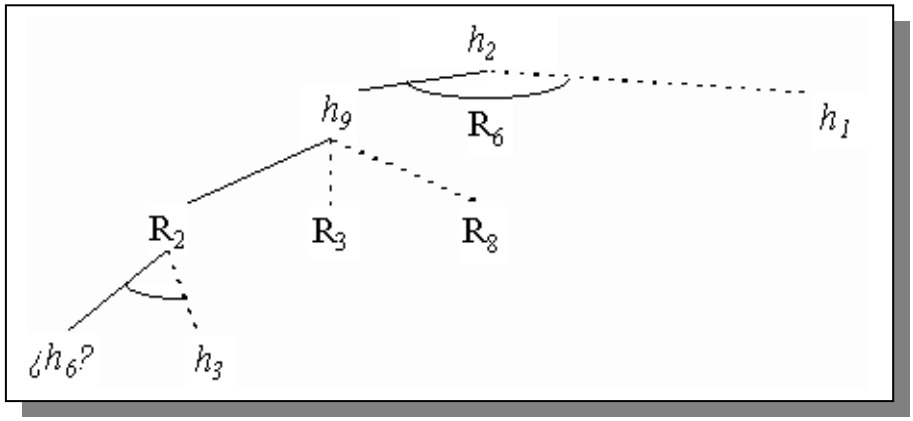


2) • h_9 no está en BH_0.

• Conjunto conflicto formado para demostrar h_9: R_2, R_3, R_8.

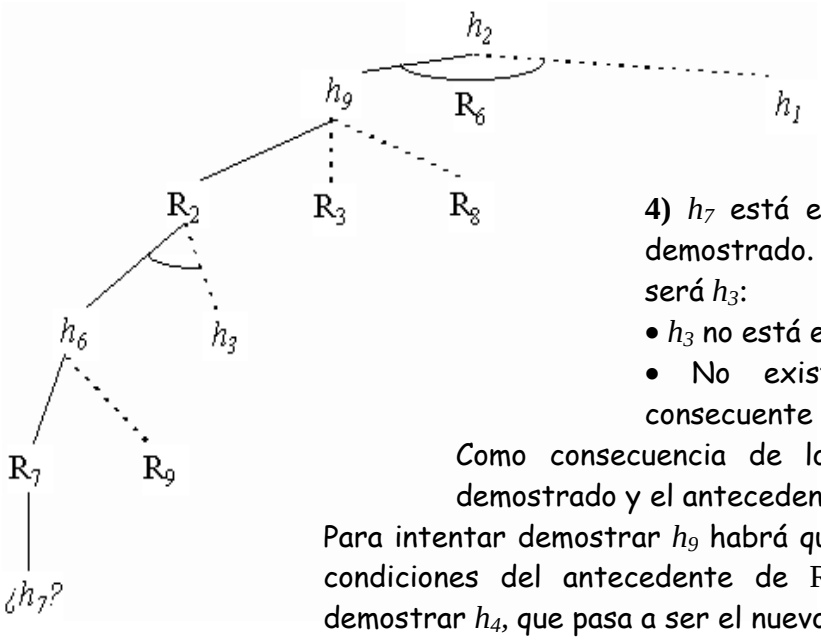
• Regla seleccionada R_2.

• Aparecen dos nuevos subobjetivos en el antecedente de R_2: h_6 y h_3. Se



estudiará en primer lugar h_6 (esta elección es arbitraria).

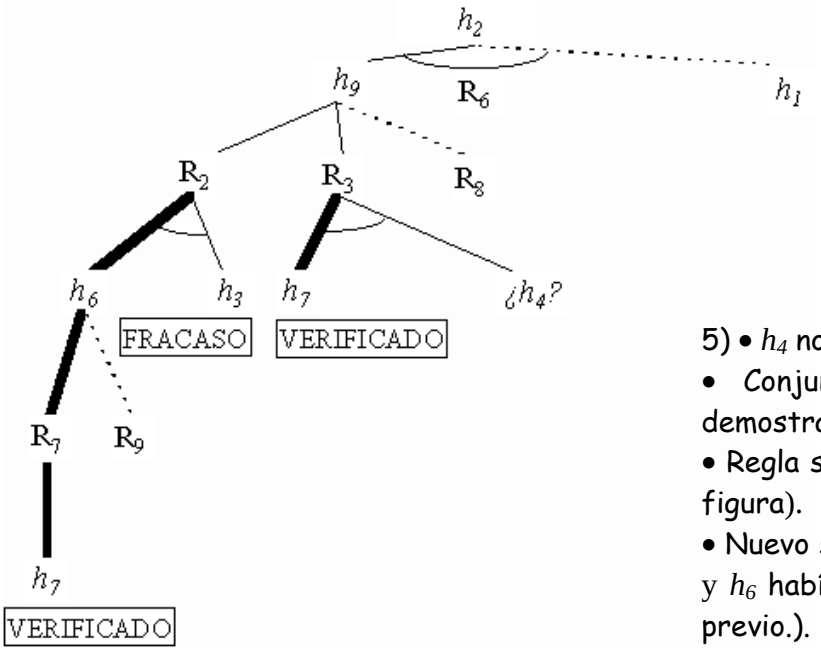
3) • h_6 no está en BH_0.
• Conjunto conflicto formado para demostrar h_6: R_7, R_9.
• Regla seleccionada: R_7.



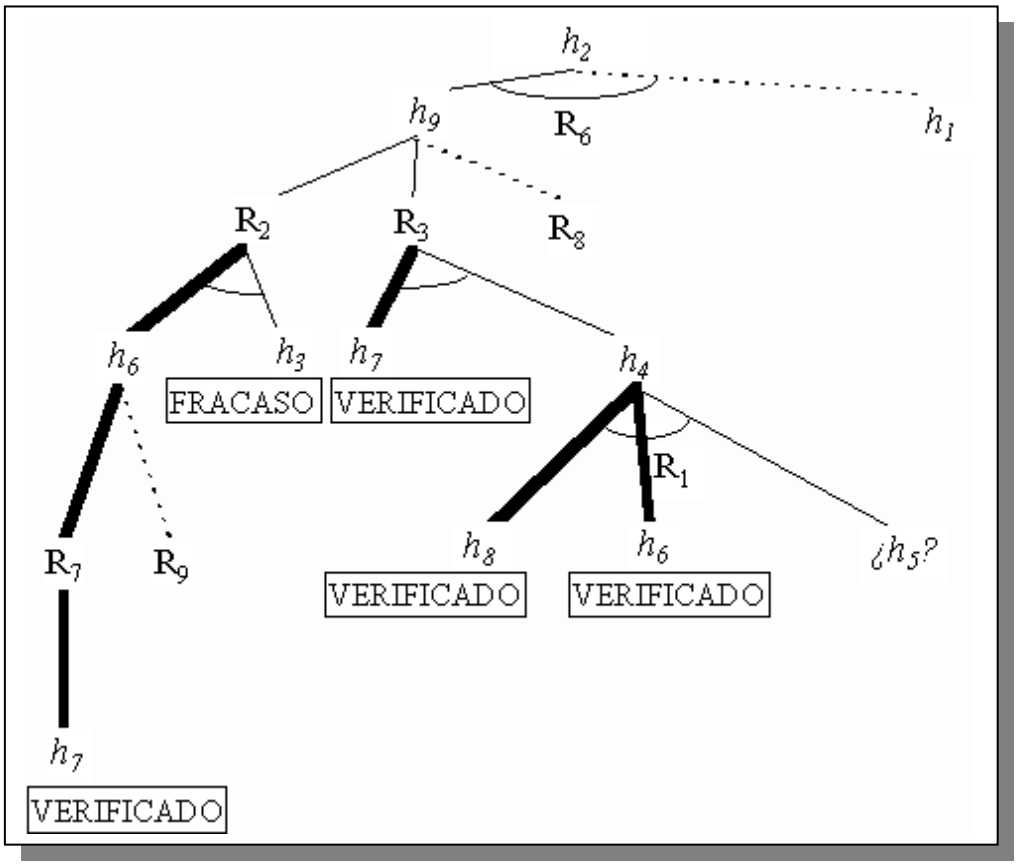
• Nuevo subobjetivo: h_7.

4) h_7 está en BH_0; por tanto, h_6 quedará demostrado. Ahora el nuevo subobjetivo será h_3:
• h_3 no está en BH_0.
• No existe ninguna regla en cuyo consecuente aparezca h_3.
Como consecuencia de lo anterior, h_3 no puede ser demostrado y el antecedente de R_2 no se cumple.
Para intentar demostrar h_9 habrá que recurrir a R_3. h_7, una de las condiciones del antecedente de R_3, está en BH_0. Queda por demostrar h_4, que pasa a ser el nuevo hecho objetivo.

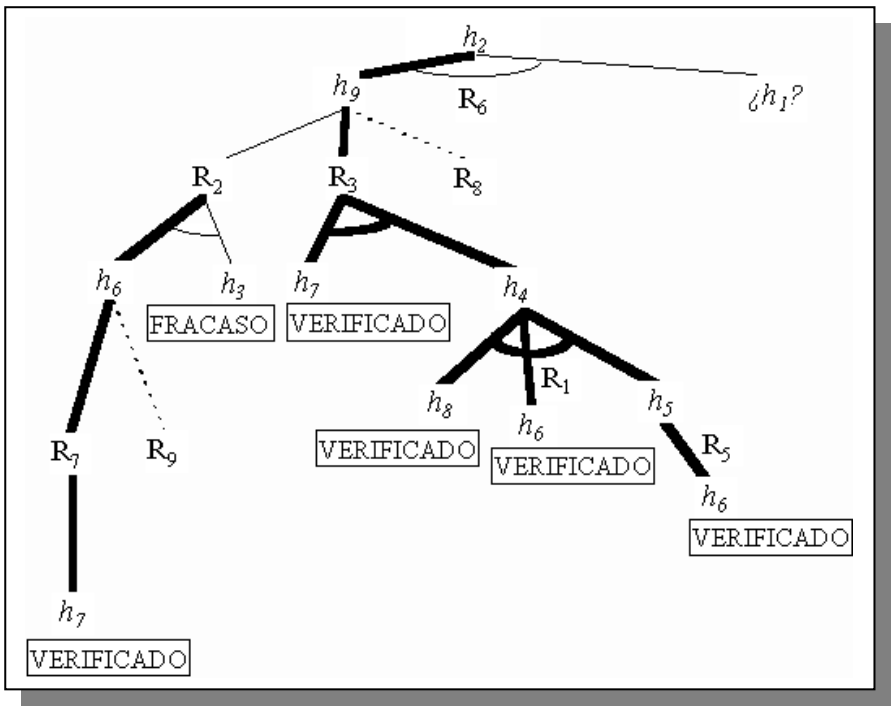
En la siguiente figura aparece reflejado el razonamiento anterior. En dicha figura las líneas gruesas reflejan los hechos que se van verificando. Las líneas con trazo normal se refieren a zonas investigadas que se pondrán con trazo grueso sólo si hay verificación. Finalmente, las líneas con trazo entrecortado reflejan zonas que no ha sido necesario investigar.



5) • h_4 no está en BH_0.
• Conjunto conflicto formado para demostrar h_4: R_1.
• Regla seleccionada: R_1 (ver siguiente figura).
• Nuevo subobjetivo: h_5 (h_8 está en BH_0 y h_6 había sido demostrado en un paso previo.).

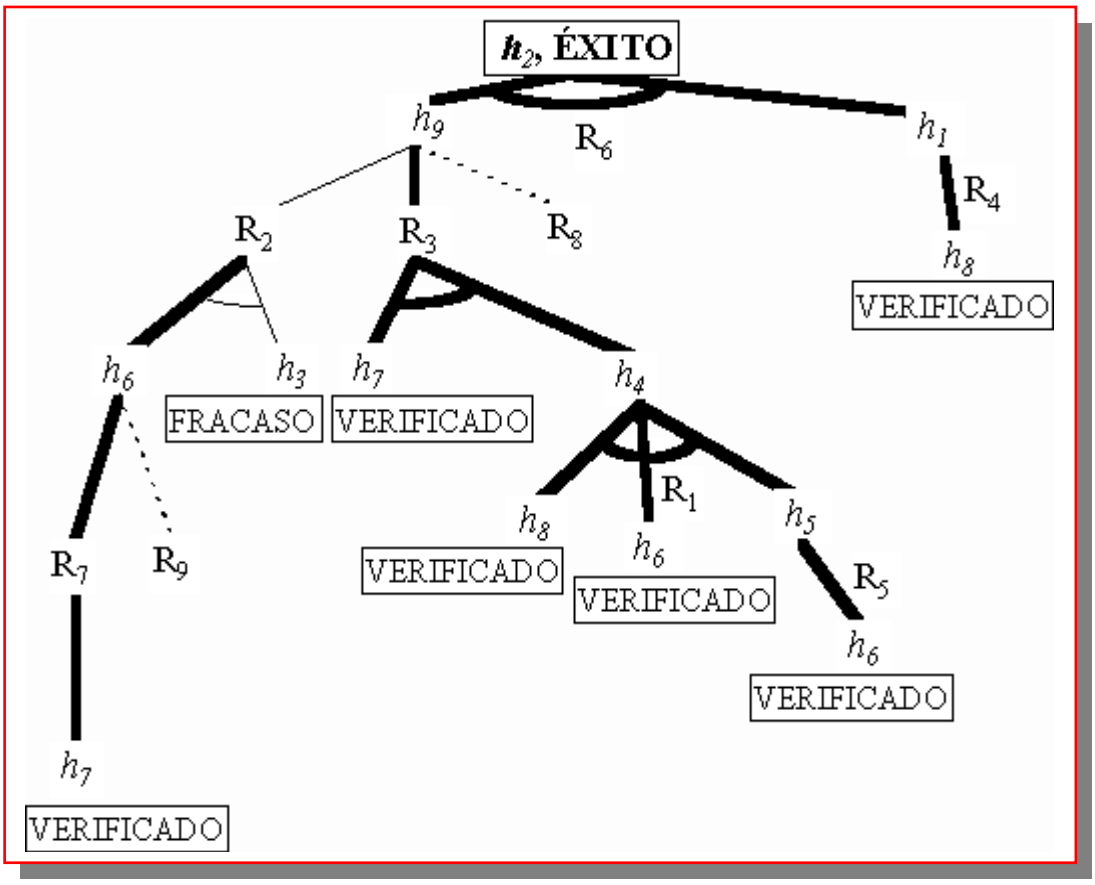


6) • h_5 no está en BH_0.
• Nuevo conjunto conflicto: R_5.
• Regla seleccionada: R_5.
• Nuevo subobjetivo: h_6, que ya se había verificado en un paso anterior. Por tanto, h_5 queda demostrado y, como consecuencia de ello, h_4 también. A su vez, h_9 quedará también demostrado. La nueva situación queda reflejada en la figura:



• Nuevo subobjetivo: h_1.

7) • h_1 no está en BH_0.
• Conjunto conflicto: R_4.
• Regla seleccionada: R_4.
• Nuevo subobjetivo: h_8, que está en BH_0. Por tanto, h_2 queda finalmente verificado (ver figura).



• **PROBLEMA 6.2:**

Un sistema basado en reglas emplea el modelo de factores de certeza de MYCIN para tratar la incertidumbre que aparece en un determinado dominio. Sean las reglas:

R_1: **Si** h_2 **o** h_3 **entonces** h_6 (0'4)

R_2: **Si** h_6 **y** h_7 **entonces** h_1 (0'7)

R_3: **Si** h_4 **o** h_5 **entonces** h_1 (0'1)

Supóngase, además, que los atributos h_2, h_3, h_4, h_5 **y** h_7 **se han establecido con factores de certeza 0'3, 0'6, 0'2, 0'8 y 0'9, respectivamente. El atributo** h_1 **es el atributo objetivo, empleándose encadenamiento hacia atrás. Calcular el factor de certeza que resulta para el atributo** h_1.

SOLUCIÓN:

El conjunto conflicto inicial estaría formado por $\{R_2, R_3\}$. Examinando en primer lugar R_2, que es la regla más prioritaria, sería necesario conocer el factor de certeza para h_6 y h_7. Se conoce este factor para h_7, a partir de la evidencia inicial, pero no para h_6. Este último valor puede calcularse mediante la regla R_1. En efecto,

$FC(h_2, e') = 0'3$, $FC(h_3, e') = 0'6 \Rightarrow FC(h_2 \textbf{ o } h_3, e') = \max\{0'3, 0'6\} = 0'6$

Por tanto, a partir de R_1:

$FC(h_6, e_1') = FC(h_6, h_2 \textbf{ o } h_3) \cdot \max\{0, FC(h_2 \textbf{ o } h_3, e')\} = 0'4 \cdot 0'6 = 0'24$

Luego,

$FC(h_6 \textbf{ y } h_7, e_1') = \min\{FC(h_6, e_1'), FC(h_7, e_1')\} = \min\{0'24, 0'9\} = 0'24$

$FC(h_1, e_1') = FC(h_1, h_6 \textbf{ y } h_7) \cdot \max\{0, FC(h_6 \textbf{ y } h_7, e_1')\} = 0'7 \cdot 0'24 = 0'168$

Ahora habrá que considerar la otra regla que formaba parte del conjunto conflicto inicial: R_3. Se tiene:

$FC(h_4, e_2') = 0'2$, $FC(h_5, e_2') = 0'8 \Rightarrow FC(h_4 \textbf{ o } h_5, e_2') = \max\{0'2, 0'8\} = 0'8$

$FC(h_1, e_2') = FC(h_1, h_4 \textbf{ o } h_5) \cdot \max\{0, FC(h_4 \textbf{ o } h_5, e_2')\} = 0'1 \cdot 0'8 = 0'08$

Combinando las dos reglas que producen evidencia sobre h_1, se obtiene finalmente:

$FC(h_1, e_1' \textbf{ comb } e_2') = FC(h_1, e_1') + FC(h_1, e_2') \cdot (1 - FC(h_1, e_1')) =$

$= 0'168 + 0'08 \cdot 0'832 =$ **0'234**

7 Redes Asociativas

En las ***redes asociativas*** cada nodo representa un concepto (o una proposición) y los enlaces corresponden a relaciones (inclusión, pertenencia, causalidad) o a categorías gramaticales (verbo principal, sujeto, objeto, complementos, etc.). entre ellas, se conocen:

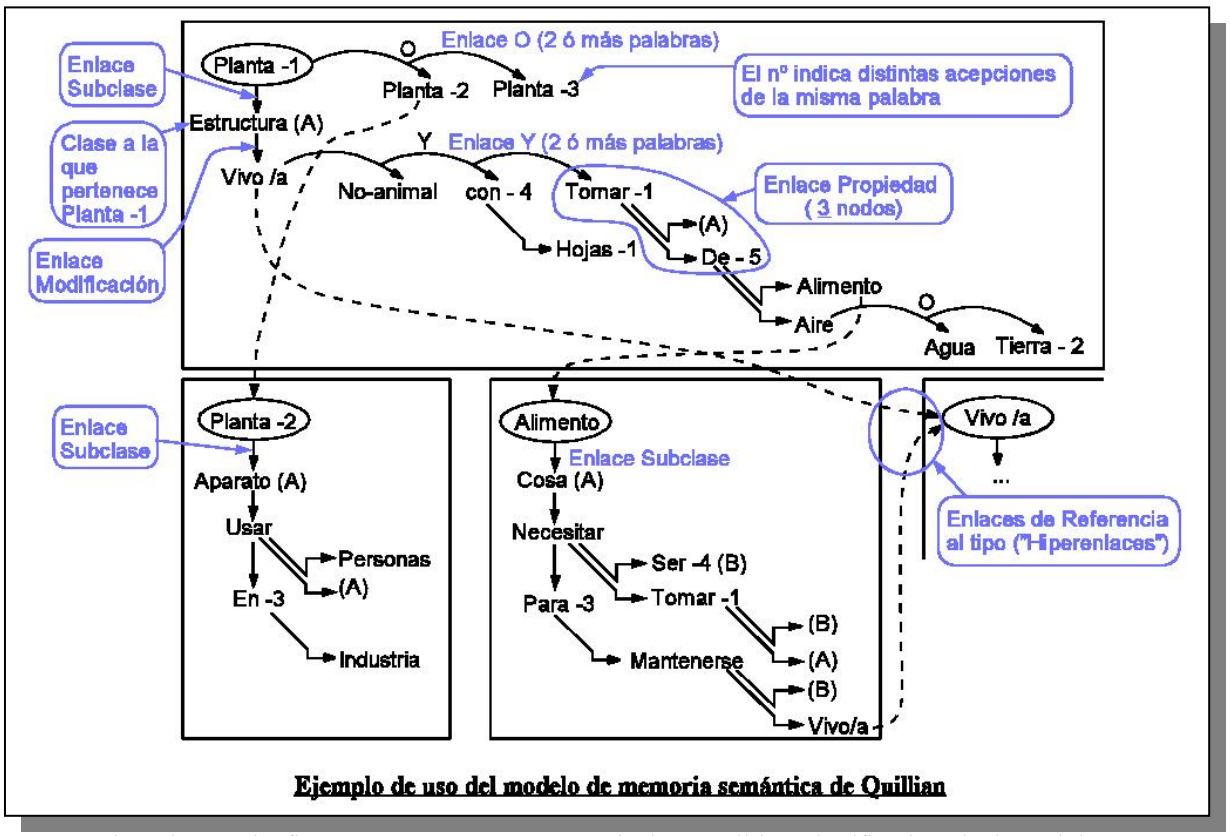
- **redes semánticas**: son las destinadas a representar o a comprender el lenguaje natural.
- **redes de clasificación**: es exactamente lo que su nombre indica, una clasificación de objetos o conceptos con sus características propias (herencia y demás).
- **redes causales**: son las que llevan asociadas, junto a sus nodos que representan variables, una relación de influencia, representada por los enlaces.

7.1 Grafos Relacionales.

7.1.1. Modelo de Memoria Semántica de Quillian.

Ross Quillian trató de construir un modelo computacional de la memoria humana, más concretamente, un programa de ordenador capaz de almacenar información sobre el significado de las palabras con la función de poder llegar a comprender el lenguaje natural.

En su sistema de representación, cada plano contiene un **nodo-tipo** (*type node*), encerrado en un óvalo, y varios **nodos-réplica** (*token nodes*). El nodo-tipo corresponde a los encabezamientos de las definiciones (palabra que se va a definir en un diccionario) y los nodos-réplica son las demás palabras que aparecen en la definición (es similar a las palabras que forman la definición de una palabra del diccionario).



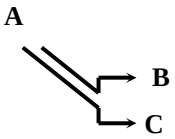
Ejemplo de uso del modelo de memoria semántica de Quillian

Por ejemplo, en la figura, se representa uno de los posibles significados de la palabra PLANTA: APARATO que las PERSONAS UTILIZAN EN la INDUSTRIA.

Conviene destacar el uso de variables para indicar conceptos que aparecen en el mismo plano de definición (por ejemplo, se define APARATO como la variable A, que es usada más tarde –Las personas usan el A, es decir, el aparato-). Los números se utilizan para distinguir los diferentes significados de un mismo término (EN-3, representa el tercer significado de la palabra "EN").

En las redes de Quillian aparecen seis tipos de enlaces:

1. **Subclase**: une un *nodo-tipo* con la clase a la que pertenece.
2. **Modificación**: une dos *nodos-réplica*, de modo que el segundo modifica el alcance del primero.
3. **Disyunción**: lleva la etiqueta "**O**", une dos o más nodos.
4. **Conjunción**: lleva la etiqueta "**Y**", también une dos o más nodos.
5. **Propiedad**: une tres nodos, **A**, **B** y **C**, donde A es la relación, B el sujeto y C el objeto. Por



ejemplo, *LAS PERSONAS (sujeto) USAN (relación) EL APARATO (objeto)*
6. **Referencia al tipo**: van desde un *nodo-réplica* al correspondiente *nodo-tipo*; se dan siempre entre planos diferentes. Por ejemplo, del *nodo-réplica* APARATO, iríamos a otro plano en el que se definiría un *nodo-tipo* APARATO.

El interés de representar computacionalmente el conocimiento consiste ante todo en poder realizar *inferencias* con el fin de obtener nueva información. La tesis de Quillian se limitó a una forma muy sencilla de inferencia: cómo comparar el significado de dos palabras, suponiendo que esta tarea significa un primer paso hacia la compresión de frases y relatos.

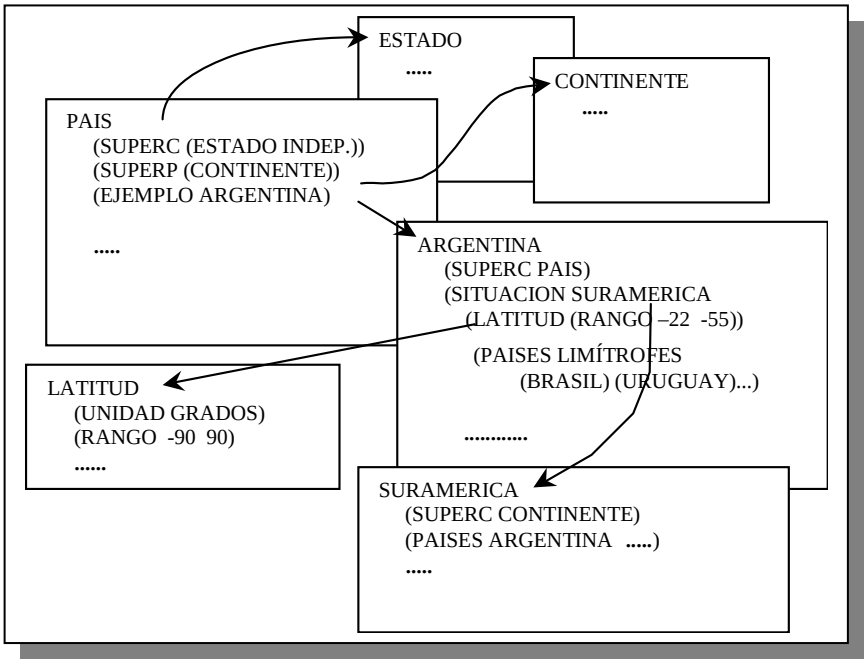
La comparación de palabras por el método de Quillian se realiza de la forma siguiente:

1. Activar los nodos vecinos de cada una de las dos definiciones creando esferas de activación de radio 1.
2. Comprobar las coincidencias. Cada intersección de esferas corresponde a un camino que une los dos nodos (una relación entre significados).
3. Si no hay coincidencias, ampliar las esferas a radio 2 y repetir la comprobación.
4. Repetir el proceso hasta encontrar coincidencias y poder relacionar los conceptos. El resultado final del programa se obtiene convirtiendo cada camino en una frase explicativa.

La estructura que utiliza Quillian es totalmente dependiente del idioma particular en que se formula el sistema. Esta dependencia se hace evidente cuando se tratan términos polisémicos. Lo ideal es que no exista esa dependencia pero para ello habría que pasar de la representación de palabras a la representación de conceptos. En las redes semánticas de Quillian no se puede extraer nueva información mediante inferencia, sino simplemente realizar comparaciones del significado de las palabras.

7.1.2 Sistema SCHOLAR de Carbonell.

Este programa poseía una base de conocimiento estructurada en forma de red (permitía generar dinámicamente preguntas para el alumno y responder a las que éste le planteaba). Además, tenía la ventaja que el módulo encargado de generar y responder a las preguntas (el motor de inferencia) era prácticamente *independiente del dominio* representado en la red (facilidad posterior para generar programas similares para otros temas). Una gran aportación de este sistema es la distinción entre los nodos que representan conceptos y los que representan ejemplos particulares.



Otra de las ideas fundamentales de SCHOLAR consiste en definir cada nodo mediante dos elementos: la clase a la que pertenece y una lista de propiedades. Una *propiedad* viene dada por un atributo, un valor y,

opcionalmente, otras subpropiedades. Por ejemplo, el nodo ARGENTINA está definido como una instancia de la clase PAÍS. La propiedad correspondiente al atributo SITUACIÓN, tiene como valor SURAMÉRICA y como subpropiedades LATITUD, PAÍSES LIMÍTROFES...etc.

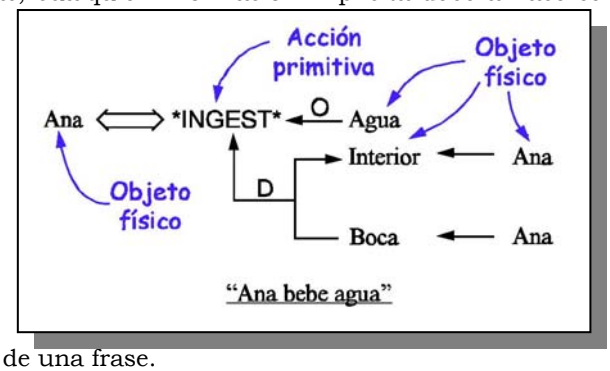
Además, cada propiedad puede llevar asociadas unas funciones destinadas a lograr inferencias más eficientes (éstas funciones se utilizan en la actualidad en la representación basada en marcos, con el nombre de *demonios*).

La distinción entre conceptos (o clases) e instancias, y la asignación de propiedades en forma de pares atributo-valor constituyen en la actualidad dos características esenciales de la representación basada en marcos y, de una u otra forma, de la mayor parte de los modelos de redes semánticas. Hoy en día no se considera una red semántica, debido a que no está dedicado a la representación del significado de los conceptos ni a la comprensión del lenguaje natural.

7.1.3 Grafos de Dependencia Conceptual de Schank.

Su perspectiva era diferente de la de Ross Quillian, puesto que Schank se apoyaba más en los estudios de la lingüística que en los modelos psicológicos de la memoria humana.

Otra diferencia es que Schank trató de abordar el problema del lenguaje independientemente de cualquier idioma particular, dicho de otro modo, a Schank no le interesaba representar las *palabras* sino los *conceptos*. En los grafos de dependencia conceptual, dos sentencias diferentes con el mismo significado, tendrán la misma representación, y además, cualquier información implícita deberá hacerse explícita en la representación del significado de una sentencia.



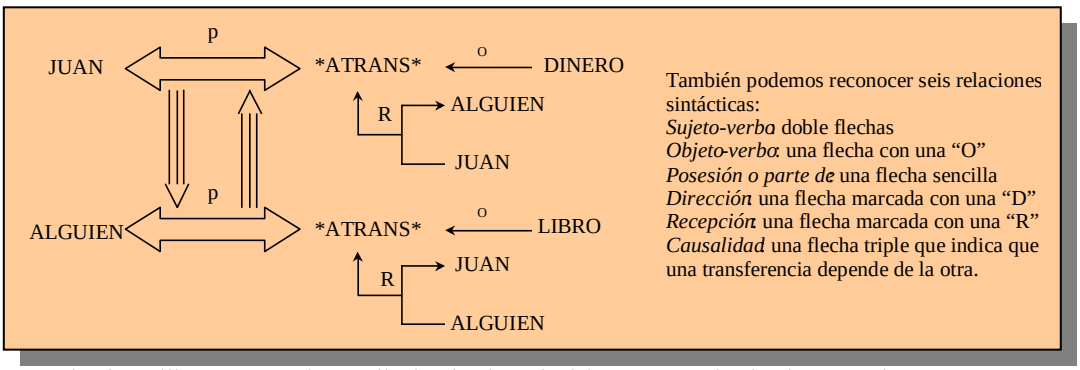
La idea principal consiste en interpretar (*representar*) cualquier frase mediante un número de *primitivas*: seis categorías conceptuales, 16 reglas sintácticas y varias acciones primitivas (Schank afirma que 12 de ellas son suficientes).

Las *categorías conceptuales* son: **objeto físico**, **acción**, **atributo de un objeto físico**, **atributo de una acción**, **tiempo** y **localización**.

Las *reglas sintácticas* determinan los diferentes tipos de relación que pueden existir entre los elementos de una frase.

Entre las *acciones primitivas* se encuentran las siguientes:

PTRANS	Transferir físicamente (cambiar de lugar un objeto)
ATRANS	Transferir una relación abstracta, como posesión o control
MTRANS	Transferir mentalmente (decir, contar, comunicar, etc.)
PROPEL	Empujar
MOVE	Mover un miembro de un animal
GRASP	Coger, atrapar
INGEST	Ingerir
CONC:	Conceptualización o pensamiento de una idea por un actor.
EXPEL:	Expulsión desde un cuerpo animal al exterior.
MBUILD:	Construcción de una información a partir de una que existía.
ATTEND:	Acción de dirigir un órgano de los sentidos hacia un objeto.
SPEAK:	Acción de producir sonidos con la boca.
.....	



La ventaja de utilizar un conjunto limitado de primitivas son principalmente dos:

1. Éstas *determinan unívocamente* la representación del conocimiento,
2. La posibilidad de construir un *intérprete* capaz de relizar *inferencias*.

Uno de los tipos de *inferencias* que pueden darse mediante estos grafos consiste en establecer las *condiciones*. Otro de los tipos de inferencias consiste en hallar las *causas* (y entre ellas las *intenciones*). También se puede inferir el *resultado* de las acciones.

Estos grafos cumplen:

- ☞ Que el sistema computacional sea eficiente (compromiso planteado entre la expresividad y la eficiencia) ya que hay un número limitado de elementos y relaciones. Se cumple utilizando un conjunto reducido de *primitivas.*
- ☞ Que el sistema tenga la potencia y flexibilidad necesaria para abordar el problema que queremos resolver.
- ☞ Que el sistema comprenda el lenguaje natural. Para ello se utiliza al *descomposición* de cualquier frase en los elementos y acciones más simples que intervienen.

Este método tiene una serie de desventajas:

1. Se cuestiona la validez de definir unas primitivas universales. Algunos autores consideran que el significado de una frase, salvo en los casos más sencillos, es inseparable de la lengua en que se encuentra formulada.
2. Los grafos conceptuales requieren una descripción demasiado detallada de las acciones (en ejemplos menos triviales que los que hemos escogido, los grafos resultan difíciles de interpretar a simple vista).
3. El número de primitivas dependerá del grado de detalle que debemos alcanzar en la descomposición. Hay autores que afirman que no tiene sentido trabajar con un número mínimo de primitivas, sino que hay que hacerlo con distintos niveles de detalle, de modo que el sistema puede explicitar los elementos cuando sea necesario.
4. Se centra la representación en torno al verbo.

7.1.4 Problemas de los Grafos Relacionales.

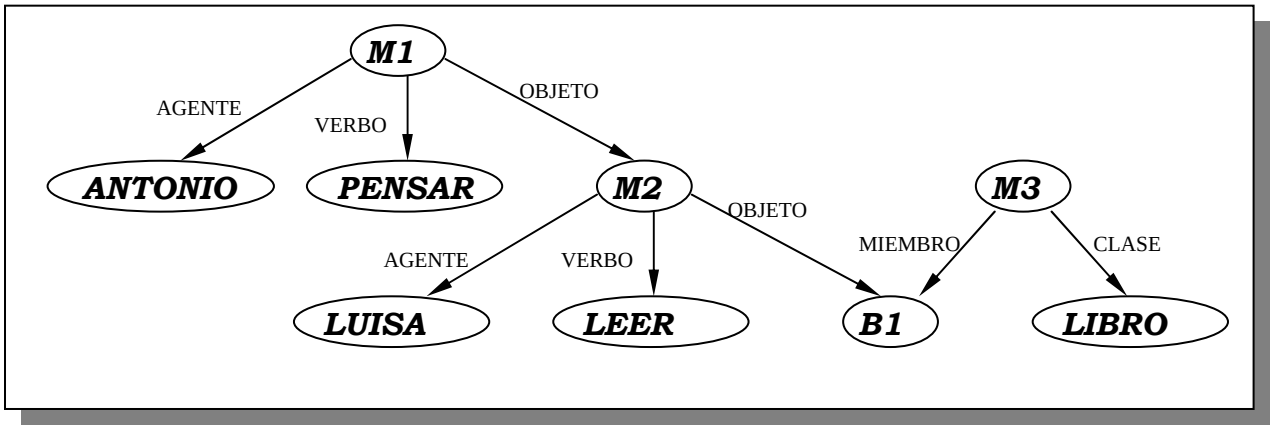
Estos tipos de redes presentan serias limitaciones a la hora de abordar el lenguaje natural. A pesar de que cada nodo corresponde a un concepto en estos tipos de redes no se pueden tratar los operadores universal y existencial de la lógica de predicados. Tampoco tratan la interacción entre varias proposiciones, por ejemplo no se puede transcribir la frase "*Cuando es de noche y hay niebla es peligroso conducir*".

7.2 Redes Proposicionales.

Aquí los nodos no sólo representan conceptos sino que también pueden representar sintagmas, cláusulas, frases, párrafos e incluso historias completas.

7.2.1 Redes de Shapiro.

Fue el primero en implementar una red asociativa que incluyera todos los operadores y cuantificadores de la lógica de primer orden. Cada nodo puede representar un concepto, una variable, una frase, una relación o una regla de inferencia.

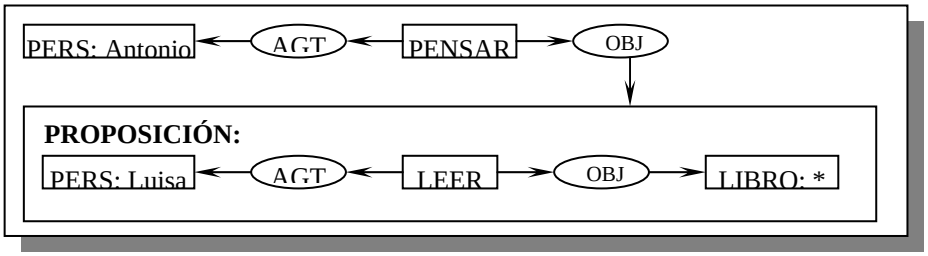


Los arcos llevan una etiqueta para indicar el tipo de relación entre nodos. En el ejemplo de la figura, se representa la frase "*Antonio piensa que Luisa está leyendo un libro*". El nodo B1 parecería sobrar, pero

es para referirse al libro concreto que está leyendo, y no a la clase formada por todos los libros, o mejor dicho, el concepto de libro.

7.2.2 Representación mediante Grafos de Sowa.

Tienen el mismo objetivo que las redes de Shapiro. La determinación de los contextos se realiza trazando rectángulos que agrupen cierto número de nodos. Existen muchas similitudes entre los grafos de Sowa y los de Shapiro. En cambio, en los grafos de Sowa, se utiliza un círculo para indicar el tipo de relación, en lugar de la etiqueta que se utiliza en los de Shapiro. Otra diferencia es que Sowa indica explícitamente en cada nodo la clase a la que pertenece. Para indicar la identidad de un nodo se escribe su nombre junto a la clase pero si la identidad no es conocida se puede utilizar un asterisco.



Tanto la representación gráfica de Shapiro como la de Sowa pueden expresarse en ***notación lineal***, de manera que los nodos aparecerán entre corchetes y las etiquetas de los arcos entre paréntesis. Esta notación ocupa menos espacio y es más fácil de manipular por un programa de ordenador. Usando esta notación tendremos, por ejemplo:

```
[PENSAR]-
    →(AGT)→[PERS: Antonio]
    →(OBJ)→[PROPOSICIÓN: [LEER]-
             (AGT)→[PERS: Luisa]
             (OBJ)→[LIBRO]]
```

```
[RÍO: *] ≡[RÍO]              Un río (operador existencial).
[RÍO]→(ATR)→[CAUDALOSO]      Un río caudaloso.
[RÍO: {*}]                   Algunos ríos.
[RÍO: {*}@3]                 Tres ríos.
[RÍO: #]                     El río (uno específico).
[RÍO: ?]                     ¿Qué río?
[RÍO: ∀]                     Todos los ríos (operador universal).
 . . . .
```

Para concretar un elemento utiliza el símbolo # ([RÍO: #]). En este tipo de proposiciones el encargado de descubrir la identidad del río es el intérprete.

La mayor parte de los modelos de redes utilizan variables con el fin de representar un objeto o un concepto que aparece en varios lugares diferentes. En este caso se utilizará una línea discontinua para identificar los nodos que representan un mismo ente. La "T" se utiliza para representar un nodo sin especificar la clase a la que pertenece). En la notación lineal, al no poder unir los nodos mediante líneas discontinuas, se utilizan variables para indicar los nodos que coinciden.

Concluyendo, en un grafo de Sowa intervienen elementos de tres clases:

- **Conceptos genéricos**, tales como LIBRO, PERS, PENSAR y COMER.
- **Conceptos individuales**, tales como [PERS: Luis] que representa a una persona individual, o [PENSAR], que representa un acto de pensamiento.
- **Relaciones conceptuales**, que pueden ser unitarias (PASADO, NEG, etc.), binarias (AGT, OBJ, etc.) e incluso ternarias (ENTRE, para indicar que A se encuentra ENTRE B y C).

Al conjunto de conceptos genéricos y de relaciones conceptuales establecido anteriormente en el sistema se le conoce como *conjunto canónico*, y a partir de él se pueden definir nuevos conceptos y relaciones.

7.2.3 Inferencia en Grafos de Sowa.

La inferencia en este tipo de grafos se implementa mediante un conjunto de *operaciones básicas*:

- ☞ · Restricción: concretar más la información del grafo. Introduce información adicional, a veces, se pierde la veracidad del grafo resultante.
- ☞ · Generalización: operación recíproca a la anterior. No introduce ninguna información adicional, por tanto si el grafo original era verdadero, el generalizado también lo será.
- ☞ · Unión y Simplificación: se aplican conjuntamente sobre un par de grafos..

Estas operaciones, junto con unas sencillas reglas de cálculo permiten implementar mediante grafos conceptuales toda la lógica de primer orden e incluso algunas características de la lógica de orden superior. Es lo que se puede denominar *razonamiento exacto* o *razonamiento deductivo* mediante grafos de Sowa. Con éste método se trabaja en la representación de razonamientos aproximados o probabilísticos.

7.3 Redes de Clasificación.

7.3.1 Extensión e Intensión.

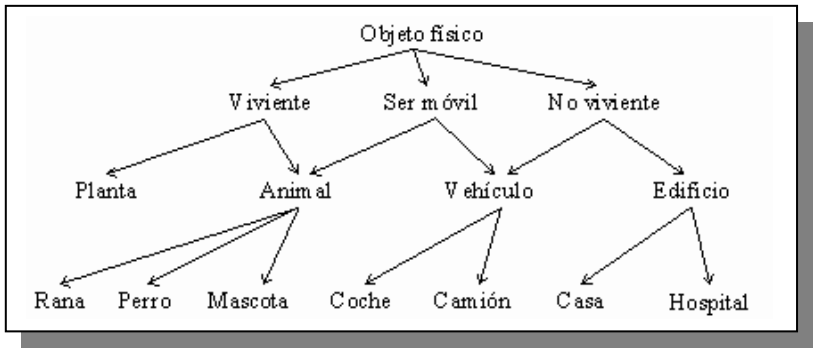
El significado intensional se refiere a la *definición* del concepto, mientras que el extensional se refiere a los elementos a los que se les puede aplicar el concepto *dentro de un universo determinado.*

	Significado intensional	Significado extensional
Hombre	Animal racional	Conjunto de todos los seres humanos que pueblan la tierra
Número primo	Número natural mayor que 1 y que sólo es divisible por sí mismo y por la unidad	{2, 3, 5, 7, 11, 13, ...}
Planetas solares	Planetas que giran alrededor del sol	{Mercurio, Venus, Tierra, Marte, Júpiter, Saturno, Urano, Neptuno, Plutón}

En la tabla se pueden destacar dos aspectos:

- El significado intensional no varía.
- El alcance extensional puede variar en el tiempo (por ejemplo el hombre).

El motivo de tratar la diferencia entre extensión e intensión es que en la literatura se ha discutido si estas redes deben representar *conceptos* (intensionales) o *clases* (extensionales). Los sistemas más importantes que existen actualmente (los grafos de Sowa y los lenguajes derivados de KL-ONE) optan por el significado intensional frente al extensional.



A pesar de estas diferencias, todas las redes de clasificación (GDA en el que hay bucles pero no ciclos) coinciden en sus aspectos esenciales. El concepto superior (el que engloba a todos los demás) se representa por "T". Los arcos indican una relación de orden parcial (un arco trazado desde un nodo *A* a un nodo *B* indica la relación $B < A$). Al pasar del significado intensional al extensional, esto implica que la clase asociada a *B* está *incluida* en la clase asociada a *A*. La red de clasificación no sólo incluye objetos físicos, sino que cada uno de los nodos que encontramos al estudiar los grafos de Sowa pertenece a un concepto de esta red, incluidos los nodos que representan proposiciones.

7.3.3 Mecanismos de Herencia.

Existen dos tipos de herencia en redes jerárquicas:

- La *herencia estricta*: consisten en que todos los conceptos descendientes de A poseen necesariamente las mismas propiedades de A.
- La *herencia por defecto*: *supone* que los descendientes de A poseen las mismas propiedades de A mientras no se indique lo contrario.

Por ejemplo: el concepto de "ser vivo", implica respirar. De aquí se infiere que el concepto "perro", también respira.

La herencia estricta se ciñe a las leyes de la lógica clásica, mientras que la herencia por defecto se sitúa dentro del razonamiento no monótono. Plantea un problema al trabajar con grafos dirigidos acíclicos, ya que el hecho de que un nodo pueda tener distintos padres hace que puedan surgir contradicciones entre los diferentes valores por defecto heredados. De ahí surge la necesidad de establecer mecanismos para resolver estos conflictos.

7.3.4 Sistemas Taxonómicos / Sistemas Asertivos.

El conocimiento contenido en una red semántica puede ser de dos clases:

1. El *conocimiento asertivo* que consiste en realizar afirmaciones particulares. En ellas no existen definiciones de conceptos ni clasificaciones jerárquicas, sino solamente afirmaciones concretas. Los grafos *Schank* son el ejemplo más claro de redes asertivas.
2. El *conocimiento taxonómico* que describe los conceptos. El sistema KL-ONE es un ejemplo de red taxonómica, ya que está diseñado especialmente para clasificar los conceptos y manejar sus propiedades.

Hay dos grupos de redes semánticas, en función del tipo de conocimiento al que den mayor importancia. No obstante, los dos grupos de redes pueden incluir ambos tipos de conocimiento, aunque generalmente de forma separada: Red de clasificación, y Sistema para el tratamiento de proposiciones (aserciones).

Por ejemplo, KRYPTON consta de dos componentes:

1. El *T-box* que contiene el conocimiento taxonómico en forma de red de clasificación de conceptos.
2. El *A-box* que contiene el conocimiento asertivo, expresado mediante predicados, que coinciden con los conceptos y los roles del *T-box*.

7.4 Redes Causales.

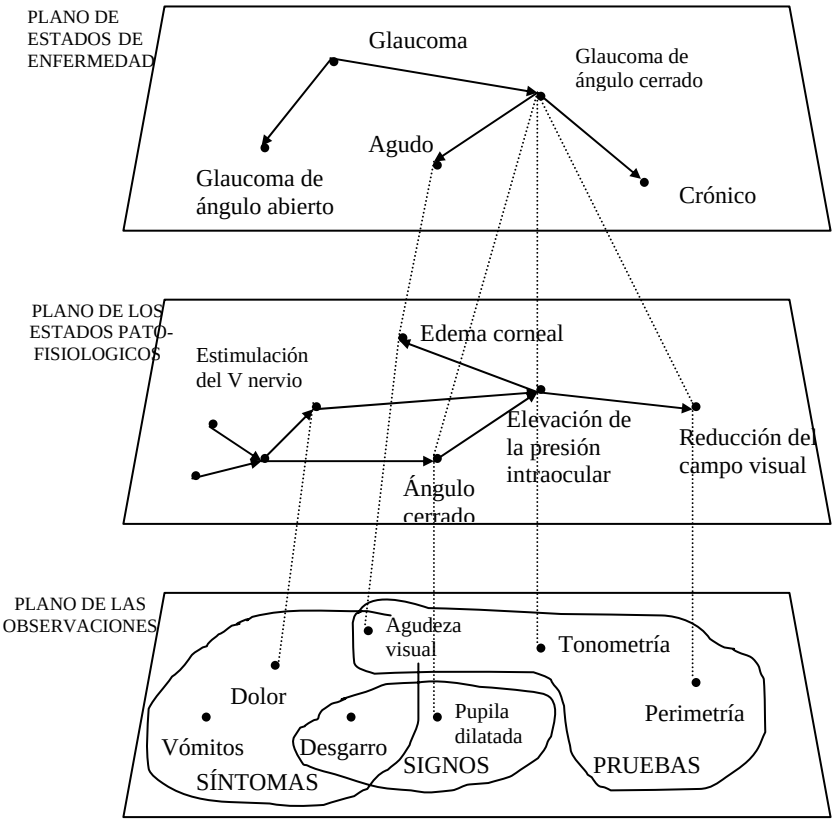
La característica esencial de una red causal es que representa un modelo en que los nodos corresponden a variables (edad, hipertensión, fiebre, ...) y los enlaces a relaciones de *influencia* (un enlace $X \rightarrow Y$ indica que el valor que toma *X influye* en el valor de *Y*); esta influencia suele ser un relación de *causalidad* (*X produce Y*). Se orientan sobre todo a problemas de diagnóstico.

7.4.1 El Sistema Experto CASNET(*Causal ASsociational NETwork*).

Su objetivo era ayudar a los médicos en el diagnóstico y el tratamiento del glaucoma (enfermedad ocular). Los programas de diagnóstico por ordenador anteriores estaban basados principalmente en métodos estadísticos y tenían deficiencias en cuanto al *razonamiento temporal* (eran incapaces de seguir la evolución de la enfermedad), en cuanto al *diagnóstico múltiple* (la mayor parte de ellos suponía que existía un único diagnóstico que excluía a los demás) y en cuanto a la *explicación* de sus conclusiones (no podían justificar cómo y por qué habían obtenido los resultados).

Uno de los rasgos más característicos de CASNET es la estructuración de los nodos en tres niveles:

- ***Observaciones***: incluye los síntomas (lo que el enfermo siente), los signos (lo que el médico observa) y los resultados de las pruebas de laboratorio.
- ***Estados patofisiológicos***: son las alteraciones que se producen en el funcionamiento de un órgano (en este caso, el ojo).
- ***Estados de enfermedad***: dentro de este nivel, las enfermedades se encuentran clasificadas en un árbol taxonómico; los nodos inferiores corresponden a especificaciones de los nodos superiores.



Existen varios tipos de enlaces:

- dentro del plano de las enfermedades, los enlaces indican relaciones de inclusión.

- los enlaces entre el plano superior y el mediano indican los estados patofisiológicos producidos por una determinada enfermedad.

- dentro del plano de los estados patofisiológicos, los enlaces corresponden a las relaciones de causalidad que determinan la evolución de la enfermedad.

- los enlaces entre el plano mediano y el inferior indican las observaciones a que da lugar cada estado patofisiológico.

Los diagnósticos pueden consistir en estados patofisiológicos (hipótesis simples) o en enfermedades (que son hipótesis complejas). El proceso consiste en interpretar los hallazgos en función de los estados patofisiológicos que han podido provocarlos. El programa asocia un valor concreto a cada zona, el cual será confirmado, indeterminado o descartado, según los estados patofisiológicos observados.

Este sistema experto es capaz, también, de ofrecer recomendaciones terapéuticas. Para ello se cuenta con un cuarto grupo de nodos, uno por cada tratamiento, que también pueden dibujarse dentro de un plano, aunque éste ya no sería paralelo a los anteriores. Para ello usa dos tipos de enlaces exclusivos. Uno de ellos se destina a unir los estados patofisiológicos y las enfermedades con los tratamientos oportunos, o para unir un tratamiento con las complicaciones a que puede dar lugar. El otro tipo de enlace se utiliza para unir los tratamientos entre sí con el fin de representar las interacciones, la toxicidad y las dependencias temporales.

7.4.2 Redes Bayesianas

7.4.2.1 Antecedentes

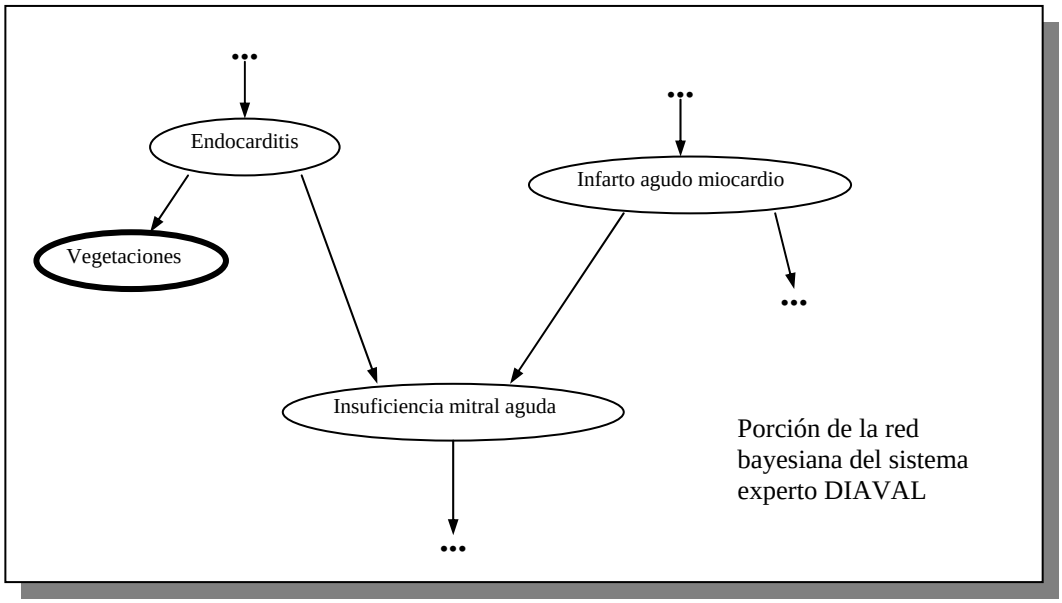
En los problemas de diagnóstico nos interesa hallar cuál es la hipótesis más probable de acuerdo con la evidencia disponible. Sin embargo, en las aplicaciones prácticas las probabilidades directas suelen ser más fáciles de obtener que las inversas. El teorema de Bayes nos permite pasar de unas a otras. Pero, es imposible de aplicar por la enorme cantidad de información que requiere. Por eso se introduce la hipótesis de que los diagnósticos son *exclusivos* (dos de ellos no pueden ser ciertos a la vez) y *exhaustivos* (no hay diagnóstico posible fuera de ellos). A pesar de estos cambios el teorema sigue siendo inaplicable por la necesidad de conocer todas las probabilidades que intervienen. Así, se vuelve tratable introduciendo como nueva hipótesis, la *independencia condicional*, que consiste en afirmar que, para cada diagnóstico, la probabilidad de encontrar un hallazgo es independiente de que se hayan encontrado otros. Por tanto, para un diagnóstico d_i y m hallazgos $E_1, \ldots, E_m$ se tiene la siguiente fórmula:

$$P(d_i|e_1, \ldots, e_m) = \frac{P(e_1|d_i) \cdot \ldots \cdot P(e_m|d_i) \cdot P(d_i)}{\sum_j P(e_1|d_j) \cdot \ldots \cdot P(e_m|d_j) \cdot P(d_j)}$$

De esta forma las redes bayesianas consiguen cierto grado de modularidad, lo cual permite computar la probabilidad de forma local.

7.4.2.2 el Sistema Experto DIAVAL

La red, que es esencialmente un modelo patofisiológico de las enfermedades cardíacas, consta en total de unos 300 nodos. Se indican los valores que toma cada variable y la información numérica, es decir, las probabilidades a priori y condicionales de la red. En DIAVAL, los nodos que no tienen descendientes corresponden a las observaciones (están representados por óvalos de trazo más grueso). La inferencia consiste en asignar los valores que se conocen y, a partir de esta información, realizar la propagación de la evidencia mediante el paso de mensajes entre nodos, con el fin de hallar la probabilidad a posteriori de los demás.



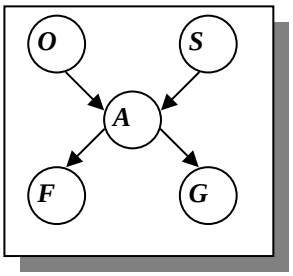
7.4.2.3 Definición de Red Bayesiana

Las redes bayesianas poseen una propiedad matemática llamada *separación direccional* que indica que la probabilidad de una variable *X* (una vez determinados los valores de los padres de *X*) es independiente de los demás nodos-variables que no son descendientes de *X*. Así, para el nodo F, la independencia condicional puede expresarse como:

$$P(f, o, s, g \mid a) = P(f \mid a)$$

o también como

$$P(f \mid a, o, s, g) = P(f \mid a)$$



Red Bayesiana: es un grafo dirigido acíclico conexo más una distribución de probabilidad sobre sus variables, la cual cumple la propiedad de separación direccional.

En las redes bayesianas la presencia de un enlace indica influencia causal, y la *ausencia* de un enlace indica implícitamente que no existen otras interacciones.

7.4.2.4 Inferencia en Redes Bayesianas

Vamos a describir un algoritmo, solamente aplicable en *poliárboles*, para fijar el valor de las variables conocidas con certeza y realizar un cálculo con el fin de hallar la probabilidad de las variables cuyo valor no se conoce con certeza. Para redes pequeñas se puede conseguir este objetivo realizando cálculos sencillos, aunque para redes mayores se utilizan algoritmos más eficientes basados en el paso de mensajes numéricos entre nodos a través de sus enlaces y donde el cálculo de la probabilidad se puede hacer de forma distribuida, asociando cada nodo a un procesador físico.

Podemos calcular la probabilidad de la siguiente manera:

$$P(a \mid \mathbf{e}) = \alpha.\pi(a).\lambda(a)$$

donde hemos introducido tres definiciones:

$$\pi(a) \equiv P(a, \mathbf{e}^{+}_{A})$$
$$\lambda(a) \equiv P(a| \mathbf{e}^{-}_{A})$$
$$\alpha \equiv P(\mathbf{e})$$

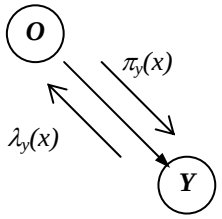
donde:

$\pi(a)$ indica cuál de los valores *a* es más probable teniendo en cuenta toda la información referente a las causas de *A*.

$\lambda(a)$ indica cuál de los valores de *a* explica mejor la información que se ha encontrado referente a sus efectos.

$\mathbf{e}^{+}_{A}$ es el subconjunto que contiene la evidencia que está "por encima" de *A* en el poliárbol.

$\mathbf{e}^{-}_{A}$ es el subconjunto que contiene la evidencia que está "por debajo" de *A* en el poliárbol.



Los valores de $\boldsymbol{\pi}$ y $\boldsymbol{\lambda}$ correspondientes a cada nodo se pueden calcular mediante el paso de mensajes (para la propagación de la evidencia) entre nodos vecinos.

7.4.3 Ventajas y Limitaciones de las Redes Causales.

La principal ventaja de razonar en base a un modelo del mundo real consiste en que el sistema posee un conocimiento profundo de los procesos que intervienen, en vez de limitarse a una mera asociación de datos e hipótesis.

Otra ventaja de los modelos causales es que pueden realizar tres tipos de razonamiento:

1. ***Razonamiento abductivo***: consiste en buscar cuál es la causa que mejor explica los efectos observados. Diríamos que se trata de un razonamiento "*hacia arriba*".
2. ***Razonamiento deductivo***: es el recíproco del anterior, pues va desde las causas hacia los efectos, es decir, "*hacia abajo*".
3. ***Razonamiento intercausal***: es un razonamiento "*en horizontal*". A partir de la seguridad de una causa se debilitan las otras posibilidades (se reduce la sospecha de otras). La evidencia disponible puede llevar a descartar todas las hipótesis menos una, que pasaría a ser el diagnóstico más verosímil.

En la práctica clínica, los tres tipos de razonamiento se realizan simultáneamente.

El principal inconveniente de las redes causales es la limitación en su rango de aplicaciones y, además, existen problemas de diagnóstico en que las redes causales no son aplicables debido a que no se conocen aún los mecanismos que intervienen.

El rendimiento es muy bueno en problemas de diagnóstico, pero no son válidas para planificación, control, o diseño. En estos casos es mejor usar reglas.

7.4.3.1 Ventajas e Inconvenientes de las Redes Bayesianas.

Además de las ventajas comunes a otros métodos de razonamiento causal, las redes bayesianas poseen una sólida teoría probabilista que les permite dar una interpretación objetiva de los factores numéricos que intervienen y dicta de forma unívoca la forma de realizar la inferencia. Sus inconvenientes son la limitación en cuanto al rango de aplicaciones. Necesitan un gran número de probabilidades numéricas, y normalmente no se dispone de toda esta información por lo que es necesario recurrir a estimaciones de expertos humanos. La presencia de bucles complica extraordinariamente los cálculos ya que los métodos de simulación estocástica resultan más eficientes que los métodos exactos, aunque resulta costoso (en términos de tiempo de computación) lograr el grado de aproximación deseado. Se hace necesario evitar una explosión combinatoria.

PROBLEMAS

• PROBLEMA 7.1:

Considérese la siguiente oración:

SAQUÉ EL LIBRO DE LA BIBLIOTECA.

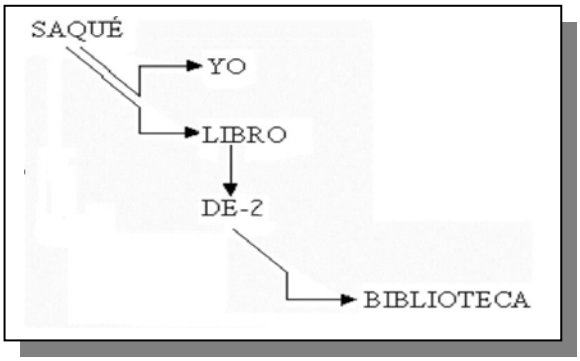
Esta sentencia puede tener dos significados distintos:

🕮 a) Saqué prestado un libro de la biblioteca.

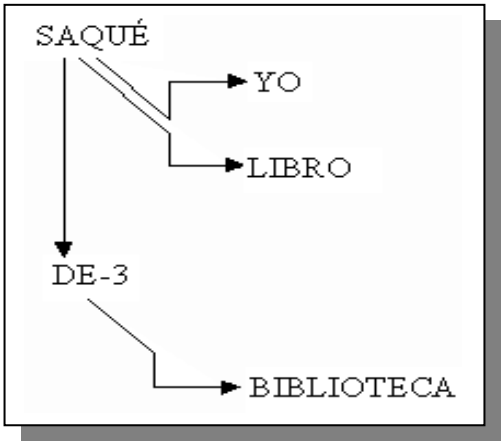
🕮 b) Saqué de la biblioteca un libro que me pertenecía y que había llevado allí para poder estudiar con él.

Utilizando el formalismo gráfico del modelo semántico de Quillian, representar los dos significados de la oración mencionada.

SOLUCIÓN:



a) En este caso "DE LA BIBLIOTECA" modifica al sustantivo "LIBRO", ya que dicho libro pertenece a los depósitos de la biblioteca. Por tanto, se tendría la representación de la figura de la izquierda.



b) Ahora "DE LA BIBLIOTECA" modifica al verbo "SAQUÉ". Por otra parte, la preposición "DE", que antes tenía un significado de pertenencia (DE-2), pasa a hacer referencia a una determinada localización (DE-3).

Los números en "DE-2" y "DE-3" se han escogido arbitrariamente.

• PROBLEMA 7.2:

Representar mediante grafos de dependencia conceptual las situaciones siguientes:

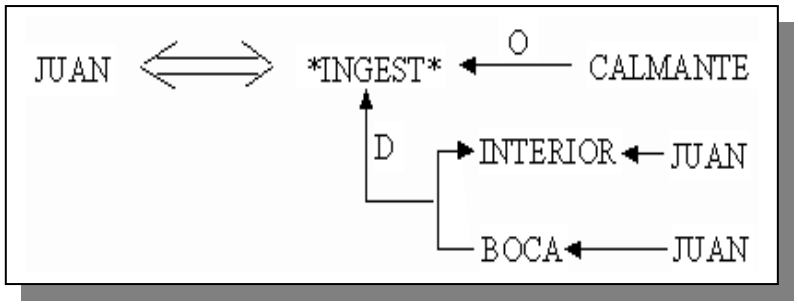
🕮 a) Juan toma un calmante.

🕮 b) Juan paga la factura.

🕮 c) Luis sacó el perro a la calle.

SOLUCIÓN:

¿Qué pretende este problema? Se consideran varios ejemplos de representación de oraciones simples mediante grafos de dependencia conceptual. En el problema planteado podrían construirse los siguientes grafos:

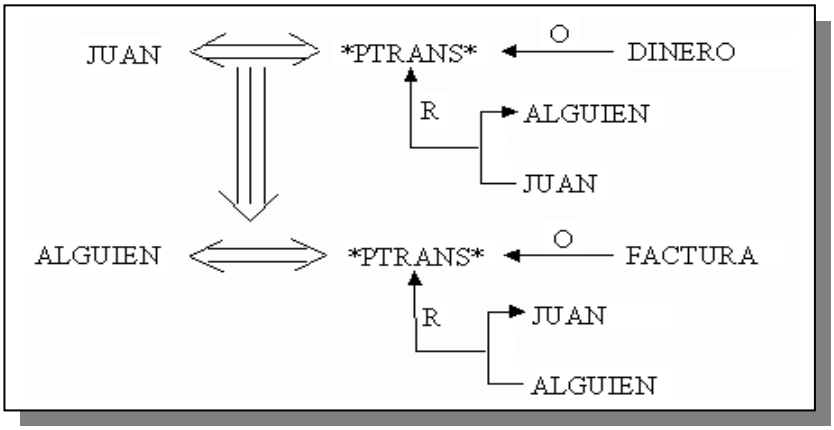


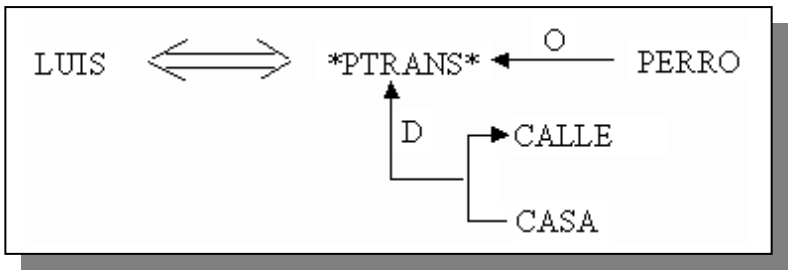
a) Juan toma un calmante.

En el grafo aparece explícitamente la forma en que el calmante es ingerido, es decir, desde la boca se introduce en el interior de Juan.

b) Juan paga la factura.

Obsérvese cómo el hecho de que Juan reciba la factura provoca el que pague el dinero correspondiente. Se está suponiendo, además, que el pago es en efectivo.





c) Luis sacó el perro a la calle.

• PROBLEMA 7.3:

Representar mediante:

🕮 a) una red de Shapiro

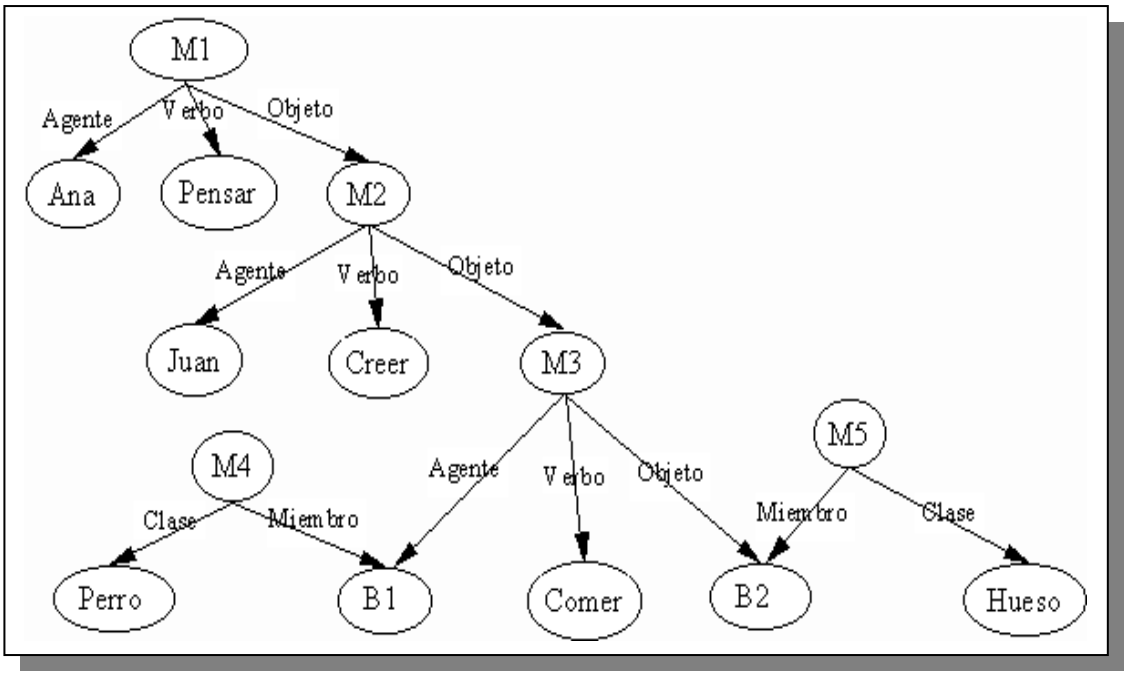
🕮 b) un grafo de Sowa

la siguiente frase:

"**Ana piensa que Juan cree que un perro está comiendo un hueso**"

SOLUCIÓN:

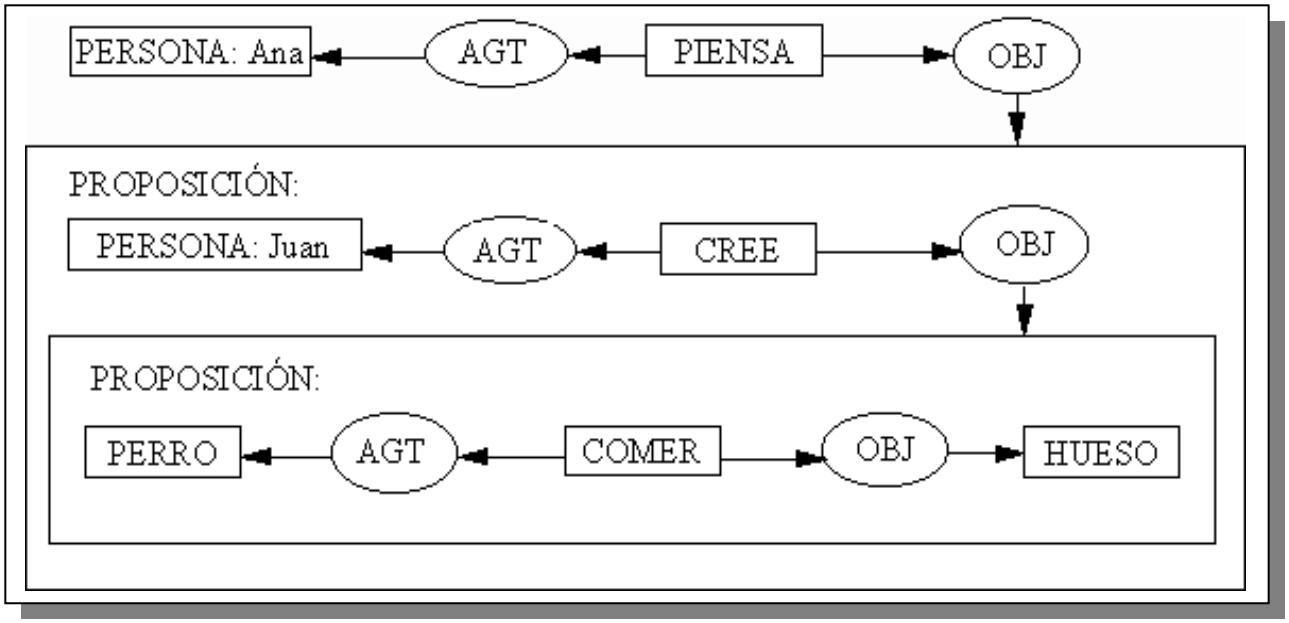
a) La frase del enunciado del problema se representaría según el formalismo de redes de Shapiro del siguiente modo:



Los nodos M1, M2, M3, M4 y M5 son nodos que representan proposiciones. M1 corresponde a la afirmación de que Ana está pensando algo. Lo que Ana piensa se representa en el nodo M2. Este nodo indica que Juan cree M3, siendo M3 la proposición correspondiente a un perro que come un hueso. Algunas cuestiones a tener en cuenta son las siguientes:

- Los nodos "Ana" y "Juan" no muestran que los conceptos que representan correspondan a personas. Si fuera necesario establecer dicha información, habría que hacerlo a través de proposiciones separadas.
- Los nodos B1 y B2 representan cierto perro y cierto hueso indeterminados.
- Los nodos M4 y M5 establecen que B1 es un miembro de la clase "Perro" y que B2 es un miembro de la clase "Hueso".
- Nótese que los conceptos que representan verbos sólo están unidos a nodos proposición y que las relaciones del tipo Agente u Objeto se enlazan, no a los nodos verbo, sino a los nodos proposición. Los componentes de una proposición se unen a un nodo externo que representa dicha proposición.

b) En el formalismo de Sowa, todo grafo que representa una proposición queda anidado dentro de un nodo concepto etiquetado con la palabra "proposición", del siguiente modo:



En este formalismo PERSONA: Ana (encerrado en un rectángulo) se refiere a algo llamado "Ana" de tipo PERSONA. Por otra parte, PERRO (encerrado en un rectángulo) se refiere a algo que no tiene nombre y que es de tipo PERRO. Obsérvese el tratamiento diferente respecto al apartado a) que se da ahora a los verbos en lo que se refiere a los enlaces asociados a los mismos.

8 MARCOS Y GUIONES

8.1 Concepto de Marco

8.1.1 La Propuesta de Minsky

Minsky propuso los marcos como método de representación del conocimiento y de razonamiento, para superar las limitaciones de la lógica a la hora de abordar problemas como la visión artificial, la comprensión el lenguaje natural o el razonamiento basado en el sentido común; problemas que requieren identificar estructuras complejas de conocimiento que representan una situación conocida. Al contrario que las reglas o la lógica, los marcos son unidades de almacenamiento suficientemente grandes como para imponer una estructura en el análisis de una situación.

Cada marco se caracteriza por un conjunto de campos (*slots*) que contienen distintas clases de información, parte de esta información indica como usar el marco, otra se refiere a lo que se espera que suceda a continuación, y otra parte indica que hacer cuando no se confirman tales expectativas. Un campo puede ser a su vez otro marco. Por ejemplo: Un marco *casa* en el que algunos de sus campos son *habitaciones*. Cada *habitación* será a su vez un marco que contendrá campos como *cama*, *nevera*, *sofá*...etc.

En su propuesta original, más que describir con detalle un método completo, trató de dar ideas y sugerencias, por ello del planteamiento original han surgido tantos modelos y tan diferentes entre sí. Las distintas formas de representación basadas en los marcos se distinguen entre si por la forma en que definen y utilizan los campos.

Los marcos están concebidos para tareas de reconocimiento. El mecanismo es el siguiente, la información recibida hace que se activen unos marcos y esta a su vez provoca la activación de otros marcos conectados con los primeros, dando lugar así a una red de activación, cuyo objetivo es predecir y explicar la información que se va a encontrar en esa situación. Este reconocimiento basado en expectativas se denomina a veces ***reconocimiento descendente***. Por ejemplo, si estamos en una *casa* que no conocemos, y encontramos una *cama*, al coincidir con un campo del marco *dormitorio*, podemos identificar de qué habitación se trata. Además, por los demás campos de este marco, podremos pensar que nos vamos a encontrar con una *mesilla*, un *armario ropero*...

Otra de las ideas novedosas de Minsky es la posibilidad de tener distintos marcos para definir una misma entidad desde distintos ***puntos de vista***. Por ejemplo, un *hombre* puede ser a la vez *profesor*, *padre de familia*, *cliente de un banco*...etc. Para cada uno de esos aspectos se definirá un marco con unos campos concretos.

El propósito de Minsky consistía en dar sugerencias e ideas más que en concretar los detalles de un cierto tipo de representación.

Comentario:

Para entender mejor el nombre de marco conviene recordar que el término original *frame* tiene muchos significados, condición o estado, armazón, estructura básica alrededor de la cual se construye "algo", conjunto de circunstancias que rodean un suceso, imagen en un rollo de película y de forma más general, los objetos en que se centra la atención de una cámara de cine. Realmente *frame*, en el sentido que le da Minsky, toma un poco de todos estos significados y por eso, al traducirlo por "marco" estamos perdiendo casi todos los matices del término original.

También conviene señalar que, según Minsky, cada marco posee varios *slots* o *terminals* que en castelano suelen denominarse "*campos*", y que significan "*ranura*", es decir, un "*hueco*" destinado a contener "algo que encaja allí".

8.1.2 El Sistema Experto PIP (Present Illness Program)

Se usa para ayudar en el diagnóstico de enfermedades. En este sistema cada enfermedad se representa por un marco, compuesto de los siguientes tipos de campos:

- **Nombre**: Identifica al marco.
- **Es-un-tipo-de**: Lo clasifica, indicando si es un estado fisiológico, enfermedad, etc.
- **Hallazgo**: Contienen los datos clínicos observados para una enfermedad dada.
- **No-debe-haber**: Reglas de exclusión, señalan que dato no debe presentarse en dicha enfermedad.
- **Criterios mayores y menores**: Parámetros de ponderación de datos clínicos.
- **Enlaces con otros marcos de la red** [campos "puede-estar-producido-por", "puede-complicarse-con"; " puede-producir"; " diagnóstico-diferencial"; …].

En un momento dado un marco puede estar en uno sólo de los 4 estados posibles: *durmiente, semiactivo, activo o aceptado*. Inicialmente todos están durmientes.

Un proceso de diagnosis comienza recogiendo información, y si se encuentra un síntoma, se *activan* todos los marcos caracterizados por ese hallazgo como candidatos, los marcos relacionados pasan al estado *semiactivo*. (Se genera un espacio de búsqueda limitado, evitando el problema de la explosión incontrolada). Posteriormente se evalúan las hipótesis generadas para encontrar el marco que mejor se adapte a los datos recogidos. El campo es-suficiente puede llevar a un marco como aceptado. El campo no-debe-haber puede excluirlo del conjunto de hipótesis. Si de los datos no se acepta ni se rechaza un diagnóstico entonces se pondera la evidencia a favor o en contra según los campos criterios-mayores y criterios-menores, hasta que la evidencia acumulada supere un cierto umbral, entonces el marco correspondiente pasa a considerarse aceptado.

Si no se llega a un umbral de aceptación se pasa a una 2ª fase de recogida de información, orientada por el conjunto de hipótesis disponible (los marcos activos) dando lugar, así, a un proceso cíclico de generación de preguntas y de ponderación de la evidencia hasta llegar a un diagnóstico satisfactorio.

Las ventajas de PIP frente a sistemas de diagnosis basados en árboles de decisión, donde cada nodo representa una pregunta con varias respuestas alternativas, son:

- Flexibilidad en el diálogo: el usuario puede introducir la información en el orden deseado.
- Generación de preguntas en función de la evidencia disponible.
- Diagnóstico de múltiples enfermedades (un árbol se basa en hipótesis excluyentes entre sí: no puede haber dos enfermedades presentes al mismo tiempo).

8.1.3 El Lenguaje KRL

Los autores pretendían que pudiera servir de base en la resolucion de gran variedad de problemas de inteligencia artificial (reconocimiento del habla o los sistemas expertos); su influencia ha sido notable en muchos de los sistemas y herramientas de la actualidad.

El propósito fundamental de KRL (*Knowledge Representation Language*) consistía en ofrecer los mecanismos necesarios para una **representación estructurada** del conocimiento. Por ello la información se agrupa en torno a los objetos o unidades "*units*", que constituyen el eje fundamental de la representación del conocimiento.

Los campos de una unidad pueden tener *valores por defecto*, que provienen de la *herencia de propiedades* o bien de la existencia de *prototipos* (unidad que en vez de corresponder a un objeto concreto representa un elemento típico de una clase).

Otro mecanismo de razonamiento muy importante es la *comparación* o *ajuste* ("*matching*"), similar a la comparación de patrones de las reglas.

Entre los objetivos básicos de KRL destaca la preocupación por la eficiencia (a costa de perder elegancia desde un punto de vista teórico). Una muestra de ello es la redundancia (incluir explícitamente cierta información para no tener que deducirla). Además, ofrece numerosas posibilidades de controlar el razonamiento. Una de ellas es la *asignación de procedimientos* o *procedural attachment* a los marcos o a sus campos los cuales pueden ser de dos tipos: los **sirvientes** (que son procedimientos activados por el marco para alcanzar cierto objetivo) y los **demonios** (se activan automáticamente en ciertas circunstancias).

Otras posibilidades de control del razonamiento son la profundidad de procesamiento variable (el usuario determina así cuantos marcos van a ser activados), la gestión de agendas, el establecimiento de niveles de prioridad, los métodos de naturaleza heurística.

8.1.4 Herramientas basadas en Marcos

Existen numerosas herramientas basadas en marcos, algunas como FrameKit usan sólo este tipo de representación, otras como ART, KEE, GoldWorks, Nexpert combinan marcos y reglas. Las propuestas actuales están inspiradas en la propuesta de Minsky aunque, en la práctica carecen de muchas de las

propiedades que él sugirió (la organización de los marcos mediante una red en la que unos marcos podían ocupar los campos de otros marcos dando lugar, a una *red de activació*n).

Los marcos originales estaban orientados al reconocimiento de imágenes y del lenguaje natural, mientras que los actuales se utilizan en la construcción de sistemas expertos, donde el diagnóstico mediante comparación de patrones desempeña un lugar menos importante.

8.2 Inferencia mediante Marcos

Los sistemas actuales basados en marcos se basan sobre todo en la herencia a partir de una red jerárquica. El punto de partida consiste en la creación de una red de marcos e instancias. Cada **marco** representa un concepto o una clase, y cada **instancia** representa un elemento dentro de la clase. En algunas herramientas, cada instancia puede pertenecer a varios marcos. Cada marco hereda los campos de todos sus antepasados en la red, y cada instancia hereda campos y valores de todos los marcos a los que pertenece. Algunas herramientas sólo permiten herencia descendente, mientras que otras admiten herencia bidireccional (el valor asignado a una instancia puede convertirse en valor por defecto del marco al que pertenece).

8.2.1 Facetas

Las facetas definen las propiedades de un **campo**. Entre las que suelen darse en las herramientas actuales destacan:

- **Valor por defecto**: es el valor que toma el campo (salvo que se indique lo contrario). En general se asigna al marco y es heredado por todas las instancias creadas posteriormente dentro de él.
- **Multivaluado**: indica si el campo es *univaluado* (la introducción de un nuevo valor desplaza el anterior) o *multivaluado* (pueden coexistir varios valores simultáneamente).
- **Restricciones**: limitan el conjunto de valores que puede recibir el campo. El sistema reconoce un error de asignación y toma las medidas oportunas.
- **Certeza**: indica la credibilidad del valor asignado al campo (es semejante al factor de cereza asociado a cada proposición dentro de un sistema basado en reglas).
- **Facetas de interfaz**: contienen texto, preguntas y mensajes destinados a la interacción con el usuario.
- **Demonios**: Por su especial importancia, se tratan más adelante.
- **Facetas definidas por el usuario**: (Por el programador)

8.2.2 Demonios

Los demonios son funciones o procedimientos que se invocan automáticamente al realizar ciertas operaciones sobre el valor de un campo. Se utilizan sobre todo para actualizar los campos que dependen de otros, manteniendo la consistencia del sistema. Cada campo puede tener asociado distintos tipos de demonios y cada uno se activa frente a un suceso concreto, entre ellos se encuentran los siguientes:

- ***Demonio de necesidad***: cuando se necesita conocer el valor de un campo y no hay ninguno asignado.
- ***Demonio de acceso***: cuando se solicita el valor de un campo aunque haya uno asignado.
- ***Demonio de asignación***: cuando se añade un valor a un campo.
- ***Demonio de modificación***: cuando se varía el valor de un campo.
- ***Demonio de borrado***: cuando se elimina el valor de un campo.

Los demonios no sólo relacionan entre sí los campos de una misma instancia; también sirven para actualizar la presentación gráfica del interfaz, buscan en una base de datos la información requerida, para controlar dispositivos externos, etc.

8.2.3 Puntos de Vista

Cada campo de una instancia puede tener asignado un valor, sin embargo algunas herramientas permiten que tenga varios valores correspondientes a distintos puntos de vista ("*view*s"). Este mecanismo de control del razonamiento está inspirado en la propuesta de Minsky, y su objetivo es similar a las *perspectivas* del lenguaje KRL, aunque la forma de implementación es completamente diferente.

Gran parte de las características de los marcos son ofrecidas por los lenguajes de **programación orientada a objetos**.

En un entorno que disponga de un lenguaje para programación orientada a objetos, los marcos son considerados objetos autónomos con capacidad para comunicarse mediante mensajes. Un mensaje consistirá en un mandato a un objeto para que ejecute un procedimiento local, que recibe el nombre de **método**, los métodos se diferencian de los demonios en que no se activan automáticamente, sino por la recepción del mensaje.

La diferencia esencial entre la perspectiva orientada a objetos y los Sistemas Basados en el Conocimiento es que en los primeros la arquitectura es plana de carácter distribuido, la solución del problema queda repartida sobre los distintos objetos y sus métodos. En los Sistemas Basados en el Conocimiento se parte de una arquitectura multicapa donde la capa de dominio contiene los objetos estructurados. Sobre esta se superpone otra responsable del esquema inferencial y de control.

Como nomenclatura o definición de marco se usa:

<marco> :: = <clase> | <instancia>
<clase> :: = **clase** <nombre - de - clase> **es superclase** <espec-super>; < atributos> **fin**
<instancia> :: = **instancia** <nombre - de - instancia> **es instancia de** <espec- super>; < atributos> **fin**
<espec - super> :: = <nombre - de - clase>{,<nombre - de - clase>}*| **nil**
<atributos**>** :: = <par - atributo - faceta>{; < par - atributo - faceta>}*| **<vacío>**
<par - atributo - faceta> :: = <nombre - de - atributo>=(<faceta>{<faceta}*)
<faceta> :: = <nombre - de faceta> < valor>| **demonio** < tipo - de - demonio > < llamada - a - demonio >
<nombre - de faceta> :: = **valor | valor - por -defecto**
<tipo - de -demonio> :: = **si - se - necesita | si - se - añade | si - se - borra**
< valor > :: = <constante - elemental> | < nombre - de - instancia >
< vacío > :: =

8.3 GUIONES

Estructura de conocimiento que contiene una secuencia estereotipada de acciones, se entiende acción en sentido amplio, es decir, incluyendo sus acepciones de conversación, pensamiento, actos de voluntad y sentimientos.

8.3.1 Planteamiento del Problema

Los guiones surgieron para ampliar las capacidades inferenciales de los *"grafos de dependencia conceptual" (gdc*) en la comprensión del lenguaje natural. Tienen, al igual que los marcos, una estructura suficiente para contener la descripción de situaciones estereotipadas (una frase necesita analizarse en un contexto). Se diferencian en que la composición de la estructura está predeterminada. Un guión está compuesto por una sucesión de escenas o eventos que describen el devenir esperado de los acontecimientos en una determinada situación. Un marco tiene campos que sirven para reconocer los objetos, y *un guión tiene escenas que sirven para reconocer situaciones*, por ello un guión puede considerarse como un tipo particular de marco, con las peculiaridades de que cada campo corresponde a un *suceso* y de que los campos-sucesos forman una *secuencia*.

8.3.2 Representación del Conocimiento

Los programas que utilizan guiones almacenan la información correspondiente a las escenas en *grafos de dependencia conceptual*; sin embargo los guiones van más allá de la representación de frases aisladas ya que enlazan secuencias de acciones.

Un guión se compone de:

- ***Escenas***: Los sucesos descritos en el guión, enlazados causalmente en forma de secuencia.
- ***Roles y objetos***: Se representan con variables y son propios de cada guión; corresponden a personajes y objetos que intervienen de modo que cada uno de ellos puede aparecer varias veces en la historia. Incluyen restricciones para indicar qué rol, objeto o lugar se puede asignar a las variables.
- ***Cabeceras***: Existen varios tipos, unas dan nombre al guión, otras representan condiciones, instrumentos y lugares; su misión es activar el guión en el momento oportuno.

  Existen varios tipos de cabeceras:

  ➔ La que da **nombre** al guión.

  Un ejemplo de su uso se puede dar en el proceso de comprensión de un texto en lenguaje natural, de modo que si apareciera en el mismo, por ejemplo, la

palabra "restaurante", el guión $RESTAURANTE sería inmediatamente activado. La activación de este guión permite una mejor comprensión del resto de la información presente en el texto.

➔La que representa **condiciones**.

Teniendo de nuevo en cuenta el guión $RESTAURANTE, una de las posibles condiciones para ir a un restaurante sería "tener hambre".

➔La que representa **instrumentos**.

Cualquier guión correspondiente al viaje en un determinado medio de transporte ($TREN, $AVIÓN...) podría activarse al tener constancia de que alguien está realizando cualquier tipo de trayecto. En este caso el tren, avión... son los posibles "instrumentos" utilizados en la acción descrita en el guión.

➔La que representa **lugares**.

El guión $CINE podría ser activado ante la aparición en un texto de palabras como: "película", "film"...

☞ ***Resultados*** : Son un conjunto de hechos que serán ciertos cuando se complete la secuencia de sucesos descritos en el guión.

8.3.3 Inferencia mediante Guiones

El objetivo del uso de guiones no se limita a la representación del conocimiento, sino que está orientado a la comprensión del lenguaje natural y se utilizan principalmente para la interpretación de textos escritos. Los métodos para comprobar la correcta interpretación de un texto (para comprobar que el sistema ha "comprendido el texto") son:

- Formulación de preguntas y ver si responde adecuadamente.
- Pedirle que repita el mismo texto con otras palabras (*paráfrasis*).

En el proceso de inferencia los pasos son:

- Seleccionar el guión que mejor explica la historia que se desea analizar. A través de las cabeceras de los guiones se decide cuál es el que pasa a estar activado.
- Asignar variables, es decir identificar los roles, objetos y lugares que intervienen. A partir de ahí, el guión "instanciado" permite extraer la información que no aparecía explícitamente en la historia escrita; en eso consiste precisamente la inferencia.
- Finalmente, a partir del guión se extrae toda la información que no se ha podido obtener del texto escrito.

La interpretación implica un proceso de reconocimiento, que en la práctica se implementa mediante la comparación de patrones. Existen tres métodos básicos de afrontar el problema de la comprensión de textos mediante guiones:

☞ *Ascendente*: Consiste en analizar palabra por palabra, activando los marcos que mejor explican toda la información que va apareciendo.

☞ *Descendente*: Selecciona un marco y trata de "encajar" en él la información, la que se ajusta a las expectativas es asimilada y el resto se ignora. Se usa cuando sólo se quiere extraer la información más relevante, pues es más eficiente y robusto, este método desprecia la información relevante que no se ajuste a las expectativas del guión.

☞ *Ascendente - Descendente*: Aprovecha las ventajas de los dos anteriores.

8.3.4 Ventajas, Inconvenientes y Extensiones

Las ventajas de uso de guiones son:

- Permite leer textos relativamente breves sobres temas concretos, extrayendo información.
- Al ser una extensión de los grafos de dependencia conceptual son independientes del idioma.
- El principal avance que han aportado los guiones frente a los grafos de dependencia conceptual es el control de inferencias (permiten integrar información para formar una historia, y realizan las inferencias relativas a la cadena causal de sucesos).

Sus inconvenientes principales son:

- Rigidez del mecanismo de representación: cada guión representa una secuencia fija de acciones.
- Incapacidad de los guiones para compartir información entre ellos.
- Incapacidad para representar motivaciones e intenciones.

Para superar la rigidez de los marcos, *Schank* aportó una solución consistente en representar el conocimiento en forma de *paquetes de organización de memoria* o **MOP's**. Al igual que los guiones, cada MOP viene descrito por una secuencia de escenas; la diferencia es que existen enlaces entre los distintos MOP's para detectar partes comunes. Estos enlaces permiten utilizar simultáneamente los MOP's aplicando en cada momento el que mejor explica los acontecimientos que aparecen en la historia. A la hora de interpretar una historia no es probable que exista un guión o un MOP que se ajuste completamente al texto, por lo que en la práctica será necesario adaptar alguno de los que existen. Si estas modificaciones se almacenan de forma ordenada, el sistema podrá aprovechar la experiencia adquirida en la comprensión de nuevos casos, teniendo así un mecanismo de aprendizaje mediante MOP's. Los MOP's sirvieron de base para el *razonamiento basado en casos* (RBC), que consiste en recordar alguna de las soluciones aplicadas a problemas del pasado y adaptarla a un nuevo problema.

8.4 COMENTARIOS

Existen dos modos en los que los **marcos** han sido y son utilizados:

1. ***Patrones para la comparación***: Cuando la información disponible indica que el problema planteado (diagnosticar una enfermedad o comprender un texto) se ajusta a un marco, el resto de la información contenida en dicho marco nos permite inferir la presencia de otros elementos.
2. ***Almacenes de información***: En este caso los marcos significan ante todo una forma estructurada de almacenar información. La distinción entre marcos e instancias y su organización en forma de red jerárquica permite utilizar la herencia de propiedades como mecanismo de razonamiento por defecto.

El lenguaje KRL participa de ambos paradigmas, pero la mayor parte de las herramientas basadas en marcos actuales se ajustan al segundo paradigma.

Valoración

Es importante la comparación de los mecanismos de representación del conocimiento. Las reglas surgen como aplicación de la lógica, reduciendo la expresividad con el fin de mejorar la eficiencia. Los esquemas basados en redes están inspirados en psicología (redes semánticas de Quillian) o en la lingüística (grafos de dependencia conceptual de Schank), otros tratan de implementar las características de la lógica de primer orden (redes de Hendrix, Shapiro y Sowa). Los marcos representan el conocimiento de forma estructurada. La información relativa a un objeto queda reunida en un marco en vez de estar dispersa en un conjunto de proposiciones sin ninguna estructura, significando una mayor eficiencia. Los marcos aportan nuevas posibilidades de razonamiento frente a la lógica con la capacidad para reconocer entidades y situaciones, los marcos son capaces de establecer una conclusión incluso con información incompleta.

PROBLEMAS

PROBLEMA 8.1:

En un periódico de información general se reciben noticias de sucesos de contenido muy variado. Se desearía disponer de algún método que captara el conocimiento general sobre dichos sucesos. Utilizar un sistema de marcos para llevar a cabo la labor mencionada.

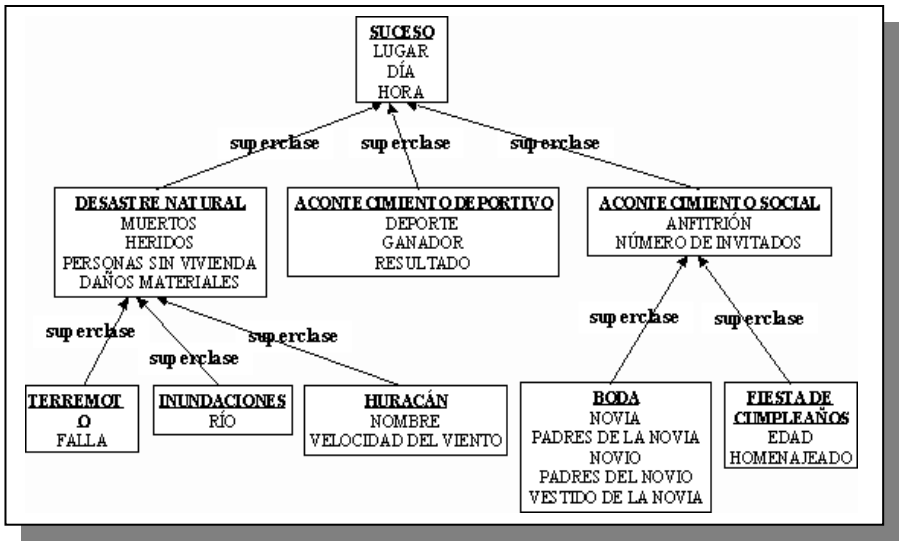
SOLUCIÓN:

¿Qué pretende este problema? Se hace hincapié en la estructuración del conocimiento de un dominio mediante una red jerárquica de conceptos dotada de herencia de propiedades.

El conocimiento general sobre las características de los sucesos que pueden dar lugar a una noticia se puede representar mediante una jerarquía de marcos donde cada nodo represente un tipo de suceso más o menos general y cada enlace una relación de superclase, de manera que se pueda construir una red de sucesos. En la parte alta de esta red figurarían los sucesos más generales, de los que otros sucesos más bajos en la misma podrían heredar información. A modo de ejemplo, en la red de sucesos debería quedar reflejada información del siguiente tipo:

- *El concepto más general que se manejará será el de suceso. Todo suceso se caracterizará por el lugar, día y hora en que ocurrió.*
- *Existen diferentes tipos de sucesos: desastres naturales, acontecimientos deportivos, acontecimientos sociales...*
- *Si se está considerando un desastre natural, habrá que tener en cuenta el número de muertos producidos, heridos, personas que se han quedado sin vivienda, daños materiales...*
- *En todo acontecimiento deportivo interesará reseñar el deporte del que se trata, quién es el ganador, el resultado final...*
- *En un acontecimiento social podría ser interesante determinar el anfitrión, número de invitados...*
- *Dentro de los desastres naturales cabría destacar los terremotos, inundaciones, huracanes...*
- *En cuanto a cualquier terremoto es interesante conocer la falla que lo produjo, la magnitud del mismo...*
- *Con respecto a las inundaciones es importante conocer, por ejemplo, el río que las ha producido...*
- *En lo que se refiere a los huracanes, dos de sus características principales son su nombre y la máxima velocidad que ha alcanzado el viento.*
- *Algunos tipos de acontecimientos sociales son las bodas, cumpleaños...*
- *En una boda podría interesar saber quién es la novia, el novio, sus respectivos padres, cómo es el vestido de la novia...*
- *Finalmente, en una fiesta de cumpleaños lo más reseñable es quién es la persona que cumple años y cuál es su edad.*

Toda la información mencionada con anterioridad aparece de forma explícita en la jerarquía de marcos de la siguiente figura, donde los nombres de las clases se refieren a distintos tipos de sucesos y los campos de los que consta cada clase vienen a representar las características más importantes de cada suceso. En dicha figura cada enlace une un suceso con otro superior en la jerarquía, que es más general que el anterior.



Como se puede observar en el árbol creado, en un nodo como pudiera ser "TERREMOTO" no es necesario señalar el lugar donde éste tuvo lugar al ser ésta una característica que hereda nodos superiores en la jerarquía.

• PROBLEMA 8.2:

Dado el sistema de marcos construido en el problema anterior, ¿de qué capacidades habría que dotar al mismo para que constituyera un sistema automático para resumir noticias?

SOLUCIÓN:

¿Qué pretende este problema? Se presenta en este problema un ejemplo de cómo puede ser útil añadir a una red jerárquica de conceptos, conocimiento procedimental del dominio a través de los llamados "demonios".

Para conseguir automatizar el proceso mencionado habría que ir completando las siguientes etapas:

1) Recepción de la noticia.

2) Identificación del tipo de noticia.

3) Llenado de los campos correspondientes de la jerarquía de marcos a partir de la información que aparece en la noticia.

4) Llenado de los huecos de un patrón resumen del suceso con los valores hallados para los campos de la jerarquía de marcos.

El paso de la etapa **1)** a la **2)** consiste en saber si la noticia hace referencia a una boda, terremoto... Para ello podría bastar con analizar el título de la noticia y las primeras frases de la misma. Una vez identificado el tipo de noticia, habría que llenar aquellos campos de la jerarquía de marcos construida en el problema anterior que tuvieran relación con el tipo de suceso en cuestión (paso de la etapa **2)** a la **3)**). Esto se puede conseguir mediante demonios de necesidad (procedimientos que actúan cuando se pretende conocer el valor de un campo y no puede obtenerse directamente o por herencia en la jerarquía). Algunos de los demonios mencionados serían:

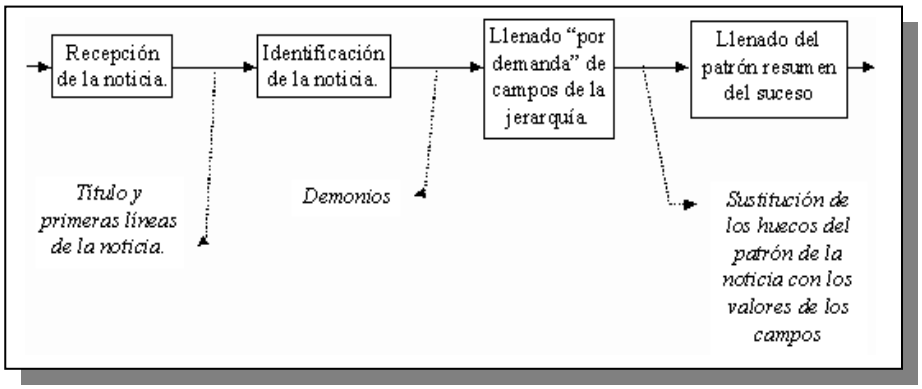
- *Para llenar el campo HORA cuando se construye la noticia SUCESO, encuentre un número que incluya dos puntos y escríbalo en el campo.*
- *Para llenar el campo DÍA cuando se construye la noticia SUCESO, encuentre una palabra como "hoy", "ayer", "mañana" o el nombre de uno de los días de la semana y escríbalo en el campo.*
- *Para llenar el campo LUGAR cuando se construye la noticia SUCESO, encuentre un nombre que aparezca en un diccionario de lugares geográficos y escríbalo en el campo.*
- *Para llenar el campo DAÑOS MATERIALES cuando se construye la noticia DESASTRE NATURAL, encuentre el número que está junto a la palabra "euros" y escríbalo en el campo.*
- *Para llenar el campo MUERTOS cuando se construye la noticia DESASTRE, encuentre un entero cercano a una palabra que tenga que ver con muerte y escríbalo en el campo.*
- *Para llenar el campo FALLA cuando se construye la noticia TERREMOTO, encuentre un nombre propio cercano a la palabra "falla" y escríbalo en el campo.*
- *Para llenar el campo MAGNITUD cuando se construye la noticia TERREMOTO, encuentre un número decimal entre 1.0 y 10.0 y escríbalo en el campo.*

Finalmente, la noticia resumida se obtendría llenando los huecos del patrón correspondiente al suceso identificado en la etapa 2) con los valores de los campos obtenidos en la etapa

anterior, la 3). Un ejemplo de patrón resumen correspondiente a un terremoto podría ser el siguiente:

> Ocurrió un terremoto en ‹valor del campo LUGAR›, el día de ‹valor del campo DÍA›. Hubo ‹valor del campo MUERTOS› muertos y ‹valor del campo DAÑOS-MATERIALES› euros de pérdidas materiales. La magnitud fue de ‹valor del campo MAGNITUD› en la escala Richter; la falla que provocó el terremoto fue la de ‹valor del campo FALLA›.

Gráficamente el proceso global que se ha descrito sería de la siguiente forma:

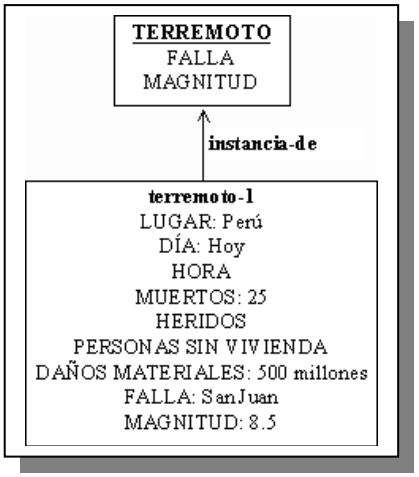


Considérese el siguiente ejemplo de aplicación del proceso por etapas visto para la siguiente noticia:

TERREMOTO EN PERÚ

Hoy, un serio terremoto de magnitud 8.5 sacudió Perú; el percance mató a 25 personas y ocasionó daños materiales por valor de 500 millones de pesetas. El Presidente del Perú dijo que el área más afectada, cercana a la falla de San Juan, ha sido durante años una zona de peligro.

En primer lugar, el sistema deduciría que el suceso noticiable se corresponde con un terremoto examinando el título de la noticia. A continuación se crearía una instancia de la clase TERREMOTO de manera que al intentarse hallar los valores de sus campos se haría uso de los demonios existentes en la jerarquía de marcos. El resultado final sería la creación de la siguiente instancia de la clase TERREMOTO:



Una vez creada la instancia, no habría más que llevar a cabo un proceso de sustitución de los valores que aparecen en los campos de la misma en los huecos de la noticia resumen patrón, que quedaría de la siguiente forma:

Patrón resumen de terremoto sustituido

Ocurrió un terremoto en *Perú* el día de *hoy*. Hubo *25* muertos y *500 millones* de euros en pérdidas materiales. La magnitud fue de *8.5* en la escala Richter; la falla que provocó el terremoto fue la de *San Juan*.

• **PROBLEMA 8.3:**

Establecer un guión que refleje los eventos producidos durante el acto de entrada a un cine.

SOLUCIÓN:

El guión $ENTRADA_A_UN_CINE constaría de los siguientes *roles* o *personajes*:

P: aquella persona o grupo de personas que van a ver la película

T: persona que está en las taquillas

E: persona que controla la entrada al cine

A: acomodador

Cada situación real particular dará lugar a que las variables anteriores acepten unos valores concretos diferentes.

Los *objetos* típicos de este guión son:

dinero

entradas

butacas

pantalla de cine

Como *condiciones* previas para una posible activación del guión, se tendría:

P tiene dinero.

P desea ver una película.

Los *resultados* que se pueden esperar una vez completada la última escena serían:

P tiene menos dinero.

T tiene más dinero.

P ha sacado una determinada impresión de la película.

Finalmente, el guión $ENTRADA_A_UN_CINE estaría formado por las siguientes *escenas*:

Escena 1: Compra de la entrada (o entradas)

P hace cola frente a las taquillas del cine.
P compra las entradas a *T*.
P se dirige hacia la entrada del cine.

Escena 2: Entrada al cine

P entrega su entrada a *E*.
E marca la entrada.
E indica a *P* la entrada a la sala de proyección.

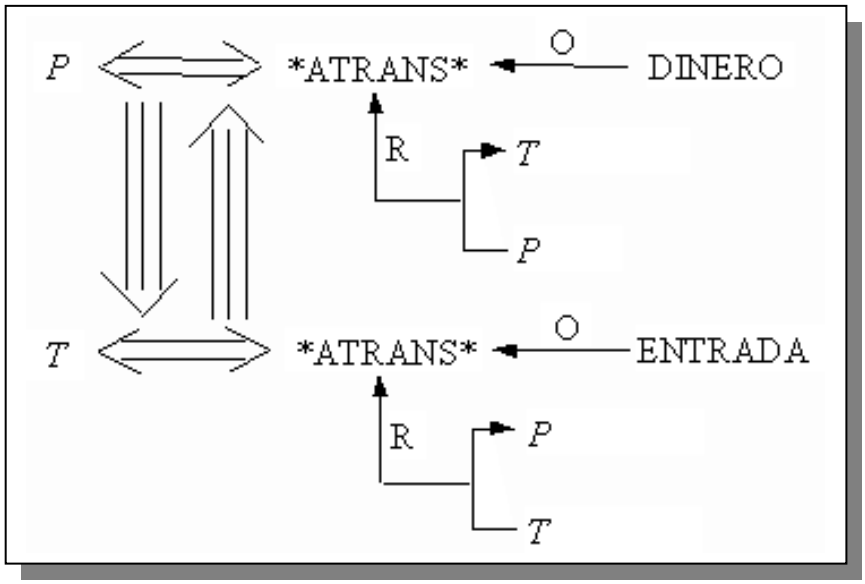
Escena 3: Tomar asiento

P entrega su entrada a *A*.
A indica a *P* qué asiento debe tomar.
P se sienta.

Escena 4: Proyección de la película

P ve la película.
P sale del cine.

Como ya ha quedado reseñado en un problema previo, cada una de las acciones especificadas en las escenas de un guión se representan internamente mediante grafos de dependencia conceptual de Schank. A modo de ejemplo, la acción en la que *P* compra la entrada se representaría de la siguiente forma:



9 SISTEMAS EXPERTOS

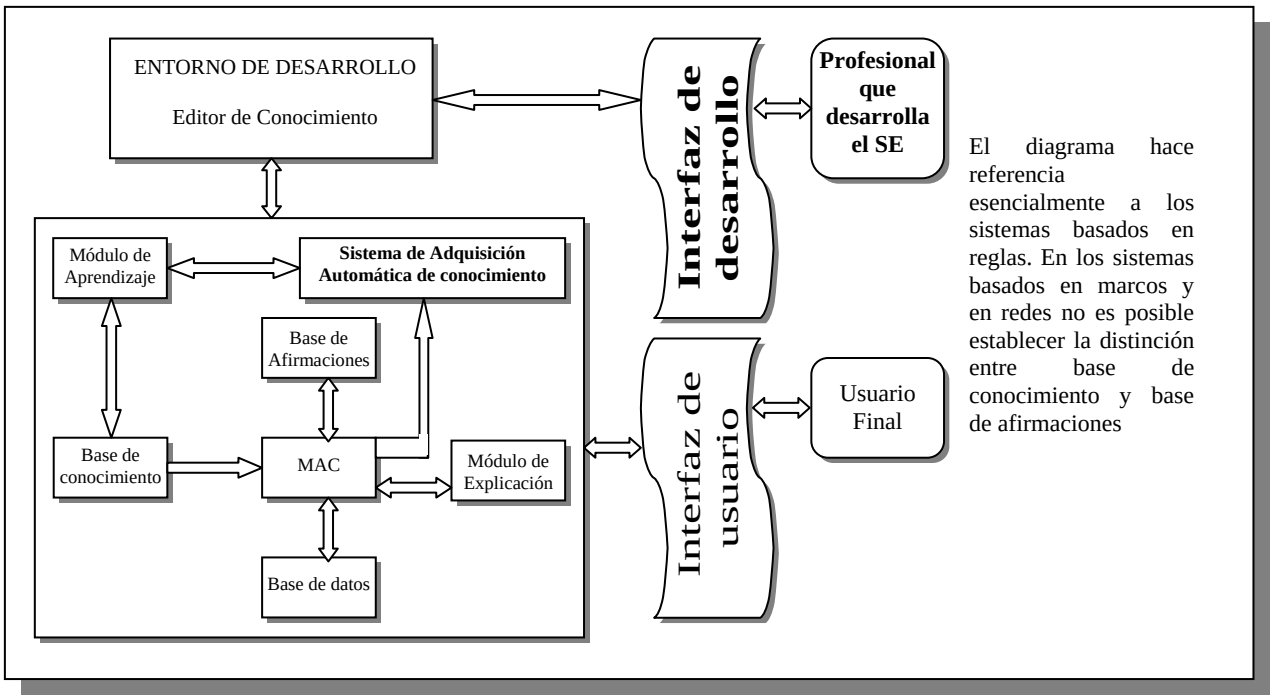
Un sistema experto es un programa de ordenador que codifica un modelo de conocimiento de un experto humano en un campo reducido. Un Sistema Experto empieza con un modelo conceptual a nivel de conocimiento y termina en una implementación, tras una etapa de codificación. La arquitectura de un sistema experto de la primera época es de hecho la de un sistema basado en reglas. Con el uso de los marcos y las reglas aparecen dudas en la distinción entre conocimiento estático y dinámico. La introducción de los Sistemas Expertos supuso cambios esenciales en la Inteligencia Artificial:

(a) Abordar directamente la complejidad del mundo real, con tareas de diagnóstico y planificación en dominios científico-técnico de extensión limitada.
(b) Aceptar que hay que modelar el conocimiento humano no analítico y hacer énfasis en la representación de ese conocimiento de forma separada del uso del mismo en inferencia.
(c) Comprobar que, mecanismos sencillos de inferencia tales como el encadenamiento de reglas, eran capaces de resolver satisfactoriamente problemas complejos en dominios reales, tales como el diagnóstico médico.

Los pasos necesarios para realizar un Sistema Experto son:

- Ver en qué tareas y escenarios es necesario utilizar Sistemas Expertos.
- Adquirir el conocimiento de los expertos humanos que realizan la tarea.
- Modelar ese conocimiento.
- Usar entornos de desarrollo para obtener prototipos.
- Validar el funcionamiento de los prototipos y evaluarlos, facilitando la posterior actualización del conocimiento.

9.1 Estructura básica



La estructura básica de un sistema experto consta de los siguientes elementos:

(1) **La *base de conocimiento*** *(BC)*: es estática y contiene la representación modular y computable de aquellos fragmentos de conocimiento que hemos sido capaces de modelar y representar a través del diálogo con el experto humano que realiza la tarea para la que ahora queremos desarrollar un sistema experto. La BC de todo sistema experto estará compuesta por un conjunto de reglas o marcos o por una combinación de ambos. En sistemas *híbridos* (simbólico-conexionistas) la BC puede incluir también una red neuronal.

(2) **Los *mecanismos de Aplicación del Conocimiento*** *(MAC)*: dependen de la aplicación y de la forma de representación del conocimiento. Si el conocimiento se representa sólo con reglas, la inferencia se realiza seleccionando reglas que se van a ejecutar después. El encadenamiento puede ser dirigido por datos (hacia delante) o dirigido por hipótesis (hacia detrás). Si el conocimiento está representado usando marcos, la BC estará organizada en una jerarquía para las distintas clases de objetos estructurados necesarias en la aplicación. La inferencia ahora está basada en la herencia directa y por defecto, el enlace dinámico mediante reglas en ciertos campos de los marcos y otros procesos de control de la inferencia y de solución de conflictos, tales como las *agendas*.

(3) **La *base de afirmaciones*** o conjunto de hechos ciertos (BA) es dinámica. Es una memoria de trabajo donde el mecanismo de inferencia (MAC) almacena las conclusiones transitorias que va obteniendo y donde, a su vez, busca las premisas que le permiten obtener otras nuevas. El contenido de la BA es distinto para cada consulta que se realiza.

Lo que hacen los hechos ciertos de la BA es activar determinados módulos de conocimiento en los que podrían estar involucrados, haciéndolos explícitos y produciendo nuevos hechos que, aunque estaban implícitos en la BC, sólo ahora aparecen activos, como consecuencia de los hechos de entrada.

(4) **La *interfaz de usuario*** de la cual depende a menudo la aceptación o el rechazo de un programa. Debe ser potente, fácil de manejar y de uso agradable. Su especificación funcional y diseño final están asociados al entorno de desarrollo usado y a las facilidades de explicación y edición de conocimiento de las que se quiera dotar al sistema.

(5) Se incluye en general una ***base de datos*** (BD), en el sentido convencional del término, para almacenar y recuperar de forma eficiente la información adicional propia de cada aplicación y no incluida ni en la BA ni en la BC.

Además de estos componentes que se encuentran en todos los Sistemas Expertos, hay tres elementos más que están incluidos en algunos casos. Son el **Módulo de Explicación** (ME), el **Módulo de Aprendizaje** (MA) y el **Módulo de Adquisición de Conocimiento** (AC).

Este esquema es utilizado básicamente con reglas; en los sistemas basados en marcos y redes, existe una separación entre inferencia y conocimiento, pero la información del dominio y las conclusiones obtenidas se almacenan en los marcos o en los nodos y enlaces de la red, por lo cual no se puede distinguir entre la Base de Conocimientos y la Base de Afirmaciones.

9.2 Características de un Sistema Experto

Cuando se termina la fase de desarrollo, tenemos efectivamente un programa, junto con la codificación del conocimiento cuya semántica se ha quedado en el dominio del observador. Como tal programa posee, al menos, las siguientes características distintivas:

- ***Competencia en su campo***: es el significado de experto. Es necesario que pueda resolver problemas con una eficiencia y calidad comparables a las de un experto humano.
- ***Dominio reducido***: ésta es la nota distintiva de los sistemas expertos frente a otros programas de inteligencia artificial. El limitarse a un dominio reducido es un requisito para alcanzar la competencia.
- ***Capacidad de explicación***: es la capacidad de explicar cómo ha resuelto el problema, es decir, qué método ha aplicado y por qué lo ha aplicado. Un SE debe ser capaz de explicar al usuario el proceso de razonamiento empleado. Los resultados en este apartado son modestos y dependen de los avances en el campo del aprendizaje simbólico.
- ***Tratamiento de la incertidumbre***: es una exigencia que se deriva de la complejidad de los problemas que van a aborda los sistemas expertos.
- ***Flexibilidad en el diálogo***: es deseable que los sistemas expertos tengan esta capacidad, llegando en la medida de lo posible a comunicarse (entender y expresarse) en lenguaje natural como un experto humano.
- ***Representación explícita del conocimiento***: es necesaria para considerar que un sistema está basado en conocimiento. Todo programa de ordenador se basa en el conocimiento de problema a tratar, pero en este caso queremos decir que la finalidad del programa es representar explícitamente el conocimiento.

9.3 Ventajas y Limitaciones

Podemos afirmar que las ventajas de los sistemas expertos son las siguientes:

- ***Permanencia***: los expertos humanos pueden morir, cambiar de empresa o perder facultades lo que no puede ocurrir con un sistema experto.
- ***Duplicación***: el experto humano se encuentra en un único lugar físico y es irreproducible, mientras que una vez construido un sistema experto, podemos fabricar un número ilimitado de copias destinadas a todos los lugares donde sean necesarias.
- ***Fiabilidad***: un sistema experto responderá siempre de la misma manera ante un cierto problema, mientras que un experto humano puede estar condicionado por factores emocionales, prejuicios personales, tensión, fatiga, etc.
- ***Rapidez***: El experto humano será rápido en los casos habituales, pero no en los casos más raros. En cambio el SE será rápido en todos los casos.
- ***Bajo coste***: aunque puede resultar caro inicialmente construir un sistema experto, una vez construido produce grandes beneficios.

En contrapartida los sistemas expertos presentan también grandes carencial frente a los seres humanos:

- ***Sentido común***: para un ordenador no hay nada obvio. Un SE carece de esta característica. Puede sacar conclusiones absurdas.
- ***Flexibilidad***: El experto humano es flexible y puede adaptarse a las nuevas circunstancias, situaciones inesperadas, buscando nuevas soluciones. Los SE son rígidos.
- ***Lenguaje natural***: todavía estamos muy lejos de tener un sistema que pueda formular preguntas flexibles y mantener una conversación informal con un usuario o con un paciente.
- ***Experiencia sensorial***: los sistemas expertos de la actualidad se limitan a recibir información mediante un teclado y un ratón. Sólo algunos disponen de tratamiento de imágenes, y es incomparable con la capacidad de visión humana. Lo mismo puede decirse de la capacidad auditiva, táctil y olfativa.
- ***Perspectiva global***: un experto humano es capaz de detectar inmediatamente cuáles son las cuestiones centrales y cuáles son secundarias (separando los datos relevantes de los detalles insignificantes).

Además existen estas otras limitaciones:

- ***Falta de capacidad de aprendizaje***: los expertos humanos son capaces de aprender de la experiencia, por caminos que aún no pueden ser modelados.
- ***Capacidad de manejar conocimiento no estructurado***: el experto humano organiza y usa la información y el conocimiento presentados de forma poco ordenada.
- ***Funciones genuinamente humanas***: por ejemplo, todo lo relacionado con el lenguaje natural, la formación de conceptos, el conocimiento de sentido común y la creación queda fuera de los sistemas expertos, al menos en el estado actual del conocimiento.

Por tanto, y en la actualidad, la idea de sustituir al humano por la máquina es aún irrealizable. Aunque el objetivo principal no es que la máquina duplique o sustituya al humano sino que colabore con él.

PROBLEMAS