La IA tiene dos concepciones claramente diferenciadas, con sus propios objetivos, métodos y enfoques. Cabe distinguir:

- **IA como Ciencia**: una tarea de análisis. Su ámbito de conocimiento engloba el conjunto de hechos asociados a la neurología y la cognición: la percepción, la memoria, el lenguaje, etc.
  Busca desarrollar una teoría computable del conocimiento humano, es decir, programas que emulen el comportamiento inteligente de los humanos (*hipótesis fuerte*). La *hipótesis débil* persigue el objetivo, más modesto, de desarrollar máquinas con un comportamiento inteligente, sin necesidad de emular el pensamiento humano.
- **IA como Ingeniería**: Dificultades:
  - El objeto formal de la Ingeniería del Conocimiento es el propio conocimiento. 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.
  - No se dispone de una sólida *teoría del conocimiento* (se debe encargar de desarrollarla la IA como ciencia), no tenemos los conocimientos suficientes sobre procesos cognitivos para apoyarse en ellos desde la parte ingenieril.

  En la mayoría de los desarrollos de la IC, Sistemas Basados en Conocimiento (SBCs) se procede sobre los mismos pasos:
  - **Análisis**: Descripción en lenguaje natural del problema, generalmente realizada por un experto.
  - **Diseño**: Modelado de la descripción mediante diferentes paradigmas para obtener un modelo conceptual.
  - **Implementación**: A partir del modelo, con operadores formales (distintas lógicas, RNAs, etc.), se llega a un modelo formal, ya computable por un sistema de cálculo.

ASPECTOS METODOLÓGICOS (PARADIGMAS)

Paradigma: *forma de abordar un problema*, aproximación metodológica a la IA y a la IC consensuada entre un grupo de profesionales del campo que la consideran como la forma normal de hacer ciencia o ingeniería.

PARADIGMA SIMBÓLICO

El conocimiento necesario puede representarse usando descripciones declarativas y explícitas en lenguaje natural (conjunto de hechos, y reglas de inferencia).

Tres tipos de tareas: análisis (disponemos de todas las soluciones y debemos clasificarlas y elegirlas en base a la descripción del problema), síntesis (diseño y construcción con restricciones (los requisitos del problema)) y *modificación (*ajuste de parámetros).

Usamos la *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*.

Aplicaciones en las que disponemos de conocimiento suficiente para especificar reglas inferenciales o para especificar las "meta-reglas" que actualizarán nuestra base del conocimiento.

EL PARADIGMA SITUADO

Toda percepción y acción están estructuralmente acopladas, mediante perceptores y efectores concretos, a un medio externo e interno también concretos.

Hay un lazo de realimentación mediante sus sensores (lo que perciben: roles de entrada) y efectores (sus acciones: roles de salida). El motor de inferencia serán esquemas de asociación pre-calculados o autómatas finitos, que permitirán que el sistema actúe de forma *reactiva*, sin tener que estar deliberando las posibles salidas.

Robótica y en aplicaciones en tiempo real simples: ámbitos donde la interfaz del sistema informático no es humana, sino que intercambia datos con el medio mediante sus sensores y efectores.

Mucha complejidad del sistema: soluciones híbridas, con componentes reactivas (rápidas) y deliberativas (lentas).

EL PARADIGMA CONEXIONISTA

Representa el conocimiento con líneas numéricas etiquetadas para la entrada y salida de una RNA, y la inferencia se resuelve mediante un clasificador numérico parametrizado en el que el valor de los parámetros se ajusta mediante aprendizaje.

Estructura modular, con muchos procesadores elementales (*neuronas*) interconectados, que evalúan una función de cálculo local. Sus características distintivas son:

- **Clasificador numérico adaptativo**: asocia valores de entrada de un conjunto de *observables* con valores de salida de otro conjunto más reducido de *clases*.
- **Fase de análisis de los datos**: el observador externo decide las variables de entrada y salida, el tipo de cálculo local, la estructura interna en capas… para conocimiento externo.
- **Balance entre datos y conocimiento disponible**. Datos etiquetados → aprendizaje supervisado y validación. Sino, no supervisado, para un preproceso de los mismos.

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.

EL PARADIGMA HÍBRIDO

Combinando los datos y el conocimiento disponible con elementos y técnicas de varios paradigmas.
- Exigencias computacionales y el conjunto de recursos disponibles.
- Descomponer la tarea en otras más simples: 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.
- Operacionalización efectiva del esquema: usando módulos simbólicos y neuronales.

TÉCNICAS DE BÚSQUEDA

Un **sistema de búsqueda** tiene 2 componentes principales:
- **Espacio de búsqueda:** generalmente representado por un grafo dirigido simple. Está formado por:
  - **Conjunto de estados** (nodos): representa las situaciones por las que se puede pasar durante la búsqueda de la solución del problema (estados objetivos o meta). Suele tener un tamaño muy grande para representarlo de forma explícita en un grafo, por lo que queda definido de forma implícita por el inicial y un conjunto de operadores.
  - **Operadores** (arcos): modelan las acciones elementales que es capaz de realizar el agente sobre el medio. Cada uno tiene un coste asociado (arbitrario o atribuido por la propia naturaleza del problema).
- **Estrategia de control:** responsable de decidir el orden en que se van explorando los estados.
  - **Búsqueda a ciegas** (*no informada*): considerará todos los nodos como igualmente prometedores.
  - **Estrategia heurística** (*informada*): debería llevar a explorar primero los nodos en el mejor camino hacia una solución óptima; pretende llegar a una buena solución del problema, visitando el menor número de nodos posibles.

Completo: siempre encuentra una solución (si existe). **Admisible:** siempre encuentra una solución óptima.

MÉTODOS DE BÚSQUEDA EN ÁRBOLES

Se parte del nodo inicial (raíz) y, desde ahí, el algoritmo va expandiendo nodos. En cada iteración, el nodo que se expande será el primero de una lista de posibles nodos candidatos a ser expandidos (*ABIERTA*), ordenada según la estrategia de control.
Consideramos que n es el número medio de sucesores de un nodo, y p la distancia del nodo inicial a la meta.

BÚSQUEDA PRIMERO EN ANCHURA

Criterio de ordenación: insertar los nuevos nodos que se van generando al final de la lista, cola FIFO.
Completo para un grafo localmente finito con un número de hijos por nodo limitado.
Admisible si el coste de todas las reglas es 1.
Tiempo: $O(n^p)$ **Espacio:** $O(n^p)$

BÚSQUEDA PRIMERO EN PROFUNDIDAD

Criterio de ordenación: insertar los nuevos nodos generados al principio de la lista, pila LIFO.
No es admisible ni completo porque puede derivar hacia una rama infinita y no terminar nunca; por eso se suele establecer una profundidad límite desde la que detener la búsqueda, aunque tampoco garantiza que sea completa.
Tiempo: $O(n^p)$ **Espacio:** O(np), en 1 hilo de ejecución solo almacena los nodos de una rama concreta, no los de todo el árbol.

BÚSQUEDA DE COSTE UNIFORME

Criterio de ordenación: insertar los nodos ordenados por el coste desde el nodo inicial, de menor a mayor. Si todas las reglas tienen el mismo (ej. coste unitario: 1), es idéntica a anchura.
Completo. Admisible. Siempre obtiene la solución de menor coste desde el nodo inicial.
Tiempo: $O(n^p)$ **Espacio:** $O(n^p)$

BÚSQUEDAS EN PROFUNDIDAD Y ANCHURA ITERATIVAS

Criterio de ordenación: limitar la profundidad/anchura y realizar varias iteraciones, incrementándolo en cada una.
Profundidad: resuelve el problema de ramas infinitas -> Admisible
Tiempo: O(n^p) **Espacio:** O(p)

Anchura: posible encontrar una solución que no sea la más próxima al nodo inicial. -> Completo, no admisible.
Tiempo: O(n^p) **Espacio:** O(n^p)

MÉTODOS DE BÚSQUEDA SIN INFORMACIÓN EN GRAFOS

En los grafos es posible encontrar varias veces el mismo nodo durante la búsqueda (como sucesor de 2 o + nodos diferentes). Debe considerar los distintos caminos y registrar solo el mejor hasta el momento. Por ello, puede ser necesario rectificar en la *TABLA_A* el coste desde el estado inicial hasta ese nodo, así como su antecesor.

ALGORITMO GENERAL DE BÚSQUEDA EN GRAFOS (AGBG)

Grafo dirigido simple definido implícitamente a partir de un nodo inicial y una serie de operadores (con costes no negativos).
Localmente finito (número de sucesores por nodo acotados) con uno o varios estados solución.
El objetivo es encontrar el camino de coste mínimo desde el nodo inicial hasta los estados objetivo.
Al expandir un nodo, hay que comprobar si alguno de sus sucesores ya se ha expandido, para ver si tiene un camino mejor mediante el nodo que se acaba de expandir: se aplica la función *Rectificar* al nodo sucesor y recursivamente a todos sus sucesores. Así se registra en la *TABLA_A* el mejor camino desde el nodo inicial hasta el momento para los nodos encontrados durante la búsqueda en el punto actual.
Estrategia de control: definida por como ordena los nodos candidatos (*ABIERTA:* profundidad, anchura o coste uniforme).

BÚSQUEDA BIDIRECCIONAL

Busca simultáneamente en dos direcciones: hacia delante desde el nodo inicial a los nodos objetivo, y hacia atrás, desde los nodos objetivo hasta el nodo inicial. Deteniéndose cuando las listas *ABIERTA* de ambos recorridos tengan algún nodo en común. Reduciendo la profundidad a *d/2*. Sin embargo, hay varios factores que condicionan esta mejora.

- Cuántos y cómo son los estados objetivo: si son muchos, la mejora de eficiencia deja de ser cierta.
- Operadores son bidireccionales, si no, dificulta decidir los sucesores de un estado en el recorrido hacia atrás.
- Decidir el algoritmo para cada dirección: Completo: si al menos uno es en anchura (en memoria los estados visitados).
- Admisible: si la búsqueda se realiza desde todos los nodos objetivo.

TÉCNICAS BASADAS EN BÚSQUEDAS HEURÍSTICAS (CON INFORMACIÓN)

Mecanismos ***heurísticos**:* permiten dirigir la búsqueda hacia las zonas más prometedoras, para poder llegar a la solución sin visitar tantos nodos. Hay que buscar un compromiso entre el coste de control que introducen y el de aplicación de las reglas.

BÚSQUEDA PRIMERO EL MEJOR (BF)

Función heurística de evaluación de los nodos *f*: para cada nodo da un valor numérico indicando lo prometedor que es para ser expandido. Ordena *ABIERTA* en base a *f*, estando los nodos candidatos más prometedores al principio.
Completo para grafos finitos. No admisible: No siempre lleva de forma directa a la mejor solución, pero si generalmente a buenas soluciones expandiendo un número de estados mucho menor que con una elección puramente aleatoria.

EL ALGORITMO A*

- ***g*(n)*** es el coste del camino más corto desde el nodo inicial a *n*.
- ***h*(n)*** es el coste del camino más corto desde *n* al nodo objetivo más cercano.
- **$f^*(n) = g^*(n) + h^*(n)$** es el coste del camino más corto desde el nodo inicial a los objetivos condicionado a pasar por *n*.
- **C*** = $f^*(inicial)$ = $h^*(inicial)$ es el coste de la solución óptima. Todos los nodos que forman parte de esta solución cumplen $f^*(n) = C^*$. (para el resto es mayor)

Para problemas complejos, los valores de estas funciones no se pueden conocer, por lo que se trabaja con aproximaciones:

- ***g(n)*** es el coste del mejor camino desde el inicial al nodo *n* obtenido hasta el momento
- ***h(n)*** es una estimación positiva del valor de $h^*(n)$ tal que $h(n) = 0$ si *n* es un nodo objetivo.
- ***f(n) = g(n) + h(n)*** es una estimación de f^* y el criterio que se utiliza para ordenar la lista *ABIERTA*.

Monótono: $\forall n, n'$ nodos, $h(n) \leq k(n, n') + h(n')$ con $k(n, n')$ el coste mínimo de *n* a *n'* => Si se elige un nodo *n* para su expansión: $g(n) = g^*(n)$, es decir, el camino hasta el momento es óptimo, luego no es necesario rectificar nodos ya expandidos => algoritmo + eficiente.

Completo: para grafos localmente finitos (nº de sucesores acotado).
Admisible: si *h* lo es, es decir, $h(n) < h^*(n)$. <= Monótono
Es posible más eficiencia, a costa de perder la admisibilidad. Existen dos métodos principales:

- **Ajuste de los pesos de *g* y *h*:** se pueden ponderar los sumandos de f: $f_w(n) = (1-w)\cdot g(n) + w\cdot h(n)$, $w \in [0,1]$. El mejor valor de *w* se puede obtener de forma experimental. *w=1 BF, w=1/2 A*, w=0 Coste uniforme*
- **Algoritmos ε-admisibles.** Se crea una sublista *FOCAL* de *ABIERTA* con todos los nodos cuyo valor de f no super (1+ ε)+ min(f(n)) para n en *ABIERTA*. Sobre *FOCAL* se aplica una segunda función heurística par determinar cuál se expande.

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. $O(n^p)$

ALGORITMO IDA* (ITERATIVE DEEPENING A*)

Extiende al de búsqueda en profundidad iterativo: se basa en realizar iteraciones de búsqueda primero en profundidad desde el nodo raíz, aumentando en cada iteración la profundidad límite de dichas búsquedas.
1ª iteración: longitud límite = *f(inicial)*, descartando los nodos cuya estimación *f(n)* la supere.
Si no se ha encontrado una solución: Nueva iteración comenzando la búsqueda desde el principio, con longitud límite el menor valor de *f* de entre todos los nodos descartados en la iteración anterior (aumentando la profundidad).
Los hijos de cada nodo expandido se introducen ordenados en *ABIERTA* según su valor *f*, actuando como una pila. Considerando antes los hijos más prometedores; así, en caso de encontrar una solución, se habrán expandido menos nodos.
Admisible: Si *h* es admisible. Completo.
Tiempo: $O(n^p)$, exponencial a la profundidad límite. **Memoria** O(np) -> ahorro.

ALGORITMO SMA* (SIMPLIFIED MEMORY-BOUNDED A*)

Limita el máximo nº de nodos que se pueden almacenar en la *TABLA_A*. Si se necesita expandir un nodo y no hay espacio suficiente, se elimina el nodo con mayor valor de *f* en *ABIERTA* de ahí y de la *TABLA_A*: *nodo olvidado*. En cada nodo recuerda el mejor *f* de sus hijos (los nodos olvidados), para solo volver a explorar un subárbol descartado si el resto de las posibilidades tienen estimaciones aún peores. Algoritmo adaptativo, capaz de evolucionar según la memoria disponible.
Completo: si la memoria disponible es suficiente para almacenar el camino a la solución menos profunda.
Admisible: si tiene suficiente memoria para almacenar el camino hasta la solución óptima menos profunda; si no, devuelve la mejor solución encontrada con la memoria disponible.

ALGORITMOS VORACES

Decisiones irrevocables, los nodos que han sido descartados no los vuelve a considerar.
No son admisibles, ni suelen ser completos, pero resultan muy eficientes => comunes en aplicaciones de tiempo real.
Diseño simple: Por ejemplo, partiendo del algoritmo A*, si en cada paso consideramos solo el nodo más prometedor en función de *h*, descartando el resto de los sucesores, obtenemos un algoritmo voraz.
No determinista: en cada ejecución puede dar resultados diferentes. Si, además, los empates que se puedan producir en cada paso se resuelven de forma aleatoria (eficiencia media dependerá de *h*).

ALGORITMOS DE RAMIFICACIÓN Y PODA

Cada nodo representa un subconjunto de soluciones del problema original planteado (el nodo raíz representa todas las soluciones posibles y un nodo hoja contiene solo una).
Ramificación: expande un nodo en sus sucesores: descompone el conjunto de soluciones en la unión disjunta de sus subconjuntos.
Un nodo hoja tiene coste concreto y conocido y los intermedios, un valor heurístico que representa una cota inferior del coste de la mejor solución contenida en el nodo.
Poda: Generalmente, cada vez que la cota inferior de un nodo supera el menor de los costes de los nodos hoja encontrados hasta el momento en el resto del árbol, se deja de considerar.
La **eficiencia** dependerá de lo ajustadas que sean las cotas inferiores obtenidas de forma heurística.
Ahorrar espacio en memoria: estrategia de control: en **profundidad**. *ABIERTA:* pila y los nodos ordenados según los valores de sus cotas inferiores (de menor a mayor).
Admisible: si recorre todo el espacio de búsqueda, excepto las partes podadas, hasta que *ABIERTA* se agote.
No Admisible: si la condición de terminación es que pare después de expandir cierto número de nodos. Devuelve la mejor solución encontrada hasta el momento, pero gana eficiencia.

ALGORITMOS DE MEJORA ITERATIVA O BÚSQUEDA LOCAL

El **camino es irrelevante**, el nodo objetivo ya contiene de por si toda la información necesaria.
Empieza en una solución inicial, y en cada iteración calcula un conjunto de *soluciones vecinas* mediante una regla de vecindad.
Cada una de ellas es evaluada, y se selecciona una de ellas con un criterio (normalmente la de menor coste).
Si la solución escogida cumple el *criterio de aceptación* (nte. ser mejor que la solución actual), reemplaza a la actual.
Continúa hasta cumplir el *criterio de finalización* (nte. Llegar a *n* iteraciones o que no haya mejoras en los últimos *n* intentos).

ALGORITMO MÁXIMO GRADIENTE

Criterio de aceptación: la solución vecina encontrada mejor o igual que la actual. Búsqueda simple, puede atascarse por:

- Óptimos locales: desde ellos, todos los vecinos son peores, y el algoritmo finaliza sin encontrar el óptimo global.
- Regiones planas: todos los vecinos tienen el mismo valor que la solución actual, y la búsqueda es aleatoria.
- Crestas: si la pendiente es suave, resulta difícil no desviarse hacia los lados al ascender.

Búsqueda multiarranque: reinicia la búsqueda desde otro punto, probablemente alcanzará un óptimo local distinto y después de varios arranques, una solución aceptable.

TEMPLE SIMULADO

Selecciona aleatoriamente un nodo entre los vecinos del estado actual:

- Si mejora la solución actual, la búsqueda sigue como en el caso de máximo gradiente.
- Si no, existe una cierta probabilidad (dependiente de la temperatura **T** y el incremento de energía **ΔE**, la diferencia entre el coste de la nueva solución y el de la actual) de que dicho nodo sea aceptado, así elude máximos locales, si los hubiera.

Empieza: temperatura con valor alto => mayor probabilidad de aceptar un nodo con solución peor, favoreciendo la exploración del espacio de soluciones.
Según avanza: decrece la temperatura => aumenta la preferencia por soluciones de calidad, convergiendo hacia que siempre mejoren a la actual.
Problema: elección de un enfriamiento adecuado (depende de la naturaleza del problema).
Temperatura alta, plan de enfriamiento lento, mayor probabilidad de llegar a una buena solución y coste temporal.

BÚSQUEDA TABÚ

Lista tabú: memoria en base a la que se evitar la generación de vecinos que conduzcan a soluciones no óptimas o ya revisadas, ahorrando tiempo y mejorando la eficiencia.
Criterio de aspiración: excepciones, para admitir nodos tabú que puedan mejorar la solución actual, expandiéndolos como si no estuviesen en la lista.
El ajuste de los parámetros (tamaño de la lista, criterio para incluir un nodo en ella, definición del criterio de aspiración…) son fuertemente dependientes del problema, por lo que se deben particularizar a cada uno.

REDES SEMÁNTICAS

Formalizar: representar simbólicamente los conocimientos de un dominio utilizando un formalismo de representación de conocimiento (pasar del *modelo conceptual* al *formal*).
Tipos: basados en conceptos (*marcos*), en relaciones (*redes semánticas*) y en acciones.

Redes semánticas: formalismo de representación de conocimiento basado en relaciones entre los conceptos o entidades de un dominio.
Los conocimientos que expresa también se pueden expresar con LP o LPO.
La información se representa en un grafo dirigido formado por un conjunto etiquetado de:

- *Nodos:* representan conceptos e instancias de dichos conceptos.
- *Arcos unidireccionales:* representan relaciones binarias entre conceptos: modela conocimientos relativos a un objeto por pares *atributo-valor*: el nodo origen es el *concepto*, el arco que los une es el *atributo*, y el nodo destino es el *valor*. Categorías:
  - **Descriptivos**: Describen entidades y conceptos.
    Ej. para personas podría ser *Profesión*. O en la img., *Come*, que indica que un *Canario Come Semillas*.
  - **Estructurales**: Enlazan las entidades o conceptos formando la estructura de la red. La semántica de estos arcos es independiente del dominio del problema. Se pueden definir tantas etiquetas como se quiera:
    - **Generalización**: relacionan 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*.
    - **Instancia**: liga un objeto con su tipo genérico, arco *Instancia (ES_UN)*.
    - **Agregación**: liga un objeto con sus componentes, arco *Parte-de (*inverso de *Tiene/n)*.

Predicados de aridad 3 o +: crear un 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.
COMPRA-VENTA(Pepe, Luis, Reloj, 45): *COMPRA_VENTA_1*, instancia de *COMPRA_VENTA*, y creamos:
Arco *Comprador* con valor *Pepe*; Arco *Vendedor* con valor *Luis*; Arco *Objeto* con valor *Reloj;* Arco *Precio* con valor *45*.

REPRESENTACIÓN DE ACCIONES

Basada en la *gramática de casos*. Toda proposición tiene una estructura formada por **1 verbo y 1 o + 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.
- *lugar*: en el que se desarrolla el evento.
- *tiempo*: fecha o momento concreto.
- *sujeto*: entidad que sufre el efecto.

Modalidad: hace referencia a características del verbo (ej. *Tiempo:* pasado, presente o futuro y *Voz:* activa o pasiva).
Se utiliza la información de la gramática de casos para representar afirmaciones que se refieren a acciones y eventos.
Cada nodo *situación* tiene como atributos el conjunto de casos y de modalidades que describen el evento.
Por ejemplo: acción *VER_1* (instancia de *VER*), con atributos de *voz, tiempo, lugar, objeto*, etc.

REPRESENTACIÓN DE CONOCIMIENTO DISJUNTO

Representar entidades del domino disjuntas entre sí: con el arco *Disjunto* de una entidad a otra.
Existe otro formalismo que utiliza arcos especiales:
arco S (subconjunto), *arco SD* (subconjunto disjunto), *arco E* (elemento) y *arco ED* (elemento disjunto).
Entidad *Ser-Vivo*, subconjuntos disjuntos *Plantas* y *Animales*: arco *SD* desde *Plantas* y otro desde *Animales* hacia la *Ser-Vivo*.

INFERENCIA DE CONOCIMIENTO

Para resolver los casos que se planteen en una red semántica se deben utilizar procedimientos que trabajen con la semántica de sus arcos. Tenemos dos técnicas diferentes según su forma de proceder:

EQUIPARACIÓN

Un *apunte* (fragmento de una red) se equipara con la red semántica si puede asociarse con un fragmento de la red semántica:

1. Se construye un apunte que responda a la pregunta que se quiere resolver, formado por conjuntos 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.
2. Se coteja el apunte con la red semántica: Los nodos variables del apunte se ligan a nodos constantes de la red semántica hasta encontrar una equiparación perfecta.
3. La respuesta a la consulta es el fragmento de red semántica con los valores con los que se rellenan los nodos variables.

HERENCIA DE PROPIEDADES

Para evitar la redundancia de propiedades en la base de conocimiento, nodos específicos pueden acceder a las propiedades definidas en otros utilizando los arcos *Instancia* y *Subclase-de*.
Para determinar la veracidad de una sentencia cualquiera sobre un atributo o propiedad de una entidad:

1. Localizar el nodo entidad al que se hace referencia, y comprobar si desde él sale un arco con la etiqueta del atributo.
2. Si no: buscar arcos *Instancia* desde el nodo hacia otro, repitiendo con arcos *instancia/Subclase*-de hacia las entidades superiores más generales. Puede heredarse una propiedad de más de un nodo, predomina la del nodo más cercano.
3. Si se han explorado todas las alternativas y no encuentra la solución: comunicar que, con la información almacenada en la red semántica, no se puede contestar sobre la veracidad o falsedad de la sentencia inicial.

Los principales errores que se suelen cometer utilizando herencia de propiedades 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.

MARCOS.

La técnica de representación del conocimiento + usada cuando se basa en conceptos, expresan un conocimiento declarativo. Organizan los conocimientos del dominio en árboles o grafos, construidos por especialización de conceptos generales.

Los conceptos básicos al formalizar la base de conocimientos son: ***marcos***, para representar conceptos o elementos, ***relaciones***, para expresar dependencias entre conce ptos, ***propiedades***, para describir cada concepto, y ***facetas***, para expresar los valores con los que se puede rellenar cada propiedad.

MARCOS: REPRESENTACIÓN DE CONCEPTOS O ELEMENTOS, 2 TIPOS

- **Clase**: para representar conceptos, clases o situaciones genéricas descritas por un conjunto de propiedades, unas con y otras sin valores asignados, que son comunes al concept o que el marco representa. Ejemplo: marco *Persona*.
- **Instancia**: la representación en el dominio real de una clase determinada. Deben estar relacionados con min. 1 marco clase. Suelen rellenar la mayoría de sus propiedades con valores específicos de la instancia que representan, el resto las hereda de los marcos clase de los que son instancias. Ejemplo: marco *Juan* (instancia del marco clase *Persona*).

RELACIONES: REPRESENTACIÓN DE DEPENDENCIAS ENTRE CONCEPTOS

Representa las relaciones del dominio mediante relaciones entre marco. Existen 2 tipos de relaciones:

- **Estándar**: independientes del dominio. Subtipos:
  - **Subclase-de** (inversa: **superclase-de**): entre marcos clase. Permite construir un SBM (sistema basado en marcos) mediante la especialización de conceptos generales en otros más específicos.
    Herencia múltiple: Pueden llegar y/o partir de un marco clase un número indefinido de relaciones de este tipo.
  - **Instancia** (inversa: **representa**): entre marco instancia y clase. El marco instancia es un elemento del conjunto o clase representado por el marco clase. Puede pertenecer a varios conjuntos simultáneamente, pueden partir tantas relaciones instancia como conceptos describan el marco consistentemente.
- **No estándar**: Algunas herramientas no las implementan (requieren de trucos para representarlas - otro marco).
  - **Fraternal**: dos marcos clase con el mismo marco clase padre.
  - **Disjunto** (inversa: **no-disjunto**): dos marcos clase cuyos conjuntos representados no tienen elementos en común.
  - **Ad-hoc:** relaciones 'a medida' entre conceptos de un dominio. Se debe comprobar previamente:
    - Definida previamente entre dos marcos clase,
    - Los marcos instancia a conectar son instancias de esos marcos clase (la relación *ad-hoc* entre dos instancias es una instancia de la definida a nivel de marco clase).
    - Si son marcos clase de diferentes jerarquías: deben cumplir las restricciones propias de la relación entre jerarquías (ej: jugadores de fútbol y equipos).

PROPIEDADES: PARA DESCRIBIR LOS CONCEPTOS, 2 TIPOS

- De **clase**: representan atributos o características genéricas de un concepto o clase.
  Rellenados en el propio marco clase, siempre toman el mismo valor en todos los elementos o instancias de la clase.
- De **instancia**: se definen en el marco clase y son comunes a todas sus instancias, pero se rellenan en cada marco instancia con valores concretos que dependen del elemento. Precedidas del *símbolo '*'* para distinguirlas.
  En el marco clase se rellenan con el tipo de valor con el que se pueden rellenar en las instancias.

FACETAS: EXPRESAR FORMAS LOS VALORES

Facetas: permiten modelar características de las propiedades y relaciones en los marcos clase. Las usa el motor de inferencia para mantener la integridad semántica de los datos (comprobar que los valores introducidos en las propiedades realmente pertenecen al tipo especificado). 3 categorías, **Facetas que definen propiedades de**:

- **Instancia y relaciones.**
  - **Tipo ranura**: Establece el tipo de datos con el que se rellenará la propiedad o relación. 3 casos diferentes:
    - **Propiedades de clase o instancia que se rellenan con valores***:* se especifica el tipo correspondiente.
    - **Propiedades de clase o instancia definidas como marcos***:* se especifica que se trata de un marco.
    - **Relaciones***:* definidas en el marco clase origen de la relación, tienen como nombre la relación, y se especifica que se trata de un marco.
  - **Cardinalidad mínima**: Establece el número mínimo de valores para rellenar la ranura, siempre que ésta se rellene.
  - **Cardinalidad máxima**: Establece el número máximo de valores con los que se puede rellenar esta ranura.
  - **Multivaluada**: Establece si la propiedad puede tener más de un valor o no.
- **Clase y relaciones.**
  - **Propiedad general**: Almacena los valores que toma una propiedad de clase o una relación.
    Propiedades de clase definidas como marcos y relaciones: la rellenan con un puntero a un marco clase.
    Propiedades de instancia: nunca la rellenan (suelen utilizar "--" para indicarlo).
- **Instancia.**

- **Valores Permitidos**: Especifica el conjunto de valores válidos (consistente con el contenido de la faceta tipo ranura) que puede tomar una propiedad de instancia. Puede almacenar un tipo de datos, un rango de valores o un puntero.
- **Valores por Omisión**: Fija el valor que toma la propiedad de instancia en un marco instancia si no se especifica otro. Puede ser anulado al asignar un nuevo valor.
- **Si Necesito**: Almacena un procedimiento que se ejecuta al solicitar el valor de una propiedad de instancia y ser desconocido dicho valor (el procedimiento puede tomar datos de otras ranuras o del usuario del sistema).
- **Si Modifico**: Almacena un procedimiento que se ejecuta al modificar el valor de una propiedad de instancia. Su ejecución puede afectar a otras ranuras.
- **Si Añado**: Almacena un procedimiento que se ejecuta al introducir un valor en una propiedad de instancia que estaba vacía. Puede afectar a otras ranuras.
- **Si Borro**: Almacena un procedimiento que se ejecuta al borrar el valor de una propiedad de instancia. Puede afectar a otras ranuras.

CRITERIOS DE DISEÑO

- Favorecer la compartición de propiedades de clase y de instancia entre marcos.
- Evitar representar conocimientos redundantes.
- Carácter local de las propiedades => Puede haber propiedades con el mismo nombre en diferentes marcos de clase.
- Se pueden redefinir las propiedades de clase/instancia en marcos clase más específicos.
- Marco instancia: se pueden rellenar, o no, todas las propiedades de instancia definidas en sus marcos clase.
- En las instancias no se pueden utilizar propiedades no definidas en los marcos clase.

INFERENCIA DE CONOCIMIENTO

El formalismo de marcos permite realizar inferencias utilizando 3 técnicas distintas:

EQUIPARACIÓN

Conocidos los valores de un conjunto de propiedades que describen parcialmente una nueva entidad o marco pregunta, clasifica el marco pregunta en el grafo que representa el dominio. Se basa en encontrar los marcos clase de la BC que describen más consistentemente el marco pregunta, para convertirlo en una instancia de dichos marcos.
Útil en SBM que clasifican o sistemas que se enfrentan a situaciones parecidas a otras anteriores. 3 etapas:

1. **Selección de los marcos candidatos**: Según si el tipo es:
   Conocido: seleccionar el marco clase en el que se ha definido, y todos los marcos clase en los que éste se ha especializado.
   Desconocido: selección arbitraria o se eligen los marcos clase en los que esté definida min. 1 propiedad conocida en el marco pregunta.
2. **Cálculo del valor de equiparación** (VE) del marco pregunta en cada uno de los marcos candidatos: Que informa del grado de idoneidad de la equiparación que se va a realizar y su cálculo varía de unas aplicaciones a otras.
3. **Elección de los marcos clase con los que se equiparará la nueva entidad**: Para un marco clase determinado:
   Si el valor VE es suficientemente alto: el sistema no buscará otros marcos e instanciará la nueva entidad convirtiéndola en un marco instancia de dicho marco clase.
   Si no: identifica otros marcos relevantes buscando en el resto de la jerarquía (vertical u horizontalmente).

HERENCIA DE PROPIEDADES

Permite compartir valores y definiciones de propiedades entre marcos de una BC con las relaciones *instancia* y *subclase-de*.

- **Simple**: solo existe 1 camino que une el marco instancia con el nodo raíz de la jerarquía (forma de árbol). Algoritmo para encontrar los valores de una cierta propiedad en un marco instancia:
  1. Buscarla en el **marco instancia**. Si se encuentra: se devuelven sus valores y fin.
     Si no: se accede al marco clase padre utilizando la relación *instancia*.
  2. Buscarla en el **marco clase**. Si se encuentra: se devuelven sus valores y fin.
     Si no: accede por la relación *subclase-de* al marco clase padre, mientras éste no sea el marco raíz del árbol.
  3. Buscarla en el **marco raíz**. Si se encuentra, se devuelven sus valores y fin.
     Si no: debe responder que con la BC actual es imposible responder.
- **Múltiple**: existen varios caminos que unen los marcos instancia con el nodo raíz de la jerarquía (forma de grafo). Algoritmos que lo recorren:
  - **Búsqueda en profundidad**: Explora en profundidad todos los posibles caminos que van desde el marco instancia al marco raíz del SBM. Criterios para el recorrido: recorrer el grafo *de izquierda a derecha*, usar el criterio de *exhaustividad* (solamente se buscará la propiedad en cada marco una vez), usar el criterio de *especifidad* (solo se puede buscar la propiedad en una clase si previamente se ha buscado en todas sus subclases).
  - **Búsqueda en amplitud**: Recorre el grafo por niveles que están a igual distancia del marco instancia:
    Primero se busca la propiedad en los padres del marco instancia, si no la encuentra, se busca en los abuelos, y así sucesivamente. El proceso termina al encontrar la propiedad o alcanzar el nodo raíz sin encontrarla.

- **La distancia 'inferencial'**: *la condición necesaria y suficiente para que la clase$_1$ esté más cercana a la clase$_2$ que a la clase$_3$, es que la clase$_1$ tenga un camino de inferencia a través de la clase$_2$ hacia la clase$_3$*. Es decir, que la clase$_2$ esté entre la clase$_1$ y la clase$_3$. Orden parcial, pueden heredar valores contradictorios definidos en ramas no conectadas.

VALORES ACTIVOS

Demonios o *disparadores*: procedimientos que recuperan, almacenan y borran información en los SBM, definidos en las facetas *Si Necesito, Si Añado, Si Modifico* y *Si Borro* de las propiedades de instancia de los marcos clase. Tipos:

- **Demonios dirigidos por eventos**: ejecutan procedimientos antes de almacenar o borrar valores en las propiedades de un marco instancia – asociados con las facetas *Si Añado, Si Modifico* y *Si Borro*.
- **Demonios dirigidos por metas**: deducen valores de propiedades a partir de valores almacenados en otras propiedades – asociados con la faceta *Si Necesito*.

Se definen en el marco clase y permanecen latentes hasta que se solicite su ejecución desde un marco instancia, momento en que el procedimiento asociado se ejecuta con los valores en las propiedades del marco instancia.
El control de ejecución va pasando de unas propiedades a otras a medida que se van ejecutando los procedimientos y produciéndose llamadas entre los mismos.

LÓGICA

Dos problemas centrales en IA:

- Representación del conocimiento: **expresividad** de los sistemas de representación
- Razonamiento automático: **corrección**, **completitud** y **complejidad** de los métodos de razonamiento

LÓGICA PROPOSICIONAL (LP)

Permite expresar ciertos razonamientos y sus métodos deductivos son particularmente sencillos y aptos para la implementación. Su carácter finito garantiza la decidibilidad del problema de la validez de fórmulas y conjuntos de fórmulas.
Proposición: expresión en lenguaje natural que sólo puede ser falsa o verdadera.
Un modelo M satisface una fórmula φ: M $\models$ p si y sólo si p = V en M; y así con el no y el y/o.
φ y ψ **equivalentes**: tienen el mismo conjunto de modelos que las satisfacen.
Los métodos deductivos, se dividen en dos clases:

1. **Sintácticos**: demostración formal de la validez de una fórmula, a través de reglas de deducción.
2. **Semánticos**: contraejemplo para una fórmula, intentando demostrar que la fórmula no es satisfacible.

Árbol semántico: Las reglas dependen directamente de la semántica de las distintas conectivas: las conjuntivas expanden la rama actual, las disyuntivas abren distintas ramas (ambas dejan el nodo viejo no activo). Si aparecen en un camino una proposición y su negación entonces el camino es cerrado y se marca con * el nodo final. Si todos los caminos son cerrados, la proposición es una contradicción.
Cláusulas de Horn: La complejidad se puede mejorar pagando el precio de tener una menor expresividad realizando una deducción lineal en vez de exponencial. Una cláusula de Horn es: Hechos y reglas ($p \wedge q \wedge r \rightarrow s$) como prolog.
Lógica no monótona: es capaz de volver atrás en sus conclusiones, la verdad de una proposición puede cambiar según vayan apareciendo nuevos axiomas.

Lógica intuicionista: sintaxis de la LPO, con la principal diferencia de que en algunas fórmulas como $\varphi \vee \neg \varphi$ o $\neg \neg \varphi \rightarrow \varphi$ no son tautologías

LÓGICA DE PRIMER ORDEN (LPO)

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 existe y universal para todo.

+expresiva; pero + esfuerzo y complejidad de las fórmulas para expresar situaciones típicas en IA en las que hay que tener en cuenta relaciones con determinadas características.
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. Necesario tener un dominio infinito, pero existen algoritmos para LPO que son correctos y completos, aunque no siempre terminan, problemas semidecidibles. Añadimos las siguientes reglas precisas para los cuantificadores:

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

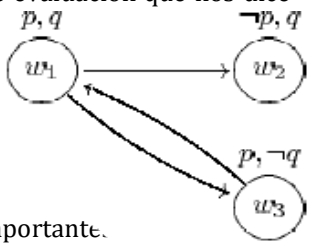
LÓGICAS MODALES

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.

Es posible indicar el modo en que una proposición es cierta o falsa (en qué condiciones) y con ello expresar conceptos de necesidad y posibilidad. 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 y una fórmula es modalmente satisfacible si lo es en al menos un mundo. Extiende con dos operadores:

- □φ: es **necesario** φ: propiedad cierta para todos los mundos alcanzables desde el actual.
- ◊φ: es **posible** φ: propiedad cierta para algún mundo alcanzable desde el actual.

M=(W,R,V): W conjunto de mundos, R relación que permite pasar de uno a otro y V función de evaluación que nos dice cuáles son los símbolos proposicionales que se satisfacen en cada mundo.



Si para todo w Є W se cumple M, w |= Φ, decimos M |= Φ

M,w_1 |= ¬ □p∧◊p ∧ □(p v q)

M |= □ (p v q)

M,w_2 |= □p ∧ □¬ p

Hay tres interpretaciones que se pueden considerar, desde un punto de vista histórico, las más importante.

- ◊ como posible y φ es necesario.
- Lógica epistémica: □φ el agente conoce φ.
- Lógica de la demostrabilidad: □φ φ es demostrable.

La variante más sencilla es la lógica K, cuyo método deductivo es la extensión del método basado en árboles semánticos para la lógica LP.

TIPOS DE LÓGICAS TEMPORALES

Interpretación de una lógica modal para modelar el tiempo: 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.

LÓGICAS TEMPORALES BASADAS EN PUNTOS

El tiempo se considera compuesto por un conjunto de puntos ordenados, permite evaluar eventos puntuales.
LTP[F,P]: sencillo, añade F futuro (F φ), P pasado (P φ): existe un mundo en el futuro/pasado que lo cumpla.
No F no = G, no P no = H: Cualquier momento futuro/pasado.
LTL: añade: S desde, U hasta . + expresivo.
M, t_i ╟ φUψ existe un mundo tj en el futuro tal que M, t_j ╟ ψ y en todo $t \in [t_i, t_j)$ se verifica M, t ╟ φ.
CTL*: Añade A φ: en todo futuro posible se cumple φ y E φ: en algún futuro posible se cumple φ. Muy expresiva, complejidad alta. Existen métodos deductivos correctos y completos. Concepción no lineal del tiempo (+ de un futuro posible).
CTL restricción que obliga a los cuantificadores sobre caminos a ser utilizados al mismo tiempo que los operadores sobre futuro de LTL. Alto poder expresivo, complejidad polinomial.

LÓGICAS TEMPORALES BASADAS EN INTERVALOS

Definidas por pares de puntos, para evaluar propiedades de cierta duración.
Lógica Proposicional de las Relaciones de Allen o HS. Añade <B> Begin y <E> End.

- M, $[d_0, d_1]$ ╟ ⟨B⟩ φ : M, $[d_0, d_2]$ ╟ φ para algún d_2 ,, $d_0 \leq d_2 < d_1$
- M, $[d_0, d_1]$ ╟ ⟨E⟩ φ : M, $[d_2, d_1]$ ╟ φ para algún d_2 ,, $d_0 < d_2 \leq d_1$
- M, $[d_0, d_1]$ ╟ $\langle \bar{B} \rangle$ φ : M, $[d_0, d_2]$ ╟ φ para algún d_2 ,, $d_1 < d_2$
- M, $[d_0, d_1]$ ╟ $\langle \bar{E} \rangle$ φ : M, $[d_2, d_1]$ ╟ φ para algún d_2 ,, $d_2 < d_0$

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.

LÓGICA BORROSA

Lógicas multivaluadas: permiten +2 valores de verdad; un tipo especial es la lógica borrosa donde las proposiciones tienen un grado de verdad que se asigna mediante una función de pertenencia que toma valores en el intervalo real $\mu_P \in [0, 1]$.

SISTEMAS BASADOS EN REGLAS.

Con las reglas se examina un conjunto de datos y solicita nueva información hasta llegar a un diagnóstico.
Ampliadas con el uso de marcos y algunos conceptos de las redes semánticas: método estándar para sistemas expertos.

Sistemas de deducción en LP o LPO, restringidos a cláusulas de Horn en primera instancia. Permiten capturar la experiencia humana en la resolución de problemas, para alcanzar decisiones consistentes y repetibles.
Estructuras de representación declarativas => separación explícita conocimiento del dominio y mecanismos de deducción (controlan las etapas para resolver un problema, aplican el conocimiento para crear una línea de razonamiento).

El conocimiento del dominio puede ser factual o heurístico.

- **Factual**: el conocimiento del dominio aceptado, generalmente consensuado por todos los expertos en un campo específico.
- **Heurístico**: mucho menos riguroso, basado en la experiencia propia de cada experto o grupo de expertos. Componentes:
  - **Interfaz de usuario:** a través del que solicitar u ofrecer información al usuario.
  - **Una base de hechos** (BH): con los hechos conocidos inicialmente y los que se van creando en el proceso de inferencia (Adelante: datos y hechos establecidos hasta el momento. Atrás: metas a alcanzar e hipótesis avanzadas).
  - **Base de conocimiento** (BC): con las reglas para representar el conocimiento disponible de un determinado dominio. Una regla consta de una condición (antecedente) y una acción (consecuente) [antecedente => consecuente].
  - **Motor de inferencias** (MI): algoritmo que examina la BH y decide las reglas que se deben disparar. La condición de finalización indica el hecho meta que debe alcanzarse (es decir, que debe estar contenido en la BH). La búsqueda del conjunto de reglas aplicables (*equiparación*) produce el **Conjunto conflicto** y en la fase de resolución de conflictos selecciona una regla (de forma informada o no) de este conjunto para disparar.

Coste computacional: suma del coste de control y el de aplicación de las reglas. Un diseño eficiente busca equilibrarlos.
Selección completamente desinformada: -de control, + de aplicación (requiere ensayar muchas para encontrar una solución).
Selección muy informada: +coste de espacio y tiempo. - de aplicación de la regla, guía a una solución de forma directa.

INFERENCIA

Establecer la verdad de determinadas conclusiones a partir de la información en la BC y BH.

Encadenamiento hacia delante: ejecuta las reglas cuyo antecedente sea cierto a partir de la información en el sistema. Fase de resolución de conflictos crítica, el disparo de las reglas favorece la explosión combinatoria y se necesita cargar la BH con todas las afirmaciones posibles sin saber si serán útiles o faltan datos.

1. **Equiparación:** determinar, a partir de la BH, qué reglas son aplicables: Son válidas aquellas para las que la equiparación ha tenido éxito. (se crea el conjunto conflicto en esta fase).
2. **Resolución de conflictos**: selecciona a partir del conjunto conflicto la regla que se disparará.
3. **Acción**: aplica la regla sobre la BH, lo cual añade o elimina hechos de la BH.

- Muchas reglas con muchas condiciones en el antecedente.
- No está claro cuáles son los objetivos que alcanzar.
- Problemas que requieren tareas reactivas (como la monitorización).

Encadenamiento hacia atrás: selecciona las reglas cuyo consecuente permita demostrar cierta condición (si no se puede demostrar desde la base de afirmaciones). Las condiciones en los antecedentes pasan a convertirse en nuevos subobjetivos a demostrar, entrando en un proceso recursivo que termina al encontrar la información o cuando se le solicita al usuario. Posibilidad de bucles infinitos, solo plantea cuestiones al usuario cuando ya ha explorado todas las posibilidades de la BC, limita el número de equiparaciones de antecedentes, disminuye la dimensión del árbol de búsqueda.

1. **Equiparación:** búsqueda de las reglas cuya conclusión se corresponda con la meta M en curso.
2. **Resolución de conflictos**: selecciona a partir del conjunto conflicto la regla.
3. **Acción**: reemplaza la meta M por la conjunción de las condiciones del antecedente de la regla seleccionada.

- Muchas reglas cuyo consecuente forma parte del antecedente de otras muchas.
- Objetivos y subobjetivos bien definidos, pero no las estructuras de los datos.
- Problemas que se basan en tareas analíticas, como la clasificación.

Mixto (bidireccional): búsquedas hacia delante y atrás simultaneas. Efectiva para búsquedas completamente desinformadas. Pero puede que no todas las reglas sean aplicables en ambos sentidos o que la de búsqueda hacia delante explore una parte distinta de la de atrás.
Si debe justificar su razonamiento, mejor en la dirección que se adapta más a la manera de pensar del usuario del sistema.

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.
En cada ciclo de inferencias es necesario examinar todas las reglas de la BC en cada ciclo de inferencias, lo cual es poco eficiente, si hay que recorrer toda la BC y ésta contiene muchas reglas. Se puede simplificar mediante 2 técnicas:

- **Indexación**: Añaden a las reglas nuevas condiciones relacionadas con el punto de inferencia, permitiendo dividir el problema en varias etapas y agrupar las reglas en función de en cuál se aplican. Sólo se puede aplicar si las condiciones de las reglas se equiparan exactamente con la BH y se pierde generalidad en la declaración de las reglas.
- **Para acelerar la equiparación, sin necesidad de examinar toda la BC**. El + conocido es el algoritmo RETE.

EL ALGORITMO RETE

Evita examinar todas las reglas de la BC con todos los datos de la BH, explotando dos propiedades de los SBR:

1) **Las condiciones de las reglas contienen muchos patrones similares**. Genera un grafo: *red RETE*, ún̂a red de clasificación estructurada en forma de árbol.

Nodos: se construye un único nodo para cada operación distinta a realizar en la equiparación.
Arcos: une un nodo con todos los que necesitan su resultado, es el camino por el que deben de fluir los datos.

2) **Suele haber pocos cambios en la BH en cada ciclo**. Guarda el resultado de la equiparación durante un ciclo, para que pueda ser usado nuevamente en el ciclo siguiente. Así, 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.

RESOLUCION DE CONFLICTOS:

1. Ordenación arbitraria de reglas en la base de conocimiento.
R11: SI h1 ENTONCES h2, R2: SI h3 ENTONCES h4, BA: {h1, h3} (Base de afirmaciones)
Adelante: R11 y R2 se podrían aplicar. Aplicaría R11 por estar antes en la BC.

2. Ordenación de cláusulas dentro de las reglas según probabilidad de fallo.
R1: SI explosión-solar O guerra-nuclear ENTONCES entrar-en-refugio-subterráneo.
Atrás: objetivo *entrar-en-refugio-subterráneo*, intentamos demostrar 1º *explosión-solar*, cuyo fallo parece más probable, para evitar desperdiciar recursos, ya que anticipamos el posible fallo del antecedente de la regla.

3. Adición de nuevas clausulas en todas las reglas, según etapas.
En un sistema experto agrícola, una cláusula para la estación del año, para definir en cuál es aplicable cada regla.
R1: SI primavera Y abril-lluvioso ENTONCES tarea10
R2: SI invierno Y disponibilidad-económica ENTONCES compra-de-maquinaria2

4. Asignación de prioridades a las reglas y utilización de agendas.
R11: SI h1 ENTONCES h2 (prioridad 50), R2: SI h3 ENTONCES h4 (prioridad 100), BA: {h1, h3}
Adelante: R 11 y R2 en una agenda, que establecería que R2 es la regla a ejecutar, por su prioridad mayor.

5. Utilización de metarreglas.
R1: SI h2 ENTONCES prioridad(R5, 20) <- metarregla que fija la prioridad de R5 a 20, siempre que h2 sea cierto
Permite razonar sobre conceptos asociados a reglas.

6. Mecanismos de refractariedad: impide, por ejemplo, que una regla se ejecute dos veces seguidas.
R1: SI h1 ENTONCES h2, R2: SI h3 ENTONCES h4, BA: {h1, h3}
Adelante: si suponemos que se ha ejecutado primero R1, a continuación, debería ejecutarse R2 con este criterio.

7. Mecanismo de actualidad: prioriza las reglas cuyo antecedente se cumpla con información de un ciclo más actual.
R1: SI h1 ENTONCES h2, R2: SI h3 ENTONCES h4, BA: {h1(2), h3(3)} <- con los hechos se guarda el ciclo en que fue inferido.
Adelante: ejecutar R2 porque h3 es una información más actual que h1.

8. Criterio de especificidad que ejecute primero las reglas más específicas.
R1: SI h1 ENTONCES h2, R2: SI h1 y h3 ENTONCES h4 BA: {h1, h3}
Habrá que ejecutar antes R2, que es más especifica que R1, (el antecedente de R2 incluye al de R1).

CARACTERÍSTICAS

- Son muy modulares (conjunto de reglas independientes) => buena mantenibilidad si su tamaño no es muy grande.
- Sistemas grandes: requieren métodos de estructuración de la BC para facilitar la depuración y evitar efectos colaterales en las fases de actualización y mantenimiento.
- La representación declarativa del conocimiento facilita la incorporación de facilidades de autoexplicación.
- Actualmente las herramientas de desarrollo integran otras técnicas de programación convencionales.
- Tienen menor potencia expresiva que la lógica de predicados (pero más eficiencia computacional), pero mayor que la lógica proposicional; e incorporan aspectos de diferentes extensiones de la lógica clásica.
- Campos de aplicación idóneos: aquellos modelables como un conjunto de estados cuyo conocimiento se puede separar claramente de la forma en que se usa.
- Diferencia importante entre los SBR y la programación imperativa 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.