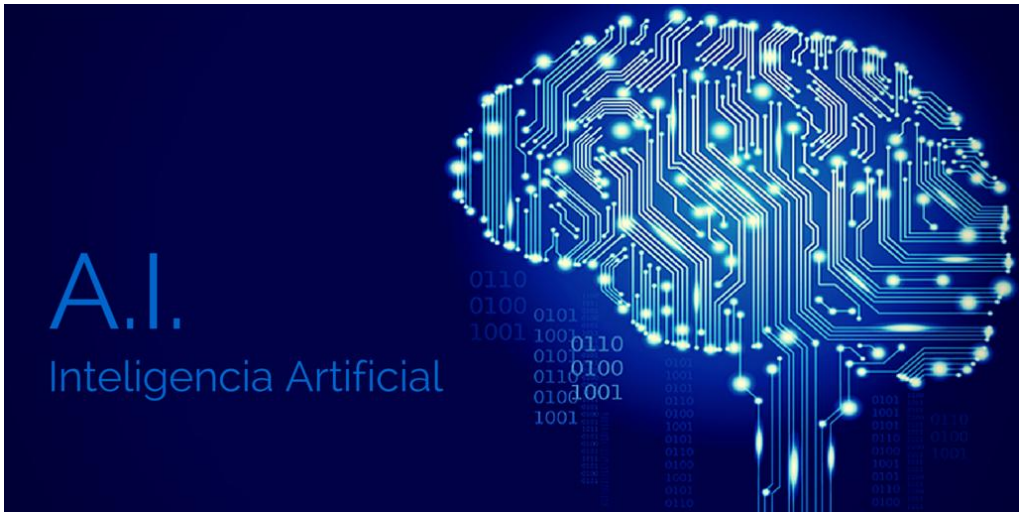
UNED

Inteligencia Artificial

Resumen del texto base de la asignatura



Manuel Dotor

23/09/2015

Este resumen está basado en el texto base del libro "INTELIGENCIA ARTIFICIAL Métodos, técnicas y aplicaciones", presentaciones del tutor de Sevilla, exámenes del equipo docente junto con otros apuntes míos y de diferentes alumnos.

Contenido

T1 – Perspectiva conceptual.

Capítulo 1 del libro base

1.2 La IA como Ciencia y como IC

Los cuatro grandes objetivos de la IA son modelar, formalizar, programar e implementar máquinas soporte capaces de interactuar de forma no trivial con el medio.

Hay que especificar qué hace que un problema sea "complejo" y por qué su solución debe de exigir procesamiento simbólico. Dos de las formas posibles de contestar a esta pregunta han dado lugar a las dos aproximaciones dominantes en IA: la simbólica o representacional y la del conexionismo situado. La primera hace énfasis en la vía descendente y en el uso de conceptos del lenguaje natural (hechos y reglas) para representar el conocimiento necesario para resolver problemas de decisión que no necesitan un robot como componente imprescindible en su implementación. La segunda hace énfasis en la vía ascendente y en el uso de conceptos de más bajo nivel semántico.

La IA tiene dos concepciones claramente diferenciadas, las cuales tienen también sus propios objetivos, métodos y enfoques diferentes. Por ello cabe distinguir entre los objetivos de la IA como ciencia e IA como ingeniería (IC).

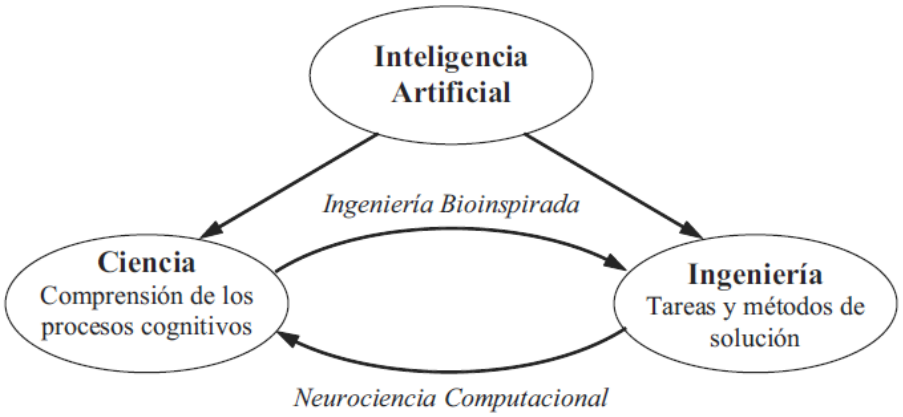


Figura 1.1: Distinción entre IA como ciencia e IA como ingeniería.

1.2.1. La IA como Ciencia

La IA entendida como ciencia es básicamente una tarea de análisis. Su ámbito de conocimiento engloba el conjunto de hechos asociados a la neurología y la cognición, como pueden ser la percepción, la memoria, el lenguaje, etc.

La perspectiva científica busca el desarrollo de una teoría computable del conocimiento humano, es decir, una teoría que pueda ejecutarse en un sistema de

cálculo con fines predictivos (construir programas que emulen el comportamiento inteligente de los humanos). Se ve, por tanto, el ambicioso objetivo perseguido por este enfoque científico de la IA, y por qué se llama hipótesis fuerte a esta idea.

Existe un enfoque conocido como hipótesis débil, que persigue un objetivo más modesto, como es el desarrollar máquinas que exhiban un comportamiento inteligente, no necesariamente emulando el pensamiento humano.

La visión de la IA como ciencia, que se basa en considerar que el término inteligencia es muy general y de carácter precientífico. Consecuentemente, nos parece conveniente:

(1) descomponerlo en un conjunto de habilidades parcialmente autónomas y accesibles al estudio experimental (percepción, abstracción, razonamiento recursivo, auto-referencia,...)

(2) aumentar el esfuerzo dedicado al estudio de los fundamentos de cada una de esas habilidades y a la construcción de un soporte teórico adecuado para las mismas.

(3) intentar desarrollar nuevas herramientas conceptuales, formales y computacionales para describir adecuadamente los procesos mentales que hayamos sido capaces de identificar, analizar y controlar experimentalmente.

1.2.2 La IA como Ingeniería

La IA como ingeniería también tiene sus dificultades. La primera, es que el objeto formal de la IC (Ingeniería del Conocimiento) es el propio conocimiento, y éste es pura forma. La idea se basa en la estructura relacional y en un punto común para los distintos observadores, que dotan de significado a los símbolos formales y físicos que constituyen el cálculo.

La segunda dificultad se basa en que todavía no se dispone de una sólida *teoría del conocimiento*, de la cual se debe encargar de desarrollar la parte de la IA como ciencia. Es decir, no tenemos los conocimientos sobre neurofisiología o procesos cognitivos suficientes para poder apoyarse en ellos desde la parte ingenieril.

En la mayoría de los desarrollos de la IC, llamados Sistemas Basados en Conocimiento (SBCs), y anteriormente conocidos como Sistemas Expertos (Ses) se procede en base a los mismos pasos:

- Se parte de la descripción en lenguaje natural del problema, descripción generalmente realizada por un experto (análisis).

- Después, se modela esta descripción mediante diferentes paradigmas (simbólico, conexionista, situado) y se llega a obtener un modelo conceptual (diseño) según el paradigma elegido.

- A partir de este modelo, y utilizando diversos operadores formales, como las distintas lógicas, las RNAs (redes neuronales artificiales), etc, se llega a un modelo formal, el cual ya es computable por un sistema de cálculo (implementación).

En la IA entendida como ingeniería (IC), los objetivos y los procedimientos son muy diferentes. Aquí no buscamos la comprensión de la inteligencia humana, sino la posibilidad de reescribir en forma computable los procedimientos usados por los

humanos para resolver un conjunto de problemas científico-técnicos, en general descritos de forma poco clara, imprecisa e incompleta. Para su solución, la IC usa todos los métodos y técnicas disponibles, igual que el resto de la informática, incluyendo los cuatro paradigmas actuales, y los combina en función del balance entre datos y conocimientos disponibles para cada aplicación específica.

Resumen

El concepto de computabilidad. A lo largo del capítulo se insiste en que los problemas adecuados para ser abordados desde la inteligencia artificial son aquellos para los cuales disponemos tan sólo, en principio, de "descripciones poco claras, incompletas, imprecisas y con alto grado de dudas y errores potenciales debido a su complejidad". Los objetivos de la inteligencia artificial consistirían, básicamente, en dar formalidad a estas descripciones, traduciéndolas a descripciones claras, completas, precisas e inequívocas. En referencia a una cita del filósofo y matemático Leibniz (a quien se debe el cálculo diferencial, cuya notación se utiliza aún hoy en día), el autor identifica el concepto anterior con el concepto de computabilidad: "todo lo que sepamos describir de forma clara, completa, precisa e inequívoca es computable". De ahí que utilice expresiones que pueden resultar difíciles de interpretar tales como "hacer computable la semántica", con lo que se refiere a la posibilidad de traducir la riqueza de significados del mundo real a un lenguaje interpretable por un ordenador.

No debe confundirse pues esta acepción de la computabilidad con la noción rigurosa establecida en las ciencias de la computación, según la cual un algoritmo es computable cuando puede ser calculado por una máquina de Turing. En cuanto a esta acepción, un algoritmo especificado de forma clara, completa, etc. puede no ser computable. Incluso un algoritmo terminante puede considerarse no computable en la práctica, por cuestiones de complejidad espacial y temporal.

El aprendizaje como proceso de metanivel. En la metáfora mente-programa, los procesos mentales se equiparan a programas informáticos. Un metaprograma es un programa que en lugar de manipular datos manipula otros programas. En este sentido, el aprendizaje es un proceso mental equiparable a un

metaprograma, ya que su efecto es modificar los propios procesos mentales. Es en este sentido que el autor lo describe como proceso de "metanivel".

Niveles de descripción. El autor hace también énfasis repetidas veces en la importancia de no confundir niveles de descripción o formalización. En particular, insiste en la importancia de distinguir entre los propios procesos fisiológicos, la descripción que de ellos se hace en lenguaje natural, la traducción que de estas descripciones se hace en términos de lenguajes matemáticos formales y en términos de lenguajes informáticos interpretables por un ordenador, y finalmente la materialización de estos procesos en la máquina. Al alumno puede resultar difícil apreciar la relevancia de esta observación y sus implicaciones desde un punto de vista práctico.

En primer lugar, los procesos fisiológicos han sido programados por la evolución natural y después por la genética, la historia personal de adaptación al medio, etc. La descripción que de ellos hacemos parece presuponer una intencionalidad de la que los procesos en sí carecen: no son autoconscientes, sino que obedecen a sus leyes internas, siguen su propia "causalidad", en palabras del autor. Por otro lado, a medida que formalizamos las descripciones evidentemente hay una pérdida de riqueza descriptiva. Finalmente, cuando un proceso se traduce a un ordenador, hay adicionalmente una pérdida de significados, de conexión con referentes reales. De nuevo en el dominio de la máquina los procesos obedecen a sus propias leyes de causalidad, que son las de la electrónica y no las de la biología.

Estas reflexiones son interesantes y muy relevantes a nivel filosófico, y advierten ante el excesivo optimismo de la inteligencia artificial en su objetivo de reproducir la vida. Es lógico que el alumno encuentre dificultades para entenderlas en toda su profundidad, y no debe pretenderlo, ya que implican cuestiones de definición imprecisa, inaprensible, grandes preguntas eternamente debatidas en la historia del pensamiento humano.

T2 – Aspectos metodológicos (paradigmas).

1.4 Paradigmas actuales en IA

Se puede considerar un paradigma como una aproximación metodológica a la IA y a la IC que ha sido consensuada entre un grupo de profesionales del campo que la consideran como la forma normal de hacer ciencia o ingeniería.

Paradigma es sinónimo de *forma de abordar un problema*. Aplicado a la IC, paradigma es una forma de modelar, formalizar, programar e implementar físicamente el soporte de esos programas (por ejemplo, en el *cuerpo* de un robot).

De forma general, los paradigmas se pueden dividir entre los basados en representaciones y los basados en mecanismos, aunque aquí vamos a considerar 4 paradigmas distintos:

1.4.1. El paradigma simbólico

Este paradigma considera que todo el conocimiento necesario para resolver una tarea de diagnóstico, planificación, control o aprendizaje, puede representarse usando descripciones declarativas y explícitas en lenguaje natural. Estas descripciones declarativas estarían formadas por un conjunto de hechos, y otro conjunto de reglas de inferencia que describen las relaciones estáticas y dinámicas entre esos hechos.

En muchas ingenierías, es usual distinguir tres tipos de tareas para resolver los problemas específicos de cada una de ellas (análisis, síntesis y modificación/mantenimiento). Así, aplicando estos términos a la IC tenemos que en la fase de *análisis* disponemos de todas las soluciones posibles al problema, y el trabajo es básicamente de clasificación y elección de esas soluciones en base a la descripción del problema. En la fase de *síntesis* nos basamos principalmente en un trabajo de diseño y construcción con restricciones (los requisitos del problema). Finalmente, tenemos que la fase de *modificación* tiene que ver sobre todo con el ajuste de parámetros en las estructuras ya diseñadas y sintetizadas.

En este paradigma, las entidades del dominio (hechos o reglas) representan roles de entrada y/o salida para las distintas inferencias (reglas), las cuales generan el resultado final del *razonamiento*. Es decir, usamos nuestra *base de conocimiento* para evaluar y obtener nuevos hechos y reglas, que actualizan nuestro modelo del medio mediante aprendizaje, y pasan a formar parte también de la *base del conocimiento*.

Este paradigma es adecuado para aquellas aplicaciones en las que disponemos de conocimiento suficiente para especificar reglas inferenciales, y en procesos de aprendizaje inductivo en los que también disponemos de conocimiento suficiente para especificar las "meta-reglas" que actualizarán nuestra base del conocimiento.

1.4.2. El paradigma situado

También llamado reactivo, o basado en conductas, está basado en que toda percepción y toda acción están estructuralmente acopladas, mediante sensores y efectores concretos, a un medio externo e interno también concretos.

El sistema que queremos modelar mediante este paradigma se encuentra en un medio que forma un lazo de realimentación mediante sus sensores y efectores. Así, todo lo que no pueda ser percibido por los sensores no existe para el sistema, y

tampoco podrá ejecutar acciones que no puedan realizar sus propios efectores.

En este esquema, los roles de entrada vienen a ser las propias percepciones captadas por sus sensores, los roles de salida serán las acciones que ejecutarán sus efectores, y todo el motor de inferencia serán esquemas de asociación precalculados, o autómatas finitos, que permitirán que el sistema actúe de forma *reactiva*, sin tener que estar deliberando las posibles salidas. Además, las posibles acciones también estarán precalculadas.

La lógica interna depende de las coordinaciones espacio-temporales entre los estados actuales de los dos tipos de mecanismos descritos: los perceptuales y los motores.

Este paradigma se usa esencialmente en robótica y en aplicaciones en tiempo real simples. Es decir, en ámbitos donde la interfaz del sistema informático no es humana, sino que intercambia datos con el medio mediante sus sensores y efectores. Cuando aumenta la complejidad del sistema, se hace necesario utilizar soluciones híbridas, combinando componentes reactivas (rápidas) con otras deliberativas (lentas).

1.4.3. El paradigma conexionista

En este paradigma, la representación del conocimiento se realiza mediante el uso de líneas numéricas etiquetadas para la entrada y salida de una RNA (red neuronal artificial), y la inferencia se resuelve mediante un clasificador numérico parametrizado en el que el valor de los parámetros se ajusta mediante aprendizaje.

Un agente conexionista tiene una estructura modular, con un gran número de procesadores elementales (*neuronas*) interconectados, los cuales evalúan una función de cálculo local. Sus características distintivas son:

- Todos los problemas resueltos con RNAs se resuelven como un clasificador numérico adaptativo, que asocia valores de entrada de un conjunto de *observables* con valores de salida de otro conjunto más reducido de *clases*.

- Mucho del conocimiento disponible se obtiene de una fase de *análisis de los datos*, en que es el observador externo quien decide cuales van a ser las variables de entrada y salida, el tipo de cálculo local, la estructura interna en capas de la RNA, etc.

- Hay que tener en cuenta el balance entre datos y conocimiento disponible. Si los datos son etiquetados (se conoce la respuesta de la red) se usan en aprendizaje supervisado y en una fase final de validación de la red. Mientras que si son datos no etiquetados, se usan en aprendizaje autoorganizativo (no supervisado), para un preproceso de los mismos.

- El paradigma conexionista tiene un fuerte carácter numérico. Las salidas numéricas de la RNA se interpretan en términos de las etiquetas asociadas a las *clases* de salida.

Esta aproximación del conexionismo nos habla de una red de estructura fija, que actúa como un clasificador. Interpreta el sistema como un mecanismo de adaptación de un agente a su medio, considerando así a la inteligencia como un medio superior de adaptación, construido sobre otros medios de adaptación más elementales. Se utiliza cuando no se sabe representar de forma explícita el razonamiento para la solución de un problema, por lo que se acude a modelos numéricos aproximativos cuyos valores se ajustan a base de la experimentación y el aprendizaje.

1.4.4. El paradigma híbrido

La mayoría de problemas en IA suelen ser de naturaleza híbrida. Por tanto, es lógico pensar en soluciones también híbridas, combinando los datos y el conocimiento disponible con elementos o técnicas de varios paradigmas distintos.

Por ejemplo, para el control de un robot, podemos necesitar aproximaciones reactivas y declarativas, combinando el paradigma situado, en el que el robot obtiene parte de la información mediante sus perceptores y actúa mediante sus efectores, con otros paradigmas, como el representacional, utilizando técnicas simbólicas y deliberativas, o el conexionista, empleando técnicas neuronales para clasificar datos.

Algunos criterios a la hora de combinar técnicas o métodos de diferentes paradigmas son:

- Analizar las exigencias computacionales del problema y el conjunto de recursos disponibles. Eso ya nos puede dar una idea de qué métodos o técnicas pueden ser los más adecuados para resolver la tarea.

- Si no es suficiente, debemos seguir descomponiendo la tarea en otras más simples, hasta que lleguemos a decidir de qué tarea disponemos del conocimiento suficiente para usar reglas simbólicas, y de cuál no tenemos suficiente conocimiento, por lo que deberemos usar redes neuronales, probabilísticas (bayesianas) o conjuntos borrosos.

- El siguiente paso es la operacionalización efectiva del esquema, usando módulos simbólicos y neuronales, resultado de las decisiones de la fase anterior.

Ejemplo de paradigmas. Pregunta examen junio 2012

Proponga paradigmas y/o técnicas específicas de la Inteligencia Artificial para abordar las distintas partes de un *hogar inteligente*, justificando brevemente su propuesta. En particular, indique qué tipo de lógica/s sería más adecuada/s para la representación del conocimiento implicado, razonando su respuesta.

Paradigma híbrido. El conocimiento experto de las diferentes áreas de conocimiento se representaría adecuadamente mediante Sistemas Basados en Reglas (paradigma simbólico) con incorporación de técnicas bayesianas (teoría de juegos, algoritmos de búsqueda, redes semánticas) y lógicas NO clásicas, ya que las decisiones multicriterio implican la argumentación sobre cuestiones conjeturales y también sobre cuestiones valorativas (razonamiento ético): lógica no monotónica, lógica intuicionista, lógica multivaluada y borrosa, lógica modal espacial y temporal. En general podrían utilizarse marcos para la representación de las entidades del dominio. El paradigma conexionista situado (reactivo o deliberativo según los casos) es idóneo para la representación de las actuaciones adaptivas función de los datos percibidos a través de sensores (robótica, redes neuronales y algoritmos genéticos). Los problemas de optimización y control, y el reconocimiento de patrones (en imágenes, voz, etc) se modelan también adecuadamente en el paradigma conexionista.
Los aspectos de personalización del sistema en infinidad de dimensiones (afectivas, físicas, de personalidad, culturales, de preferencias…) sugiere el uso de técnicas de aprendizaje supervisado (conexionista o simbólico, con incorporación de nuevos hechos y reglas en los Sistemas Basados en Reglas) o autoorganizativo y minería de datos.
Considerando el sistema en conjunto, con sus diferentes partes interconectadas, sería sin duda útil una visión en términos de sistema multiagentes (inteligencia artificial distribuida).

T3 – Introducción a las técnicas de búsqueda.

Capítulo 8 del libro base

8.1 Introducción

En IA, la resolución de problemas y búsqueda se refiere a un conjunto de técnicas y métodos que se utilizan en diferentes dominios de aplicación, como la deducción, la elaboración de planes de actuación, razonamientos, etc.

Dependiendo del problema concreto, la ejecución de la secuencia de acciones o decisiones tiene asociado un coste que se tratará de minimizar, o bien tiene asociado un beneficio que se tratará de maximizar.

8.3 Formulación de problemas de búsqueda

Un **sistema de búsqueda** tiene 3 componentes principales:

- El conjunto de estados, que representa todas las situaciones por las que el agente puede pasar durante la búsqueda de la solución del problema y deben basarse en un modelo de representación con un nivel de detalle adecuado.

- Los operadores, que modelan las acciones elementales que es capaz de realizar el agente sobre el medio, y, además, cada operador debe tener un coste asociado, que puede ser arbitrario o atribuido por la propia naturaleza del problema. Cada operador debe quedar perfectamente definido a partir de una especificación precisa de las condiciones que debe cumplir el estado en el que se aplica y del estado resultante de su aplicación.

- La estrategia de control, que es la responsable de decidir el orden en que se van explorando los estados. Una estrategia de control inteligente (*heurística o informada*) debería llevar a explorar primero aquellos nodos que están en el mejor camino hacia una solución óptima. Sin embargo, una búsqueda a ciegas (*no informada*) generalmente considerará todos los nodos como igualmente prometedores. Por tanto, con una búsqueda informada lo que se pretende es llegar a una buena solución (generalmente óptima) del problema, visitando el menor número de nodos posibles.

Los dos primeros (estados y operadores), conforman lo que se llama el *espacio de búsqueda*, que generalmente se representa mediante un grafo dirigido simple, en donde los nodos son los distintos estados por los que puede pasar el sistema, y los arcos son las reglas que provocan la transición. Generalmente, el grafo que representa el espacio de estados del problema tiene un tamaño tan grande, que no es posible representarlo de forma explícita, por lo que queda definido de forma implícita por un estado inicial y un conjunto de operadores. En este grafo, hay uno o varios nodos que representan soluciones del problema: son los llamados objetivos.

Un ejemplo de un sistema de búsqueda podría ser el caso de un robot que tiene un espacio de bloques, y, dada una situación inicial, tiene que conseguir colocar los bloques en una determinada posición. El conjunto de estados serían las distintas configuraciones de posiciones de los bloques, los operadores serían los movimientos que podría realizar el robot para trasladar los bloques, y la estrategia de control sería la definida por el diseñador, que podría ser no informada o informada.

Un sistema de búsqueda inteligente es encontrar una buena solución del problema, a ser posible óptima, visitando la menor cantidad de estados posible utilizando los mínimos recursos computacionales.

Algoritmos completos y admisibles.

- Un algoritmo de búsqueda es completo si siempre que existe una solución la encuentra
- Un algoritmo de búsqueda es admisible (o exacto) si siempre encuentra una solución óptima

8.4. Métodos de búsqueda sin información

Para realizar búsquedas sin información se utilizan métodos que realizan un recorrido del espacio de búsqueda de una forma sistemática, pero sin tener en cuenta ningún tipo de información sobre el dominio del problema que se está resolviendo. Se considera que el espacio de búsqueda es un árbol, luego se tendrán en cuenta las modificaciones que hay que introducir para tratar con espacios de búsqueda más generales, es decir grafos.

Se utiliza una lista denominada ABIERTA que contendrá al principio de cada iteración los estados candidatos a ser desarrollados o expandidos y que estará ordenada con un determinado criterio en cada caso. El desarrollo, o expansión de un estado, consiste en calcular todos sus sucesores mediante la aplicación de todos los operadores posibles, para ello se dispone de la función Sucesores que para un determinado estado retorna una lista con todos sus sucesores. Se utiliza otra estructura denominada TABLA_A, que es en realidad un conjunto ordenado y se suele implementar mediante una tabla hash, para registrar toda la información necesaria acerca de cada uno de los nodos encontrados en el proceso de búsqueda, es posible realizar reorientaciones de enlaces si se encuentra un camino mejora hacia un cierto nodo.

En resumen:

Lista ABIERTA. Nodos generados y no expandidos. Se gestiona de forma distinta según los algoritmos:
– Primero en anchura. Funciona como una cola, se saca el primer nodo de la cola y se introducen sus hijos al final de la misma.
– Primero en profundidad. Funciona como una pila, siempre se saca el primer nodo de la pila y se introducen sus hijos al principio de la misma.
– Coste uniforme. Los estados se ordenan teniendo en cuenta el coste del mejor camino desde el nodo inicial

TABLA_A, subgrafo parcial que muestra el mejor camino encontrado desde cada estado hasta el estado inicial. Se va ampliando al expandir nuevos nodos. Es posible que se produzcan reorientaciones de enlaces si se encuentra un camino mejor hacia un cierto nodo N:
– Si N estaba en ABIERTA, se anota el nuevo camino mejor en TABLA_A
– Si N había sido expandido el proceso se extiende recursivamente a sus hijos

8.4.1 Recorrido de árboles

En primer lugar se coloca en la lista ABIERTA el estado inicial. Este estado se genera a partir de los datos del problema. A partir de ese momento el algoritmo va expandiendo nodos, de forma que en cada iteración el nodo que se expande es el primero de la lista ABIERTA. De este modo, la estrategia de control viene definida por el criterio que se utilice para ordenar ABIERTA, o lo que es igual por el criterio que se utilice para insertar los nodos en esta lista.

En una exploración el nodo meta no es realmente expandido (es decir, sus hijos no son generados), ya que justo antes de su expansión se comprueba que es un nodo meta y, por tanto, el algoritmo termina en ese momento sin que se lleguen a generar sus hijos.
Se muestran las tres formas más comunes de realizar una búsqueda a ciegas y se analiza su comportamiento en términos del espacio y el tiempo requeridos para una búsqueda, así como de sus propiedades de completitud y admisibilidad.

– Primero en anchura. Completo y admisible (si los operadores del mismo nivel tienen igual coste)

– Primero en profundidad. No completo (y, por tanto, no admisible)

– Coste uniforme. Admisible (y, por tanto, completo)

– Búsqueda en profundidad iterativa. Admisible

– Búsqueda en anchura iterativa. Completo pero no admisible

– Búsqueda bidireccional. Completo si una de las búsquedas es en anchura

Búsqueda primero en anchura

En esta búsqueda, el criterio que define la ordenación de *ABIERTA* es insertar los nuevos nodos que se van generando simplemente al final de la lista, tratada como una cola FIFO.

Este algoritmo, al realizar una exploración del árbol de búsqueda por niveles de profundidad, siempre es completo (para un grafo localmente finito, número de hijos por nodo limitado). Además, si el coste de todas las reglas es 1, también es admisible. Sin embargo, tanto el tiempo de ejecución como el espacio de memoria necesario crecen de forma exponencial con el tamaño del problema. Este crecimiento está relacionado con la profundidad del árbol que representa el espacio de búsqueda.

Esta búsqueda también proporciona una solución más cercana al nodo inicial.

Búsqueda primero en profundidad

Para esta búsqueda, el criterio de ordenación de la lista *ABIERTA* es el de insertar los nuevos nodos generados al principio de la lista, tratándola así como una pila LIFO.

El algoritmo de búsqueda en profundidad, al realizar una exploración que intenta bajar de nivel de profundidad siempre que sea posible, no es admisible, y ni siquiera es completo. Esto es debido a que la búsqueda puede derivar hacia una rama infinita, y el algoritmo no terminaría nunca. Por ello, se suele establecer una profundidad límite a partir de la cual se detiene la búsqueda, se haya o no encontrado la solución, pero esto no garantiza que sea completo ya que la solución puede encontrarse más allá de la profundidad límite elegida.

En cuanto al espacio en memoria, este algoritmo tiene un coste lineal en relación a la profundidad, puesto que en cada hilo de ejecución solo almacena los nodos de una rama concreta, no los de todo el árbol. En relación al tiempo de ejecución, en el peor de los casos, este algoritmo tiene, como en la búsqueda en anchura, una complejidad exponencial a la profundidad del árbol. Además, la búsqueda en anchura siempre proporciona la solución más cercana al nodo inicial (la de menor altura en el árbol), lo que no siempre ocurre con la búsqueda en profundidad.

Esta búsqueda se suele realizar mediante algoritmos de tipo backtracking(recursivos) que son muy utilizados debido al bajo consumo de espacio de almacenamiento.

Búsqueda de coste uniforme

Este algoritmo no trata la lista *ABIERTA* como una cola ni una pila, sino que inserta los nodos en la lista, ordenados directamente por el coste desde el nodo inicial a cada uno de los nodos, de menor a mayor. Si el coste de todas las reglas es el mismo, la estrategia de coste uniforme y anchura son idénticas.

El coste computacional en espacio y memoria es similar al de la búsqueda en anchura (exponencial), y también es un algoritmo completo. Sin embargo, este algoritmo siempre obtiene la solución de menor coste al nodo inicial por lo que además es admisible.

Búsquedas en profundidad y anchura iterativas

Estas búsquedas se basan en limitar la profundidad o anchura límite (respectivamente), y realizar varias iteraciones del mismo algoritmo, incrementando dichos límites en cada iteración. Ambas versiones iterativas de dichos algoritmos tienen los mismos costes computacionales que sus respectivas no iterativas.

Sin embargo, en el caso de la búsqueda en profundidad se resuelve el problema de su profundidad límite (ramas infinitas) y en el caso de la búsqueda en anchura, ahora es posible encontrar una solución que no sea la más próxima al nodo inicial.

El principal inconveniente de este método de búsqueda es que para cada uno de los valores de la profundidad límite hay que visitar de nuevo todos los nodos visitados en la iteración anterior.

Búsqueda bidireccional

La idea de este algoritmo consiste en buscar simultáneamente en dos direcciones: por un lado, hacia delante desde el nodo inicial a los nodos objetivo, y por el otro lado, hacia atrás, desde los nodos objetivo hasta el nodo inicial. La búsqueda se detendrá cuando las listas *ABIERTA* de ambos recorridos tengan algún nodo en común. Esto consigue que el tiempo de ejecución, en lugar de ser exponencial en

relación a una profundidad *d*, lo sea en relación a una profundidad *d/2*, lo que supone una mejora. La búsqueda bidireccional es completa y, si la búsqueda se realiza desde todos los nodos objetivo, es admisible.

Sin embargo, hay varios factores que condicionan esta mejora. Por un lado, hay que tener claro cuántos y cómo son los estados objetivo, puesto que si son muchos, la mejora de eficiencia deja de ser cierta. Por otro lado, hay que considerar si los operadores son bidireccionales, puesto que si no lo son, a veces resulta difícil decidir cuáles son los sucesores de un estado en el recorrido hacia atrás. Y por último, hay que decidir que algoritmo de búsqueda emplear en cada dirección, puesto que para asegurar que ambos recorridos se encuentren en algún momento, al menos uno de ellos debe almacenar en memoria todos los estados visitados, es decir, al menos uno debe ser en anchura.

Complejidad de los diferentes métodos de búsqueda

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.

De cara a comparar las complejidades de los tres algoritmos de búsqueda del enunciado, definimos los siguientes conceptos:

- *Factor de ramificación* (*n*): es el número medio de sucesores/hijos de los nodos del árbol de búsqueda.
- *Profundidad de la solución* (*p*): número de arcos desde el nodo inicial hasta la solución, que suponemos única.

Anchura

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 $1 + n + n2 + ... + n^p$ que es del orden de $O(n^p)$.

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 $O(n^p)$.

El algoritmo primero en anchura tiene el inconveniente de requerir espacio y tiempo exponenciales con la profundidad de la solución.

Profundidad

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 por el factor de ramificación n, ya que cada vez es necesario guardar constancia de todos los sucesores del nodo que se está expandiendo, es decir O(n x p).

El algoritmo primero en profundidad tiene la ventaja de requerir espacio proporcional a la profundidad de la solución.

Algoritmo	Tiempo	Espacio
Anchura	$O(n^p)$	$O(n^p)$
Profundidad	$O(n^p)$	$O(n \cdot p)$
Coste uniforme (todas las reglas igual coste)	$O(n^p)$	$O(n^p)$
Profundidad iterativa	$O(n^p)$	$O(p)$
Anchura iterativa	$O(n^p)$	$O(n^p)$
Bidireccional (dos búsquedas en anchura)	$O(n^{p/2})$	$O(n^{p/2})$

Consideramos que n es el número medio de sucesores de un nodo, y p la distancia del nodo inicial a la meta.

8.4.2. Métodos de búsqueda sin información en grafos

En los grafos, al contrario que en árboles, es posible que durante la búsqueda se pueda encontrar varias veces el mismo nodo *n* como sucesor de dos o más nodos diferentes. En este caso, se deben considerar los distintos caminos hasta dicho nodo *n*, y registrar solo el mejor obtenido hasta el momento. Por ello, puede ser necesario rectificar en la *TABLA_A* el coste desde el estado inicial hasta ese nodo *n*, así como su antecesor.

Algoritmo general de búsqueda en grafos (AGBG)

Este algoritmo parte de un grafo dirigido simple definido implícitamente a partir de un nodo inicial y una serie de operadores o reglas de producción. Cada regla tiene un coste no negativo. El grafo debe ser localmente finito (número de sucesores por nodo acotados), pero no necesariamente finito (se puede extender infinitamente en profundidad). En el grafo hay uno o varios estados solución y el objetivo de la búsqueda es encontrar el camino de coste mínimo desde el nodo inicial hasta los estados objetivo.

Al expandir un nodo *n* determinado, hay que comprobar si alguno de sus sucesores, por ejemplo *s* ya fue expandido con anterioridad, para comprobar si dicho nodo *s* tiene un camino mejor mediante el nodo *n* que se acaba de expandir que mediante su camino anterior. Esto se hace con una función *Rectificar*, que se aplica a dicho nodo *s* en el caso de ya haber sido expandido con anterioridad, y también se aplica recursivamente a todos los sucesores de *s*, como por ejemplo un nodo *q*, porque puede que también mejore su camino de coste mínimo al nodo inicial pasando por *n* y por *s*. De esta forma, en la *TABLA_A* se registrará el mejor camino obtenido hasta

el momento desde el nodo inicial a cada uno de los nodos encontrados durante la búsqueda en el punto actual.

La búsqueda en grafos puede provocar que se descubra más de un camino desde el estado inicial hasta cualquier otro nodo. Esto implica que en TABLA_A deba anotarse para cada nodo n cuál de sus padres conocidos marca el mejor camino encontrado hasta el momento desde n al nodo inicial. Estos enlaces desde cada nodo a uno de sus padres forman un árbol en el espacio de búsqueda, que cambia a medida que avanza el proceso de búsqueda. Pueden surgir tres casos a la hora de expandir el siguiente nodo de ABIERTA, que siempre deberá ser sacado de ABIERTA después de su expansión y al que llamaremos "nodo padre expandido":

1. Que el "nodo hijo generado" correspondiente no haya sido generado o incluso expandido previamente, no esté en ABIERTA. En este caso, el nodo hijo generado es introducido en ABIERTA y el enlace a su nodo padre recién expandido es incluido en TABLA_A.
2. Que el nodo hijo generado ya haya sido generado pero no expandido previamente, es decir, el nodo hijo generado ya estaba en ABIERTA. En este caso hay que evaluar si hemos encontrado un camino mejor desde el nodo inicial al nodo hijo recién generado; en caso afirmativo, hay que enlazar en TABLA_A el nodo hijo recién generado con su nodo padre recién expandido. Este último proceso no es más que una reorientación de enlace.

3. Que el nodo hijo generado ya haya sido expandido previamente, es decir, el nodo hijo no está en ABIERTA. Significa que este ya se había expandido anteriormente. Si el nuevo camino tiene menor coste, además de llevar a cabo los pasos del caso anterior, hay que actualizar los nodos hijo por si hubiera que redirigir los enlaces.

La estrategia de control del AGBG queda definida por la forma de ordenar la lista *ABIERTA* de nodos candidatos a ser expandidos. Así, si ordenamos la lista insertando los nuevos nodos al principio de abierta (pila) tendremos una búsqueda en profundidad. Si ordenamos la lista insertando los nuevos nodos al final (cola), tendremos una búsqueda en anchura. Y por último, si ordenamos la lista *ABIERTA* mediante el coste de cada nodo al inicial, de menor a mayor, se obtiene una búsqueda de coste uniforme.

La complejidad temporal de estos métodos es exponencial, lo que hace necesario desarrollar otras estrategias para tratar problemas de tamaño medio o grande (heurísticas).

T4 – Técnicas basadas en búsquedas heurísticas.

Capítulo 9 del libro base

9.1 Introducción

Los llamados *heurísticos* son aquellos mecanismos que permiten, en un espacio de estados y empleando conocimiento del dominio del problema, dirigir en cierta manera la búsqueda hacia las zonas más prometedoras, de modo que se pueda llegar a la solución sin necesidad de visitar tantos nodos como en general requeriría una búsqueda a ciegas o no informada.

Sin embargo, estos *heurísticos* también introducen un cierto coste de control, por lo que se debe alcanzar un punto de compromiso entre el coste de control y el coste de aplicación de las reglas, el cual es a veces difícil de alcanzar.

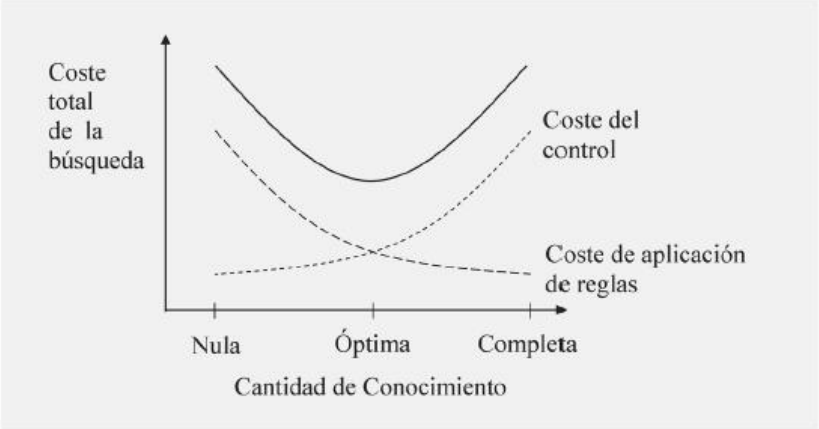


Figura 9.1: Representación del coste total de la búsqueda frente al grado de conocimiento utilizado.

La búsqueda heurística utiliza conocimiento del dominio para guiar la exploración de un espacio de estados.
Es importante tener en cuenta que el conocimiento heurístico no suele producir valoraciones siempre correctas, sino que intenta proporcionar información lo más cercana posible a la realidad. Dicha cercanía determina la calidad del heurístico.

9.2. Búsqueda primero el mejor (BF)

En este algoritmo es una modificación del algoritmo general de búsqueda en grafos que emplea una función de evaluación de los nodos *f*, tal que para cada nodo *n*, la función *f(n)* da un valor numérico que indica lo prometedor que es ese nodo para ser expandido. Así, la lista *ABIERTA* se ordena en base a *f*, estando los nodos candidatos más prometedores al principio. Esta medida *f* de lo prometedor que es un nodo se denomina *función heurística de evaluación*, y se puede estimar de varias formas.

Este modo de expansión de nodos candidatos no siempre llevará de forma directa a la mejor solución, pero si permiten generalmente llegar a buenas soluciones expandiendo un número de estados mucho menor que con una elección puramente aleatoria.

*9.3. El algoritmo A**

El algoritmo A* es a su vez una especialización del algoritmo BF en el que la función de evaluación f se define de una forma particular.

A continuación se introducen algunos conceptos previos (consideremos un nodo *n cualquiera del espacio de búsqueda*):

- ***g*(n)*** es el coste del camino más corto, o de menor coste, que existe desde el nodo inicial a *n*.

- ***h*(n)*** es el coste del camino más corto que existe desde *n* al nodo objetivo más cercano a *n*.

- **f*(n) = g*(n) + h*(n)**. Es decir, *f*(n)* es el coste del camino más corto desde el nodo inicial a los nodos objetivos condicionado a pasar por *n*.

- **C*** = f**(inicial)* = h**(inicial)*. Es decir, *C** es el coste de la solución óptima.

Todos los nodos que forman parte de la solución óptima cumplen que *f*(n)=C**, mientras que aquellos que no están en el camino óptimo cumplen que *f*(n) > C**. Para problemas complejos, los valores de estas funciones no se pueden conocer, por lo que el algoritmo A* lo que hace es trabajar con aproximaciones a estos valores.

g(n) es el coste del mejor camino encontrado desde el inicial al nodo *n* obtenido hasta el momento.

h(n) es una estimación del valor de *h*(n)* tal que *h(n)=0* si *n* es un nodo objetivo.

f(n) = *g(n) + h(n)*, siendo *f* una estimación de *f**, y siendo el criterio que se utiliza para ordenar la lista *ABIERTA*. Su valor cambia durante la ejecución del algoritmo, ya que mientras h(n) es fijo, el valor de g(n) puede variar al expandir nuevos nodos.

La función *h* se suele denominar función heurística.

9.3.1 Descripción del algoritmo A*

El algoritmo A* es una variante del algoritmo general de búsqueda de grafos, donde la lista ABIERTA se ordena según f(n), y las reorientaciones en TABLA_A están producidas por los cambios en g(n).

El algoritmo A* es el mismo que el AGBG con algunos cambios debidos a la forma de definir la función de evaluación f, en esta versión se hace una ligera modificación del contenido de la TABLA_A para registrar, para cada nodo n encontrado, el valor de g(n) que es lo mismo que Coste(inicial, n) del algoritmo general, y el valor de h(n).

9.3.2 Propiedades formales

Terminación y completud

El algoritmo BF es completo en grafos finitos, ya que en el peor de los casos terminaría después de expandir todos los nodos. Esto es evidente porque el algoritmo expande un nodo en cada iteración y ningún nodo se expande más de una vez.

Sin embargo, esto no puede ocurrir en el algoritmo A*, debido a la definición de f = g + h. La componente h siempre es positiva, y la componente g crece de forma no acotada, por lo que si la búsqueda se fuese a través de un camino infinito, en algún momento el valor de f sería suficientemente grande para que el siguiente nodo a través de ese camino no fuese el más prometedor, y por lo tanto se consideraría la búsqueda a través de otros caminos.

Teorema. A* es completo en grafos localmente finitos (nº sucesores acotado),si hay solución la encuentra.

Admisibilidad del algoritmo A*

- A* es admisible, encuentra la solución óptima, siempre que utilice una función heurística *h* admisible → $\forall n,\ h(n) \leq h^*(n)$.

Teorema. A* es admisible

Comparación de heurísticos admisibles

- Una función heurística h1 está más informada que h2 si:

Ambas h1 y h2 son admisibles.

Para todo n, h1(n)> h2(n)

Entonces, todo nodo que haya que reorientar usando h1, también hay que reorientarlo con h2.

Cuanto más informada esté la función heurística, menos nodos deberán expandirse, y por tanto, más eficiente será el algoritmo A* que la utilice.

- Un algoritmo A^*_2 domina a otro algoritmo A^*_1 si los dos son admisibles y todo nodo expandido por A^*_2 es también expandido por A^*_1.

Teorema. Una condición necesaria de expansión de un nodo n es que f(n) ≤ C*.

Heurísticos Consistentes o Monótonos

Un nodo expandido puede sufrir muchas rectificaciones posteriores, lo que evidentemente condiciona la eficiencia del algoritmo A*. La consistencia y la monotonía son equivalentes y, si el heurístico las cumple, aseguran que no es preciso rectificar el contenido de la TABLA_A para aquellos nodos que ya han sido expandidos.

- Un heurístico es monótono si para todo par de nodos *n* y *n'* se cumple que:

h(n) ≤ k(n, n') + h(n'). Siendo *k(n, n')* el coste mínimo de *n* a *n'*. En el caso de que no existiera camino de n a n' sería infinito.

- Un heurístico h es consistente si para todo par de nodos n y n' se cumple que

$$h(n) \leq c(n, n') + h(n')$$

donde c(n, n') representa el coste de la regla que lleva de n a n', y por lo tanto es infinito si no existe esta regla.

- Monotonía y Consistencia son propiedades equivalentes.

- Todo heurístico que sea monótono, también es admisible.

- Si *h* es monótono y A* elige por ejemplo un nodo *n* para su expansión, se cumple que *g(n) = g*(n)*. Es decir, el camino del inicial a *n* encontrado hasta el momento es óptimo. Debido a esto, si *h* es monótono, no es necesario rectificar nodos ya expandidos, por lo que el algoritmo A* es más eficiente utilizando un heurístico monótono.

9.3.4 Relajación de las condiciones de optimalidad

En muchos problemas A* pasa demasiado tiempo discriminando entre caminos cuyos costes no varían significativamente, por esto se puede plantear conseguir más eficiencia con el algoritmo A*, a costa de perder la admisibilidad que proporciona. Existen dos métodos principales:

9.3.4.1 Ajuste de los pesos de g y h.

La búsqueda de A* está basada en la fórmula f(n)=g(n)+ h(n)

Donde:

- **g** trata de ajustar el algoritmo a una estrategia de búsqueda en anchura o de coste uniforme, lo que garantiza soluciones óptimas.

- **h** trata de acercar la búsqueda directamente hacia la solución sin que, necesariamente, se expandan todos los nodos que disten igual del inicial.

El ajuste de pesos se basa en asociar unos pesos a las funciones g y h, que permitan adaptar el tipo de exploración realizada en función de lo avanzado que se encuentre el proceso de búsqueda.

Esta variante de A* pondera ambos sumandos, de tal forma que se obtenga una mayor eficiencia de la siguiente forma:

$$f(n) = (1-w)\cdot g(n) + w\cdot h(n), \quad \text{con } 0 \leq w \leq 1$$

Valor de w	Tipo de búsqueda
0	Coste uniforme
1/2	A* estándar
1	Primero el mejor

El problema de este tipo de ponderación (que se llama estática, pues el peso ω no varía a lo largo del proceso), es la gran dificultad que presenta encontrar, a priori, un valor adecuado para ω. En la práctica el mejor valor de ω se puede obtener de forma experimental.

9.3.4.2 Algoritmos ε-admisibles.

Se sacrifica la obtención de una solución óptima en favor de alguna mejora en el rendimiento del algoritmo, controlando el deterioro de la solución obtenida a través de un factor ε que representa la distancia al coste óptimo.

Utiliza una función heurística adicional para seleccionar uno de entre los nodos de una sublista de ABIERTA que contiene los nodos cuyo valor de f no difiere en más de un factor 1+ε del valor mínimo de f en ABIERTA.

Un algoritmo de búsqueda es ε-admisible si siempre encuentra una solución con un coste que no excede el valor $(1+\varepsilon)C^*$ (una solución de este tipo se denomina solución ε-óptima).

Las dos versiones más conocidas de este tipo de algoritmos son; la ponderación dinámica y el algoritmo $A\varepsilon^*$

Ponderación dinámica

- Los pesos cambian de manera dinámica a lo largo del proceso de búsqueda.

- h un mayor peso al principio de la búsqueda hasta que se esté en las proximidades de la solución, en donde se reduce este peso para asegurar una búsqueda en anchura que afine lo más posible la solución: $f(n)= g(n) + h(n) + \varepsilon (1 -d(n)/N)h(n)$

  d(n) es la profundidad del nodo n

  N es una cota superior de profundidad de la mejor solución.

  El número ε >0 indica la desviación que estamos dispuestos a admitir.

Teorema. Si el heurístico h es admisible, el algoritmo de ponderación dinámica es ε-admisible.

Algoritmo Aε*

La idea consiste en considerar dentro de ABIERTA una sublista, llamada FOCAL, formada por aquellos nodos cuyo valor de f no se separa más de un factor (1+ε) del valor mínimo de f en ABIERTA. (1+ε).min Siendo min el valor de f en ABIERTA

La estrategia de Aε* es idéntica a la de A*, salvo que en Aε* se desarrollan primero los nodos de la sublista FOCAL según los valores de un segundo heurístico h', donde h'(n) es una estimación del esfuerzo computacional que se necesitaría para llegar desde n a una solución.

Los nodos de FOCAL tienen todos más o menos las mismas posibilidades de estar en un camino óptimo.

Teorema. Si h es admisible, Aε* es ε-admisible.

Comparación entre los dos algoritmos.

El algoritmo de ponderación dinámica es más simple ya que solamente tiene una lista de estados y un heurístico, pero tiene el inconveniente de que es necesario conocer la profundidad de la solución óptima, o al menos una buena cota superior.

Aε*, al utilizar un segundo heurístico h', que no tiene que ser admisible, ofrece un mecanismo más flexible del que ofrecen las funciones g y h, para hacer estimaciones del coste computacional de la búsqueda en una determinada dirección.

9.4. Búsqueda con memoria limitada

El principal problema del algoritmo A* es el requerimiento de memoria, que crece de forma exponencial con la profundidad, aunque dispongamos de buenos heurísticos.

9.4.1 Algoritmo IDA* (Iterative Deepening A*)

Este algoritmo extiende al de búsqueda en profundidad iterativo, y se basa en realizar iteraciones de búsqueda primero en profundidad desde el nodo raíz, aumentando en cada iteración la profundidad límite de dichas búsquedas. En la primera iteración, se establece una longitud límite igual al valor de *f(inicial)*, descartando todos aquellos nodos cuya estimación *f(n)* supere dicha longitud límite. A continuación, si no se ha encontrado una solución, se realiza una nueva iteración en la que la búsqueda comienza otra vez desde el principio, pero ahora la longitud límite será el menor valor de *f* de entre todos los nodos descartados en la iteración anterior (aumentando la profundidad, por tanto).

Los hijos de cada nodo expandido se introducen ordenados en *ABIERTA* según su valor *f*, actuando *ABIERTA* como una pila. Así, se consideran antes los hijos más prometedores, y, en caso de encontrar una solución, se habrán expandido menos nodos.

Si *h* es admisible, el algoritmo *IDA** que lo utilice también lo será. Por otro lado, el consumo de memoria de este algoritmo es proporcional al producto de la profundidad de la solución y del factor de ramificación (nº de hijos por nodo), lo que supone un ahorro de memoria. Sin embargo, el tiempo de búsqueda es exponencial con la profundidad límite.

El principal problema del algoritmo IDA* es la cantidad de nodos que tiene que volver a expandir, debido a que lo único que recuerda de una iteración a la siguiente es el valor del menor valor de f de los nodos descartados.

En líneas generales, IDA* amplía la búsqueda iterativa en profundidad asociando a cada nodo n su valor de la función heurística propia de A*,f(n)=g(n)+ h(n)

9.4.2 Algoritmo SMA* (Simplified Memory-Bounded A*)

En este caso, el algoritmo se basa en limitar el tamaño de la *TABLA_A*, es decir, el máximo nº de nodos o memoria que se pueden almacenar en ella. Si se necesita expandir un nodo, y no hay espacio suficiente en la *TABLA_A*, se elimina un nodo de *ABIERTA* y de la *TABLA_A*, aquel con mayor valor de *f* en *ABIERTA*, el cual se conoce como *nodo olvidado* al ser el menos prometedor. El algoritmo recuerda en cada nodo el mejor *f* de los hijos de ese nodo (los nodos olvidados). De este modo, solo

reexplora un subárbol descartado si el resto de posibilidades tiene estimaciones que son aún peores de acuerdo a las estimaciones.

Las propiedades principales de este algoritmo son las siguientes:

- Es capaz de evolucionar utilizando la memoria que tenga disponible.
- Evita estados repetidos en la medida en que la cantidad de memoria disponible se lo permita.
- Es completo si la memoria disponible es suficiente para almacenar el camino a la solución menos profunda.
- Es admisible si tiene suficiente memoria para almacenar el camino hasta la solución óptima menos profunda. En otro caso, devuelve la mejor solución que se puede alcanzar con la memoria disponible.
- Si la memoria es suficiente para el árbol de búsqueda completo, la eficiencia de la búsqueda es óptima.

Similitudes y qué diferencias existen entre los algoritmos SMA* e IDA*.

Como similitudes se encuentran las siguientes:

- Tanto un algoritmo como el otro son variantes del algoritmo A*.
- Estos dos algoritmos intentan solventar el problema que tiene el algoritmo A* de requerimiento exponencial de memoria con la profundidad.
- En relación al punto anterior, incorporan mecanismos para limitar la cantidad de memoria necesaria durante la ejecución.
- Fueron propuestos con sólo siete años de diferencia, IDA* en 1985 y SMA* en 1992.

Como diferencias se pueden citar las siguientes:

- Mientras que IDA* es una extensión del método de búsqueda iterativa en profundidad, SMA* se basa en limitar el tamaño de TABLA_A a un valor máximo prefijado.
- IDA* utiliza ABIERTA como una pila y no necesita establecer ningún límite a la capacidad de TABLA_A. SMA* trata ABIERTA como una lista ordenada según los valores de f y limita la capacidad de TABLA_A a un cierto número de nodos prefijado.
- Como novedad respecto a A*, IDA* establece en cada iteración un coste límite para la búsqueda. En la primera iteración el coste límite es el valor de "f" del estado inicial y, en sucesivas iteraciones, el coste límite es el menor valor de f de los nodos descartados en la iteración anterior. Por su parte, la novedad que introduce SMA* con respecto a A* consiste en establecer un límite en el número máximo de nodos que se pueden almacenar en TABLA_A. Si se necesita expandir un nodo y no hay espacio en TABLA_A, se elimina de ABIERTA y de TABLA_A el nodo con mayor valor de f en ABIERTA. Estos nodos eliminados se denominan "olvidados" y SMA* recuerda en cada nodo el mejor f de los hijos de ese nodo que han sido olvidados.
- Mientras que IDA* es completo, SMA* es completo si la memoria disponible es suficiente para almacenar el camino a la solución menos profunda.
- Mientras que IDA* es admisible si h es admisible, SMA* es admisible si además tiene suficiente memoria para almacenar el camino hasta la solución óptima menos profunda. En caso contrario, devuelve la mejor solución que se puede alcanzar con la memoria disponible.

9.5. Algoritmos voraces

Estos algoritmos se basan en la idea básica de tomar decisiones de forma irrevocable en la exploración no considerando ninguna alternativa. Es decir, los nodos que han sido descartados no los vuelve a considerar. Por tanto, no son admisibles, y tampoco suelen ser completos, pero a cambio resultan muy eficientes, por lo que se suelen usar en aplicaciones de tiempo real (planificación del procesador).

El diseño de este tipo de algoritmos es simple. Por ejemplo, si partimos del algoritmo A*, y en cada paso consideramos solo el nodo más prometedor en función de *h*, descartando el resto de sucesores, obtenemos un algoritmo voraz. Si, además, los empates que se puedan producir en cada paso se resuelven de forma aleatoria, tenemos un algoritmo no determinista, pues en cada ejecución del mismo sobre un problema, nos puede dar resultados diferentes, aunque con una eficiencia media que dependerá de *h*.

La elección irrevocable de uno de los sucesores del nodo actual se hace en función de un heurístico, de manera que el sucesor más prometedor suele ser el elegido, aunque otras opciones de naturaleza probabilista son posibles.

9.6. Algoritmos de ramificación y poda

Los algoritmos de ramificación y poda interpretan cada estado o nodo como un subconjunto de soluciones de todo el conjunto posible de soluciones del problema original planteado. De esta manera, el nodo raíz representa todas las soluciones posibles al problema original, mientras que un nodo hoja sería aquel que no puede ser expandido por contener una única solución. Así, el proceso de ramificación consiste en descomponer un determinado conjunto de soluciones (un nodo) en la unión disjunta de varios subconjuntos suyos (los nodos sucesores), con lo que el espacio de búsqueda adquiere forma de árbol. De este modo, la ramificación de un nodo sería equivalente a la expansión de un nodo para generar sus nodos hijos.

Mientras que un nodo hoja tendrá asociado un coste concreto y conocido, cada nodo intermedio del árbol tiene asociado un valor heurístico que representa una cota inferior del coste de la mejor solución (la de menor coste) contenida en el nodo. Generalmente, cada vez que la cota inferior de un nodo rebasa el menor de los costes de los nodos hoja encontrados hasta el momento en el resto del árbol, dicho nodo es podado.

Para ahorrar espacio en memoria, generalmente la estrategia de control para la exploración del árbol suele ser en profundidad. En este caso, *ABIERTA* actúa como una pila y se introducen en ella los nodos generados en cada expansión ordenados según los valores de sus cotas inferiores (de menor a mayor). La eficiencia del algoritmo de ramificación y poda dependerá de lo ajustadas que sean las cotas inferiores obtenidas de forma heurística. Por otra parte, los algoritmos de ramificación y poda son admisibles si recorren todo el espacio de búsqueda, excepto las partes podadas, hasta que *ABIERTA* se agote; en caso contrario, si la condición de terminación es que el algoritmo pare después de expandir cierto número de nodos, se pierde la admisibilidad, ganando eficiencia, y únicamente se puede devolver la mejor solución encontrada hasta el momento.

Este algoritmo tiene cuatro componentes fundamentales:

Esquema de ramificación. La ramificación de un nodo es equivalente al cálculo de sucesores en los algoritmos de búsqueda anteriores.

Cálculo de cotas inferiores. El cálculo de cotas inferiores para un nodo n es análogo al cálculo de heurísticos admisibles.

Cálculo de cotas superiores. Las cotas superiores están asociadas a soluciones reales del problema completo.

Estrategia de control. La estrategia de control se refiere a la gestión de la lista ABIERTA, que contiene los nodos candidatos a ser desarrollados mediante el esquema de ramificación, es decir a la estrategia que se utiliza para decidir qué nodos entran en ABIERTA y cuál es el siguiente nodo expandido. Generalmente, por razones de eficiencia los algoritmos de ramificación y poda utilizan una estrategia de búsqueda en profundidad, es decir ABIERTA se trata como una pila. Para asegurar la condición de admisibilidad, los algoritmos de ramificación y poda deben recorrer todo el espacio de búsqueda, salvo las partes que se podan, antes de terminar, ya que, en general, la primera solución que encuentran no es la óptima.
Al disponer de la información que proporcionan las cotas inferiores, se pueden ordenar los sucesores de cada nodo antes de insertarlos en ABIERTA.

9.7. Algoritmos de mejora iterativa o búsqueda local

La búsqueda local se suele emplear en problemas donde lo que interesa no es el camino desde el nodo inicial hasta el nodo objetivo, sino que el nodo objetivo ya contiene de por si toda la información necesaria, y el camino es irrelevante sólo interesando el nodo meta. Es el caso de los algoritmos de búsqueda local.

La búsqueda local consiste en realizar cambios en el estado actual que nos lleven a un estado vecino de cara a acabar en el estado meta tras una serie de iteraciones.

Estructura básica:

1. El algoritmo parte de una solución inicial.

2. En cada iteración calcula un conjunto de *soluciones vecinas* mediante una regla de vecindad.

3. Cada una de estas soluciones vecinas es evaluada, ésta es la tarea más crítica del algoritmo por el elevado tiempo de ejecución que puede suponer.

4. S*e selecciona una* de ellas con un criterio de aceptación, que suele ser elegir la de menor coste.

5. Si la solución escogida cumple el *criterio de aceptación* (normalmente ser mejor que la solución actual *S*), la solución elegida reemplaza a dicha solución *S*, pasando dicho vecino a ser el nodo actual.

6. El proceso continúa hasta que se cumple el *criterio de finalización*, que generalmente es agotar un número *n* de iteraciones, o que no se produzcan mejoras en la calidad del nodo actual en los últimos *n* intentos.

Los algoritmos de búsqueda local realizan una búsqueda en un espacio de soluciones potenciales del problema tratando de encontrar aquella que maximice, o minimice, una determinada función objetivo.

La búsqueda local realiza una búsqueda en un espacio de estados a los que se puede asociar una función objetivo. La función objetivo puede contener numerosos óptimos locales, lo cual puede dificultar enormemente la consecución del óptimo global. En cualquier caso, la búsqueda local es un método eficiente que puede encontrar soluciones aceptablemente buenas en un tiempo reducido. Además, se puede combinar con otros métodos de búsqueda más sofisticados, como pueden ser los algoritmos evolutivos. De forma ideal, la regla de vecindad debería cumplir la denominada "regla de conectividad", según la cual desde cualquier estado se debe poder alcanzar el objetivo óptimo mediante una secuencia de transformaciones.

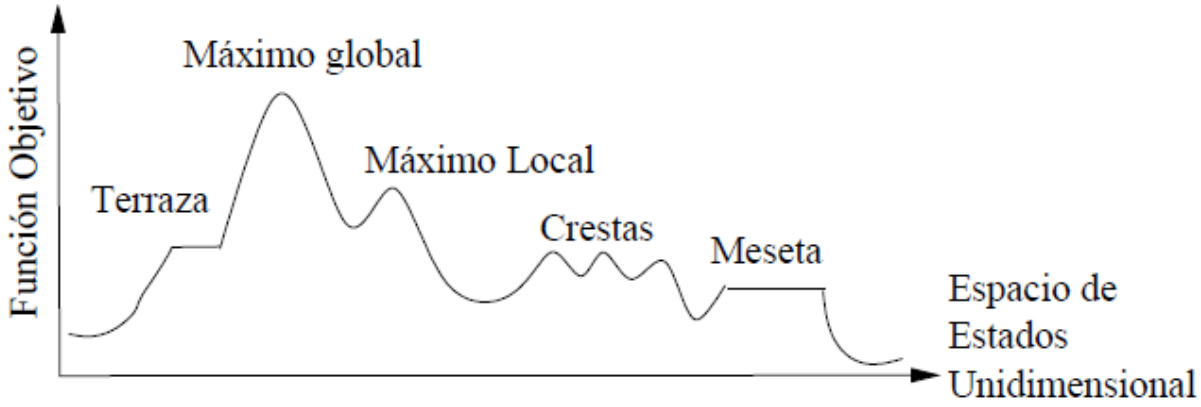
Algoritmo de escalada o máximo gradiente

En este tipo de búsqueda, el criterio de aceptación es que la solución vecina S_1 encontrada sea mejor o igual que la solución *S* actual. Previamente en la selección se intenta elegir una solución que mejore a S, o bien la mejor de todas las soluciones de vecindad. El criterio de aceptación se tiene que indicar, podría ser h(n).

Es una búsqueda simple, pero que tiene algunos problemas:

- óptimos locales: pico que es más alto que cada uno de sus estados vecinos, pero más bajo que el máximo local.Si se alcanza uno de estos puntos del espacio de búsqueda, todos los vecinos serán peores, y el algoritmo finaliza sin encontrar el óptimo global
- regiones planas(Meseta): zona del espacio de estados con función de evaluación plana, en este caso todos los vecinos tendrán el mismo valor que la solución actual, y la búsqueda es aleatoria
- crestas: zona del espacio de estados con varios óptimos locales. Si hay una cresta con pendientes muy marcadas, puede ser fácil alcanzar la cima de la cresta, pero si la pendiente es suave, en la dirección del óptimo global, resulta difícil no desviarse hacia los lados de la cresta al ascender

Debido a esos problemas, la búsqueda puede quedarse atascada en un óptimo local. Una solución puede ser reiniciar la búsqueda a partir de otro punto de inicio, lo que se denomina *búsqueda multiarranque*. Con este método se alcanzaría probablemente otro óptimo local distinto, y así, después de varios arranques se alcanzaría una solución aceptable.



Temple simulado

La idea de este método consiste en admitir, con una cierta probabilidad, algunas transiciones en las que la solución actual empeore; de este modo se puede salir de óptimos locales.

El método de temple simulado tiene las siguientes peculiaridades:

- Realiza una selección aleatoria entre los vecinos del nodo actual. Si el nodo seleccionado mejora la solución actual, la búsqueda sigue como en el caso de máximo gradiente.
- Si no mejora, el criterio de aceptación permite admitir, con una cierta probabilidad, algunas transiciones entre estados en las que empeore el valor de la función objetivo. Dicha probabilidad depende de dos parámetros: la temperatura T y el incremento de energía ΔE = coste(vecino-seleccionado) – coste(estado-actual). Esto evita caer en óptimos locales durante la búsqueda. En cualquier caso, si el valor de la función objetivo (o "coste") del estado vecino elegido fuera mejor que el valor del estado actual ($\Delta E < 0$), la transición se realizaría siempre al igual que en el caso del algoritmo de escalada.
- Al principio de la búsqueda, la temperatura tiene un valor alto, de modo que la probabilidad de aceptar un estado peor que el actual sea grande, favoreciendo así la exploración del espacio de soluciones.. La temperatura va decreciendo a lo largo del proceso de búsqueda según un determinado plan de enfriamiento, que puede ser más rápido o más lento. Por tanto, a medida que la búsqueda progresa, cada vez es más probable que sólo se admitan soluciones que mejoren o igualen a la actual.
- Generalmente, la probabilidad de llegar a una buena solución mediante el temple simulado es mayor si la temperatura inicial es alta y el plan de enfriamiento es lento. Sin embargo, en dicho caso la búsqueda consume demasiado tiempo, ya que el criterio de terminación suele ser que la temperatura alcance un valor suficientemente pequeño.

Un inconveniente del temple simulado es que requiere un ajuste apropiado de sus parámetros en función del problema abordado, sobre todo del plan de enfriamiento, de cara a que el proceso de búsqueda resulte eficaz y se obtenga una buena solución. Este ajuste depende fuertemente de la forma de la función objetivo y de la distribución de sus óptimos locales, es decir, del problema particular que estemos intentando resolver.

Búsqueda tabú

Lo que caracteriza a este algoritmo es que dispone de una memoria, denominada *lista tabú*. En base a dicha lista, se puede evitar la generación de vecinos que conduzcan a solución no óptimas o ya revisadas, ahorrando tiempo y mejorando la eficiencia.

En algunos casos, se pueden aplicar excepciones a dicha lista, utilizando lo que se conoce como *criterio de aspiración*, que consiste en admitir nodos tabú que puedan mejorar la solución actual, expandiendo dichos nodos como si no estuviesen en la lista.

Un inconveniente de la búsqueda tabú es que requiere un ajuste apropiado de sus parámetros en función del problema abordado (sobre todo del tamaño de la lista tabú, del criterio para la construcción de la lista tabú o de la definición del criterio de aspiración), de cara a que el proceso de búsqueda resulte eficaz y se obtenga una buena solución. Este ajuste depende fuertemente de la forma de la función objetivo y de la distribución de sus óptimos locales, es decir, del problema particular que estemos intentando resolver.

T5 – Lógica.

Capítulo 2 del libro base

2.1 Introducción: ¿Por qué la Lógica?

El objetivo de la IA es la construcción de sistemas, tanto hardware como software, que sean capaces de replicar aspectos de lo que se suele considerar inteligencia.

La IA es el conjunto de técnicas, métodos, herramientas y metodologías que nos ayudan a construir sistemas que se comportan de manera similar a un humano en la resolución de problemas concretos.

Razonar significa obtener conclusiones (correctas) a partir de ciertas premisas (que consideramos correctas). Si el método (sea cual sea) que utilicemos nos devuelve siempre conclusiones correctas, entonces diremos que el método (de deducción) es correcto; si, además, el método es capaz de devolvernos todas las conclusiones correctas, entonces lo llamamos completo

Dos problemas centrales en IA:
– Representación del conocimiento → **expresividad** de los sistemas de representación
– Razonamiento automático → **corrección**, **completud** y **complejidad** de los métodos de razonamiento

Una logica se caracteriza por:
– Lenguaje (formal)
– Semántica (significado)
– Calculo (razonamiento, deducción)

2.2 Lógica proposicional

En esta sección nos centraremos en el lenguaje lógico más sencillo, es decir, el lenguaje de la lógica proposicional o de orden 0, denotada también como LP.

2.2.1 Sintaxis y semántica

La gramática nos indica qué secuencias de signos del vocabulario están bien construidas. A estas secuencias se las llama fórmulas bien formadas. El lenguaje de la lógica proposicional clásica (LP), está basado en un conjunto numerable (no necesariamente finito) de proposiciones AP (del inglés Atomic Propositions), y sus fórmulas bien formadas son todas y solamente todas las fórmulas que se pueden obtener a través de la siguiente gramática abstracta:

$$\phi ::= p \mid \neg \phi \mid \phi \wedge \phi, \qquad (2.1)$$

siendo $p \in AP$. Por ahora, podemos pensar que las fórmulas de LP cuentan con un conjunto extendido de conectivas, es decir, el conjunto $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$, por lo que la gramática abstracta se convierte en la siguiente:

$$\phi ::= p \mid \neg \phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid \phi \rightarrow \phi \mid \phi \leftrightarrow \phi \qquad (2.2)$$

donde $p \in AP$.

El primer problema es distinguir entre una fórmula bien formada y otra que no lo es.

Por ejemplo, la fórmula

$$\phi = (p \wedge q) \vee (p \rightarrow \neg \neg (q \vee r))$$

está bien formada con respecto a la gramática (2.2), pero no con respecto a la gramática(2.1), mientras la fórmula

$$\phi = \neg (p \wedge \vee (p \rightarrow \neg \neg (q \vee r)))$$

no está bien formada para ninguna de las dos.

Una proposición es una expresión en lenguaje natural que sólo puede ser falsa (F) o verdadera (V) (en la semántica del sentido común). Reconocer las proposiciones y las conectivas en una frase del lenguaje natural permite obtener una formulación correcta, y pasar sin pérdida de información a un lenguaje formal como el de LP.

¿Cómo se evalúa una fórmula ϕ de LP? Un modelo (o mundo) es una asignación de valores de verdad a todos los símbolos proposicionales que aparecen en la fórmula o en el conjunto de fórmulas del que estamos hablando. Se dice que una fórmula es satisfacible si existe, al menos, un modelo que la satisface; en caso contrario, se dice que la fórmula es insatisfacible. Para saber si un modelo M satisface una fórmula ϕ aplicamos las siguientes reglas de forma recursiva:

- $M \models p$ si y sólo si $p = V$ en M;

- $M \models \neg \phi$ si y sólo si no es verdad que $M \models \phi$;

- $M \models \phi \wedge \varphi$ si y sólo si $M \models \phi$ y $M \models \varphi$.

Por ejemplo, si evaluamos la fórmula $p \wedge \neg q$ en un modelo $M = \{p = V, q = V\}$, obtenemos que $M \models p \wedge \neg q$ si y sólo si $M \models p$ y $M \models \neg q$, es decir, si y sólo si

p = V en M (que es cierto) y no es verdad que M |= q (que no es cierto). Por lo tanto, M |= p∧¬q. Para poder estudiar el comportamiento global de una fórmula es necesario evaluarla en todos los modelos posibles.

Se dice que dos fórmulas ϕ y ϕ son equivalentes si y sólo si tienen el mismo conjunto de modelos que las satisfacen, es decir, si para todo modelo M de ϕ y ϕ se cumple que M|= ϕ si y sólo si M|= ϕ.

Una tautología (o fórmula válida) es una fórmula de LP tal que, independientemente del modelo en la que la evaluamos, siempre es verdadera, como por ejemplo pv¬p. La negación de una tautología, que por lo tanto siempre es falsa, recibe el nombre de contradicción. Las tautologías son el eje central de cualquier lógica; de hecho, el problema de la decisión para una lógica consiste en determinar si una fórmula dada (bien formada) es, o no, una tautología en esa lógica. Desde un punto de vista práctico, podemos considerar que un razonamiento es un conjunto de fórmulas

$\Gamma = \{\phi 1, . . . , \phi n\}$ del cual se puede deducir una conclusión ϕ, es decir,

$$\phi 1 \wedge . . . \wedge \phi n \rightarrow \phi . \qquad (2.3)$$

Supongamos ahora que queremos preguntarnos si el razonamiento (2.3) es válido o no. Por lo que hemos dicho anteriormente, cualquier fórmula es válida si y sólo si su negación es una contradicción, es decir, en nuestro caso, si y sólo si la fórmula

$$\phi 1 \wedge . . . \wedge \phi n \wedge \neg \phi \qquad (2.4)$$

es una contradicción o, si preferimos, no hay ningún modelo que la satisface. Los métodos deductivos, que veremos más adelante, se dividen en dos clases:

1. Métodos sintácticos: buscan una demostración formal de la validez de una fórmula, a través de reglas de deducción

2. Métodos semánticos: buscan un contraejemplo para una fórmula, intentando demostrar que la fórmula no es satisfacible

2.2.2 Poder expresivo y límites de la Lógica Proposicional

LP es un lenguaje que permite expresar ciertos razonamientos y cuyos métodos deductivos, como veremos, son particularmente sencillos y aptos para la implementación. Además, el carácter finito de LP garantiza la decidibilidad del problema de la validez de fórmulas y conjuntos de fórmulas en LP.

2.2.3 Métodos deductivos semánticos y coste computacional

El método del árbol semántico.

Uno de los métodos más importantes para decidir la validez de una fórmula en LP es el método del árbol semántico, también conocido como tablero semántico o simplemente tableau, una posible técnica para demostrar que φ es una fórmula válida consiste en demostrar que $\neg \phi$ es una fórmula para la cual no hay modelos posibles que la satisfacen. Esto significa que, cada vez que intentamos construir un modelo para $\neg \phi$ asignando valores de verdad, nos encontramos con alguna contradicción. El ejemplo más sencillo consiste en demostrar que

$p \wedge \neg p$ *no es satisfacible*.

Las reglas dependen directamente de la semántica de las distintas conectivas:

• Conectivas conjuntivas(regla α), es decir, $\neg\neg$ y $\wedge$, expanden la rama actual: por ejemplo, si estamos evaluando $\phi \wedge \phi$, entonces, en la misma rama, evaluaremos ϕ y ϕ.

• Conectivas disyuntivas(regla β),, es decir, $\vee$, $\rightarrow$, $\leftrightarrow$, abren distintas ramas: por ejemplo, si estamos evaluando $\phi \vee \phi$, entonces, elejimos seguir o bien una rama en la que evaluamos ϕ, o bien una en que evaluamos ϕ.

Cada vez que una fórmula (o un nodo) se expande, su estado cambia, pasando de activo a no activo. Una fórmula no activa no se considera nunca más para el desarrollo del árbol semántico.

La conectiva $\wedge$ se representa por una arista vertical:

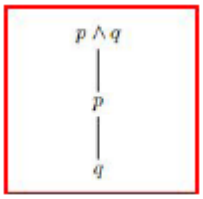


Figura 4: Conectiva ∧.

La conectiva V por un par de aristas de la forma:

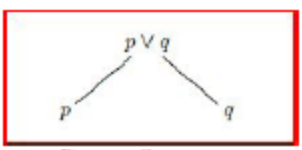


Figura 5: Conectiva V.

El resto de las conectivas se traducen a esta forma. Así, el condicional p→q se representa como:

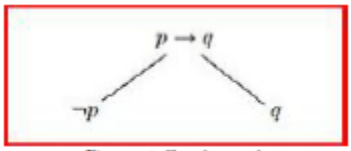


Figura 6: Condicional.

Se trata de ir descomponiendo cada proposición compuesta de acuerdo con las reglas anteriores, al hacerlo se irá generando el árbol semántico. Al descomponer una proposición cambiaremos su estado de activa a no activa, con lo que no volverá a considerarse más para el desarrollo del árbol.

Conviene descomponer primero los bicondicionales y sus negaciones antes que otras conectivas que creen ramas.

Si en una sucesión de nodos del árbol (camino), aparece una proposición y su negación, se dice que es un camino cerrado y se marca con * el nodo final.

Si al final del proceso todos los caminos están cerrados, la proposición de partida es una contradicción, con lo que queda validada la fórmula original. En caso contrario, cada camino abierto es un modelo que satisface la proposición inicial, así que la fórmula original no es válida.

Reglas de formación de árboles. La construcción comienza como un árbol (que se irá ramificando hacia abajo) cuya raíz es el conjunto de fórmulas de partida (cada fórmula en un nodo de la rama inicial), y se aplican las siguientes reglas:

Regla α: Si en una rama hay una fórmula de la clase α, se añaden al final de la rama sus componentes α_1 y α_2.

Regla β: Si en una rama hay una fórmula de la clase β, la rama se divide en dos, y en cada una de las ramas resultantes se añade una de las fórmulas β_1 y β_2.

Regla σ (Doble Negación): Si en una rama aparece una doble negación $\neg\neg\lambda$, se añade λ a la rama.

Regla de Cierre: Si en una rama aparecen una fórmula λ y su negación $\neg\lambda$, la construcción de la rama termina y decimos que queda cerrada. Se puede indicar con una marca $\otimes$.

La construcción del árbol semántico continúa hasta que todas sus ramas se cierran (decimos que es un **árbol cerrado**) o no quedan reglas por aplicar (a cada fórmula se ha debido aplicar la regla que sea posible en todas las ramas donde se encuentre). Observar que las únicas fórmulas a las que no se puede aplicar ninguna regla son los *literales* (variables proposicionales y sus negaciones). Si tras terminar la construcción queda alguna rama abierta, decimos que es un **árbol abierto**.

Teorema: Un conjunto de fórmulas Γ es satisfactible si y sólo si el árbol de Γ es abierto.

Teorema: Si el árbol semántico de Γ contiene una rama completa y abierta, se puede construir una interpretación que satisface Γ a partir de los literales de la rama. Toda interpretación que satisfaga los literales de la rama abierta, satisface Γ.

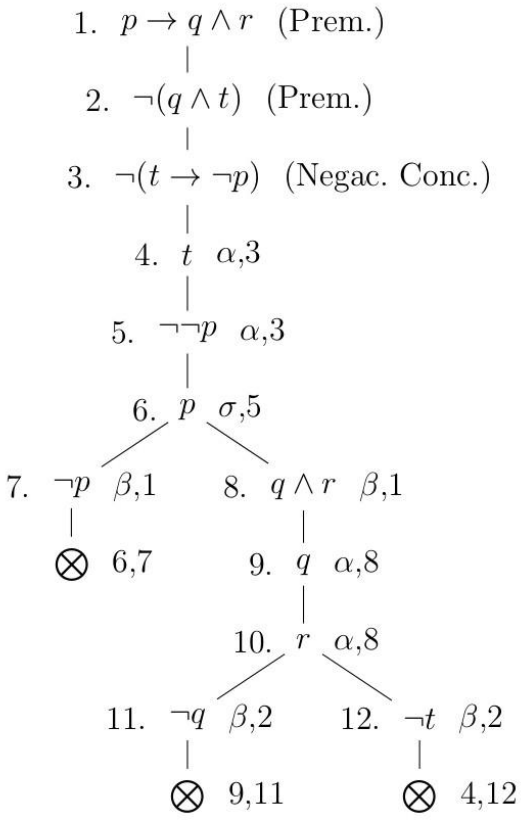
α	α_1	α_2
$\varphi \wedge \psi$	φ	ψ
$\neg(\varphi \vee \psi)$	$\neg\varphi$	$\neg\psi$
$\neg(\varphi \to \psi)$	φ	$\neg\psi$

β	β_1	β_2
$\varphi \vee \psi$	φ	ψ
$\neg(\varphi \wedge \psi)$	$\neg\varphi$	$\neg\psi$
$\varphi \to \psi$	$\neg\varphi$	ψ
$\varphi \leftrightarrow \psi$	$\varphi \wedge \psi$	$\neg\varphi \wedge \neg\psi$
$\neg(\varphi \leftrightarrow \psi)$	$\varphi \wedge \neg\psi$	$\neg\varphi \wedge \psi$

Algunos usos de los árboles semánticos

- La fórmula α es consecuencia lógica del conjunto de fórmulas Γ si y sólo si el árbol de $\Gamma \cup \{\neg\alpha\}$ es cerrado. Si el árbol es abierto, a partir de una de las ramas abiertas de puede construir una interpretación que satisface Γ pero no α (contraejemplo).
- La fórmula α es universalmente válida (tautología) si y sólo si el árbol de $\{\neg\alpha\}$ es cerrado. Si el árbol es abierto, a partir de una rama abierta se puede construir una interpretación que hace falsa α.
- La fórmula α es satisfacible si y sólo si el árbol de $\{\alpha\}$ es abierto. A partir de cada una de las ramas abiertas, se puede construir una interpretación que hace verdadera α.

Comprobamos $\{p \to q \wedge r,\ \neg(q \wedge t)\} \models t \to \neg p$ mediante un árbol semántico. Tenemos, por tanto, que construir el árbol de las premisas y la negación de la conclusión:



El método de resolución. La idea básica del método de resolución para el lenguaje de LP es la siguiente. Supongamos que Γ_1 y Γ_2 son dos conjuntos de

disyunciones de fórmulas de LP (p.e., $\Gamma 1 = \{\phi 1, \phi 2, . . . , \phi k\} = (\phi 1 \vee \phi 2 \vee . . . \vee \phi k)$, y ϕ una fórmula de LP. Si sabemos que las fórmulas $\Gamma \vee \phi$ y $\Gamma \vee \neg \phi$ son válidas a la vez, las cuales podemos expresar en notación de conjuntos como:

$\Gamma 1 \cup \{\phi\} \Gamma 2 \cup \{\neg \phi\}$,

entonces podemos deducir que también es válido (regla de resolución)

$\Gamma 1 \cup \Gamma 2$.

$\Gamma \rightarrow$ Significa *cualquier cosa*

El método de la resolución está basado en el reconocimiento de pares (fórmula, negación); por lo tanto, es necesario utilizar reglas adecuadas de transformación que nos permitan llegar a una forma común para las fórmulas de LP en la que estamos interesados. Las reglas más importantes son:

- $\phi \leftrightarrow \phi = \phi \rightarrow \phi \wedge \phi \rightarrow \phi$;
- $\phi \rightarrow \phi = \neg \phi \vee \phi$;
- $\neg (\phi \wedge \phi) = \neg \phi \vee \neg \phi$ (ley de De Morgan);
- $\neg (\phi \vee \phi) = \neg \phi \wedge \neg \phi$ (ley de De Morgan);
- $\neg \neg \phi = \phi$;
- $\phi \wedge (\phi \vee \varphi) = (\phi \wedge \phi) \vee (\phi \wedge \varphi)$;
- $\phi \vee (\phi \wedge \varphi) = (\phi \vee \phi) \wedge (\phi \vee \varphi)$;

Por ejemplo, la fórmula $\neg (p \rightarrow (q \wedge r))$ puede transformarse del siguiente modo:

1. $\neg (p \rightarrow (q \wedge r))$ (fórmula de salida);
2. $\neg (\neg p \vee (q \wedge r))$ (elminación de $\rightarrow$);
3. $\neg \neg p \wedge \neg (q \wedge r))$ (ley de De Morgan);
4. $p \wedge \neg (q \wedge r))$ (eliminación de $\neg$);
5. $p \wedge (\neg q \vee \neg r)$ (ley de De Morgan).

Una disyunción de literales ($\phi 1 \vee . . . \vee \phi k$) se denomina cláusula, y una fórmula expresada como conjunción de cláusulas se dice que está en forma normal conjuntiva.

Cuando queramos demostrar que una fórmula es insatisfacible a través de la resolución, primero hay que negar dicha fórmula y después hay que transformarla en conjunción de cláusulas de la misma manera que se hizo en el ejemplo anterior.

Como ejemplo, consideremos la fórmula

$$\phi = (p \wedge (\neg q \rightarrow \neg p) \wedge (q \rightarrow r)) \rightarrow r.$$

Si queremos demostrar que ϕ es un razonamiento válido, procederemos aplicando el algoritmo de resolución a la fórmula $\neg \phi$, es decir, a la fórmula

$$\phi = (p \wedge (\neg q \rightarrow \neg p) \wedge (q \rightarrow r)) \wedge \neg r.$$

Después de los pasos necesarios de trasformación, obtenemos el siguiente conjunto de cláusulas:

$\{p\}, \{q, \neg p\}, \{\neg q, r\}, \{\neg r\},$

al que se puede aplicar la resolución como se muestra en la Figura 2.2

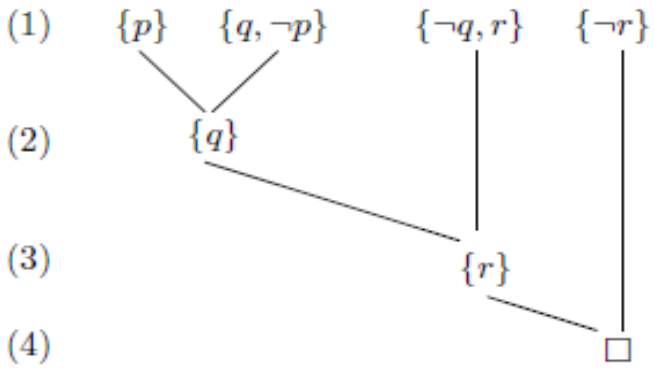


Figura 2.2: Un ejemplo de resolución.

Optimizar la resolución: cláusulas de Horn. La complejidad de LP se puede mejorar pagando el precio de tener una menor expresividad realizando una deducción lineal en vez de exponencial. De esta manera perdemos en expresividad (ya no permitimos cualquier tipo de fórmula) a cambio de ganar en complejidad. Es posible demostrar que si en un conjunto de cláusulas todas tienen una forma llamada de Horn, entonces el problema de la satisfacibilidad tiene una complejidad inferior. Una cláusula de Horn es:

- Un átomo. Hechos: p
- Una implicación tal que el antecedente es una conjunción de literales positivos, y el consecuente es un solo literal positivo. Reglas: $p \wedge q \wedge r \rightarrow s$

• Una implicación tal que el antecedente es una conjunción de literales positivos, y el consecuente es falso. Reglas: $p \wedge q \wedge r \rightarrow \neg s$

Por ejemplo, $p, p \wedge q \wedge r \rightarrow s$ son cláusulas de Horn, mientras que $p \wedge q$ no lo es.

El lenguaje prolog se basa en resolución con cláusulas de Horn.

2.3 Lógica de primer orden

En esta sección, nos centraremos en la clásica extensión del lenguaje de LP, el lenguaje de la lógica de primer orden o lógica de predicados LPO.

2.3.1 Sintaxis y semántica

La lógica de primer orden es una extensión de LP, en la cual se introducen variables para denotar elementos del dominio, cuantificadores y predicados.

Enriquecemos el lenguaje de LP con:

– Constantes (a, b, c, …) y variables (x, y, z, …)

– Functores (f, g, h, …) y predicados (p, q, r, …)

– Cuantificadores: existencial $\exists$ y universal $\forall$.

Son términos: las constantes, las variables, y los functores con sus argumentos instanciados por términos (f(a,x), f(b,g(a)), …)

Las formulas atómicas se componen de un predicado con sus argumentos instanciados por términos (p(a,b), q(x), …)

Ejemplos de fórmulas: $\exists x\ p(x)$, $\forall x\ (p(x) \rightarrow \neg q(x,a))$, …

Considérese el ejemplo siguiente:

El robot se encuentra con el brazo mecánico vacío. Si el robot está sujetando algo en el brazo mecánico, no puede levantar nada más desde el suelo. Si existe un objeto que se encuentra en el suelo, entonces el robot tiene que levantarlo o pasar a su lado. Si el robot encuentra un obstáculo y no puede levantarlo, entonces tiene que pasar a su lado, dejar lo que tiene en el brazo en el punto P, y volver a por el objeto que ha encontrado, hasta que todos los objetos están en el punto P.

Mediante LPO, somos capaces de formalizar el anterior párrafo de manera muy sencilla.

La propiedad vacío del brazo mecánico podría ser indicada con p (un predicado

0-ario, es decir, una proposición). Levantar un objeto, o sujetarlo, es un predicado unario q(x), donde el parámetro x indica el objeto sujetado. Entonces, tenemos:

$$\exists x(q(x) \rightarrow \neg p).$$

Asimismo, la premisa si existe un objeto que se encuentra en el suelo, entonces el robot tiene que levantarlo o pasar a su lado se puede formalizar con

$$\exists x(r(x) \rightarrow s(x) \vee t(x)),$$

donde r(x), s(x), t(x) son los respectivos predicados unarios.

Las fórmulas de LPO se interpretan a partir de un dominio D o universo de discurso (conjunto no vacío).

En un modelo M:

- Cada constante c se interpreta como c^M, un elemento de D.
- Cada functor f con n argumentos, se interpreta como f^M, una función de D^n en D.
- Cada predicado p de n argumentos se interpreta como p^M, un elemento de D^n.

Las constantes que aparecen en los términos se interpretan directamente; por ejemplo, en la fórmula

Hombre(Sócrates)

la constante Sócrates es un símbolo que viene interpretado como el famosos filósofo Sócrates en un dominio en el que aparecen objetos, personas, animales,.

En el lenguaje de LPO se puede expresar una gran variedad de situaciones. Por ejemplo:

Si el robot tiene la batería cargada, y cumple el conjunto de acciones A,

entonces luego tendrá la batería descargada.

En este ejemplo queremos tener en cuenta la componente temporal. Supongamos que modelamos el tiempo como un conjunto de puntos D (nuestro dominio) linealmente ordenado; los predicados pA (el robot cumple el conjunto de acciones A), pC (batería cargada), y pD (batería descargada) indican las situaciones que queremos modelar, y dependen de un parámetro que indica el momento temporal. El problema es que debemos añadir las condiciones para que el dominio sea linealmente ordenado:

$\forall x(\neg(x < x))$ (2.5) Irreflexividad

$\forall x, y, z(x < y \wedge y < z \rightarrow x < z)$ (2.6) Transitividad

$\forall x, y(x < y \vee y < x \vee x = y)$, (2.8) Disyuntividad

y, finalmente, podemos formalizar el ejemplo anterior como sigue:

$(2.5)\wedge(2.6)\wedge(2.8)\wedge\exists x(pC(x) \wedge \exists y(x < y \wedge pA(y) \rightarrow \exists z(y < z \wedge pD(z)))$.

2.3.2 Poder expresivo y límites de la lógica de primer orden

El gran problema de la IA es qué expresar, y no cómo expresarlo. La lógica LPO proporciona un lenguaje uniforme con el que se puede formalizar el conocimiento. El primer paso siempre consiste en conceptualizar el conocimiento en términos de objetos, funciones y relaciones. Luego, utilizamos las expresiones y las fórmulas cuyo significado involucra a dichos objetos, funciones y relaciones. Finalmente, nos preguntamos sobre la satisfacibilidad de las fórmulas teniendo en cuenta únicamente los modelos que respetan ciertas características de nuestro interés. En el lenguaje de LPO podemos expresar gran variedad de situaciones de complejidad arbitraria; sin embargo, los límites más evidentes de LPO, consisten en el esfuerzo (que se traduce en complejidad de las fórmulas) que tenemos que hacer para expresar situaciones típicas en IA en las que hay que tener en cuenta relaciones con determinadas características. Expresar características algebraicas necesarias supone fórmulas relativamente largas; por último, como hemos dicho anteriormente, se cumple la ecuación: *más poder expresivo = más complejidad computacional* (en casos como LPO también indecidibilidad)= métodos deductivos más complicados.

2.3.3 Métodos deductivos y coste computacional

El método del árbol semántico. Los métodos de deducción para el lenguaje de LPO son básicamente extensiones de los métodos para el lenguaje de LP. En el caso del método basado en el desarrollo de un árbol semántico para averiguar si cierta fórmula es insatisfacible, debemos de tener en cuenta que, en general no es posible dar una metodología de decisión para LPO que termine en todos los casos. Los límites de las metodologías para LPO están en el carácter no finito de los dominios; de hecho, no es difícil escribir una fórmula en el lenguaje de LPO que sólo admita modelos infinitos; supongamos por ejemplo que el predicado < ha sido formalizado sobre un orden lineal en el dominio (véase el último ejemplo de la sección 2.3.1), y consideremos la fórmula:

$$\exists x(p(x)) \wedge \forall x(p(x) \rightarrow \exists y(x < y \wedge p(y))). \qquad (2.9)$$

Para satisfacer la fórmula anterior es necesario tener un dominio infinito. A pesar de esta limitación en los algoritmos de deducción, el problema de la satisfactibilidad de la lógica LPO se encuentra en una clase de problemas 'no demasiado complicados'; en otras palabras, existen algoritmos (como los que veremos) para LPO que son correctos y completos, aunque no siempre terminan. Esta clase de problemas se denominan semidecidibles,

El método del árbol semántico para el lenguaje LPO contiene las mismas reglas que en el caso de LP, más las siguientes reglas precisas para los cuantificadores:

- Cuantificador universal, es decir, $\forall x\, \phi(x)$: en este caso, para todas las constantes a que han sido utilizadas en la rama considerada evaluaremos la fórmula $\phi(a)$, teniendo en cuenta que la fórmula $\forall x(p(x))$ permanece activa.
- Cuantificador existencial, es decir, $\exists x\, \phi(x)$: en este caso, es preciso introducir una constante a nueva (que no haya sido utilizada en ninguna rama abierta hasta el momento), y evaluar la fórmula $\phi(a)$.

A la hora de la resolución de los tableaux, primero utilizar las reglas que instancian las nuevas constantes del cuantificador existencial y luego las del universal.

El método de resolución. En el caso de LPO el método de resolución es un poco más complicado que en el caso de LP. Básicamente hay dos problemas nuevos con respecto al caso proposicional:

1. Los cuantificadores en la fase de transformación de la fórmula de salida en forma clausal.

2. El reconocimiento de las fórmulas atómicas contradictorias, cuando éstas contengan variables o términos.

Empezamos con la transformación de ϕ en forma clausal. El algoritmo que permite esta transformación debe tener en cuenta tanto las reglas utilizadas en el caso de LPO, más algunas reglas que permiten poner todos los cuantificadores en la parte izquierda de la fórmula. Una fórmula que tiene esta estructura se dice que está en forma prenexa.

Las reglas que permiten poner ϕ en forma prenexa son las siguientes:

• $\phi \circ Qx\varphi(x) = Qx(\phi \circ \varphi(x))$ (donde x no aparece libre en ϕ);

• $\phi \circ Qx\varphi(x) = Qy(\phi \circ \varphi(y))$ (donde x aparece libre en ϕ, y la variable y es nueva), donde $\circ \in \{\wedge, \vee\}$, y $Q \in \{\forall, \exists\}$.

Veamos un ejemplo; consideremos la fórmula

$$\neg(\exists x(p(x) \rightarrow \forall y(q(x, y)))):$$

1. $\neg(\exists x(\neg p(x) \vee \forall y(q(x, y))))$ (eliminación de $\rightarrow$);
2. $(\forall x \neg(\neg p(x) \vee \forall y(q(x, y))))$ (interdefinición de $\forall$ y $\exists$);
3. $(\forall x(\neg \neg p(x) \wedge \neg \forall y(q(x, y))))$ (ley de De Morgan);
4. $(\forall x(p(x) \wedge \exists y \neg(q(x, y))))$ (eliminación de $\neg \neg$, e interdefinición de $\forall$ y $\exists$);
5. $\forall x \exists y(p(x) \wedge \neg(q(x, y)))$ (cuantificador $\exists$).

Para aplicar resolución, primero hay que transformar la formula a forma normal de Skolem:

– Mediante equivalencias, se comienza llevando todos los cuantificadores al inicio de la formula (forma prenexa)

– Aplicamos recursivamente:

- $\forall x \Phi(x)$ se transforma en $\Phi(x)$, es decir, se quita el cuantificador y la variable queda libre.
- $\exists x \Phi(x)$ se transforma en $\Phi(f(x1, \cdots, xn))$, donde f es un nuevo functor y las variables x1,…, xn son todas aquellas que estaban cuantificadas universalmente a la izquierda del existencial que se elimina. Si no había ninguna, f se convierte en una constante nueva.

2.3.5 Fragmentos de LPO

Con respecto al lenguaje de LPO es posible identificar algunos subconjuntos (de carácter sintáctico) que permiten mejorar las propiedades computacionales.

Cuando consideramos una gramática abstracta para un cierto lenguaje L que resulta ser un subconjunto de la gramática para un lenguaje L', entonces denominamos L un fragmento (sintáctico) de L'.

Ejemplos en este sentido son:

– Lógicas con número limitado de variables. En cada formula se permite un máximo de n variables.

– Fragmentos con guardia. Se limita la cuantificación, por ejemplo

$\forall x\ Px$ se transforma en $\forall x\ (R(x,a) \rightarrow Px)$

– Fragmentos donde se limita la aridad máxima de los predicados.

Ejemplo: LPO con predicados monádicos es decidible.

Las lógicas de primer orden con un número limitado de variables son básicamente lógicas de primer orden en las que se permiten utilizar solamente algunos símbolos de variable, los cuales se pueden reutilizar un número arbitrario de veces.

Los fragmentos con guardia, por otra parte, han sido propuestos como resultado de los estudios de decidibilidad de las lógicas modales.

Un fragmento de LPO se define con guardia si la cuantificación de las variables está limitada por alguna relación. Por ejemplo, en un fragmento con guardia la fórmula

$$\forall x\ \phi(x)$$

no se admitiría; en su lugar, se supone la presencia de una relación, digamos, binaria, R, y se cuantifica así:

$$\forall x(xRy \rightarrow \phi(x))$$

2.4 Extensiones de las lógicas clásicas

2.4.1 ¿Por qué extender las lógicas clásicas?

El lenguaje LP es muy poco expresivo, pero tiene buenas propiedades computacionales y sus métodos deductivos son fácilmente implementables; por otro lado, prácticamente todo lo que podemos expresar con respecto a los razonamientos de un sistema inteligente encuentra su formalización en el lenguaje LPO, cuyos sistemas deductivos tienen malas propiedades computacionales. Pero, ¿cuáles son exactamente las características que nos gustaría poder expresar? Podríamos enumerar algunas de ellas:

- Capacidad de expresar situaciones hipotéticas, mundos (realidades) posibles, relacionadas de alguna forma.
- Capacidad de expresar situaciones de carácter temporal.
- Capacidad de expresar situaciones de carácter espacial.

Entonces, ¿es posible alcanzar dichos objetivos utilizando alguna extensión del lenguaje de LP pero que no suponga utilizar todo el poder expresivo de LPO (que, por lo visto, es la fuente de la indecidibilidad de LPO)?

Las lógicas modales permiten expresar, de forma sencilla, situaciones hipotéticas y mundos posibles, y, en alguna de sus especializaciones, también situaciones de carácter espacio/temporal.

2.4.2 Lógicas no monotónicas, razonamiento del sentido común y otras consideraciones

Una lógica se dice no monótona cuando es capaz, de alguna manera, de volver atrás en sus conclusiones. De hecho, en el mundo real, donde nos regimos por el sentido común, algunas conclusiones son ciertas hasta que una nueva información nos hace cambiar de idea, y pone en duda toda la línea de razonamiento seguida hasta el momento. Dicho de otra forma, las creencias o conclusiones son derrotables o anulables. En la lógica clásica este tipo de razonamiento no puede ser expresado, siendo la lógica clásica monótona, es decir, permite solamente alcanzar, a través de la deducción, nuevas verdades. Intuitivamente, la propiedad de monotonicidad de la lógica clásica nos dice que el hecho de aprender algo nuevo, no puede reducir el conjunto de cosas que ya sabemos. Por el contrario, en una lógica no monótona, la verdad de una proposición puede cambiar según vayan apareciendo nuevos axiomas. Mientras que en una lógica clásica temporal o espacial este comportamiento está regulado por alguna estructura fija (como un orden lineal), en una lógica no monótona no tenemos esta limitación. La lógica clásica (en todas sus formas) utiliza reglas del tipo: si p es un teorema, entonces q es un teorema; en una lógica no monótona, una regla de inferencia puede ser del tipo q es un teorema si p no es un teorema.

Otras extensiones de la lógica clásica son las lógicas multivaluadas, las cuales, se caracterizan por permitir más de dos valores de verdad; un tipo especial de lógica multivaluada es la lógica borrosa o difusa donde las proposiciones tienen un grado de verdad que se asigna mediante una función de pertenencia que toma valores en el intervalo real [0, 1].

Por último, mencionamos la lógica intuicionista cuya sintaxis es la misma que la de la lógica proposicional o de primer orden, con la principal diferencia de que en esta lógica algunas fórmulas como $\phi \vee \neg \phi$ o $\neg \neg \phi \rightarrow \phi$ no son tautologías.

2.4.3 Lógicas modales y mundos posibles

La lógica modal intenta acercarse más al razonamiento humano completando la lógica clásica al introducir el concepto modalidad. Esto significa que es posible indicar el modo en que es cierta o falsa una proposición (cuándo, dónde, bajo qué condiciones) y con ello expresar los conceptos de necesidad y posibilidad. Lo anterior se modela considerando un conjunto de mundos posibles relacionados por una relación de accesibilidad y cuantificadores existenciales que se aplican a las proposiciones permitiendo desplazarse entre mundos. Cada proposición tiene asignado un valor de verdad en cada mundo. Una fórmula es modalmente satisfacible si lo es en al menos un mundo.

Una lógica modal es un sistema formal que intenta capturar el comportamiento deductivo de algún grupo de operadores modales. Los operadores modales son expresiones que califican la verdad de los juicios. Por ejemplo, en la oración «es necesario que 2+2=4», la expresión «es necesario que» es un operador modal que califica de necesaria a la verdad del juicio «2+2=4».

En un sentido más restringido, sin embargo, se llama lógica modal al sistema formal que se ocupa de las expresiones «es necesario que» y «es posible que».

Imaginemos una gramática formal extendida para LP, como la siguiente:

$$\phi \ ::= p \mid \neg \phi \mid \phi \wedge \phi \mid \Diamond \phi$$

donde p es un símbolo proposicional, y ◇ es el cuantificador existencial que permite desplazarse entre los mundos.

El lenguaje proposicional se extiende en lógicas modales con dos operadores

- $\Box \phi$: es necesario ϕ
- $\Diamond \phi$: es posible ϕ

Si disponemos de varios mundos ,W={w1,w2,…} y de una relación R que permite pasar de un mundo wi a otro mundo wj, un modelo de la lógica modal es una terna M=(W,R,V) donde:

- W es un conjunto de mundos posibles (o modelos de LP), W={w1,w2..}
- R es una relación binaria de accesibilidad entre mundos, $R \subseteq W \times W$. Si estamos en el mundo w1 resulta posible w2. La función de la relación de accesibilidad es ayudar a expresar una necesidad o posibilidad relativa. En principio, no todo lo que es posible en un mundo es posible en otro mundo.
- V es una función de evaluación que, por cada uno de los mundos, nos dice cuáles son los símbolos proposicionales que se satisfacen en ellos.

Dado un modelo M = (W, R, V) y un mundo wϵW,

- M, w |= p syss pϵV(w) → En el mundo w ¿Cuándo es verdadero p? *Sí y sólo sí, cuando p es verdadero en todos los mundos.*

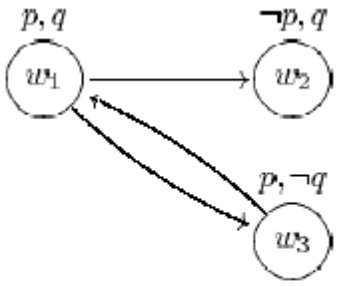
- M, w |= ¬ Φ syss no se cumple M, w |= Φ → En el mundo w ¿Cuándo no es verdadero Φ? *Sí y sólo sí, cuando Φ no es verdadero en todos los mundos*

- M, w |= Φ ^ψ syss M, w |= Φ y M, w |= ψ → En el mundo w ¿Cuándo es verdadero Φ y ψ? *Sí y sólo sí, cuando Φ y ψ son verdaderos en todos los mundos.*

- M, w |= □ Φ syss para todo w'ϵW, wRw' implica M, w' |= Φ → En el mundo w ¿Cuándo es necesario que Φ sea verdadero? Cuando Φ es verdadero en todos los mundos accesibles.

- M, w |= ◇ Φ syss existe un w'ϵW, tal que wRw' y M, w' |= Φ → En el mundo w ¿Cuándo es posible que Φ sea verdadero? *Cuando Φ es accesible desde algún mundo.*

Si para todo wϵW se cumple M, w |= Φ, decimos M |= Φ. → Φ es verdadero en todos los mundos



M,w_1 |= ¬□p ^◊p ^ □(p v q)

M |= □ (p v q)

M,w_2 |= □p ^ □¬p

Figura 2.5: Un modelo modal.

El modelo representado tiene tres mundos, w1, w2 y w3, y es un modelo para un lenguaje en el que sólo figuran dos símbolos proposicionales, es decir, p y q. Vemos que en el mundo w1 ambas, p y q, se realizan, en el mundo w2 se realiza q pero no p, mientras que en el mundo w3 se realiza p pero no q. Además, sabemos que desde w1 podemos llegar a w2 y w3, pero, por ejemplo, desde w2 no podemos llegar a w1.

La satisfacibilidad de una fórmula ϕ en un modelo modal M, depende de una serie de parámetros:

Las propiedades de la relación R.

El mundo en que evaluamos la fórmula.

Es decir, la misma fórmula puede ser evaluada a verdadero en un modelo M y en un mundo wi, y a falso en el mismo modelo pero en otro mundo wj .
Formalmente:

$$M, wi \Vdash \Diamond\phi \text{ si y sólo si existe } wj \text{ tal que } R(wi, wj) \text{ y } M, wj \Vdash \phi .$$

Para expresar una propiedad que es cierta para todos los mundos que son alcanzables a partir del mundo actual, utilizamos el símbolo $\Box = \neg\Diamond\neg$.

En un lenguaje modal, decimos que una fórmula ϕ es satisfacible si y sólo si existe un modelo modal M y un mundo wi tales que M,wi $\Vdash \phi$.

Asimismo, decimos que ϕ es válida en un modelo M si y sólo si M,wi $\Vdash \phi$ para cualquier mundo wi, y válida si y sólo si es válida en todos los modelos modales posibles.

Las propiedades computacionales de una lógica modal dependen directamente de las propiedades de la relación R.

Existen varias lógicas modales, que han sido aplicadas en varios contextos, que se pueden clasificar según las propiedades de R; algunos ejemplos de lógicas modales pueden verse en la Tabla 2.2.

Nombre	Propiedad
K	No hay condiciones sobre la relación R
K4	R es transitiva, es decir, $\forall x, y, z(r(x, y) \wedge r(y, z) \rightarrow r(x, z))$
T	R es reflexiva, es decir, $\forall x R(x, x)$
B	R es simétrica, es decir, $\forall x, y(R(x, y) \leftrightarrow R(y, x))$

Tabla 2.2: Algunas lógicas modales.

El lenguaje de LM ha sido interpretado de diferentes maneras, cada una de las cuales tiene diferentes aplicaciones. En particular, hay tres interpretaciones que se pueden considerar, desde un punto de vista histórico, las más importantes.

- Primero, el símbolo $\Diamond$ puede interpretarse como posible: en este caso, la fórmula $\Diamond\phi$ se lee ϕ es posible, y la fórmula $\Box\phi$ se lee ϕ es necesario. En esta interpretación, podemos expresar conceptos como todo lo que es necesario es posible, con la fórmula $\Box\phi \rightarrow\Diamond\phi$ (o sea, requiriendo que la fórmula sea válida).

- Segundo, en la lógica epistémica, la fórmula $\Box\phi$ se interpreta como el agente (inteligente) conoce ϕ. Esta interpretación se utiliza para la representación del conocimiento.

- Tercero, en la lógica de la demostrabilidad, la fórmula $\Box\phi$ se interpreta como ϕ es demostrable (en cierta teoría aritmética).

2.4.4 Métodos deductivos y coste computacional de la Lógica Modal

La variante más sencilla es la lógica K

El método deductivo para K es la extensión del método basado en árboles semánticos para la lógica LP.

Árboles semánticos para K. El primer paso siempre consiste en poner la fórmula considerada en una forma más sencilla, queremos obtener es una forma que tenga los operadores ¬ sólo delante de símbolos proposicionales. Las reglas son las mismas que vimos en el caso de la resolución para LP, más las dos reglas siguientes:

- transforma $\neg \Diamond \phi$ en $\Box \neg \phi$;
- transforma $\neg \Box \phi$ en $\Diamond \neg \phi$.

Por ejemplo, si tenemos la fórmula $\neg \Box (p \wedge \neg \Diamond q)$, podemos aplicar las reglas que hemos visto, obteniendo $\Diamond \neg (p \wedge \Box \neg q)$, y, con otro paso, $\Diamond (\neg p \vee \neg \Box \neg q)$, y, finalmente, $\Diamond (\neg p \vee \Diamond q)$.

Exactamente como en el caso de LP, una rama quedará cerrada en cuanto se encuentre un símbolo proposicional y su negación; la diferencia está en que esta contradicción habrá que encontrarla en el mismo mundo de evaluación.

La regla que permitirá el desarrollo de un caso como $\Diamond \phi$ será la encargada de construir nuevos mundos de evaluación, mientras que la regla para el caso $\Box \phi$ aplicará la evaluación de ϕ a todos los mundos accesibles a partir del mundo actual.

La lógica K no aplica ninguna restricción a la relación R, lo que significa que, por ejemplo, a partir de un mundo wi es posible acceder a través de $\Diamond$ también al mismo mundo wi.

Para tener una idea de cómo funcionan las reglas, supongamos que tenemos que evaluar la fórmula $\phi = \Diamond (p \vee q) \wedge \Box \neg p \wedge \Box \neg q$, que ya se encuentra en NNF. Esta fórmula no es satisfacible, y por lo tanto, para cualquier elección no determinista que hacemos, nos encontramos con una contradicción en algún mundo. Supongamos que ϕ es satisfacible.

Entonces, ϕ se evaluará a verdadero en un mundo w0. Esto supone que el mundo w0 también satisfará a las (sub)fórmulas $\Diamond (p \vee q)$, $\Box \neg p$, y $\Box \neg q$. Ahora, la única fórmula activa que se puede expandir es $\Diamond (p \vee q)$; verificamos que, en este momento, no hay mundos que satisfacen la fórmula $(p \vee q)$, y, por lo tanto, lo

creamos; nuestro modelo parcial ahora tiene los mundos w0 y w1, donde w1 satisface sólo (p v q). En este punto, las fórmulas universales se pueden expandir, y en dos pasos verificamos que el mundo w1 también debe de satisfacer ¬p y ¬q. Para desarrollar p vq, hacemos una elección no determinista: el algoritmo elige una situación en la que w1 satisface p, llegando a una contradicción, y, con un paso de backtracking (vuelta atrás), verifica que también la situación en la que w1 satisface q supone una contradicción. Por lo tanto la fórmula no es satisfacible. En la Figura 2.6 vemos desarrollado este ejemplo, puesto en forma de árbol para evidenciar las elecciones no deterministas; cada nodo del árbol es un modelo modal parcial.

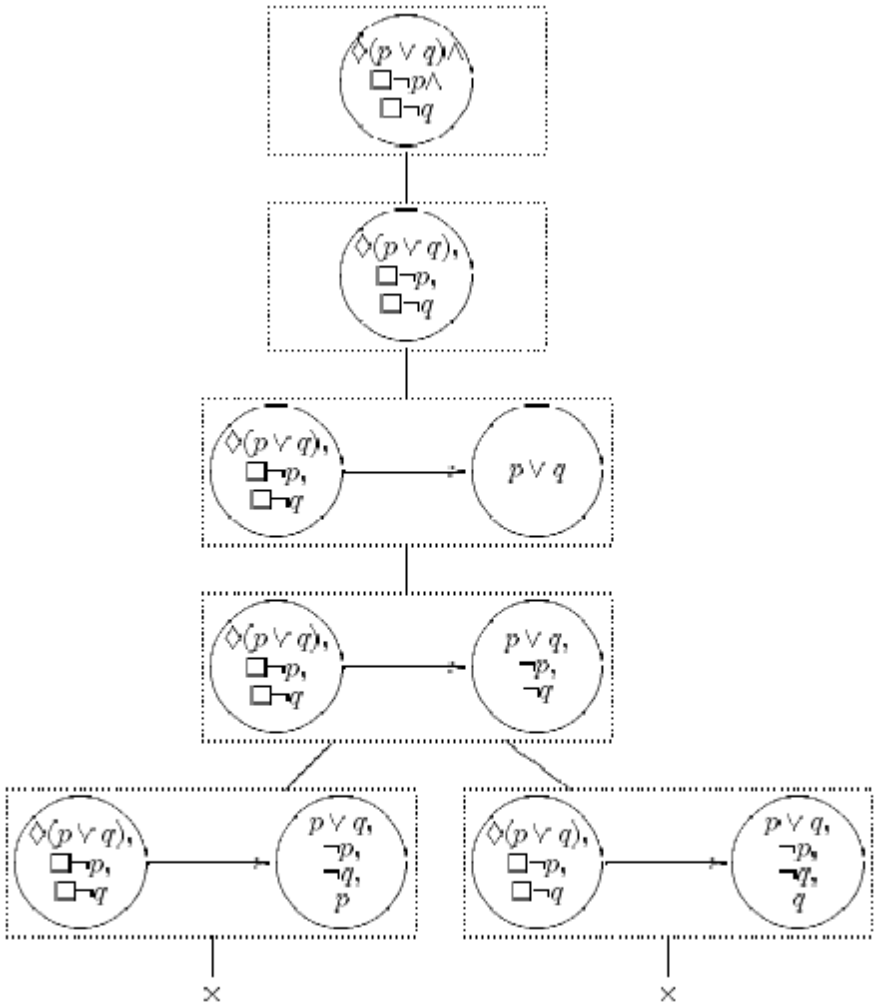


Figura 2.6: Un ejemplo de árbol semántico modal.

La lógica modal en K es decidible, ya que el algoritmo es correcto, completo y termina siempre.

2.5 Aplicaciones: el ejemplo de las lógicas temporales

La lógica modal K tiene numerosas variantes, de las que hemos nombrado sólo algunas. Las lógicas temporales pueden verse como variantes muy peculiares de K (que, además de propiedades sobre la relación, presentan lenguajes con más de un operador modal). Su importancia desde el punto de vista de las aplicaciones es tanta que merecen ser destacadas desde el conjunto de las lógicas modales.

La interpretación de una lógica modal que permite utilizarla para modelar el tiempo es la siguiente: los mundos posibles se sustituyen por instantes de tiempo y la accesibilidad entre mundos por la sucesión temporal. Los operadores modales se convierten en operadores temporales.

2.5.1 Tipos de lógicas temporales

La interpretación de una lógica modal que permite utilizarla para modelar el tiempo es la siguiente: los mundos posibles se sustituyen por instantes de tiempo y la accesibilidad entre mundos por la sucesión temporal. Los operadores modales se convierten en operadores temporales.

Se distingue entre lógicas temporales basadas en puntos (el tiempo se considera compuesto por un conjunto de puntos ordenados) y lógicas temporales basadas en intervalos (pares de puntos). El primer caso permite expresar eventos puntuales, situaciones pasadas y futuras con respecto al instante presente. Destacan en esta variedad la lógica LTL (Linear Time Logic), para la cual se utiliza el método deductivo decidible basado en árboles semánticos; y la lógica CTL, de gran expresividad y alta complejidad computacional, ampliamente aplicada en el campo de control de modelo o Model Checking (metodología de verificación automática de proyectos software).

Las lógicas temporales basadas en intervalos se utilizan cuando no es preciso modelar eventos puntuales sino propiedades con una cierta duración. Destaca entre ellas la Lógica Proposicional de las Relaciones de Allen o HS. En este caso la deducción automática es mucho más compleja y plantea problemas de indecibilidad. No obstante la capacidad expresiva de estas lógicas es muy alta, y resultan de gran interés en algunas áreas, particularmente en aplicaciones de tiempo real.

Las estructuras temporales no lineales permiten modelar diferentes futuros o pasados.

2.5.2 Lógicas temporales basadas en puntos

Una lógica temporal basada en puntos debe, como mínimo, permitir la expresión de situaciones como en el futuro del momento actual o en el pasado del momento actual. Se suele denotar con F la modalidad unaria sobre puntos para el futuro, y con P su correspondiente en el pasado. Representan **instantes** temporales. Considera que el eje temporal está constituido por una secuencia de puntos discretos de forma que los eventos suceden en instantes concretos de dicho eje.

Uno de los lenguajes temporales más sencillo, llamado LTL[F,P] (Linear Time Logic), se obtiene con la siguiente gramática abstracta:

$$\phi = p \mid \neg \phi \mid \phi \wedge \phi \mid F\phi \mid P\phi$$

donde las otras connectivas proposicionales se expresan como en el caso de LP, y las relaciones que expresan las modalidades F y P son la una inversa de la otra.

Un modelo M y un mundo wi satisfacen la fórmula ϕ si y sólo si:

- $M,w_i \Vdash F\phi$ si y sólo si existe un mundo wj en el futuro de wi tal que $M,w_j \Vdash \phi$, es decir, si y sólo si $\exists w_j(w_i < w_j \wedge M,w_j \Vdash \phi)$;
- $M,w_i \Vdash P\phi$ si y sólo si existe un mundo wj en el pasado de wi tal que $M,w_j \Vdash \phi$, es decir, si y sólo si $\exists w_j(w_j < w_i \wedge M,w_j \Vdash \phi)$.

La expresión $\neg F \neg$ se suele denotar con G(cualquier instante futuro), y la expresión $\neg P \neg$ con H(cualquier instante pasado).

- M, w |= Gf syss M, w |= $\neg F \neg f$
- M, w |= Hf syss M, w |= $\neg P \neg f$

En un modelo M, todos los símbolos proposicionales interesados toman valor verdadero o falso en cada punto.

Cuando el tiempo viene modelado como un conjunto discreto de puntos, se puede añadir un operador modal X que se interpreta como en el próximo instante, y/o su correspondiente en el pasado X−1.

Existen dos operadores binarios, llamados since(durante el instante anterior) y until(durante el instante posterior), y denotados con S y U, que resultan fundamentales en lógica temporal porque permiten añadir un alto poder expresivo al lenguaje. La fórmulas de la lógica llamada LTL (Linear Time Logic) se puede obtener con la gramática abstracta:

$$\phi = p \mid \neg \phi \mid \phi \wedge \phi \mid \phi S \phi \mid \phi U \phi$$

y tiene la siguiente semántica:

- $M,w_i \Vdash \phi U \phi$ si y sólo si existe un mundo w_j en el futuro de w_i tal que $M,w_j \Vdash \phi$, y, en todos los mundos w entre w_i y w_j, tenemos que $M,w \Vdash \phi$;
- $M,w_i \Vdash \phi S \phi$ si y sólo si existe un mundo w_j en el pasado de w_i tal que $M,w_j \Vdash \phi$, y, en todos los mundos w entre w_i y w_j, tenemos que $M,w \Vdash \phi$.

LTL es más expresiva que LTL[F,P]

Los operadores binarios representan una dificultad en este sentido, y necesitan reglas muy especiales para ser expandidos.

En IA, merece destacar el trabajo sobre las lógicas temporales basadas en puntos interpretadas sobre árboles donde es posible expresar situaciones en las cuales hay más de un futuro posible. Además de los operadores que permiten, para un cierto futuro, expresar las mismas situaciones que en el caso de LTL, en una lógica interpretada sobre árboles precisamos cuantificadores para los diferentes futuros. Normalmente, se suele denotar con A el operador modal que permite expresar una propiedad de todos los futuros posibles a partir del momento actual, mientras con E el operador que permite elegir un futuro posible. Naturalmente, tenemos que $A\phi = \neg E \neg \phi$.

Un ejemplo es la lógica CTL*, cuyas fórmulas se obtienen a través de la gramática abstracta:

$$\phi = p \mid \neg \phi \mid \phi \wedge \phi \mid \phi S \phi \mid \phi U \phi \mid A\phi \mid E\phi$$

Lógica CTL*, concepción no lineal del tiempo, que se ramifica en un árbol

(más de un futuro posible). A los operadores S y U se añaden:

– $A\phi$: en todo futuro posible se cumple ϕ

– $E\phi$: en algún futuro posible se cumple ϕ

La lógica CTL* es muy expresiva, y su complejidad computacional es muy alta; sin embargo, existen métodos deductivos correctos, completos y que terminan siempre para esta lógica.

El problema del control del modelo tiene una complejidad alta para la lógica CTL*; por eso, ha sido estudiada una restricción llamada CTL que, a pesar de su alto poder expresivo, tiene complejidad polinomial para este problema. La lógica CTL es una restricción muy sencilla de CTL*, en la que simplemente se obliga a los cuantificadores sobre caminos a ser utilizados al mismo tiempo que los operadores sobre futuro que provienen de LTL. La versión que sólo permite expresar propiedades del futuro de CTL tiene la siguiente gramática:

$$\phi = p \mid \neg \phi \mid \phi \wedge \phi \mid A(\phi U \phi) \mid E(\phi U \phi)$$

2.5.3 Lógicas temporales basadas en intervalos

Cuando las propiedades en las que estamos interesados tienen una duración y no es preciso modelarlas como eventos puntuales, entonces podemos utilizar una lógica temporal basada en intervalos. El eje temporal es una secuencia continua de intervalos, de forma que los eventos suceden en alguno de tales segmentos temporales.

Si el modelo del tiempo es lineal, hay trece relaciones diferentes entre dos intervalos (como se ve en la Figura 2.7).

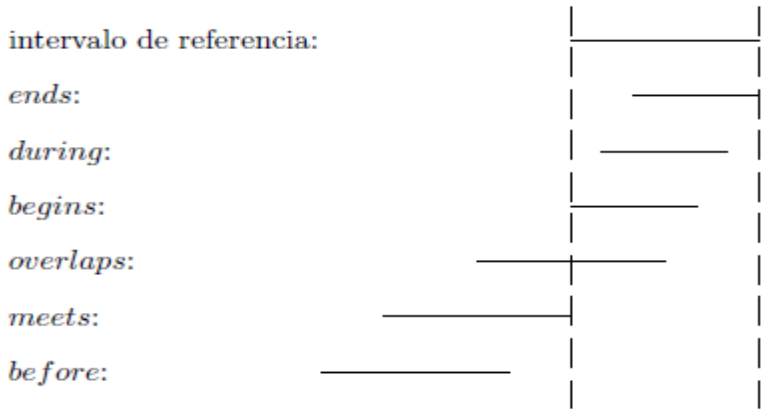


Figura 2.7: Relaciones binarias entre intervalos.

En las lógicas temporales basadas en intervalos el tiempo se considera compuesto por un conjunto de puntos ordenados, permitiendo expresar eventos puntuales y situaciones pasadas y futuras con respecto al instante presente.

Una lógica importante es la Lógica Proposicional de las Relaciones de Allen, conocida también como HS. Su sintaxis abstracta es:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \psi \mid \langle B\rangle\phi \mid \langle E\rangle\phi \mid \langle \overline{B}\rangle\phi \mid \langle \overline{E}\rangle\phi$$

Como se puede ver, son suficientes cuatro modalidades, porque las demás pueden expresarse en términos de éstas.

La semántica para una lógica temporal basada en intervalos viene dada en términos de un modelo M y un mundo que se suele denotar como un par ordenado de puntos. Por lo tanto, tenemos que

- $M, [d_0, d_1] \Vdash \langle B\rangle\phi$ si $M, [d_0, d_2] \Vdash \phi$ para algún d_2 tal que $d_0 \leq d_2 < d_1$;
- $M, [d_0, d_1] \Vdash \langle E\rangle\phi$ si $M, [d_2, d_1] \Vdash \phi$ para algún d_2 tal que $d_0 < d_2 \leq d_1$;
- $M, [d_0, d_1] \Vdash \langle \overline{B}\rangle\phi$ si $M, [d_0, d_2] \Vdash \phi$ para algún d_2 tal que $d_1 < d_2$;
- $M, [d_0, d_1] \Vdash \langle \overline{E}\rangle\phi$ si $M, [d_2, d_1] \Vdash \phi$ para algún d_2 tal que $d_2 < d_0$.

La deducción automática en lógicas temporales basadas en intervalos es mucho más compleja que en el caso de las lógicas temporales basadas en puntos. En la mayoría de los casos, la interpretación en términos de intervalos genera sistemas altamente indecidibles, como en el caso de HS.

T5 – Lógica borrosa.

Capítulo 7 del libro base

7.1 Introducción

Tradicionalmente, han quedado excluidos de toda consideración formal otro tipo de predicados, que como 'alto', no permiten realizar una división satisfactoria de una población de individuos en dos subconjuntos, aquéllos para los cuales 'alto' es cierto, y aquéllos para los que es falso. Y sin embargo, estos predicados, que llamaremos predicados vagos, protagonizan el lenguaje ordinario, lo dotan de flexibilidad y constituyen un elemento fundamental en la capacidad de abstracción del ser humano.

La falta de nitidez de estos predicados supone una fuente de inconsistencias; pensemos sin ir más lejos que un individuo puede ser al mismo tiempo alto, y no alto.

"Fuzzy Sets" plantea la necesidad de utilizar herramientas formales para su análisis, y así surge la Teoría de los Conjuntos Borrosos, como la teoría que ha de proporcionar dichas herramientas, y la Lógica Borrosa, como la ciencia de los principios formales del razonamiento aproximado.

7.2 Conjuntos borrosos

Todo predicado preciso P, aplicado a una colección U, tiene asociado un conjunto preciso que denotaremos de la misma forma P ⊂ U, y que puede describirse mediante una función μP (u), cuyo valor viene determinado por la pertenencia a P de los distintos elementos u ∈ U, y definida como:

$$\mu_P(u) = \begin{cases} 0 & \text{si u} \notin \text{P} \\ 1 & \text{si u} \in \text{P} \end{cases}$$

Para aquellos u para los que μP (u) = 0 el predicado P es falso, y para aquellos u para los que μP (u) = 1 el predicado P es verdadero. En cambio, un predicado vago V , aplicado a la misma colección U, tiene asociado un conjunto borroso que denotaremos de la misma forma V ⊂ U, y que puede describirse mediante una función μV (u), cuya representación de la pertenencia a V de los distintos elementos de U es una cuestión de grado, de modo que:

$$\mu v : U \subset [0, 1]$$

Además de aquellos elementos para los que el predicado V es cierto (μV (u) = 1), y aquéllos para los que V es falso (μV (u) = 0), existe un conjunto de elementos para los que el predicado V es cierto en un determinado grado 0 < μV (u) < 1.

La función de pertenencia encuentra sentido en su capacidad para ordenar los elementos de U según su grado de pertenencia. La elección de la forma de la función de pertenencia estará en función de los requerimientos del problema.

Lo que sí parece evidente es que su forma debe representar adecuadamente el significado que proporcionamos al predicado correspondiente, por lo que cualquiera de las representaciones que vemos en la Figura 7.1 podrían ser ejemplos perfectamente válidos.

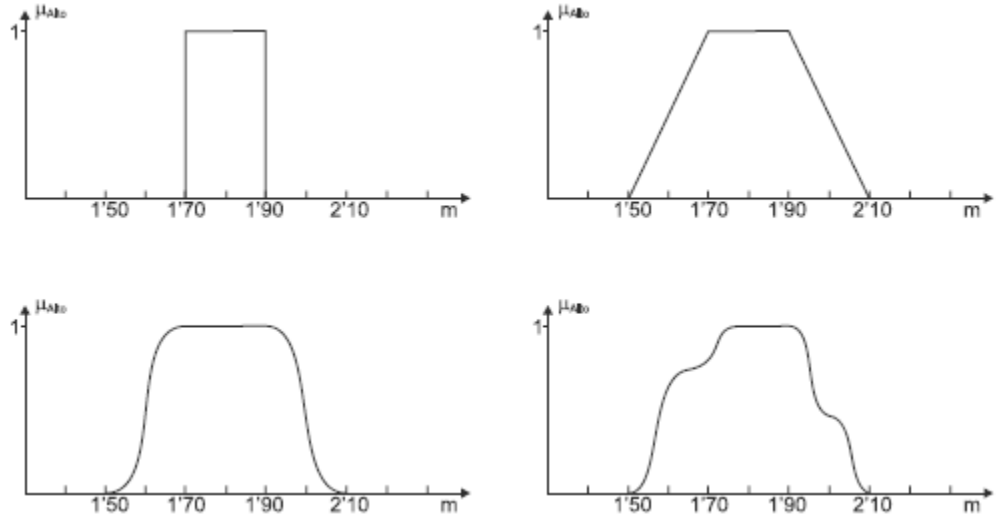


Figura 7.1: Diferentes representaciones de un mismo conjunto borroso, correspondiente al predicado 'alto', donde la primera de ellas coincide con una representación clásica.

Resulta útil definir los siguientes conjuntos asociados a una función de pertenencia μV :

su soporte, definido como {u ∈ U : μV (u) > 0}

su núcleo, definido como {u ∈ U : μV (u) = 1}.

Aquellos elementos que pertenecen al núcleo de V se consideran prototipos de V.

Es habitual caracterizar un conjunto borroso mediante las propiedades de convexidad y normalidad.

Un conjunto V se dice convexo si

$$\forall u,\ u',\ u'' \in U, u' \in [u,\ u''],\ \mu_V(u') \geq \min\{\mu_V(u),\ \mu_V(u'')\}.$$

Un conjunto V se dice normalizado si $\exists u \in U$, tal que $\mu V(u) = 1$.

Una manera interesante de representar un conjunto borroso es a través de una familia de conjuntos precisos anidados, haciendo uso de la noción de α-corte. El α-corte Vα del conjunto borroso V se define como el conjunto $\{u \in U : \mu v(u) \geq \alpha\}$, para cualquier $0 < \alpha \leq 1$ (ver Figura 7.2).

Los miembros de un determinado α-corte son aquellos elementos cuyo grado de pertenencia es mayor o igual a α

El 1-corte de V coincide con su núcleo.

Vemos cómo los miembros de un determinado α-corte son aquellos elementos cuyo grado de pertenencia es mayor o igual que α.

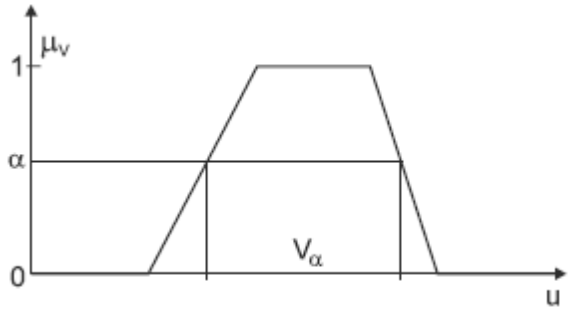


Figura 7.2: Representación del concepto de α-corte.

Si un conjunto borroso V es convexo, entonces cualquiera de sus α-cortes es un intervalo.

7.3 Semántica de los conjuntos borrosos

La idea de conjunto es por definición abstracta: una reunión de elementos que comparten alguna propiedad. Esta propiedad es la que determina la pertenencia de un elemento particular al conjunto. Si el conjunto es borroso, la pertenencia es una cuestión de grado.

Existen algunos tipos reconocibles en la ingeniería, correspondientes a ciertas tareas de interés. Estos tipos definen algunas semánticas:

- Similitud: Sea V un conjunto borroso. Si pensamos en los elementos del núcleo de V como en prototipos de V, entonces μ_V (u) es el grado de proximidad de u al tipo de elementos prototipo de V . Esta semántica que caracteriza a los sistemas de control, donde la diferencia, complementaria de la similitud, entre la situación actual y la deseada (prototipo), determina la realización de acciones de corrección.

- Preferencia: Supongamos que V reúne a un conjunto de elementos entre los cuales existe alguna preferencia. En este caso, μ_V (u) representa el grado de preferencia entre dichos elementos. Esta semántica es propia de tareas de optimización, diseño o planificación.
- Incertidumbre: Dada una variable x que toma valores en U, el predicado "x toma un valor pequeño" se modela mediante un conjunto borroso V ⊂ U, con función de pertenencia $\mu_{V\ =\text{pequeño}}$. La teoría de la posibilidad nos dice que si la única información disponible es que x es pequeño, y su valor preciso es desconocido, entonces $\mu_{V\ =\text{pequeño}}$ se puede utilizar como una medida de la posibilidad de que el valor de x sea u $\in$ U. Esta semántica está ligada a la presencia de incertidumbre en aquellas tareas de razonamiento que llevan a cabo los sistemas basados en conocimiento.

7.4 Teorías de conjuntos borrosos

La lógica borrosa es una extensión de la lógica clásica donde las proposiciones tienen un grado de verdad que se asigna mediante una función de pertenencia que toma valores en el intervalo real [0,1]; Realmente, los límites de edad que definen un alumno joven, un alumno adolescente y un alumno de mediana edad son imprecisos, de modo que podríamos definir, p.e., los correspondientes conjuntos borrosos del siguiente modo:

- Pertenencia de un alumno al grupo de alumnos jóvenes:

μj(edad) = 1 si 18 < edad < 30

μj(edad) = o si 35 < edad o bien 15 > edad

μj(edad) = edad/3 – 5 si 15 ≤ edad ≤ 18

μj(edad) = 7 – edad/5 si 30 ≤ edad ≤ 35

- Pertenencia de un alumno al grupo de alumnos adolescentes:

μa(edad) = 1 si 13 < edad < 16

μa(edad) = o si 18 < edad o bien 11 > edad

μa(edad) = edad/2 – 11/2 si 11 ≤ edad ≤ 13

μa(edad) = 9 – edad/2 si 16 ≤ edad ≤ 18

- Pertenencia de un alumno al grupo de alumnos de mediana edad:

μm(edad) = 1 si 35 < edad < 60

$\mu m(edad) = o$ si $65 < edad$ o bien $30 > edad$

$\mu m(edad) = edad/5 - 6$ si $30 \leq edad \leq 35$

$\mu m(edad) = 13 - edad/5$ si $60 \leq edad \leq 65$

Para cada predicado P, Q, etc., podemos definir una función de pertenencia μP, μQ, etc.

La forma más habitual de interpretar las conectivas:

- El valor de verdad de $P(a)$ es $\mu P(a)$
- El valor de verdad de $\neg\phi$ es $1-x$, siendo x el valor de verdad de ϕ
- El valor de verdad de $\phi \wedge \psi$ es el minimo de los valores de verdad de ϕ y ψ.
- El valor de verdad de $\phi \vee \psi$ es el maximo de los valores de verdad de ϕ y ψ.
- El valor de verdad de $\phi \rightarrow \psi$ es $min(1-a+b, 1)$, siendo a el valor de verdad de ϕ y b el valor de verdad de ψ.

Definición 7.1. Podemos definir una teoría de conjuntos borrosos como la tupla ([0, 1]U, T, S, n), donde [0, 1]U = {μ| μ : U → [0, 1]} es el conjunto de funciones de pertenencia entre U y [0, 1], T es una t-norma, S es una t-conorma, y n es una negación fuerte.

La estructura algebraica originada no es un álgebra de Boole, salvo en el subconjunto {0, 1}U.

Vemos, por tanto, que no existe una única teoría de conjuntos borrosos, sino múltiples teorías, resultado de una elección de aquellas propiedades deseables en los distintos usos del lenguaje. Puesto que los conjuntos borrosos nos permiten formalizar el significado de los predicados vagos que utilizamos en el lenguaje, esto nos puede sugerir que cada hablante incorpora alguna teoría particular de la significación que proviene del aprendizaje, y que obedece a aquellas propiedades o leyes que le han de permitir razonar mediante el lenguaje de un modo consistente.

Por otra parte, el significado de un predicado tiene no sólo un carácter subjetivo, sino también local, en el sentido de que su validez está restringida a un uso particular del lenguaje. Así, la definición que hacemos del predicado "alto" mediante un determinado conjunto borroso puede variar según el contexto en el que hablamos de la altura: una clase de educación infantil, un equipo de baloncesto, etc. Por tanto, asignamos una función de pertenencia a un uso particular de un determinado predicado.

T6 – Sistemas basados en reglas.

Capítulo 3 del libro base

3.1 Introducción

Los Sistemas Basados en Reglas constituyen una de las principales áreas de la inteligencia artificial simbólica. Su surgimiento y la enorme difusión de sus aplicaciones en los años setenta, bajo el nombre de Sistemas Expertos y posteriormente Sistemas Basados en Conocimiento, suponen uno de los hitos más importantes de la inteligencia artificial.
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.

El uso de las reglas como un esquema de representación del conocimiento.

Los Sistemas Basados en Reglas (SBR) permiten capturar la experiencia humana en la resolución de problemas, con el fin de alcanzar decisiones consistentes y repetibles.

En los SBR, las estructuras de representación son declarativas, por lo que existe una separación explícita entre el conocimiento del dominio y los mecanismos de deducción utilizados.

El conocimiento del dominio puede ser factual o heurístico.

- Factual: el conocimiento del dominio aceptado, y generalmente consensuado, por todos los expertos en un campo de aplicación específico.
- Heurístico: es mucho menos riguroso y está basado en la experiencia propia de cada experto o grupo de expertos.

Por otra parte, los mecanismos de deducción utilizados o motores de inferencia controlan las etapas que se deben llevar a cabo para resolver un problema y son los que aplican el conocimiento para crear una línea de razonamiento.

Los SBR son una aplicación de los sistemas de deducción en lógica proposicional o de primer orden, restringidos a cláusulas de Horn en primera instancia.

Utilizan las reglas de inferencia de la lógica (razonamiento deductivo) para llegar a obtener conclusiones lógicas, interpretando dichas reglas de la siguiente manera:

a) como reglas de la forma condición-acción en un control con encadenamiento hacia delante

b) como conjuntos de implicaciones lógicas a partir de las cuales se obtienen las deducciones, en un control con encadenamiento hacia atrás.

Se realiza una separación del conocimiento (hechos y reglas) y los mecanismos de inferencia (encadenamiento)

Un lenguaje que utiliza los sistemas SBR es el lenguaje declarativo Prolog

3.2 Componentes básicos de los SBR

Un SBR es un sistema que tiene:

- Una base de conocimiento (BC), contiene las reglas utilizadas para representar el conocimiento disponible de un determinado dominio.
- Motor de inferencias (MI), coordina la información procedente del resto de módulos mediante algoritmos determinando qué regla aplicar.
- Una base de hechos (BH) o memoria de trabajo, contiene los hechos o afirmaciones conocidos inicialmente y aquéllos que se van creando en el proceso de inferencia.
- Interfaz de usuario, a través del cual se puede solicitar u ofrecer información al usuario.

Para que un SBR llegue a ser realmente útil ha de estar dotado de facilidades de entrada/salida sofisticadas, conocidas como interfaces de usuario.

Figura 3.1: Componentes básicos de un SBR.

En la figura 3-1 aparece un esquema de la composición de un sistema basado en reglas, donde el sentido de las flechas da idea del flujo de información que se produce entre los diferentes módulos. Obsérvese que normalmente no existirá este flujo desde el motor de inferencia a la base de conocimientos, a no ser que el sistema tenga capacidad de aprendizaje.

3.2.1 Base de Hechos

La base de hechos (BH) o memoria de trabajo contiene toda la información actual del problema o tarea a resolver, es decir, los datos y hechos establecidos hasta el momento en la tarea, las metas a alcanzar y las hipótesis avanzadas en el curso de una solución. Este conjunto de hechos puede existir como parte de las condiciones iniciales del sistema, se pueden añadir durante el proceso de inferencias del sistema, el usuario puede introducirlos durante el uso del sistema o pueden proceder de sistemas externos, tales como sensores o bases de datos.

- Datos y hechos establecidos hasta el momento en la tarea (hacia delante)
- Metas a alcanzar e hipótesis avanzadas en el curso de una solución (hacia atrás)

En la BH también es donde se guardan todos los cambios de información que se producen en el sistema durante cada proceso de inferencias obtenidos al aplicar las reglas. El sistema no se guarda más información que la que existe en la BH, que representa el estado actual del problema en curso de solución.

3.2.2 Base de Conocimiento

Es el conocimiento declarativo de experto en forma de reglas que operan sobre la BH.

Una regla consta de dos partes:

- izquierda, denominada condición o antecedente,
- derecha, llamada consecuente o de acción.

Así, definiremos una regla como un par condición-acción. El antecedente contiene una lista de cláusulas a verificar y el consecuente una lista de acciones a ejecutar.

Condición→ Acción

Las reglas operan sobre el espacio de trabajo de la BH. La condición expresa algún tipo de test sobre el contenido de la BH, que se puede verificar o no. Si se verifica el test de una regla, se puede ejecutar su parte de acción. La ejecución de una acción puede cambiar el contenido de la BH.

La BH contiene:

- Encadenamiento hacia delante: el conjunto de hechos que han podido ser deducidos por el sistema
- Encadenamiento es hacia atrás: un conjunto de metas que deben alcanzarse e hipótesis avanzadas en el curso de una solución.

Una diferencia importante entre la BH y la BC es que la primera contiene información puntual sobre la tarea a realizar, como memoria de trabajo que es, mientras que la BC almacena segmentos de conocimiento relacional entre datos y conceptos.

Una diferencia importante entre los SBR y la programación imperativa (basada en procedimientos) es la forma de seleccionar una condición entre varias que se satisfacen simultáneamente, en un SBR el motor de inferencias tiene en cuenta diferentes factores para tomar la decisión y ésta se realiza durante la ejecución. El proceso en el que se toma la decisión se llama resolución de Conflictos.

3.2.3 Motor de inferencias

El motor de inferencias (MI), (estrategia de control o el intérprete de reglas) es el mecanismo (algoritmo) que sirve para examinar la BH y decidir qué reglas se deben disparar.

La condición de finalización indica el hecho meta que tiene que alcanzarse. Esta meta se alcanzará cuando esté contenida como hecho en la BH.

La búsqueda del conjunto de reglas que se pueden aplicar a la BH se denomina equiparación y consiste en seleccionar las reglas cuyas condiciones o acciones sean compatibles con los datos almacenados en ese ciclo en la BH.

El conjunto de reglas que se obtiene durante el proceso de equiparación se denomina **conjunto conflicto**. De todo este conjunto selecciona una regla, fase que se conoce como resolución del conjunto conflicto.

La selección de las reglas puede considerarse como un proceso de búsqueda, y ésta puede realizarse sin ningún conocimiento (selección desinformada) o con algún tipo de conocimiento acerca del problema (selección informada).

El coste computacional de un SBR es el resultado de la suma del coste de control y del coste de aplicación de las reglas.

Selección completamente desinformada tiene un pequeño coste de control, coste de aplicación de las reglas elevado, ya que se requiere ensayar un gran número de reglas para encontrar una solución. Selección muy informada, implica un alto coste, tanto en espacio de almacenamiento como en tiempo computacional. Sin embargo, el coste de aplicación de la regla es mínimo, ya que la selección informada guía a una solución de forma directa.

Generalmente, el diseño de un SBR eficiente busca el equilibrio entre ambos costes.

El mecanismo de inferencias es un algoritmo independiente del conocimiento almacenado en la BC. Por tanto, un SBR desarrollado para una aplicación se puede usar nuevamente en otra aplicación sin más que sustituir la BC.

3.3 Inferencia

El proceso de inferencia en un SBR consiste en establecer la verdad de determinadas conclusiones a partir de la información que se tiene en la base de afirmaciones y la base de conocimientos. Es el motor de inferencia el encargado de llevar a cabo dicho proceso, que por lo general puede realizarse de las dos formas siguientes:

- El encadenamiento hacia delante, se basa en ejecutar aquellas reglas cuyo antecedente sea cierto a partir de la información que hay en el sistema. Por otra parte, la introducción de nueva información a partir del consecuente de cada regla ejecutada, permitirá la ejecución de otras reglas. Cada vez que se aplica una regla se almacenan en la BH los resultados de las acciones del consecuente de dicha regla.

- El encadenamiento hacia atrás, se basa en seleccionar aquellas reglas cuyo consecuente permita demostrar cierta condición, si ésta no puede ser demostrada a partir de la base de afirmaciones. A su vez, las condiciones o cláusulas que figuren en los antecedentes de las reglas seleccionadas pasarán a convertirse en nuevos subobjetivos a demostrar, entrándose en un proceso recursivo que finaliza cuando se encuentra la información buscada en la base de afirmaciones o cuando se le solicita dicha información al usuario.

Si el número de reglas no es muy grande, uno puede obtener un grafo dirigido con las reglas (llamado red de inferencia), en el cual;

- las cláusulas del antecedente son las entradas a los nodos
- las acciones del consecuente son las salidas de los nodos que, a su vez, pueden ser las cláusulas del antecedente de otros nodos.
- el antecedente que no es consecuente de ninguna otra regla de la red, se convierte en los hechos de partida.
- el consecuente, que no es antecedente de ninguna otra regla, se convierte en la meta a alcanzar por el sistema.

La Figura 3.2 muestra un ejemplo de red de inferencia con un conjunto de reglas y hechos simbólicos. En ella se representan los hechos o elementos de la BH por medio de las letras que van desde la a hasta la q; aunque, en general, los antecedentes suelen ser más complejos. En la figura se muestra una red con las siguientes cinco reglas, representadas gráficamente mediante puertas de implicación:

Regla 1: a&b → p

Regla 2: b&c → m

Regla 3: d&e&f → n

Regla 4: n&g → m

Regla 5: h&m → q

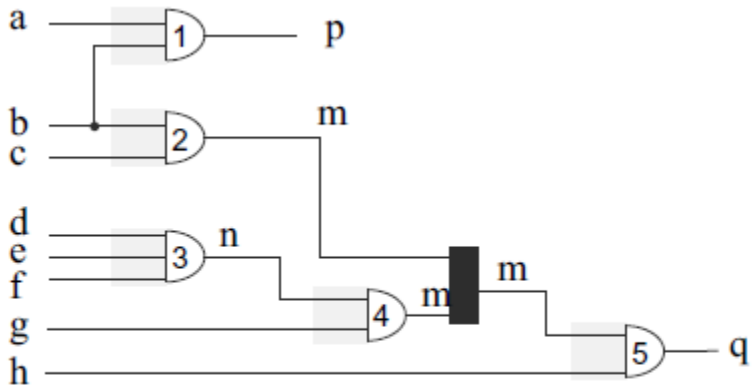


Figura 3.2: Ejemplo de una red de inferencia.

Las reglas 2 y 4 tienen el mismo consecuente, m, que se corresponde con el caso en que se llega a una misma conclusión, siguiendo dos líneas de razonamiento independientes.

Esto se representa por medio de un rectángulo negro. A partir de esta red de inferencia, hay dos posibles formas de razonamiento, que consisten en:

- Buscar el conjunto de objetivos que se verifican a partir de un conjunto de entradas. Es lo que se conoce como encadenamiento hacia delante. En este tipo de razonamiento, la inferencia progresa en la red de izquierda a derecha.
- Determinar si se verifica un objetivo dado con ciertas entradas. Es lo que se conoce como encadenamiento hacia atrás. Aquí, la inferencia progresa en la red de derecha a izquierda.

3.3.1 Encadenamiento hacia delante

La particularidad de este algoritmo es la etapa de equiparación, en donde se seleccionan las reglas cuyos antecedentes se verifican, teniendo en cuenta el contenido de la BH.

La ejecución repetida de una regla no modifica en nada la BH (en cada repetición se genera un hecho que ya está contenido en la BH desde la primera vez que se ha disparado la regla). Con esta suposición evitamos que una regla esté continuamente ejecutándose sin producir modificaciones en la BH (esta aproximación práctica se conoce como principio de refracción).

3.3.1.2 Ciclo de reconocimiento-acción

El proceso iterativo de aplicación del encadenamiento hacia delante se denomina ciclo de reconocimiento-acción (Figura 3.11). Durante la fase de equiparación el mecanismo de inferencias examina la BH y selecciona el conjunto de reglas cuya parte antecedente es compatible con la BH. Esta operación es relativamente

sencilla cuando las reglas no contienen variables, pero se vuelve más compleja cuando se permite la utilización de variables y cuantificadores en las reglas.

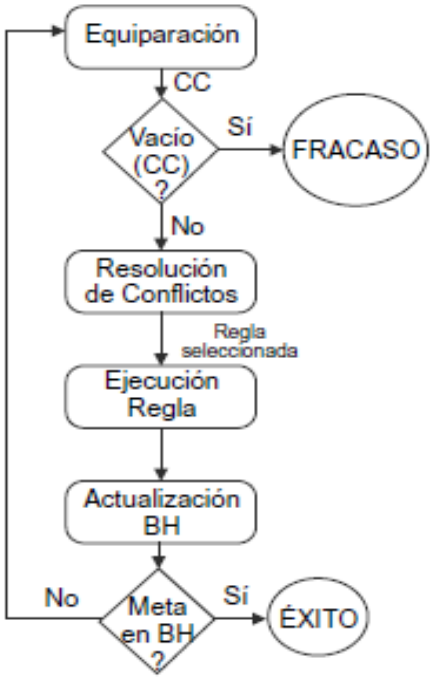


Figura 3.11: Ciclo de reconocimiento-acción (CC es el conjunto conflicto).

Ciclo de reconocimiento-acción:

- **Paso 1**: equiparación, determinar, a partir de la BH, qué reglas son aplicables para las cuales la equiparación ha tenido éxito son válidas y forman el conjunto conflicto.
- **Paso 2**: resolución de conflictos, selecciona a partir del conjunto conflicto obtenido de todas las instancias la regla que se disparará.
- **Paso 3**: una vez elegida la regla, el mecanismo de inferencias aplica la regla sobre la BH, es decir, activa la parte de acción de la regla, la cual añade o elimina hechos de la BH (acciones indicadas en el consecuente o cabeza de la regla).
- **Paso 4**: actualización de la BH (la regla puede añadir o quitar hechos)

El ciclo de reconocimiento-acción se repite hasta que el hecho que se ha fijado como meta se añade a la BH

Los inconvenientes de este método de encadenamiento proceden de la falta de focalización hacia la meta buscada. Podemos citar los siguientes:

1. La fase de resolución de conflictos es crítica.

2. El disparo de las reglas elegidas, a pesar de que las conclusiones a las que llegan no tengan interés, favorece la explosión combinatoria.

3. Se necesita cargar la BH con todas las afirmaciones que se poseen, sin saber si los elementos que se introducen serán útiles o no, o si faltan datos.

3.3.2 Encadenamiento hacia atrás

En un modo de razonamiento con encadenamiento hacia atrás, se especifica una meta objetivo y se trata de determinar si ese objetivo se verifica o no, teniendo en cuenta el contenido de la BH, se investigan los consecuentes de todas las reglas, con el fin de encontrar aquellas cuyos consecuentes contengan el objetivo a verificar. El subconjunto de reglas resultantes se examina para descubrir alguna que verifique todos sus antecedentes, teniendo en cuenta los contenidos de la BH. Si existe tal regla, entonces se verifica el objetivo; en caso contrario, los antecedentes que no se pudieron verificar pasan a ser nuevos objetivos a verificar, en un control recursivo. Este tipo de búsqueda se corresponde con una búsqueda en profundidad

3.3.2.2 Ciclo de encadenamiento hacia atrás

En el encadenamiento hacia atrás el ciclo de inferencias se convierte en:

1. *Etapa de equiparación*: búsqueda de las reglas cuya conclusión se corresponda con la meta M en curso, creando así el conjunto conflicto.

2. *Etapa de resolución de conflictos*: selección de una regla entre las dadas por el conjunto conflicto.

3. *Etapa de acción*: con reemplazamiento de la meta M por la conjunción de las condiciones del antecendente de la regla seleccionada.

El ciclo finaliza cuando la meta inicial se ha reducido a submetas elementales verificadas en la BH o cuando no se ha podido encontrar ninguna regla que llegue a alguna submeta válida.

El problema de este modo de razonamiento es entrar en bucles infinitos, en los cuales una meta es, a su vez, una submeta en su propio árbol de búsqueda.

El encadenamiento hacia atrás tiene una serie de ventajas con respecto al encadenamiento hacia delante:

1. El sistema sólo planteará cuestiones al usuario cuando tiene necesidad de ello, es decir, después de haber explorado todas las posibilidades de la BC.

2. Limita el número de equiparaciones de antecedentes de las reglas, ya que sólo se equiparan aquellas cuyo consecuente se requiere verificar.

3. Disminuye la dimensión del árbol de búsqueda, como consecuencia de la limitación en las equiparaciones.

Ejemplo de encadenamientos

Ejercicio 2. Segunda Semana. (Valoración: 3 puntos)

La base de conocimiento de un sistema basado en reglas contiene las siguientes reglas:

R1: Si h_2 entonces h_4	R5: Si h_3 y h_{11} entonces h_2	R9: Si h_1 entonces h_{13}
R2: Si h_1 entonces h_3	R6: Si h_{10} y h_{16} entonces h_{13}	R10: Si h_4 entonces h_7
R3: Si h_3 y h_8 entonces h_5	R7: Si h_{10} entonces h_{14}	R11: Si h_1 y h_8 entonces h_{10}
R4: Si h_3 y h_{10} entonces h_5	R8: Si h_2 entonces h_{15}	R12: Si h_{18} entonces h_8

donde hi representa un hecho. Cada hecho se almacena en la Base de Afirmaciones de la siguiente forma: $h_i(t)$, que significa que hi fue inferido en el ciclo t. Inicialmente, BA_0= $\{h_1, h_9, h_{11}\}$.

a) Suponiendo que nuestro objetivo es obtener h_{13}, indicar detalladamente cómo evoluciona la ejecución del método de encadenamiento hacia adelante, a partir de BA_0. ¿Es posible obtener h_{13}? ¿Hasta qué ciclo llegaríamos? Como mecanismo de control consideramos el criterio de refractariedad (no se puede ejecutar en el presente ciclo una regla que fue ejecutada en el ciclo anterior) y tienen preferencia las reglas de menor subíndice.

b) Considerando la misma base de conocimiento y la base de afirmaciones BA= $\{h_1, h_{11}\}$, averiguar, aplicando un método de encadenamiento hacia atrás, si en algún momento podríamos llegar a tener h_7.

a) Teniendo en cuenta la base de afirmaciones inicial BA_0= $\{h_1, h_9, h_{11}\}$:

Ciclo 1: Podríamos aplicar las siguientes reglas (conjunto conflicto): {R2,R9}. Resolución de conflictos: Como es el primer ciclo sólo debemos tener en cuenta la preferencia de las reglas con menor subíndice. Aplicamos por tanto, la regla R2. La base de afirmaciones sería: BA_1= $\{h_1(0), h_9(0), h_{11}(0), h_3(1)\}$

Ciclo 2: Podríamos aplicar las siguientes reglas: {R2,R9,R5}.Tras refractariedad el conjunto conflicto es {R5,R9}. Por el criterio de preferencia de las reglas con menor subíndice, aplicamos la regla R5. La base de afirmaciones sería: BA_2= $\{h_1(0), h_9(0), h_{11}(0), h_3(1), h_2(2)\}$.

Ciclo 3: Podríamos aplicar las siguientes reglas: {R1,R2,R9,R5,R8}. Tras refractariedad el conjunto conflicto es {R1,R2,R9,R8}. Por el criterio de preferencia de las reglas con menor subíndice, aplicamos la regla R1. La base de afirmaciones sería: BA_2= $\{h_1(0), h_9(0), h_{11}(0), h_3(1), h_2(2), h_4(3)\}$

Ciclo 4: Podríamos aplicar las siguientes reglas: {R1,R2,R5,R8,R9,R10}. Tras refractariedad el conjunto conflicto es {R2,R5,R8,R9,R10}. Por el criterio de preferencia de las reglas con menor subíndice, aplicamos la regla R2. La base de afirmaciones sería: BA_2= $\{h_1(0), h_9(0), h_{11}(0), h_3(4), h_2(2), h_4(3)\}$

Podemos comprobar que seguiríamos aplicando sucesivamente las reglas R1 y R2 con lo que no podríamos llegar nunca a aplicar R8 y obtener por tanto el objetivo h_{15}.

b) Suponemos ahora que nuestro objetivo es h_7. Aplicando encadenamiento hacia atrás y teniendo en cuenta la regla R10, obtendríamos como subobjetivo el hechos h_4. En la base de afirmaciones no se encuentra h_4 así que, teniendo en cuenta la regla R1 obtenemos como nuevo subojetivo el hecho h_2. Teniendo en cuenta la regla R5 obtenemos como nuevos subojetivos los hechos h_3 y h_{11}. El hecho h_{11} lo tenemos en la base de afirmaciones dada, así que nuestro objetivo ahora es ver si podemos obtener h_3. Volviendo a aplicar encadenamiento hacia atrás, y teniendo en cuenta la regla R2, vemos que podemos obtener h_3 ya que tenemos $h_1$en la base de afirmaciones dada. Por tanto, sí podemos llegar a obtener h_7 con la base de afirmaciones inicial y aplicando encadenamiento hacia atrás.

3.3.3 Reversibilidad

Dado un problema concreto, no es lo mismo usar un modo de razonamiento con encadenamiento hacia delante o hacia atrás, ya que puede ser significativamente más eficiente buscar en una dirección que en otra. El uso de un tipo u otro depende del problema concreto.

Sólo podemos indicar un conjunto de estrategias generales, basadas en la forma de la red de inferencia y en el grado de conocimiento que el usuario del sistema tiene de los datos y de los objetivos del problema.

1. Si la red tiene, en promedio, un número elevado de reglas con muchas condiciones en el antecedente, se debe elegir un encadenamiento hacia delante; mientras que si hay muchas reglas cuyo consecuente forma parte del antecedente de otras muchas, entonces se debe elegir un encadenamiento hacia atrás.

2. En muchos problemas conocemos perfectamente la naturaleza de los datos de entrada, mientras que no está tan claro cuáles son los objetivos a alcanzar, es preferible un encadenamiento hacia delante. Por otra parte, si los objetivos y subobjetivos están bien definidos, pero no las estructuras de los datos, es preferible un encadenamiento hacia atrás.

3. Un encadenamiento hacia delante es apropiado para problemas que requieren tareas reactivas como la monitorización; y un tipo de encadenamiento hacia atrás es aconsejable para problemas que se basan en tareas analíticas como la clasificación.

4. Los SBR que deban justificar su razonamiento, deben proceder en la dirección que se adapta más a la manera de pensar del usuario del sistema.

Además de los mecanismos de razonamiento hacia delante y hacia atrás, existe otra posibilidad que consiste en razonar siguiendo ambos modos de razonamiento a la vez por la misma de red de inferencia, hasta que los dos caminos de búsqueda se encuentran. Este modo de razonamiento mixto se llama búsqueda bidireccional o reversibilidad. Para aplicar este tipo de razonamiento, se requiere:

1. Incorporar a la BH global tanto las descripciones de los datos iniciales como de las metas a alcanzar.

2. Definir cuidadosamente las reglas, distinguiendo entre las reglas aplicables en el encadenamiento hacia delante (reglas tipo A) y en el encadenamiento hacia atrás (reglas tipo B).

3. Diseñar la condición de terminación de la inferencia para que considere tanto la parte de descripción de los datos inferidos como de las metas a alcanzar.

4. Modelar el mecanismo de control para que decida en cada etapa si aplica una regla de tipo A o una regla de tipo B.

Es efectiva para búsquedas completamente desinformadas, esta estrategia basada en divide y vencerás es efectiva.

Puede que no todas las reglas sean aplicables en ambos sentidos, algunas sólo serán aplicables hacia delante y otras hacia atrás.

Resaltar que uno de los problemas que tiene esta búsqueda es que el encadenamiento hacia delante pueda explorar una parte de la red de inferencia distinta de la parte investigada por el encadenamiento hacia atrás.

Ejemplo Junio 2013:

e) Imagine que la implementación de un sistema recomendador educativo o una parte de ella se realiza mediante un Sistema Basado en Reglas. Describa brevemente el sistema en cuestión. Explique qué tipo o tipos de encadenamiento de reglas podrían resultar de utilidad para el funcionamiento del sistema, ilustrando su argumentación con ejemplos. Explique brevemente, a nivel teórico, la diferencia entre los distintos tipos de encadenamiento.

e) La Base de Hechos contendría en cada momento las condiciones del contexto de aprendizaje y los datos del perfil del estudiante. El encadenamiento hacia adelante podría servir para identificar las recomendaciones oportunas en un cierto contexto de aprendizaje y para un alumno concreto. El encadenamiento hacia atrás podría permitir identificar al conjunto de alumnos a los que una recomendación dada es aplicable. Mediante el encadenamiento mixto se podría responder con eficiencia a la pregunta de si una recomendación concreta es aplicable a un alumno concreto.

Definición teórica de los tipos de encadenamiento:

Encadenamiento hacia adelante. Modo de inferencia de un Sistema Basado en Reglas que partiendo de una colección de hechos o afirmaciones de partida aplica las reglas de la Base de Conocimiento repetidas veces hasta que no se generan nuevos hechos.

Encadenamiento hacia atrás. Modo de inferencia de un Sistema Basado en Reglas que parte de un conjunto de hipótesis e intenta verificar estas hipótesis usando datos de la Base de Hechos o datos externos (obtenidos, por ejemplo, del usuario).

Encadenamiento mixto. Modo de inferencia de un Sistema Basado en Reglas que consiste en realizar encadenamiento hacia atrás y hacia adelante simultáneamente por la misma red de inferencia, hasta que los dos caminos de búsqueda se encuentran.

3.4 Técnicas de equiparación

La equiparación del antecedente de las reglas con el estado de la BH no siempre es obvia ya que el antecedente puede no describir situaciones particulares sino generales.

Además, en cada ciclo de inferencias es necesario examinar todas las reglas de la BC en cada ciclo de inferencias. Este proceso es poco eficiente, si hay que recorrer toda la BC y ésta contiene numerosas reglas. Se puede simplificar mediante:

1. Técnicas de indexación: Consisten en añadir a las reglas nuevas condiciones relacionadas con el punto de inferencia. Este recurso permite dividir el problema en varias etapas y agrupar las reglas en función de la etapa en la que se aplican. Lamentablemente, esta forma de indexar las reglas sólo se puede aplicar si las condiciones de las reglas se equiparan exactamente con la BH. Además, con esta aproximación se pierde generalidad en la declaración de las reglas.

2. Técnicas que aceleran el proceso de equiparación, sin necesidad de examinar toda la BC. El método más conocido es el algoritmo RETE.

3.4.1 Equiparación con variables

Generalmente, el antecedente de las reglas describe situaciones generales que requieren ser expresadas mediante variables.

3.4.1.1 Sintaxis de reglas con variables

La sintaxis propuesta basada en CLIPS es la siguiente:

1. Cada elemento de condición presente en el antecedente de una regla debe ir encerrado entre paréntesis y empezar por una constante.

2. Todos los elementos de condición del antecedente de una regla deben ir unidos por el operador lógico Y (representado por el símbolo $\wedge$).

3. Los elementos de condición están formados por átomos, donde un átomo puede ser una constante o una variable. Para distinguir las variables de las constantes, es usual señalar las primeras con un prefijo de interrogación.

4. Los elementos de condición pueden expresar algún tipo de prueba sobre las variables.

5. Las acciones del consecuente también pueden contener variables.

6. Las acciones del consecuente pueden incluir dos predicados: Añadir (para agregar nuevos hechos a la BH) y Borrar (para eliminar hechos existentes de la BH).

3.4.1.2 Equiparación de reglas utilizando variables con los contenidos de la BH

La equiparación de reglas usando variables es un caso particular del problema de la unificación de términos en la resolución de primer orden.

Durante la equiparación, pueden darse las siguientes situaciones:

1. Una variable aparece una sola vez en la parte de condición de una regla. En este caso, la variable se equipara con cualquier valor que ocupe la misma posición en un elemento de la BH.

2. Una variable aparece más de una vez en la parte de condición de una regla. En este caso, la variable debe equipararse siempre con el mismo valor en todas las ocurrencias de la regla.

3. Varias variables aparecen en la parte de condición de una regla. Se pueden equiparar variables distintas con un mismo valor.

4. Una o varias variables aparecen en una o varias condiciones negadas de una regla. La equiparación se da si se cumplen las dos siguientes restricciones:

(a) Se equiparan todas las condiciones no negadas para algún o algunos elementos de la BH.

(b) No existe ningún elemento en la BH que pueda equiparar las condiciones negadas.

La aplicación de estas dos restricciones se conoce como Hipótesis del Mundo Cerrado: Todo hecho que no esté en la BH, durante la equiparación de una regla, se considera falso.

Se llama instanciación (conjunto conflicto) al par formado por una regla y los elementos de la BH que equiparan dicha regla.

3.4.2 El algoritmo RETE

El algoritmo de equiparación RETE evita examinar todas las reglas de la BC con todos los datos de la BH. Para hacer esto, explota dos propiedades comunes a todos los SBR:

1. Las condiciones de las reglas contienen muchos patrones similares, aunque no idénticos. RETE agrupa estos patrones comunes con el fin de evitar la equiparación de patrones idénticos repetidas veces. Genera un grafo llamado red RETE, que es una red de clasificación estructurada en forma de árbol. Los nodos de la red indican las operaciones que se realizan en la equiparación, y los arcos indican el camino de nodo a nodo por el que deben de fluir los datos. Se eliminan las operaciones redundantes construyendo un único nodo por cada operación distinta a realizar y uniendo dicho nodo con todos los demás nodos que requieren su resultado.

2. En cada ciclo los cambios en la BH suelen ser muy pocos, aunque los contenidos sean muchos. Es muy ineficiente equiparar, en cada ciclo, toda la BH, RETE guarda el resultado de la equiparación durante un ciclo, para que pueda ser usado nuevamente en el ciclo siguiente. De esta manera, el coste computacional depende de la velocidad de cambio de la BH, que suele ser baja, y no de su tamaño, que suele ser grande.

RETE opera de la siguiente manera:

(a) Guarda las instanciaciones, resultado de las equiparaciones, de ciclo a ciclo (tanto de las equiparaciones completas de las condiciones de las reglas como de las parciales).

(b) Actualiza esta información con los cambios que se producen en la BH después de la ejecución de cada ciclo.

Para cada nuevo dato en la BH, obtiene las nuevas instanciaciones que se generan y las añade al conjunto de las instanciaciones resultado de previos ciclos.

Para cada dato que se borra de la BH, RETE elimina de dicho conjunto las instanciaciones a las que dio lugar en ciclos previos.

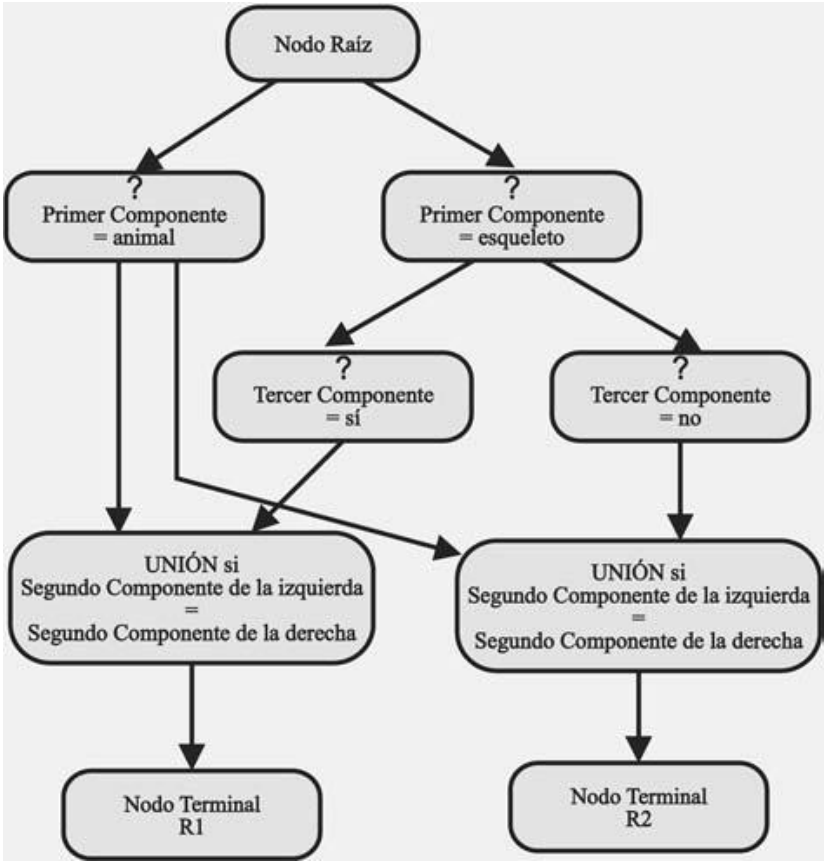


Figura 3.16: Red RETE para las reglas 1 y 2 del ejemplo de clasificación de animales.

Para aplicar el algoritmo RETE, es necesario modificar el ciclo de reconocimiento-acción (véase Figura 3.17).

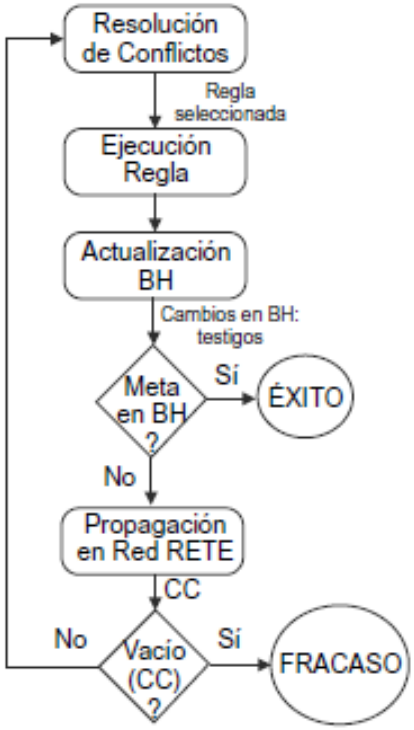


Figura 3.17: Ciclo de reconocimiento-acción del algoritmo RETE (CC es el conjunto conflicto).

A partir del conjunto conflicto, la etapa de resolución selecciona la regla a aplicar. La etapa de acción ejecuta las acciones especificadas en la parte derecha de la regla seleccionada. Estas acciones suelen producir cambios en la BH. Estos cambios se notifican a la red RETE, para que ésta los equipare con las condiciones de las reglas y genere los cambios oportunos en el conjunto conflicto.

La red RETE puede considerarse como una función que traslada los cambios de la BH a cambios en el conjunto conflicto. Cada vez que se producen cambios en la BH se crean estructuras, llamadas señales o testigos, que describen dichos cambios.

Un testigo es un par de elementos, donde el primero de ellos puede tomar un valor en el conjunto {+, −}, y el segundo es una lista de elementos que representan los cambios de la BH. El signo + indica que los elementos han sido añadidos a la BH; mientras que el signo − que han sido eliminados de la BH. Todos los nodos de la red, excepto el nodo raíz y el terminal, tienen asociada una memoria, donde se almacenan los testigos.

3.5 Técnicas de resolución de conflictos

Se trata de seleccionar, a partir del conjunto conflicto, la instanciación a aplicar.

Las principales técnicas de resolución de conflictos que más se han usado son las siguientes:

1. Seleccionar la primera regla que se equipara con los contenidos de la BH. En este caso el coste de control es pequeño, ya que no implica cálculos costosos. Sin embargo, conlleva un coste de aplicación de las reglas elevado, ya que es necesario ensayar un gran número de reglas para encontrar la solución buscada.

A veces, esta técnica se combina con otras que consisten en ordenar la BC, tales como:

(a) Ordenación de las reglas en la BC, colocando en primer lugar las que se deben examinar antes.

(b) Ordenación de las condiciones dentro de cada regla. Se utiliza en los SBR con encadenamiento hacia atrás. Durante la ejecución del sistema, si no se verifica una condición, ya no se necesita investigar el resto de las condiciones que aparecen a continuación dentro de la misma regla. Para que este método sea eficaz, durante la definición de las reglas, hemos de ser cuidadosos a la hora de colocar primero las condiciones que tienen menos probabilidades de verificarse.

2. Seleccionar la regla de prioridad más alta. La prioridad se definirá durante la construcción del SBR.

3. Seleccionar las instanciaciones más específicas. En este caso, se supone que las instancias más específicas se adaptan mejor a la situación planteada.

4. Seleccionar arbitrariamente una regla, dentro de un conjunto de reglas con igual posibilidad de ser efectivas.

5. Seleccionar las instanciaciones que tienen elementos más recientemente añadidos a la BH. Para ello, la BH debe poseer un contador que indique el ciclo que se está ejecutando en cada momento.

6. Seleccionar una instanciación que no se haya ejecutado previamente (refracción). Con esta técnica se evita la ejecución de la misma instanciación indefinidamente.

Todos los métodos comentados no se suelen usar aisladamente, sino combinados.

Para ello, se establece un orden de prioridad en los métodos: se aplica el primer método escogido al conjunto conflicto, y sobre el subconjunto de instanciaciones obtenido se aplica el segundo método, y así sucesivamente.

Ejemplo de resolución de conflictos.

Ejercicio 3. Segunda Semana 2007. (Valoración: 3 puntos)

Enumere las soluciones que conozca para aplicar control de razonamiento en reglas, también conocido como resolucion de conflictos. Explique el funcionamiento de cada una de dichas soluciones a partir de un ejemplo sencillo.

Esquema de solución

a) Ordenación arbitraria de reglas en la base de conocimiento. Ejemplo:
R11: SI h1 ENTONCES h2
R2: SI h3 ENTONCES h4
BA: {h1, h3} (Base de afirmaciones)
En encadenamiento hacia delante, tanto R11 como R2 se podrían aplicar. Finalmente se aplicaría R11 por estar antes en la base de conocimiento.

b) Ordenación de cláusulas dentro de las reglas según probabilidad de fallo. Ejemplo:
R1: SI explosión-solar O guerra-nuclear ENTONCES entrar-en-refugio-subterráneo
En encadenamiento hacia atrás, si nuestro hecho objetivo fuera "entrar-en-refugio-subterráneo", intentaríamos primero demostrar la primera clausula, "explosión-solar", cuyo fallo parece más probable. Esto nos evita desperdiciar recursos, ya que anticipamos el posible fallo del antecedente de la regla.

c) Adición de nuevas clausulas en todas las reglas, según etapas. Ejemplo: en un sistema experto agrícola, podría interesar utilizar una clausula referida a la estación del año en la que nos encontramos, para definir en qué estación es aplicable cada regla.
R1: SI primavera Y abril-lluvioso ENTONCES tarea10
R2: SI invierno Y disponibilidad-económica ENTONCES compra-de-maquinaria2

d) Asignación de prioridades a las reglas y utilización de agendas. Ejemplo:
R11: SI h1 ENTONCES h2 (prioridad 50)
R2: SI h3 ENTONCES h4 (prioridad 100)
BA: {h1, h3}
En encadenamiento hacia adelante podríamos introducir R11 y R2 en una agenda, la cual establecería que R2 es la regla a ejecutar, debido a que su prioridad es mayor.

e) Utilización de metarreglas. Ejemplo:
R1: SI h2 ENTONCES prioridad(R5, 20)
R1 es una metarregla porque permite razonar sobre conceptos asociados a reglas, "prioridad" en este caso. Lo que hace R1 es fijar la prioridad de R5 a 20, siempre que h2 sea cierto.

f) Aplicación de un mecanismo de refractariedad que impida, por ejemplo, que una regla se ejecute dos veces seguidas. Ejemplo:
R1: SI h1 ENTONCES h2
R2: SI h3 ENTONCES h4
BA: {h1, h3}
En encadenamiento hacia adelante, si suponemos que se ha ejecutado primero R1, a continuación debería ejecutarse R2 si se aplica este criterio.

g) Aplicación de un mecanismo de actualidad que obligue a que se ejecuten primero aquellas reglas cuyo antecedente se cumpla gracias a información obtenida en un ciclo más actual. Ejemplo:
R1: SI h1 ENTONCES h2
R2: SI h3 ENTONCES h4
BA: {h1(2), h3(3)}
En la base de afirmaciones, al lado de cada hecho se almacena el ciclo en que fue inferido. En encadenamiento hacia delante, habría que ejecutar R2 y no R1, ya que h3 es una información más actual que h1.

h) Aplicación de un criterio de especificidad que ejecute primero aquellas reglas más específicas.
Ejemplo:
R1: SI h1 ENTONCES h2
R2: SI h1 y h3 ENTONCES h4
BA: {h1, h3}
Según este criterio habrá que ejecutar antes R2, ya que es más especifica que R1, es decir, el antecedente de R2 incluye al de R1.

3.6 Ventajas e inconvenientes

Una de las características más importantes que presentan las reglas es la independencia entre ellas. Las reglas sólo se comunican a través de la BH: una regla no puede disparar directamente la ejecución de otra regla, es decir, el consecuente de una regla no puede contener otra regla. Por tanto, la BC está formada por un conjunto de reglas independientes entre sí. Esto proporciona una gran modularidad a los SBR, en donde el conocimiento puede introducirse y modificarse en cualquier orden.

Un SBR con grandes BC, la modificación de reglas (adición o borrado) puede originar efectos colaterales inesperados sobre las otras reglas almacenadas previamente, ya que el único punto de unión entre éstas es la BH. Son necesarios métodos para estructurar la BC, que faciliten la depuración y modificación del SBR.

Otra diferencia importante entre los SBR y la programación convencional basada en procedimientos es la forma de selección de una condición entre varias que se satisfacen simultáneamente. En la aproximación basada en procedimientos, se selecciona la siguiente condición en una secuencia lineal de condicionales (cuyo orden ha sido establecido), mientras que en un SBR el mecanismo de control tiene en cuenta diferentes factores para tomar la decisión, y ésta se realiza en el momento de la resolución.

Por otra parte, la representación declarativa del conocimiento es más cercana al pensamiento humano.

Las reglas son fáciles de leer y entender por gente que no es especialista.

Sin embargo, el uso de los SBR también posee inconvenientes, los SBR puros no permiten usar estructuras de control usuales en los sistemas de programación convencionales, tales como condicionales, iteraciones, llamadas a funciones, recursiones, etc.

El diseño de SBR no es trivial debido a la dificultad en el modelado del conocimiento.

El tipo de granularidad escogido influye en su eficiencia: una granularidad alta puede no captar los conceptos básicos del problema y una granularidad excesivamente fina aumenta considerablemente el número de reglas del sistema y conlleva una pérdida de generalidad entre éstas, que obliga a una frecuente actualización del sistema.

En su formulación más simple, las reglas pueden considerarse como una versión reducida de la lógica de predicados. La diferencia fundamental entre ambos métodos es que las reglas reducen la capacidad de representación (potencia expresiva) y la capacidad de inferencia de la lógica de predicados con el fin de obtener una mayor eficiencia. En cuanto a la expresividad, las reglas contienen variables y cuantificadores universales implícitos, pero en general no permiten representar cuantificadores existenciales, por lo que ofrecen mayor expresividad que la lógica de proposiciones pero no llegan a tener la expresividad de la lógica de predicados.

Otra ventaja de los SBR con respecto a la lógica, además de la eficiencia, es la posibilidad de tratamiento de la incertidumbre. Los SBR suelen manipular datos obtenidos de la experiencia, que pueden ser conocidos sólo de forma aproximada. Es decir, no reflejan implicaciones lógicas sino, más bien, las convicciones de un experto.

Los SBR son también principalmente sintácticos, ya que usan únicamente información sintáctica para decidir qué reglas desestiman.

Comparando la representación mediante marcos y los SBR, la principal ventaja de los primeros es que representan el conocimiento de forma estructurada. Toda la representación mediante un objeto se reúne en un marco, en vez de estar dispersa en un conjunto de reglas sin ninguna o poca estructura. Esto proporciona más eficiencia y mayor facilidad de mantenimiento de la BC. Las representaciones mediante marcos son normalmente más semánticas, es decir, sus procedimientos para el razonamiento son más variados, más eficaces y están más estrechamente relacionados con los tipos específicos de conocimiento. En contraste, es difícil expresar afirmaciones más complejas que la herencia de propiedades.

En resumen;

1. Los Sistemas Basados en Reglas, al consistir en un conjunto de reglas independientes, son muy modulares y por tanto presentan buenas propiedades de mantenibilidad si su tamaño no es muy grande

2. En sistemas grandes se requieren métodos de estructuración de la Base de Conocimientos para facilitar la depuración y evitar efectos colaterales en las fases de actualización y mantenimiento.

3. La representación declarativa del conocimiento facilita la incorporación en estos sistemas de facilidades de autoexplicación.

4. Actualmente las herramientas de desarrollo de Sistemas Basados en Reglas integran otras técnicas de programación convencionales.

5. Los Sistemas Basados en Reglas tienen menor potencia expresiva que la lógica de predicados, en aras de la eficiencia computacional, pero mayor que la lógica proposicional; e incorporan aspectos de diferentes extensiones de la lógica clásica.

6. Los campos de aplicación idóneos de los Sistemas Basados en Reglas son aquellos que se pueden modelar como un conjunto de múltiples estados, y el conocimiento se puede separar claramente de la forma en que se usa.

3.7 Dominios de aplicación

Hay dominios en los cuales los problemas se modelan de forma sencilla mediante los SBR, mientras que en otros dominios no sucede esto. Podemos distinguir dos clases diferentes de problemas:

1- aquellos que se pueden modelar como un conjunto de múltiples estados. Un ejemplo es la medicina clínica, en donde hay muchos estados asociados a las acciones que se pueden llevar a cabo. Esto suele deberse a la falta de una teoría concisa o a la complejidad del sistema a modelar. En estos casos, es sencillo asociar múltiples reglas modulares a los diferentes estados independientes.

2- Aquellos que vienen descritos por medio de una teoría concisa y unificada. Ejemplos son las áreas de la física o las matemáticas, en las que unos pocos principios contienen mucho del conocimiento requerido.

Teniendo en cuenta la complejidad del flujo de control, se puede distinguir entre problemas representados por un conjunto de acciones independientes (por ejemplo, tareas tales como la monitorización) y problemas representados por una colección compleja de múltiples procesos paralelos con varios subprocesos dependientes. En el primer caso, la comunicación entre las acciones es muy

limitada y, como vimos, ésta es una característica importante de los SBR. Sin embargo, los problemas con flujos de control (bucles y saltos) complicados requieren comunicación explícita entre las acciones (ya que unas acciones invocan otras). Los SBR no proporcionan mecanismos para establecer estas comunicaciones.

Resumiendo, los SBR se muestran adecuados para modelar dominios donde las tareas se caracterizan por el reconocimiento de un número elevado de estados distintos.

3.8 Resumen

Los SBR permiten modelar gran cantidad de conocimiento útil que se suele expresar mediante reglas sencillas del tipo SI ... ENTOCES. Se debe a que utilizan el razonamiento deductivo para llegar a obtener conclusiones lógicas. Como hemos visto, los SBR pueden interpretar estas reglas de dos formas. Una primera alternativa consiste en aplicar las reglas de la forma condición-acción en un control con encadenamiento hacia delante. La segunda opción considera las reglas como conjuntos de implicaciones lógicas, a partir de las cuales se obtienen las deducciones en un control con encadenamiento hacia atrás. En ambos casos, se requieren técnicas de equiparación entre el estado actual de la BH y las condiciones de las reglas para determinar, entre todas las reglas, cuáles de ellas se pueden aplicar en un ciclo dado de la inferencia.

El algoritmo de equiparación RETE evita examinar todas las reglas de la BC con todos los datos de la BH. Para ello, genera una red que agrupa las condiciones iguales de las reglas en un único nodo y equipara, en cada ciclo, sólo las modificaciones más recientes de la BH, almacenando dichas modificaciones de ciclo en ciclo.

El resultado del proceso de equiparación es una lista de reglas cuyas partes izquierdas se han equiparado con la descripción del estado actual. Los métodos de resolución de conflictos permiten determinar, entre varias reglas seleccionadas, cuál de ellas se debe aplicar.

T7 – Redes semánticas.

Capítulo 4 del libro base

4.1 Introducción

En representación de conocimientos se puede hablar de tres modelos distintos:

- El modelo **conceptual** que es una representación de los conocimientos de un dominio utilizando estructuras no computables que modelizan el problema y la solución en un dominio concreto.
- El modelo **formal** que es una representación "semiinterna" o "semicomputable" de los conocimientos de un dominio. Este modelo formal se ha de obtener a partir del modelo conceptual.
- El modelo **computabl**e que hace que el modelo formal sea totalmente operativo, y que está formado por una base de conocimientos, un motor de inferencias y una serie de estrategias de control.

La formalización de conocimientos consiste en representar simbólicamente los conocimientos de un dominio utilizando alguno de los formalismos de representación de conocimientos existentes, organizarlos de acuerdo a algún modelo de diseño y determinar los métodos de inferencia adecuados para manejar eficiente y efectivamente los conocimientos representados.

Para representar los conocimientos de un determinado dominio se debe utilizar uno o varios formalismos o técnicas de representación de conocimiento.

En sistemas pequeños y sencillos, lo normal es que solamente se utilice un formalismo para representar el conocimiento; pero a menudo se combinan dos o tres formalismos de modo que se complementen unos a otros para lograr una representación que se ajuste de la forma más natural posible a los conocimientos del dominio.

Cada formalismo tiene asociadas unas técnicas básicas de inferencia. Estas técnicas son independientes del dominio representado por el formalismo.

En función de si el formalismo es más adecuado para representar conceptos, relaciones o acciones, se puede realizar la siguiente clasificación:

- Formalismos basados en **conceptos**, los cuales representan las principales clases o entidades del dominio, sus propiedades, y los posibles valores que puede tomar cada propiedad. Marcos.
- Formalismos basados en **relaciones**, que centran su atención en las relaciones que aparecen entre los conceptos o entidades del dominio. Los más importantes son: la **Lógica** y **las Redes Semánticas.**
- Formalismos basados en **acciones**, que describen los conocimientos del dominio como un conjunto de acciones básicas. Los principales formalismos de este tipo son: los Sistemas de Producción(reglas) y los Guiones.

El formalismo de las Redes Semánticas fue definido por Quillian como un grafo orientado formado por nodos y arcos unidireccionales, ambos etiquetados. Los nodos representan conceptos y los arcos relaciones entre conceptos. Las redes semánticas se están utilizando en IA como una técnica de representación de conocimientos para expresar relaciones entre los conceptos de un dominio.

El formalismo de Marcos fue definido por Minsky como una estructura de datos para representar estereotipos.

Cada marco está formado por un nombre y un conjunto de propiedades. Los valores de cada propiedad se describen en el marco utilizando un conjunto de facetas. Los marcos se unen unos con otros mediante relaciones. La relación subclase-de permite crear jerarquías de conceptos por especialización y proporciona un camino para la herencia de propiedades.

Las principales características que han hecho de los marcos el formalismo de representación más utilizado cuando los conocimientos del domino están basados en conceptos son: la organización de los conceptos que intervienen en el problema en jerarquías o taxonomías de conceptos, y la posibilidad de representar conocimientos procedimentales y declarativos en los marcos de una forma integrada.

Formalizar consiste en representar simbólicamente los conocimientos de un dominio utilizando alguno de los formalismos de representación de conocimientos existentes (paso del *modelo conceptual* al *modelo formal*). Hay distintos tipos de formalismos: basados en conceptos (*marcos*), basados en relaciones (*redes semánticas*) y basado en acciones.

4.2 Redes semánticas

Las redes semánticas (o redes conceptuales) son un formalismo o paradigma de representación de conocimiento basado en relaciones; es decir, este formalismo centra su atención en las relaciones que aparecen entre los conceptos o entidades de un dominio.

Básicamente, una red semántica es un grafo orientado formado por:

- nodos etiquetados, que representan **conceptos** e **instancias**.
- arcos unidireccionales etiquetados, que representan **relaciones entre conceptos** o instancias. La Figura 4.1 muestra los conceptos básicos de representación en redes semánticas.

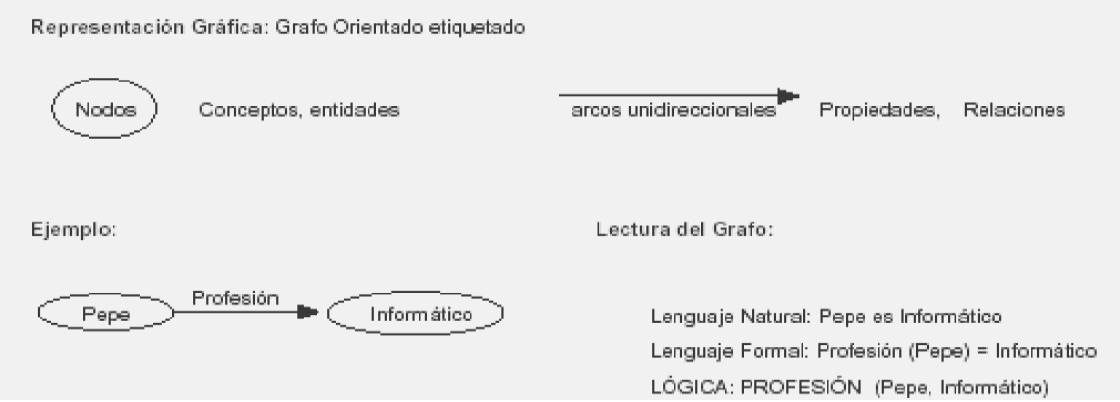


Figura 4.1: Conceptos básicos en redes semánticas.

4.2.1. Representación del conocimiento

En una red semántica, la información se representa en un grafo dirigido formado por un conjunto de nodos y arcos unidireccionales, ambos etiquetados. Los nodos representan conceptos e instancias de dichos conceptos, y los arcos conectan los nodos y representan relaciones binarias entre ellos (predicados de aridad dos). Ejemplo:

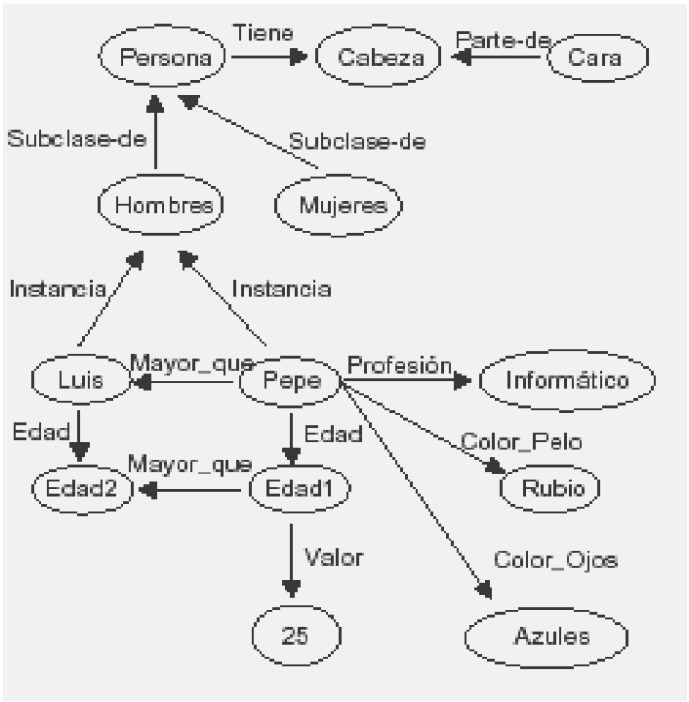


Figura 4.2: Ejemplo de red semántica.

La base de la representación consiste en modelar conocimientos relativos a un objeto (concepto) mediante pares *atributo-valor*. Así, el nodo origen es el *concepto*, el arco que los une es el *atributo*, y el nodo destino es el *valor* para dicho atributo de ese objeto.

Los arcos en las redes conceptuales (semánticas) se agrupan en dos categorías:

- **Arcos descriptivos**. Describen entidades y conceptos. Por ejemplo, un

arco descriptivo para una red que representase de alguna forma personas podría ser Profesión. Ejemplos de arcos descriptivos en la red de la Figura 4.2 son profesión y color-pelo. En la red conceptual de la misma figura podemos ver dos tipos de arcos descriptivos: arcos que relacionen dos entidades independientes ya existentes y arcos utilizados para definir una nueva entidad. Por ejemplo, el arco profesión une dos nodos (Pepe, informático) con existencia propia. Sin embargo, los nodos Edad1 y Edad2 son nuevos conceptos que representan la edad de Pepe y de Luis respectivamente, y que se definen por sus relaciones con dichos nodos.

- **Arcos estructurales**. Enlazan las entidades y conceptos formando (modelando) la arquitectura de la red. Ejemplos de arcos estructurales en la Figura 4.2 son los etiquetados como subclase-de, instancia y parte-de, que se corresponden respectivamente con los procesos básicos de generalización, instanciación, y agregación. La semántica de estos arcos es independiente del dominio del problema. Se pueden definir tantas etiquetas estructurales como se necesiten:

    - **Generalización**, ponen en relación una clase con otra más general. Las propiedades definidas en los nodos generales son heredadas por deducción por los nodos específicos, mediante arcos *Subclase-de*. Por ejemplo en la Figura 4.2 se puede deducir que Pepe por ser Hombre y por ser Persona tiene Cabeza.
    - Arco **instancia**, liga un objeto con su tipo genérico. Se llama arco *Instancia*. Por ejemplo, la aserción "Pepe es un Hombre" se representa en la Figura 4.2 mediante el arco instancia desde el nodo Pepe hacia el nodo Hombre.
    - **Agregación**, liga un objeto con sus componentes. Se llama arco *Parte-de*. la aserción "la Cara forma parte de la Cabeza" se representa utilizando el arco parte-de desde el nodo Cara hacia el nodo Cabeza.

Los conocimientos expresados en una red semántica también pueden expresarse utilizando lógica proposicional y cálculo de predicados de primer orden.

4.2.2 Representación de predicados no binarios

Generalmente, en las redes semánticas solo es posible representar predicados de aridad dos. Para representar predicados de aridad 3 o superior, es necesario crear un nuevo objeto que represente al predicado de aridad mayor que dos, y definir nuevos predicados binarios que describan las relaciones entre este nuevo objeto y sus argumentos.

Por ejemplo, al representar el predicado COMPRAVENTA(Pepe, Luis, Reloj1, 45, euros) en una red semántica, se crearía un nodo que representa la compra-venta (compra-venta1) y cinco predicados (vendedor, comprador, objeto, precio, unidad) que representan las relaciones con las cinco piezas de información, como se muestra en la Figura 4.3.

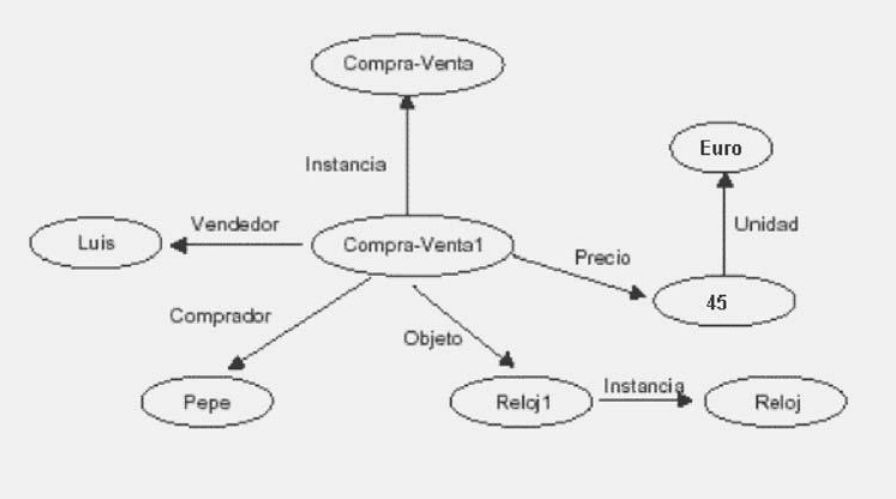
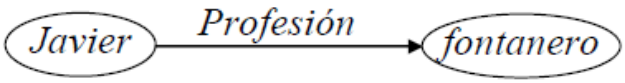


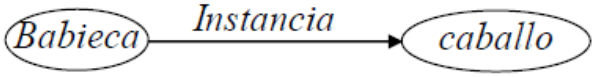
Figura 4.3: Representación de predicados no binarios

En resumen(examen septiembre 2013):

La etiqueta asociada a un arco de una red semántica suele representar un predicado binario cuyo par de argumentos son el objeto origen y el objeto destino unidos por el arco. Por ejemplo, el hecho Profesión(Javier, fontanero), que consta del predicado binario “Profesión”, se puede representar mediante la siguiente red semántica:

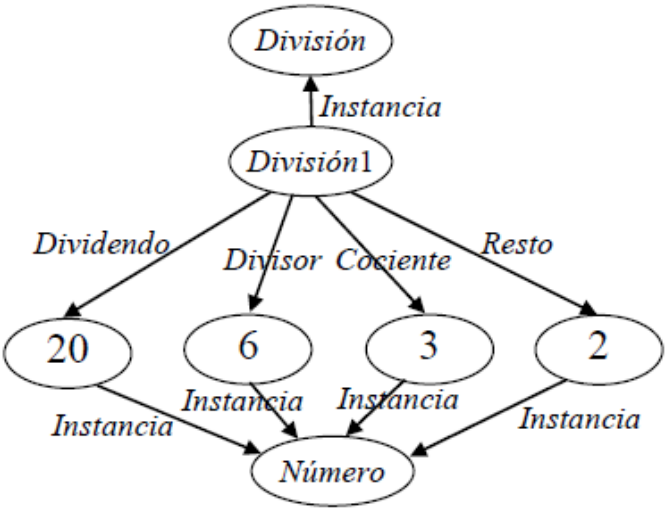


Pero también es posible representar predicados no binarios mediante redes semánticas. Por ejemplo, el hecho Caballo(Babieca) es equivalente a Instancia(Babieca, Caballo). Por tanto, en este ejemplo el predicado unario “Caballo” se representaría del siguiente modo mediante una red semántica:



Si se quisieran representar predicados de aridad mayor que dos, habría que operar de un modo diferente. En este caso, el predicado de aridad mayor que dos se representa mediante un objeto y se utilizan predicados binarios adicionales para describir las relaciones entre este objeto y sus argumentos.

A modo de ejemplo, si consideramos el hecho División(20, 6, 3, 2), que representa la operación de división de 20 entre 6 cuyo cociente es 3 y cuyo resto es 2, la red semántica que lo representa sería:



Como puede comprobarse en la figura, se ha creado un nodo que representa el predicado no binario (División1) y cuatro predicados binarios (Dividendo, Divisor, Cociente y Resto) que representan las relaciones con los cuatro argumentos del predicado no binario.

4.2.3 Representación de acciones

Se basa en la *gramática de casos*. En ella, toda proposición tiene una estructura formada por un verbo y una o varias frases nominales. Cada frase nominal se relaciona con el verbo mediante un conjunto de casos, que pueden ser:

- *agente* (persona que realiza la acción)
- *contra-agente* (resistencia contra la que se ejecuta la acción)
- *objeto* (entidad cuya posición o existencia se considera)
- *Resultado*: la entidad que aparece como consecuencia de la acción.
- *Instrumento*: el estímulo o causa física inmediata de un evento.
- *Origen*: el lugar del que procede el evento.
- *Propósito*: el motivo por el que se ejecuta la acción.
- *Lugar*: sitio en el que se desarrolla la acción.
- *tiempo* (fecha o momento concreto)
- *sujeto* (entidad que sufre el efecto).

La modalidad por su parte hace referencia a características que presenta el verbo, como por ejemplo: *tiempo del desarrollo de la acción* (pasado, presente o futuro) y *voz del verbo* (activa o pasiva).

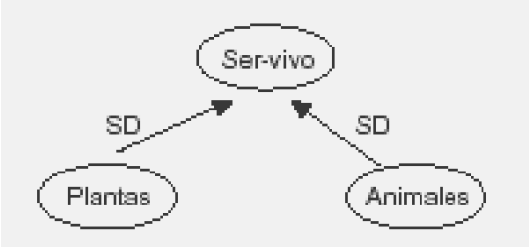
Para representar acciones y eventos se utilizan nodos situación o suceso. Cada nodo situación tiene como atributos el conjunto de casos y de modalidades que describen el evento. Este esquema de nodos situación permite representar sentencias compuestas.

4.2.4 Representación de conocimiento disjunto

En algunas ocasiones debemos representar entidades del domino que son disjuntas entre sí (no tienen elementos comunes), se puede utilizar simplemente el arco *Disjunto* de una entidad a otra.

Sin embargo, existe otro formalismo definido por Hendrix que utiliza arcos especiales, como sigue:

- *arco S* (subconjunto)
- *arcoSD* (subconjunto disjunto)
- *arco E* (elemento)
- *arco ED* (elemento disjunto).



Por ejemplo, tendríamos una entidad *Ser-Vivo* que representa el conjunto de seres vivos, la cual tiene dos subconjuntos *Plantas* y *Animales* disjuntos entre sí. Por tanto, tendríamos un arco *SD* desde la entidad *Plantas* y otro arco *SD* desde la entidad *Animales* que irían ambos hacia la entidad *Ser-Vivo*.

4.3. Inferencia de conocimiento

Un sistema que utilice como formalismo de representación de conocimientos las redes semánticas, debe utilizar los conocimientos almacenados en la red para resolver los casos que se planteen. La eficacia del razonamiento en las redes depende de los procedimientos que trabajan con la semántica de sus arcos. Las técnicas más empleadas son la equiparación y la herencia de propiedades.

4.3.1 Equiparación

Se dice que un *apunte* (fragmento de una red, ej. una pregunta) se equipara con la red semántica si el apunte puede asociarse con un fragmento de la red. Los pasos a seguir en el proceso de equiparación si queremos que una red responda a una pregunta son:

- Paso 1. Se construye un apunte que responda a la pregunta que se quiere resolver, formado por un conjunto de:

  - nodos *constantes* (datos conocidos de la pregunta)
  - nodos *variables* (valores que se requieren, son desconocidos)
  - *arcos etiquetados* (como arco *agente*, arco *lugar*, arco *objeto*, etc, que unen nodos constantes y variables).

- Paso 2. A continuación, se coteja el apunte con la red semántica en la base de

conocimientos.

- Paso 3. Los nodos variables del apunte se ligan a nodos constantes de la red semántica hasta encontrar una equiparación perfecta.

- Paso 4. La respuesta a la consulta es el fragmento de red semántica construido con los valores con los que se rellenan los nodos variables. Si no hubiera ninguna equiparación del apunte con la red semántica la respuesta dada por la técnica de equiparación sería "No se ha encontrado".

4.3.2 Herencia de propiedades

La herencia de propiedades está basada en la regla del modus ponens.

Permite que nodos específicos de una red accedan a las propiedades definidas en otros nodos utilizando los arcos *Instancia* y *Subclase-de*, evitando así la redundancia de propiedades en la base del conocimiento.

La herencia de propiedades se puede utilizar en sistemas que razonan dirigidos por la meta o por los datos.

Para verificar una sentencia habrá que buscar un nodo que corresponda con el sujeto sobre el que se solicita validar la afirmación, después localizar un arco relativo a la materia sobre la que se necesita verificación, si no existe se buscarán arcos *instancia* hacia algún otro nodo, se elegirá uno de estos nodos y se volverá a buscar el arco relativo a la materia en ese nodo, y así sucesivamente hasta que podamos afirmar la veracidad de la afirmación o tengamos que concluir que con el conocimiento almacenado no puede afirmarse.

La herencia de propiedades trabaja muy bien con propiedades que presentan excepciones en sus valores.

La distribución de propiedades en la red permite excepciones, ante la posibilidad de heredar un valor de dos nodos distintos, que se herede el valor de la propiedad del nodo más cercano al nodo que sirvió como punto de partida en la inferencia permitiendo razonamientos no monótonos.

Los principales errores que se suelen cometer utilizando herencia de propiedades por formalizar mal los conocimientos son:

- No distinguir bien los nodos que son instancias de aquellos que son conceptos.
- Que el nombre etiquetado tenga una semántica diferente al conocimiento representado.
- Establecer arcos en sentido contrario al natural o adecuado.
- No representar situaciones empleando nodos situación o evento.

T8 – Marcos

Capítulo 4 del libro base

4.4 Marcos

El formalismo de marcos es la técnica de representación de conocimientos más utilizada en IA cuando los conocimientos del dominio están organizados sobre la base de conceptos, se definen como una estructura de datos que representa situaciones esteorotipadas construidas sobre situaciones similares ocurridas anteriormente, permitiendo así aplicar a situaciones nuevas los conocimientos de situaciones, eventos y conceptos previos.

Los marcos organizan los conocimientos del dominio en árboles, también llamados jerarquías, o en grafos, ambos construidos por especialización de conceptos generales en conceptos más específicos.

Las técnicas de inferencia utilizadas para razonar con los conocimientos de la base de conocimientos son:

- equiparación, para clasificar entidades en una jerarquía.
- herencia simple y herencia múltiple, para compartir propiedades que están distribuidas en la jerarquía de conceptos o en el grafo, respectivamente.
- valores activos y métodos para representar la conducta del sistema.

Los marcos utilizan para representar conocimiento grafos dirigidos en cada uno de cuyos nodos se estructura la información correspondiente a las características de cierta entidad del dominio. Estos grafos son construidos por especialización de conceptos generales en conceptos más específicos. Asociado a este **conocimiento declarativo** de carácter jerárquico y estereotipado, existe generalmente **conocimiento procedimental** adicional, de manera que un sistema de marcos utiliza cuatro elementos fundamentales para representar conocimiento: *marcos*, *relaciones*, *propiedades* y *facetas*.

Hay dos tipos de marcos, **clases** e **instancias**, y mientras que las primeras expresan conceptos genéricos, las segundas constituyen ejemplos concretos de dichos conceptos genéricos. Los diferentes marcos quedan enlazados entre sí por medio de relaciones. Las relaciones estándar más importantes son las de "*Subclase-de*", que une una clase específica con otra clase más genérica que la engloba, y la de "*Instancia*", que une una instancia con la clase a la que pertenece. Sobre este tipo de relaciones se puede aplicar herencia de propiedades. Existen otras relaciones no estándar, que no se pueden aplicar herencia de propiedades, que expresan dependencias entre conceptos del dominio: la relación "Fraternal" une dos clases cuya clase padre coincide, la relación "Disjunta" une dos clases que no pueden tener una instancia común y diferente, relaciones "Ad-hoc" entre dos clases pueden ser definidas por el

usuario. Las propiedades de un marco permiten describir el concepto asociado al mismo, de manera que cada propiedad represente un atributo o característica genérica del marco. Las propiedades se crean en las clases de un sistema de marcos, pero pueden ser definidas bien para que los valores que toman sólo puedan ser rellenados en clases bien para que sólo puedan ser rellenados en instancias de clases. Cada propiedad y cada relación de un sistema de marcos tiene un conjunto de facetas asociadas. Una faceta puede definir cierta característica como: el tipo de dato con el que se rellenará la propiedad o relación, la cardinalidad mínima o máxima del dato y si es multivaluada o no. Existen otras facetas que afectan a propiedades de instancias y que indican el conjunto de valores válidos de la propiedad, el valor por omisión de la propiedad, o qué procedimiento o regla hay que ejecutar si se necesita, modifica, añade o borra el valor de la propiedad.

La inferencia en marcos se lleva a cabo mediante tres tipos principales de procesos: *equiparación*, *herencia* y *demonios*. La equiparación es un método de inferencia utilizado en sistemas de marcos para la clasificación de entidades en la base de conocimiento. Dados los valores conocidos de un subconjunto de propiedades de un marco pregunta, la equiparación identifica qué marco de la base de conocimiento corresponde al marco pregunta. En muchas ocasiones, este proceso no es un proceso exacto y debe llevarse a cabo en tres fases: selección de marcos candidatos, cálculo de valores de equiparación y toma de la decisión final. Otro método de inferencia en sistemas de marcos es la herencia de propiedades, que permite compartir propiedades y valores de propiedades usando las relaciones "Subclase-de" e "Instancia".

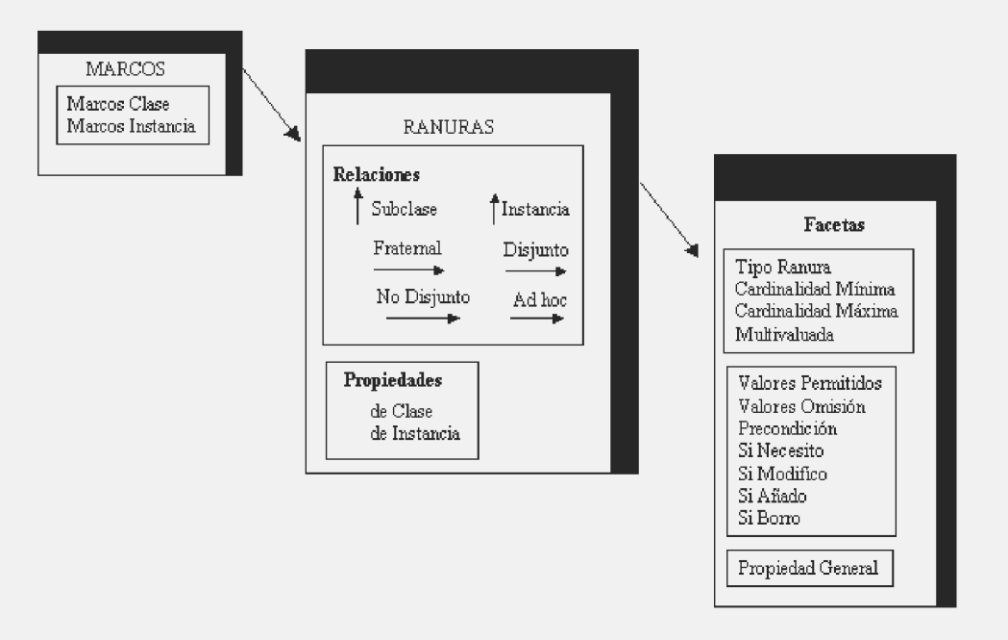
La herencia es simple cuando el sistema de marcos tiene forma de árbol, considerando exclusivamente las relaciones "*Subclase-de*" e "*Instancia*". En este caso, el valor de una propiedad de una instancia se busca en primer lugar en dicha instancia y, en caso de fracaso, se toma de la clase más cercana a la instancia donde figure dicho valor. Si un marco es subclase o instancia de más de una clase, se dice que la herencia es múltiple. En este caso, dado que una instancia puede tener más de una clase padre que contenga la propiedad buscada, el valor de la propiedad que hereda la instancia depende del método de búsqueda utilizado para recorrer el grafo: en profundidad, en anchura o utilizando el concepto de "distancia inferencial".

Por último, otra técnica de inferencia en marcos son los demonios, que son procedimientos asociados a propiedades de marcos y cuya función es recuperar, almacenar o borrar información en las mismas.

4.4.1 Representación de conocimiento

Los conceptos que se utilizan al formalizar la base de conocimientos en marcos son:

- **marcos** para representar conceptos(clases) y elementos(instancias)
- **relaciones** para expresar dependencias entre conceptos
- **propiedades** para describir cada concepto
- **facetas** para expresar formas y valores con los que se puede rellenar cada propiedad.



La Figura 4.11 muestra los conceptos básicos de representación en marcos.

4.4.1.1 Representación de conceptos e instancias

Existen dos tipos de marcos:

- **Marcos clase:** se utilizan para representar **conceptos**, clases o situaciones genéricas (entidades acerca de las cuales se desea describir cierto tipo de información), descritos por un conjunto de propiedades, unas con valores y otras sin valores asignados, que son comunes al concepto, clase o situación que el marco representa. Los marcos clase representan conceptos, es decir, entidades acerca de las cuales se desea describir cierto tipo de información. Los conceptos pueden ser de cualquier índole, ya se refieran a entidades físicas, descripción de tareas, procesos de razonamiento, entidades abstractas, etc.
- **Marcos instancia**: son la representación en el dominio real de una clase determinada (individuo) y deben estar relacionados, como mínimo, con un marco clase. Además, han rellenado la mayoría de sus propiedades con valores específicos; es decir, con información concreta del elemento, instancia o individuo que representan, y que pertenece al concepto, clase o situación representado por un marco clase. El resto de propiedades que no estén almacenadas en él, las hereda de los marcos clase de los cuales son instancias.

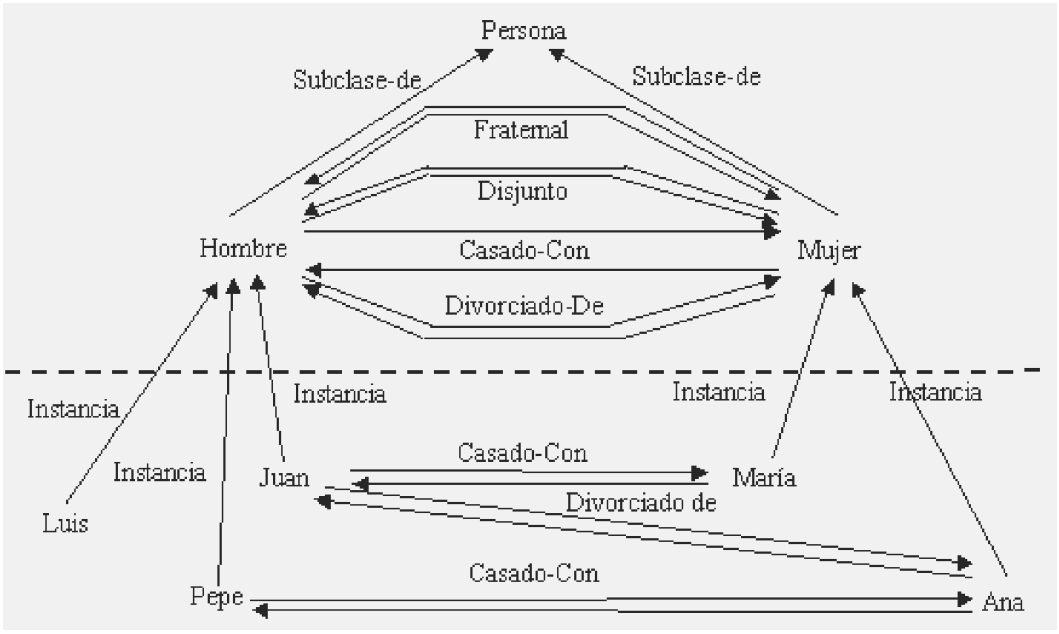


Figura 4.12: Ejemplo de una jerarquía en marcos.

4.4.1.2 Representación de relaciones entre conceptos

El formalismo de marcos representa las relaciones del dominio mediante relaciones entre marcos clase, entre marcos instancia, y entre marcos clase y marcos instancia, formando así un sistema basado en marcos (SBM). Intuitivamente, el significado del SBM de la Figura 4.12 es el siguiente:

- Los marcos clase Hombre y Mujer son subclases del marco clase Persona, y el marco clase Persona es una superclase de los marcos clase Hombre y Mujer.
- Los marcos instancia Luis, Pepe y Juan son instancias del marco clase Hombre y, por lo tanto, el conjunto de los Hombres está formado por Luis, Pepe y Juan.
- Los marcos instancia Ana y María son instancias del marco clase Mujer y, por lo tanto, el conjunto de las Mujeres está formado por María y por Ana.
- Los marcos clase Hombre y Mujer son hermanos, pues tienen el mismo padre.
- Los marcos clase Hombre y Mujer son disjuntos, pues no hay instancias que pertenezcan simultáneamente a ambos marcos clase.
- Las relaciones casado-con y divorciado-de definidas entre marcos clase representan que los Hombres y las Mujeres se casan y se divorcian. Las relaciones casado-con y divorciado-de, definidas entre marcos instancia, representan que Juan está casado con María y divorciado de Ana, y que Ana está casada con Pepe.

Hay ciertas relaciones (*subclase-de* e *instancia*) que son independientes del dominio, y que reciben el nombre de relaciones estándar. La técnica de inferencia basada en herencia de propiedades razona sobre dichas relaciones.

También existen otras relaciones llamadas relaciones no estándar, para representar relaciones "a medida" entre conceptos de un dominio, sobre las que no se puede aplicar herencia de propiedades. La Figura 4.13 representa gráficamente la sintaxis de las relaciones más utilizadas en el paradigma de marcos, y que son las siguientes:

- Relaciones estándar
  - *Subclase-de*, y su relación inversa *superclase-de*.
  - *Instancia*, y su relación inversa **representa**.

- Relaciones no estándar
  - *Fraternal.*
  - *Disjunto.*
  - *No disjunto.*
  - *Ad-hoc*, o relaciones "a medida".

Muchas de las herramientas existentes para construir SBM no implementan relaciones no estándar. Esto significa que el IC debe buscarse un "truco" para representarlas (por ejemplo, mediante otro marco), si realmente las necesita en su sistema. En este caso, a diferencia de las relaciones estándar, el motor de inferencias no trabajará con ellas, y el IC deberá implementar las inferencias bien con procedimientos programados, con reglas, con demonios, o con cualquier otro modo de expresar conocimientos procedimentales.

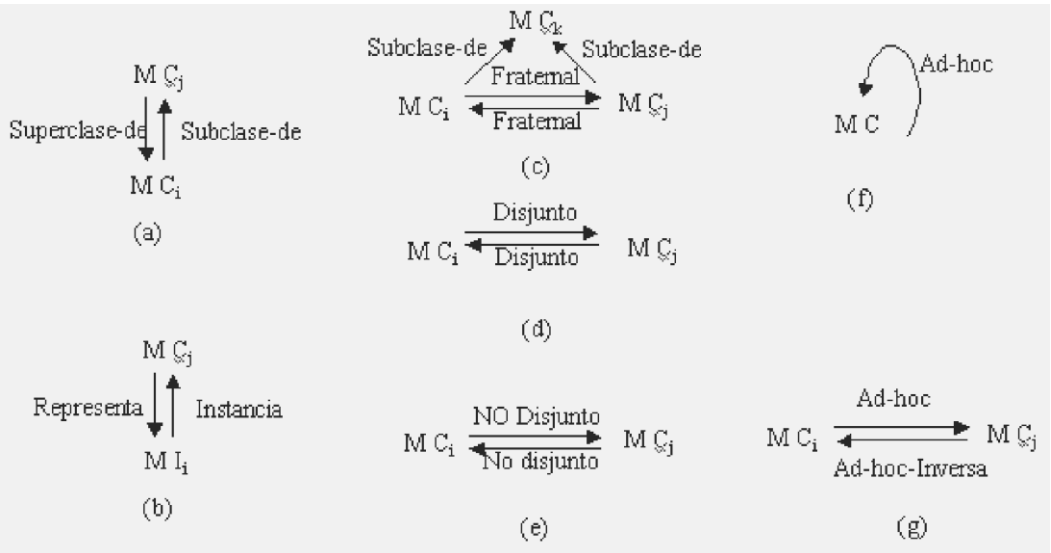


Figura 4.13: Sintaxis de relaciones en marcos.

Relación estándar *Subclase-de* (Figura 4.13a)

La relación *subclase-de* permite construir un SBM mediante la especialización de conceptos generales en conceptos más específicos. Sintácticamente, la relación *subclase-de* está bien definida si los marcos origen y destino son marcos clase. La semántica correcta es que el concepto representado por MCi es subconjunto del concepto representado por MCj. La relación inversa de la relación *subclase-de* es la relación *superclase-de*. Dado que un concepto puede especializarse en varios conceptos o ser clasificado desde varios puntos de vista, a un marco clase pueden llegar y/o partir un número indefinido de relaciones *subclase-de* y *superclase-de*. Las relaciones subclase-de en los SBM definen un camino para la herencia de propiedades. Dada la jerarquía de la Figura 4.12, la herencia de propiedades permitirá que todas las propiedades que se definen en el marco clase Persona sean accesibles desde Hombre y desde todas sus instancias, definiéndose en Hombre sólo aquellas propiedades que distinguen a esta clase de la clase Persona, y en la clase Persona aquellas propiedades que son comunes a los marcos clase Hombre y Mujer.

Relación estándar *Instancia* (Figura 4.13b)

Sintácticamente, una relación *Instancia* está bien definida si tiene como origen un marco instancia MIi y como destino un marco clase MCj. Semánticamente, esta relación representa que el marco instancia es un elemento del conjunto o clase representado por el marco clase. De un marco instancia pueden partir tantas relaciones *instancia* como conceptos lo describan (un elemento puede pertenecer a varios conjuntos). Permiten construir jerarquías o clasificaciones de conocimientos estáticos del dominio que se quiere modelar. Esta jerarquía será un árbol si existe un único camino que une cada marco instancia con el nodo raíz de la jerarquía, y un grafo en caso contrario.

Relaciones no estándar

Las relaciones no estándar representan dependencias entre conceptos de un dominio.

Como ejemplo de relaciones que representan la estructura de los conceptos del dominio se tienen las siguientes:

- La relación *fraternal*(c) está sintácticamente bien definida si los marcos origen y destino son marcos clase, y ambos están unidos mediante relaciones *subclase-de* con el mismo marco clase padre. Semánticamente, representa que dos conceptos son hermanos y, por consiguiente, tienen que tener el mismo padre.
- Una relación *disjunto*(d) está sintácticamente bien definida si los marcos origen y destino son marcos clase. Semánticamente, representa que las clases son disjuntas, (su intersección es el conjunto vacío), así que los conjuntos o clases representados por ambos marcos no tienen ningún

elemento en común. Esta relación está bien definida si no se ha definido previamente una relación no-disjunto entre ambos marcos clase.
- La relación *no-disjunto*(e) es opuesta semánticamente a la relación disjunto. Por consiguiente, marcos clase conectados por relaciones *no-disjunto* pueden tener instancias comunes.

Relaciones a medida o ad-hoc(Figura 4.13 f,g) (pertenece, casado-con, divorciado-de). Expresan dependencias a medida entre conceptos de un dominio. Cuando se definen relaciones ad-hoc en un dominio hay que asegurarse de definirlas entre marcos clase, y no entre marcos instancias.

Cuando se definan relaciones ad-hoc entre dos marcos instancia, previamente hay que comprobar lo siguiente:

- Que la relación ad-hoc haya sido definida previamente entre dos marcos clase.
- Que los marcos instancia que se quiere conectar sean instancias de dichos marcos clase.

Es importante mencionar que cada relación ad-hoc entre dos marcos instancia es una instancia de la relación ad-hoc definida a nivel de clase. Por ejemplo, en la jerarquía de la Figura 4.12 existen cuatro instancias de la relación ad-hoc casado-con. Las relaciones ad-hoc también se utilizan para conectar marcos clase de diferentes jerarquías.

4.4.1.3 Representación de las propiedades de los conceptos

El IC utiliza dos tipos de propiedades cuando formaliza su BC en marcos para describir los conceptos:

- **Las propiedades de clase** representan atributos o características genéricas de un concepto o clase. Se rellenan en el marco clase, y toman siempre el mismo valor en todos los elementos o instancias de la clase. En la jerarquía de la Figura 4.16, la propiedad animal-racional en el marco clase Persona y la propiedad sexo en los marcos clase Hombre y Mujer son propiedades de clase.
- **Las propiedades de instancia**, aunque el IC las define en el marco clase y son comunes a todas las instancias del marco clase, se rellenan en cada instancia con valores concretos que dependen del elemento de la clase que se esté representando. Gráficamente, si la propiedad va precedida del símbolo "*" se trata de una propiedad de instancia. En el SBM de la Figura 4.16, las propiedades nombre, edad, nacionalidad, estado-civil, tipo-de-matrimonio, religión, fecha-de-boda y fecha-de-nacimiento son propiedades de instancia en el marco clase Persona. A veces no se rellenan todas las propiedades definidas en los marcos clase por desconocimiento o para evitar repetir información que puede obtenerse mediante inferencia, p.e repetir la fecha de la boda sólo en uno de los cónyuges.

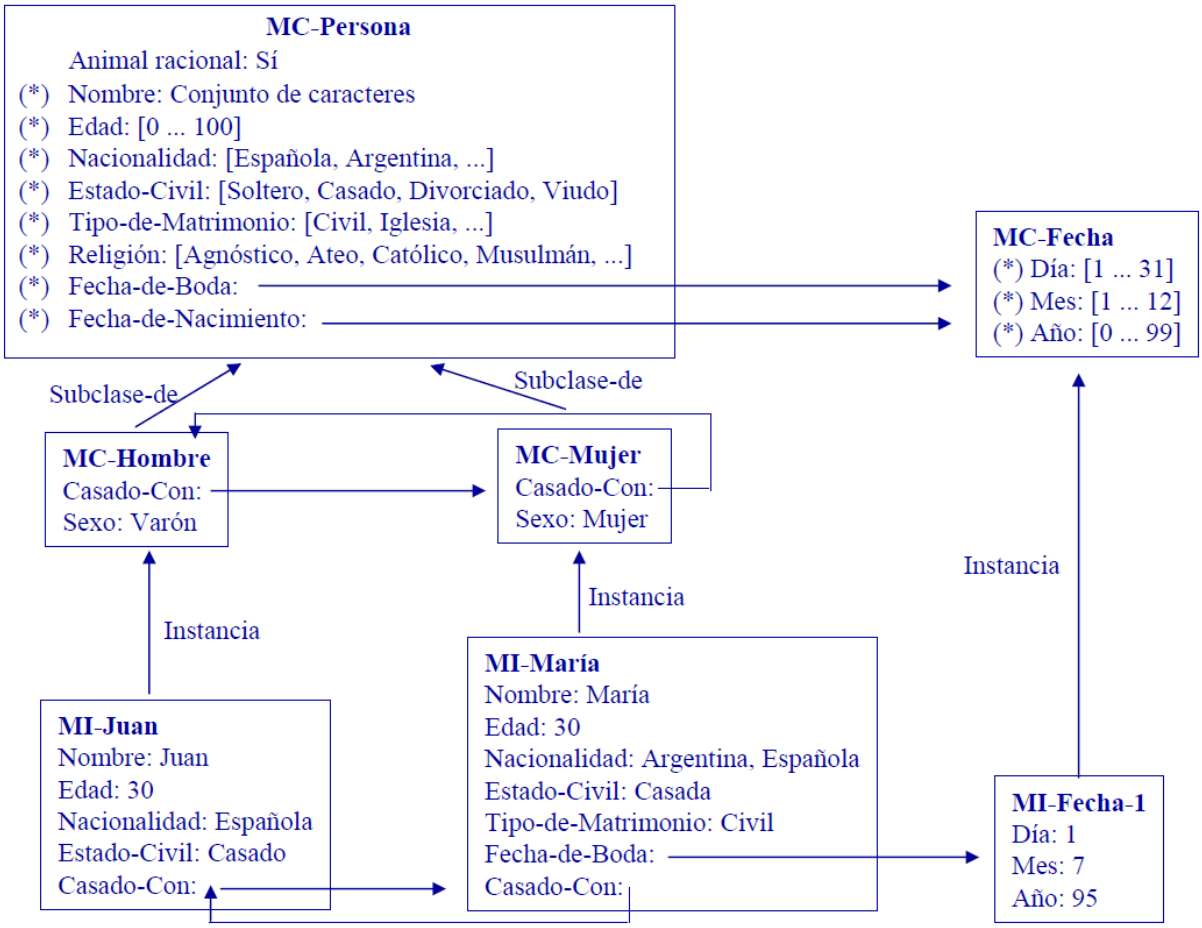


Figura 4.16: Ejemplos de propiedades en un SBM.

Al definir las propiedades de clase y de instancia en los marcos clase hay que tener en cuenta lo siguiente:

- La distribución de las propiedades en el SBM debe favorecer que unos marcos compartan propiedades entre ellos, evitando la presencia de conocimientos redundantes.
- Las propiedades deben proporcionar la suficiente información como para determinar la entidad que el marco clase está representando, es decir, que no podamos confundir una entidad representada por él de una que no lo es.
- El nombre de cada propiedad refleja su semántica, pero no indica la importancia que tiene la propiedad en el marco clase.
- El carácter local de cada propiedad permite tener propiedades con el mismo nombre en diferentes marcos. Por ejemplo, la propiedad sexo en los marcos clase Hombre y Mujer.
- Un marco clase rellena las propiedades de clase con valores concretos.
- En un marco clase se puede definir una propiedad de instancia utilizando otros marcos clase. Por ejemplo, las propiedades fecha-de-nacimiento y fecha-de-boda se definen a través del marco clase Fecha.
- Un marco instancia puede rellenar o no todas las propiedades de instancia definidas en los marcos clase con los que está conectado. Por ejemplo, en el marco instancia Juan no se ha rellenado las propiedades fecha-de-boda y tipo-de-matrimonio porque toman los mismos valores que los del marco instancia María.

4.4.1.4 Representación de facetas de propiedades

Las facetas permiten modelar características de las propiedades y relaciones en los marcos clase.

El motor de inferencias usa las facetas para mantener la integridad semántica de los datos, es decir, para comprobar que los valores introducidos en las propiedades realmente pertenecen al tipo especificado, en la obtención de valores, y en la asignación de valores por defecto.

Las facetas se clasifican según las siguientes categorías:

1. Facetas que definen propiedades de clase, de instancia y relaciones
2. Facetas que definen propiedades de clasey relaciones
3. Facetas que definen propiedades de instancia

En cualquier propiedad se debe especificar las facetas que definen propiedades de clase, de instancia y relaciones.

1.- Facetas que definen propiedades de clase, de instancia y relaciones

Tipo ranura: establece el tipo de datos con el que se rellenará la propiedad o relación con unos valores concretos.

- Si se trata de propiedades de clase o de instancia que se rellenan con unos valores pertenecientes a determinados tipos de datos, en esta faceta se especificará el tipo correspondiente. Ej.fig.4.16, en el marco clase Persona, las propiedades nombre, estado-civil, nacionalidad y tipo-dematrimonio se definen como un conjunto de caracteres; la propiedad edad se define como un entero corto; y, la propiedad animal-racional como de tipo lógico.
- Si se trata de propiedades de clase o de instancia definidas como marcos, en esta faceta se especifica, precisamente, que se trata de un marco. Ej.fig.4.16, la definición de las propiedades fecha-de-nacimiento y fecha-de-boda como un marco evitará repetir la descripción del concepto Fecha en cada una de ellas.
- Si se trata de relaciones, la relación se definirá siempre en el marco clase origen de la relación, tendrá como nombre el de la relación, y en esta faceta se deberá especificar que se trata de un marco. Ej.fig.4.16, se tiene las relaciones subclase-de, casado-con.

- Cardinalidad mínima: establece el número mínimo de valores con los que se rellena la ranura, siempre que ésta se rellene.
- Cardinalidad máxima: esta faceta informa del número máximo de valores con los que se puede rellenar la ranura.
- Multivaluada: establece si la propiedad puede tener más de un valor o no. Si la cardinalidad mínima es igual a la máxima, y ambas son iguales a uno, entonces la propiedad no es multivaluada. En caso contrario entonces la ranura sí es multivaluada. Ej.fig.4.16, de Persona, las únicas propiedades multivaluadas son las de nacionalidad y tipo-de-matrimonio. El resto de las propiedades no son multivaluadas; y se pueden denominar simples.

2.- Existe una faceta llamada propiedad general, común a las **propiedades de clase y relaciones**, que se rellena con los siguientes valores:

- Las propiedades de clase definidas en la faceta *tipo ranura* como un tipo de datos la rellenan con los valores que toma la propiedad en el marco clase.
- Las propiedades de clase definidas como marcos y relaciones rellenan esta faceta con un puntero a un marco clase.
- Las propiedades de instancia no la rellenan, esto se indica colocando el símbolo "_ _".

3.- Para cada una de las **propiedades de instancia** definidas en un marco clase, se definirán las correspondientes facetas que definen propiedades de instancia y que se utilizan para modificar, añadir o borrar un valor de una propiedad:

- Valores Permitidos. especifica el conjunto de valores válidos que puede tomar la propiedad de instancia. Concretamente, se utilizará: un tipo de datos, un rango de valores, o un puntero a un marco clase. Independientemente se debe comprobar que el conjunto de valores especificados es consistente con el tipo almacenado en la faceta tipo ranura. Los rangos de valores se utilizan para restringir el número de posibles valores con los que se rellena la propiedad en la instancia. Es preferible un rango de valores frente a un tipo de datos.
- Valores por Omisión. Define los valores que debe tomar la propiedad de instancia en un marco instancia si no se conoce de forma explícita otro valor. Es aconsejable rellenar las propiedades de instancia que no tienen asociados valores por omisión, las propiedades de clase y las relaciones con el símbolo "_ _". El valor asignado en esta faceta será siempre del tipo especificado en la faceta tipo ranura.
- Si Necesito. almacena un procedimiento o regla que se ejecuta al solicitar el valor de una propiedad de instancia en un marco instancia y ser desconocido dicho valor. Se utilizan para:

  - *Requerir el valor de una propiedad*. Si el valor de una propiedad de instancia en un marco instancia es desconocido, los procedimientos pueden preguntar dicho valor al usuario del sistema, o bien, pueden calcular el valor a partir de otros valores conocidos almacenados en la BC.

  - *Mantener la integridad semántica de la BC*. Cada vez que el usuario introduzca un valor en alguna propiedad de una instancia, el SBM comprobará que se cumplen las restricciones impuestas en las facetas de dichas propiedades, y sólo si los valores son correctos se aceptan.

  - *Gestionar dinámicamente de valores*. Los procedimientos almacenados en la faceta si necesito también se utilizan para obtener dinámicamente el valor de una propiedad de un marco instancia si el conjunto de valores con los que se rellena la propiedad en la instancia

se puede obtener a partir de los valores almacenados en otras propiedades y relaciones.

- *Determinar dinámicamente el rango de valores*. Si el conjunto de valores con los que se rellena una propiedad en el marco instancia dependen de los valores que se han rellenado en otras propiedades y relaciones del mismo o de otros marcos instancia, dinámicamente se puede determinar el rango de valores permitidos.

- Si Modifico. almacena el procedimiento que se ejecuta al modificar un valor de una propiedad de un marco instancia. La ejecución de este procedimiento puede añadir, modificar y borrar valores de otras ranuras, disparando así sus procedimientos asociados.
- Si Añado. almacenan procedimientos que se ejecutan al introducir un valor en una propiedad de un marco instancia que estaba vacía. Se utiliza para mantener la integridad semántica de la BC, y añadir, modificar y borrar valores de otras ranuras.
- Si Borro. Esta faceta almacena el procedimiento que se ejecuta al borrar un valor en una propiedad de un marco instancia. La ejecución de este procedimiento puede añadir, modificar y borrar valores en otras ranuras, disparando de este modo sus procedimientos asociados.

4.4.2 Criterios de diseño

Se indican algunos criterios de diseño para representar el conocimiento utilizando el formalismo de marcos:

- Se debe favorecer la compartición de propiedades de clase y de instancia entre marcos, también denominado herencia.
- Se debe evitar representar conocimientos redundantes.
- En cada marco clase debe haber una propiedad de clase que identifique a los elementos de dicha clase.
- Semántica de las propiedades; las propiedades deben tener o aportar significado a la representación.
- Debido al carácter local de las propiedades, se pueden tener propiedades repetidas con el mismo nombre en diferentes marcos clase.

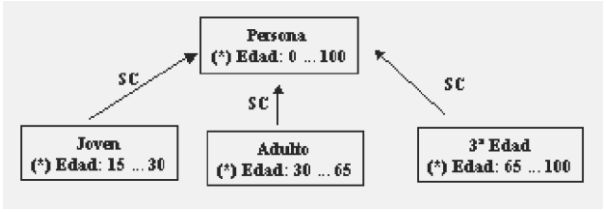


Figura 4.17: Ejemplo de redefinición de propiedades.

- En caso necesario, se pueden redefinir las propiedades (de clase e instancia) en marcos clase más específicos (tal y como muestra el ejemplo de la Figura 4.17).

- Si se desea conocer información o propiedades de las propiedades de un marco, entonces dichas propiedades deben ser definidas como marcos
- No se deben mezclar conceptos.
- Hay que tener en cuenta que en un marco instancia se pueden rellenar, o no, todas las propiedades de instancia definidas en los marcos clase con los que está conectado.
- Hay que recordar que las propiedades de instancia se rellenan con un valor concreto.
- No se pueden utilizar en las instancias propiedades que no se hayan definido en los marcos clase.

4.5 Inferencia de conocimiento en SBM

El formalismo de marcos permite realizar inferencias utilizando los conocimientos almacenados en la BC. Las tres técnicas que utilizan los marcos son:

- La equiparación para clasificar entidades en la BC
- La herencia de propiedades para compartir propiedades entre marcos
- Los valores activos (demonios) para representar la conducta del sistema y mantener la integridad de los datos almacenados.

4.5.1 Equiparación

Equiparar significa clasificar. En este sentido, conocidos los valores de un conjunto de propiedades que describen parcialmente una nueva entidad o marco pregunta, la técnica de equiparación clasifica el marco pregunta en el árbol o en el grafo que representa los conceptos del dominio. Para ello, la técnica de equiparación tiene que encontrar los marcos clase (uno o varios) de la BC que describen más consistentemente al marco pregunta. Encontrados, el marco pregunta se convierte en un marco instancia de dichos marcos clase.

Ocurre que si los valores de todas las propiedades de una entidad son conocidos y dichas propiedades se encuentran almacenadas en un único marco clase de la BC, entonces la equiparación de la nueva entidad con el marco clase es perfecta.

La técnica de equiparación se descompone en tres etapas:

1. La selección de marcos candidatos se realizará siguiendo alguno de los siguientes procedimientos:

- Si el tipo de una nueva entidad es conocido, se puede seleccionar el marco clase en el que se ha definido el tipo, y todos los marcos clase en los que éste se ha especializado.
- Si el tipo de la nueva entidad es desconocido, la selección de marcos clase se realiza arbitrariamente, o se eligen aquellos marcos clase en los que, como mínimo, se encuentre definida una propiedad conocida en el marco pregunta. En este caso, los marcos clase que no tengan al menos una propiedad en común con el marco pregunta no se seleccionan.

2. Seleccionados los marcos clase candidatos, se pasa a calcular el valor de equiparación (VE) de la nueva entidad en cada una de las clases seleccionadas. El VE es una medida que informa del grado de idoneidad de la equiparación que se va a realizar.

3. Se decide con qué marcos clase se equiparará la nueva entidad. Si el VE es lo suficientemente alto y el marco es lo suficientemente específico, entonces el sistema no buscará otros marcos e instanciará la nueva entidad convirtiéndola en un marco instancia. Sin embargo, si el VE no es lo suficientemente alto, o el marco no es lo suficientemente específico, entonces la técnica tendrá que identificar marcos relevantes. Esto se puede realizar de diferentes maneras:

- Ascender a lo largo de la jerarquía hasta encontrar una equiparación relevante.
- Comenzar en el marco raíz y seleccionar el marco con el mejor VE
- Buscar marcos relacionados utilizando las relaciones *fraternal, disjunto, no-disjunto* y relaciones *ad-hoc*.

4.5.2 Herencia de propiedades

Esta técnica permite compartir valores y definiciones de propiedades entre marcos de una BC usando las relaciones *instancia* y *subclase-de*.

Se puede diferenciar entre:

- herencia simple, que se aplica a BC en forma de árbol
- herencia múltiple, que se aplica a BC en forma de grafo.

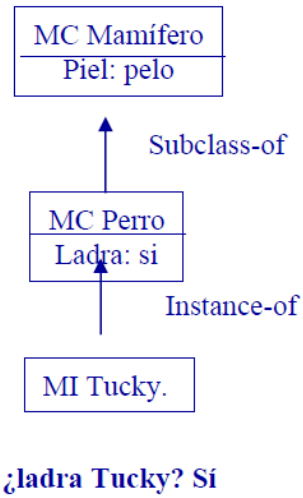
Herencia simple. Cuando el sistema tiene forma de árbol, considerando exclusivamente las relaciones *Subclase-de* e *Instancia*. Es decir, sólo existe un único camino que une el marco instancia con el nodo raíz de la jerarquía.

El algoritmo que realiza la inferencia para encontrar los valores de una cierta propiedad de un marco instancia es el siguiente:

1. Se busca la propiedad en el marco instancia. Si se encuentra, se devuelven sus valores. En caso contrario, se accede al marco clase padre utilizando la relación instancia.

2. Se busca la propiedad en el marco clase. Si se encuentra la propiedad, entonces se devuelven sus valores y el algoritmo finaliza. En otro caso, se utiliza la relación subclase-de para acceder al marco clase padre. Este paso se repite mientras el padre no sea el marco raíz del árbol.

3. Se busca la propiedad en el marco raíz. Si se encuentra, se devuelven sus valores. Si no, la técnica debe



responder que con los conocimientos almacenados en la BC es imposible proporcionar una respuesta.

La ejecución de un procedimiento asociado a la faceta *Si Necesito*, o la herencia de un valor por omisión, permitirán, en última instancia, responder a la pregunta. En el caso de estar definidas ambas facetas, debe existir una estrategia de control que ayude al SBM a decidir si ejecuta un procedimiento o si toma el valor por omisión.

• **Herencia múltiple**. Cuando el sistema de marcos tiene forma de grafo, bien porque un marco instancia es instancia de varios marcos clase, o bien porque cada marco clase es especialización de una o varias clases, puede ocurrir que un hijo puede heredar propiedades de varios padres, la dificultad se encuentra en determinar el orden en el cual las clases se deben explorar.

- Si los padres tienen propiedades con distinto nombre nunca habrá conflicto.
- Si los padres tienen propiedades con el mismo nombre, el valor de la propiedad que hereda el marco hijo dependerá de la técnica de búsqueda que se utilice para recorrer el grafo.

Por consiguiente, existen diferentes algoritmos que recorren de forma diferente el grafo, y el IC deberá elegir uno de ellos:

- Búsqueda en profundidad; consiste en explorar en profundidad todos los posibles caminos que parten del marco instancia y acaban en el marco raíz. Para recorrer estos marcos se puede seguir alguno de los siguientes criterios:

- Recorrer el grafo en un determinado sentido (de izquierda a derecha o viceversa)
- Usar el criterio de exhaustividad para evitar la búsqueda reiterada de propiedades en marcos clase ya analizados, es decir, solamente se buscará la propiedad en cada marco una vez.
- Las instancias pueden heredar valores de clases generales en vez de clases específicas. El criterio de especificidad impone la restricción de que sólo se puede buscar la propiedad en una clase si previamente se ha buscado en todas sus subclases.

Un algoritmo que cumple los tres requisitos es el procedimiento de ordenación topológica. Dada una distancia, este procedimiento utiliza la topología del grafo para formar su lista de criterios de preferencias. Esta lista almacena el orden en el que se recorrerán todos los marcos clase accesibles desde un marco instancia concreto.

- Búsqueda en anchura; consiste en recorrer el grafo por niveles que estén a igual distancia del marco instancia.

Dado un marco instancia, se mirará si la propiedad se encuentra en alguno de los marcos clase con los que está unido. Si se encuentra, entonces se devuelve su valor. En caso contrario, utilizando las relaciones *subclase-de*, comienza a buscar

la propiedad en todas las superclases de dichas clases y así sucesivamente, terminando al encontrar la propiedad o al alcanzar el nodo raíz sin encontrarla.

El problema de este método es cómo resolver las ambigüedades que surgen cuando existen al menos dos clases a un mismo nivel desde las que heredar la propiedad. Es importante en este método que el IC se asegure de que en todo el grafo el nivel de profundidad de las clases se corresponda con su grado de especialización; en caso contrario, se podrían heredar propiedades más generales situadas en el mismo nivel que propiedades idénticas más específicas, lo cual no es deseable.

– La distancia "inferencial"; consiste en aplicar el concepto de *distancia inferencial de Touretzky* para determinar qué propiedades se heredan. Este método detecta situaciones ambiguas, pero no permite resolverlas. El concepto de distancia inferencial se basa en el siguiente razonamiento: "la condición necesaria y suficiente para que la clase1 esté más cercana a la clase2 que a la clase3 es que exista un camino desde clase1 hacia la clase3 a través de la clase2". Al no permitir comparar marcos no conectados, el orden introducido por el concepto de distancia inferencial es un orden parcial en el sistema de marcos y, por tanto, se pueden heredar valores contradictorios definidos en ramas que no están conectadas

4.5.3 Valores activos (demonios)

Además de la estructura declarativa de los marcos, también existe un aspecto dinámico o procedimental de los mismos. Con los nombres de demonios, valores activos o disparadores se denominan a los procedimientos que recuperan, almacenan, y borran información en los SBM.

Estos procedimientos se definen en las facetas *Si Necesito, Si Añado, Si Modifico* y *Si Borro* de las propiedades de instancia de los marcos clase.

La ejecución de estos procedimientos presenta las siguientes características:

- Los procedimientos se definen en el marco clase y permanecen latentes, sin hacer nada, a menos que, por algún motivo, se solicite su ejecución desde un marco instancia. Se puede solicitar la ejecución de un procedimiento para:
    - Recuperar, almacenar, o borrar valores de propiedades en marcos instancia
    - Mantener la integridad semántica de la BC al impedir que se introduzcan valores en la BC que no satisfacen unas determinadas restricciones.
    - Garantizar que cambios en los valores de una propiedad se reflejan correcta y automáticamente en los valores de otras propiedades diferentes.

- Calcular dinámicamente valores de propiedades que se requieren y que se obtienen a partir de los valores de otras propiedades almacenadas en la BC.
- Gestionar errores.

- Los procedimientos se despiertan al recibir una solicitud de ejecución. Cuando un marco instancia solicita la ejecución de un valor activo, el procedimiento asociado se ejecuta con los valores almacenados en las propiedades del marco instancia.

- Los procedimientos pueden simular encadenamiento hacia adelante, y así tener demonios dirigidos por los eventos, y encadenamiento hacia atrás, y tener demonios dirigidos por las metas.

  - Los demonios dirigidos por los eventos ejecutan procedimientos antes de almacenar o borrar valores en las propiedades de un marco instancia, y están asociados a las facetas *Si Añado, Si Modifico* y *Si Borro,* y tiene un comportamiento análogo al encadenamiento hacia delante de las reglas.

  - Los demonios dirigidos por las metas tienen lugar cuando se necesita conocer el valor de una propiedad de un marco instancia. Están asociados a la faceta *Si Necesito*, y tiene un comportamiento análogo al encadenamiento hacia delante de las reglas.

- El control va pasando de unas propiedades a otras a medida que se van ejecutando los procedimientos. En cada instante, el control lo tiene la propiedad del procedimiento que se está ejecutando. Una vez ejecutado un procedimiento, el control vuelve al procedimiento que lo llamó, procedimiento que sigue ejecutándose en el mismo lugar en el que se detuvo.

EJEMPLO

MC Persona	Si Necesito	Si Añado	Si Modifico
(*) Edad	$P.edad:= Fecha-$P.Fecha_Nacimiento	Si $P.edad <18 Entonces $P.Mayor-Edad:=no Sino $P.Mayor-Edad :=si	Si $P.edad <18 Entonces $P.Mayor-Edad:=no Sino $P.Mayor-Edad :=si
(*) Mayor de Edad	Si $P.edad <18 Entonces $P.Mayor-Edad:=no Sino $P.Mayor-Edad :=si	Si $P.Mayor-Edad:=no Entonces $P.PuedeVotar:= no Sino $P.PuedeVotar:=si	Si $P.Mayor-Edad:=no Entonces $P.PuedeVotar:= no Sino $P.PuedeVotar:=si
(*) Puede Votar	Preguntar_Usuario ($P.Mayor.edad)		

4.6 Resumen

En líneas generales, la mayoría de los SBC suelen representar los conocimientos del dominio utilizando marcos frente a redes semánticas debido a que:

- Los marcos permiten construir taxonomías de conceptos por especialización de conceptos generales en conceptos más específicos, lo que posibilita la herencia de propiedades. Con la herencia de propiedades se comparten propiedades entre marcos y se evita la presencia de conocimientos redundantes en la BC. Aunque esta característica es común a las redes semánticas, la diferencia se encuentra en que en los marcos las propiedades están recogidas dentro del marco, mientras que en las redes semánticas las propiedades se encuentran dispersas por toda la red uniendo nodos que representan conceptos con otros nodos que representan los valores que toma la propiedad.
- En las BC formalizadas en marcos las propiedades se pueden definir de forma declarativa y procedimental; sin embargo, en las redes semánticas sólo se pueden definir de forma declarativa. Además, la cantidad de información asociada a las propiedades en un marco supera con creces a la de las propiedades en una red semántica. En los marcos se puede introducir información sobre el tipo de datos al que pertenecen los valores, número de valores mínimo y máximo con los que se rellena cada propiedad, valores por omisión, excepciones, y procedimientos y reglas que permiten inferir los valores de dichas propiedades, mientras que en las redes semánticas no se puede.
- Los SBM permiten el uso de valores por omisión y excepciones a dichos valores; sin embargo, en las redes semánticas no se pueden representar valores por omisión, aunque sí excepciones. La propia estructura interna de los marcos permite mantener internamente las restricciones de integridad semántica que existen entre los elementos de un dominio, mientras que en las redes semánticas, no.
- Uno de los principales inconvenientes que presentan las redes semánticas es que a medida que la BC del sistema aumenta, la comprensión de la red se hace cada vez más difícil. No ocurre lo mismo en los SBM, los cuales facilitan el diseño y mantenimiento de la BC.

Apéndice. Glosario de términos de IA

A*, algoritmo

Especialización de la búsqueda primero el mejor, basada en usar una función heurística que estima para cada nodo *n* el coste menor para ir del nodo inicial a un nodo meta pasando por el nodo *n*.

abductiva, inferencia

Razonamiento que deriva explicaciones de los hechos conocidos.

ABIERTA

Lista que contiene los nodos generados pero no expandidos.

admisible, búsqueda

La que siempre encuentra una solución óptima.

anchura iterativa, búsqueda en

La que realiza iterativamente búsquedas primero en anchura aumentando en cada iteración en una unidad el número de sucesores de cada nodo que son generados.

anchura, búsqueda primero en

La que usa ABIERTA como una cola.

aprendizaje

Proceso de modificación de las funciones cognitivas con el fin de adaptar un organismo a su entorno.

árbol semántico, método deductivo del

Método de deducción no determinista de la validez de una fórmula, basado en la reducción al absurdo.

arco descriptivo

Arco de una red semántica que representa un atributo y une el objeto al que hace referencia el atributo con el valor asociado al atributo.

arco estructural

Arco de una red semántica que une dos conceptos y determina la arquitectura de la red.

autoepistémica, lógica

Extensión de la lógica clásica basada en la hipótesis del mundo cerrado.

Aε*, algoritmo

Variante del algoritmo A*, que utiliza una función heurística adicional para elegir el siguiente nodo a expandir de entre un subconjunto de ABIERTA formado por sus elementos más prometedores que difieren del mejor elemento de ABIERTA como máximo en un factor $(1+\varepsilon)$.

basada en intervalos, lógica temporal

Aquella en que el tiempo se define mediante pares de puntos, permitiendo modelar propiedades de cierta duración.

basada en puntos, lógica temporal

Aquella en que el tiempo se considera compuesto por un conjunto de puntos ordenados, permitiendo expresar eventos puntuales, y situaciones pasadas y futuras con respecto al instante presente.

base de Conocimientos

Componente de un Sistema Basado en Reglas que almacena segmentos de conocimiento relacional entre datos y conceptos.

base de Hechos

Memoria de trabajo de un Sistema Basado en Reglas que acumula un conjunto de hechos establecidos que se usan para determinar qué reglas puede aplicar el mecanismo de inferencias.

borrosa, lógica

Extensión de la lógica clásica donde las proposiciones tienen un grado de verdad que se asigna mediante una función de pertenencia que toma valores en el intervalo real [0,1].

búsqueda bidireccional

La que simultanea la búsqueda del estado objetivo desde el estado inicial y viceversa.

Ver reversibilidad.

búsqueda, algoritmo de

Estrategia de control que decide el orden en que se exploran los estados de un espacio de búsqueda.

cibernética

Ciencia que estudia la construcción de sistemas electrónicos y mecánicos a partir de su comparación con los sistemas de comunicación y regulación automática de los seres vivos.

ciclo de reconocimiento-acción

Proceso iterativo de aplicación del encadenamiento hacia delante.

Circunscripción de predicados

Ver autoepistémica, lógica.

clase

Marco que expresa un conocimiento genérico.

cláusula

Fórmula consistente en una disyunción de literales.

Complejidad espacial
Cuantos nodos tengo que recordar para llegar a la solución.

complejidad Temporal
Cuanto tarde en llegar a la solución.

complejidad NP

Denominación de la clase de problemas que pueden ser resueltos en tiempo polinómico por una máquina de Turing no determinista.

completa, búsqueda

La que siempre encuentra solución en el caso de que exista.

conexionista, paradigma de la inteligencia artificial

Aquel en que el conocimiento se representa mediante líneas numéricas etiquetadas para la entrada y salida de una red y la inferencia se realiza mediante un clasificador numérico de naturaleza paramétrica en el que el valor de esos parámetros se ajusta mediante un algoritmo de aprendizaje supervisado o no supervisado.

conjunto conflicto

Conjunto de reglas seleccionadas por un proceso de equiparación.

conocimiento, Ingeniería del

Disciplina de la Inteligencia Artificial cuyo fin es el diseño y desarrollo de Sistemas Expertos (o Sistemas Basados en el Conocimiento).

coste uniforme, búsqueda de

La que expande el nodo cuyo camino al nodo inicial es el de menor coste.

CTL, lógica

Lógica temporal basada en puntos de gran expresividad y alta complejidad computacional, ampliamente aplicada en el campo de control de modelo o ModelChecking.

débil, inteligencia artificial

Postura adoptada sobre los avances de la inteligencia artificial que sostiene que las computadoras sólo pueden aspirar a mimetizar la inteligencia, comportarse como si fueran inteligentes, sin serlo realmente.

Decibilidad
Capacidad de encontrar algo y su contrario. No todos los problemas son decidibles pro cuano susamos lógica y encontramos algo que es verdad y no verdad a la vez, se dice que no es resoluble.

declarativa, representación del conocimiento

Aquella en que está especificado el conocimiento pero no el modo en que debe ser usado.

deductiva, inferencia

Razonamiento que deriva conclusiones a partir de unas premisas.

deliberativo, agente

Aquel que contiene un modelo simbólico del mundo, explícitamente representado, en donde las decisiones se toman utilizando mecanismos de razonamiento lógico basados en la concordancia de patrones y la manipulación simbólica.

demonio

Procedimiento asociado a una propiedad de un marco, que se encarga de recuperar, almacenar o borrar información asociada a dicha propiedad.

difusa, lógica

Ver borrosa, lógica

encadenamiento hacia atrás de reglas

Modo de inferencia de un Sistema Basado en Reglas que parte de un conjunto de hipótesis e intenta verificar estas hipótesis usando datos de la Base de Hechos o datos externos (obtenidos, por ejemplo, del usuario).

encadenamiento hacia delante de reglas

Modo de inferencia de un Sistema Basado en Reglas que partiendo de una colección de hechos o afirmaciones de partida aplica las reglas de la Base de Conocimiento repetidas veces hasta que no se generan nuevos hechos.

equiparación

Método de inferencia en redes semánticas o marcos consistente en la comparación de datos, identificación de aquella red semántica o marco que mejor los contiene y asignación de valores.

Equiparación en Sistemas Basados en Reglas

Selección de reglas compatibles con la Base de Hechos por sus condiciones y acciones.

equivalencia, entre fórmulas

Tiene lugar si y sólo si las fórmulas tienen el mismo conjunto de modelos que las satisfacen.

espacio de búsqueda

Conjunto de estados, operadores aplicables a cada estado y coste de cada operador, que están asociados a un problema de búsqueda.

Expresividad
Un lenguaje es cuanto más expresivo, cuanto más se parezca el lenguaje formal. La lógica es muy poco expresiva, la lógica modal gana algo de expresividad.

faceta

Característica de una propiedad o relación asociada a un marco.

formal, lenguaje

Aquel que se define mediante unas reglas fijas de formación de expresiones y significados (una sintaxis y una semántica formales).

fragmento de lógica proposicional

Subconjunto de la lógica proposicional, que supone restricciones mediante guardias, limitación de aridad de los predicados, o limitación de variables con el fin de combatir la indecibilidad.

fuerte, inteligencia artificial

Postura adoptada sobre los avances de la inteligencia artificial que sostiene que la inteligencia que posee un ser vivo procede de un programa y que es reproducible en todas sus dimensiones mediante un computador.

gradiente, búsqueda del

Tipo de búsqueda local que acepta una solución vecina sólo si es mejor o igual que la actual.

herencia de propiedades

Método de inferencia en redes semánticas o marcos consistente en que los elementos más específicos reciben propiedades y valores de los elementos más genéricos.

heurística

Conjunto de técnicas para la resolución de un problema.

heurística, búsqueda

La que utiliza información del dominio.

heurístico

Criterio para establecer la calidad de un estado en un problema de búsqueda.

heurístico, conocimiento

Aquel conocimiento no riguroso ni consensuado por los expertos de un campo sino basado en heurísticas propias de un experto o grupo de expertos.

híbrido, paradigma de la inteligencia artificial

Aquel que incorpora elementos de distintos paradigmas.

hipótesis del mundo cerrado

Hipótesis asumida en la lógica Autoepistémica o la Circunscripción de predicados, consistente en suponer como falso todo aquello que no esté explícitamente afirmado.

horn, cláusula de

Cláusula consistente en un átomo, una implicación donde el antecedente es una conjunción de literales positivos y el consecuente un solo literal positivo, o bien una implicación donde el antecedente es una conjunción de literales negativos, y el consecuente es vacío.

IDA*, algoritmo

Variante del algoritmo A*, basado en realizar una búsqueda en profundidad iterativa acotada en cada iteración por ciertos valores conservadores de la función heurística.

indexación, técnica de equiparación de

Aquella que consiste en añadir a las reglas nuevas condiciones relacionadas con el punto de inferencia.

inductiva, inferencia

Razonamiento que deriva conclusiones generales a partir de premisas que contienen datos particulares.

Inferencia

Razonamiento, obtención de una conclusión. Mecanismo que a partir del conocimiento nos lleva a la toma de una decisión.

instancia

Marco que constituye un ejemplo concreto de un concepto genérico.

Inteligencia

Medida global de la calidad de todos los procesos cognitivos de un ser vivo y de su capacidad de adaptación a los cambios del medio en el que existen otros seres vivos de complejidad comparable.

intérprete de reglas

Ver Motor de inferencias.

intuicionista, lógica

Extensión de la lógica clásica que redefine el concepto de tautología.

K, lógica

Variante de uso extendido de la lógica modal que utiliza un conjunto simplificado de axiomas.

local, búsqueda

Aquella que persigue generar el estado óptimo a partir de un estado inicial cualquiera al que se realizan pequeños cambios sucesivos.

lógica

Cálculo definido sobre unos símbolos mediante un conjunto de reglas que establecen las inferencias válidas (razonamientos correctos). Una lógica es en matemáticas un lenguaje, donde las reglas de inferencia son equivalentes a reglas sintácticas que determinan la validez de las proposiciones del lenguaje, y una semántica que asigna significados a los símbolos del lenguaje.

lógica proposicional

Es un sistema formal diseñado para analizar ciertos tipos de argumentos. En lógica proposicional, las fórmulas representan proposiciones y las conectivas lógicas son operaciones sobre dichas fórmulas, capaces de formar otras fórmulas de mayor complejidad.

LTL, lógica

Lógica temporal basada en puntos para la cual se utiliza el método deductivo decidible basado en árboles semánticos.

marco

Método de representación del conocimiento e inferencia que se centra en la representación de conceptos referidos a entidades del dominio.

modal, lógica

Extensión de la lógica clásica donde es posible indicar el modo en que es cierta o falsa una proposición (cuándo, dónde, bajo qué condiciones) y con ello expresar los conceptos de necesidad y posibilidad.

modelo

Asignación de valores de verdad a todos los símbolos proposicionles que aparecen en una fórmula lógica.

motor de Inferencias

Mecanismo que implementa la estrategia de control de un Sistema Basado en Reglas, examina la Base de Hechos y determina qué reglas se deben disparar.

multivaluada, lógica

Extensión de la lógica clásica donde se permiten más de dos valores de verdad.

no informada, búsqueda

La que no utiliza información del dominio.

no monótona, lógica

Extensión de la lógica clásica caracterizada por la posibilidad de volver atrás en las conclusiones.

no situado, conocimiento

El asociado a aquellas situaciones en las que la interfaz entre el sistema de inteligencia artificial y el medio es humana, por lo que no hay que preocuparse de sensores y de efectores.

ontología

Formulación de un exhaustivo y riguroso esquema conceptual de un dominio considerado.

orden superior, lógica de

Extensión de la lógica de primer orden que se caracteriza por tener variables relacionales de uno o varios órdenes, todas las cuales pueden cuantificarse.

paradigma, de la inteligencia artificial

Aproximación metodológica a la inteligencia artificial consensuada entre un amplio grupo de profesionales del campo que establece una forma de modelar conocimiento, formalizar los modelos, programar los operadores formales e implementar físicamente el soporte de

esos programas, junto con la hipótesis de partida acerca de qué se entiende por conocimiento. Forma de abordar un problema.

Paradigma Simbólico
Vía descendente y uso del lenguaje natural. Todo el conocimiento para la resolución de un problema se puede representar usando descripciones en lenguaje natural usando un conjunto de "conceptos". Tres tipos de tareas (Análisis, Síntesis y Modificación) Este paradigma se usa en orientaciones a objetos y en sistemas multiagentes.

Paradigma Situado
Basado en conductas. Se basa en el hecho de que toda percepción y toda acción están estructuralmente acopladas, a través de sensores y efectores. Tiene la función de decisión. Autómata finito que asocia, ante un conjunto de percepciones, las acciones correspondientes. Este paradigma se usa en robótica y aplicaciones real-time.

Paradigma conexionista
Está basado en rendes neuronales artificiales (RNA's). Asocia conjuntos observables con conjuntos reducidos de clases. Su arquitectura es modular, en capas. Balance entre datos y conocimiento disponible. Se usa en arcos reflejos/generadores de patrones de respuesta.

Paradigma hibrido
Son la mayoría. Principalmente simbólicos--conexionistas. Analiza exigencias computacionales. Descompone la tarea hasta el nivel de inferencias primitivas. Operacionalizacion efectiva del esquema inferencial con módulos simbólicos y neuronales.

ponderación dinámica, algoritmo de

Variante del algoritmo A*, que utiliza pesos dinámicos en la definición de la función heurística.

por defecto, lógica

Extensión de la lógica clásica donde es posible establecer conclusiones y excepciones por defecto.

predicados, lógica de

Extensión de la lógica proposicional en la cual se introducen variables para denotar elementos del dominio, cuantificadores y predicados.

primer orden, lógica de

Ver Predicados, lógica de.

primero el mejor, búsqueda

Especialización del algoritmo general de búsqueda en grafos, en la que se usa una función heurística numérica cualquiera para ordenar ABIERTA.

procedimental, representación del conocimiento

Aquella que incluye la información de control necesaria para utilizar el conocimiento.

proceso cognitivo

Proceso psicológico de orden superior que implica principalmente a las funciones de percepción, memoria, razonamiento y aprendizaje.

profundidad iterativa, búsqueda en

La que realiza iterativamente búsquedas primero en profundidad aumentando en cada iteración en una unidad la profundidad límite.

profundidad, búsqueda primero en

La que usa ABIERTA como una pila.

progresivo, razonamiento

Ver Encadenamiento hacia delante de reglas.

propiedad

Atributo asociado a una característica genérica de un marco, que ayuda a describir el concepto representado por dicho marco.

proposición

Expresión lógica que puede ser falsa o verdadera.

ramificación y poda, algoritmo de

Aquel que interpreta cada estado como un subconjunto de soluciones del problema original, ramifica dichas soluciones en un árbol, asocia un valor heurístico a cada nodo del árbol que representa una cota inferior del coste de la mejor solución contenida en el nodo y poda aquellos nodos que se sabe que no pueden mejorar la mejor solución encontrada hasta el momento.

reactivo, agente

Aquel que no incluye un modelo simbólico del mundo, actuando en base a un patrón estímulo-respuesta.

red semántica

Método de representación del conocimiento e inferencia que se centra en la representación de relaciones entre entidades del dominio.

regla

Par que consta de un antecedente o condición, y un consecuente o acción. Equivale a uncondicional IF-THEN de los lenguajes de programación.

regresivo, razonamiento

Ver encadenamiento hacia a tras de reglas.

relacional, paradigma de la inteligencia artificial

Ver simbólico, paradigma de la inteligencia artificial.

relaciones de Allen o HS, lógica de las

Lógica temporal basada en intervalos de alta capacidad expresiva y aplicabilidad en problemas de tiempo real.

reorientación

Cambio del enlace que parte de un nodo hacia aquel padre suyo que está contenido en el camino parcial más corto desde el nodo al nodo inicial, debido a que se ha encontrado un nuevo padre que acorta el camino hacia el nodo inicial.

resolución de conflictos

Selección de una regla de entre las resultantes de un proceso de equiparación.

resolución, método de

Método deductivo basado en averiguar la satisfacibilidad de un conjunto de fórmulas.

Rete, algoritmo de equiparación

Técnica de aceleración del proceso de equiparación ampliamente utilizada que aumenta la velocidad de procesamiento a costa de consumir ingentes cantidades de memoria.

reversibilidad

Modo de inferencia mixto que combina encadenamiento de reglas hacia adelante y hacia atrás.

semántica

Correspondencia entre expresiones de símbolos y entidades de un mundo físico o abstracto.

semántico, método deductivo

Aquel que busca un contraejemplo para una fórmula, intentando demostrar que la fórmula no es satisfacible.

simbólico, paradigma de la inteligencia artificial

Aquel enque se considera que todo el conocimiento necesario para resolver una tareapuede representarse usando descripciones declarativas y explícitas en lenguaje natural formada por un conjunto deconceptos, los hechos, y otro conjunto de reglas de inferencia que describenlas relaciones estáticas y dinámicas conocidas entre esos hechos.

símbolo

Representación de una entidad de un mundo físico abstracto.

sintáctico, método deductivo

Aquel que busca una demostración final de la validez de una fórmula a través de reglas de deducción.

sintaxis

Conjunto de reglas de una gramática que define las combinaciones válidas de símbolos.

sistema basado en reglas

Aplicación de los sistemas de deducción en lógica proposicional restringidos a cláusulas de Horn que utiliza reglas de inferencia de la lógica para obtener conclusiones lógicas. Tiene una base de conocimiento con reglas y algún mecanismo de inferencias que selecciona las reglas que se pueden aplicar y las ejecuta con el objetivo de obtener alguna conclusión.

Sistemas expertos

Emulan el razonamiento de un experto en un tema. Se trata de una aplicación que, en sobre una base de conocimientos, domina un tema específico.

SMA*, algoritmo

Variante del algoritmo A*, que utiliza memoria limitada y descarta el nodo menos prometedor de ABIERTA cuando se necesita liberar memoria.

TABLA_A

Tabla que permite obtener el mejor camino parcial encontrado desde cada nodo al nodo inicial.

tableaux

Ver árbol semántico, método deductivo del.

tablero semántico

Ver árbol semántico, método deductivo del.

tabú, búsqueda

Tipo de búsqueda local que descarta ciertos vecinos del nodo actual a partir del historial reciente de la búsqueda.

tautología

Fórmula que independientemente del modelo en que se evalúe siempre es verdadera.

temple simulado

Tipo de búsqueda local que realiza una elección aleatoria entre los vecinos del nodo actual y puede llegar a elegir un vecino peor con cierta probabilidad que decrece progresivamente a lo largo del proceso de búsqueda.

temporal, lógica

Interpretación de una lógica modal que permite utilizarla para modelar el tiempo sustituyendo los mundos posibles por instantes de tiempo y la accesibilidad entre mundos por la sucesión temporal, y convirtiendo los operadores modales en operadores temporales.

voraz, algoritmo

Aquel que toma decisiones irrevocables en la exploración y no considera alternativas al camino actual.

Alfabeto griego

Α	α	alfa	Ν	ν	ni
Β	β	beta	Ξ	ξ	xi
Γ	γ	gamma	Ο	ο	ómicron
Δ	δ	delta	Π	π	pi
Ε	ε	épsilon	Ρ	ρ	ro
Ζ	ζ	dseta	Σ	σ	sigma
Η	η	eta	Τ	τ	tau
Θ	θ	zeta	Υ	υ	ípsilon
Ι	ι	iota	Φ	φ	fi
Κ	κ	kappa	Χ	χ	ji
Λ	λ	lambda	Ψ	ψ	psi
Μ	μ	mi	Ω	ω	omega