

# **INTRODUCCIÓN A LA INTELIGENCIA ARTIFICIAL (RESUMEN)**

Autor: Miguel Ángel García Marcos



# TEMA 3

## FUNDAMENTOS Y TÉCNICAS DE BÚSQUEDA BÁSICAS.

### 3.1. Planteamiento del problema.

En la formulación de cualquier problema se parte de la declaración del mismo en lenguaje natural. Esto representa la idea que se tiene de ese problema en el nivel de conocimiento.

A partir de esta descripción “en abstracto” se hacen distintos planteamientos del problema, en los que aparecen elementos concretos que están representados por entidades que el sistema puede manipular directamente.

Principio de racionalidad: - Si el sistema tiene el conocimiento de que una de sus acciones conduce a una de sus metas, entonces realizará esa acción.  
- Se utiliza en el nivel de conocimiento.

La IA estudia métodos que permiten resolver problemas en los que no se conoce un procedimiento sistemático de una solución analítica o “tradicional”. Estos métodos:

- Se consideran débiles, pues realizan búsquedas no informadas (no aprovechan la información disponible).
- Suelen ser poco eficientes.
- Ventaja: aplicables en gran variedad de problemas.

Ante un problema: - Se puede saber de antemano cómo resolverlo, o  
- Se tiene que buscar un método que lo resuelva.

Técnicas declarativas: · Describen los aspectos conocidos del problema. Declaran “el qué”.  
· Claridad y modularidad.  
· Se usan cuando se desea un tratamiento heurístico.

Técnicas procedimentales: · Describen el proceso que hay que seguir para encontrar las soluciones del problema. Indican “el cómo”.  
· Son más fáciles de mantener que las técnicas declarativas.  
· Se usan cuando se desea un tratamiento algorítmico.

En todos los sistemas se utilizan ambas técnicas, y a veces son intercambiables.

Un problema requiere:

- Un agente con unos objetivos que se desean alcanzar.  
Agente es el sistema (programa) concreto que se está ejecutando.
- Conjunto de acciones que permiten obtener los objetivos.
- Un procedimiento de elección entre las diferentes acciones mencionadas.

Para llegar a una solución se necesita una secuencia de acciones.

Solucionador: módulo encargado de construir esa solución.

Por lo tanto se requiere buscar en cada momento la acción adecuada, hasta conformar la secuencia completa de acciones.

En IA, la búsqueda se considera una tarea genérica (TG).

Dados:

- Un conjunto de estados (todas las configuraciones posibles de las situaciones que pueden plantearse en el dominio).
- Uno o más estados iniciales.
- Uno o más estados finales.
- Un conjunto de operadores o reglas que describen las acciones posibles.

Encontrar:

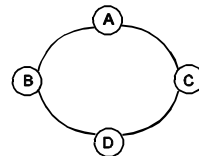
- Una secuencia de operadores que permitan pasar del estado inicial al estado final, o a uno de ellos, si es que hay varios.

Se suele utilizar un grafo para representar un proceso de búsqueda.

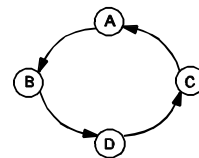
## 3.2. Espacios de representación.

### 3.2.1. Nociones básicas sobre grafos:

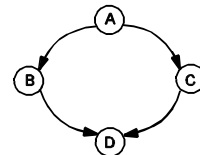
Ciclo: - En un grafo no dirigido: cualquier camino cerrado.



- En un grafo dirigido: si se puede recorrer el camino cerrado en la dirección indicada por los arcos y se vuelve al punto de partida.

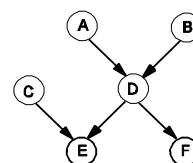


Bucle: En un grafo dirigido: un camino cerrado que no es un ciclo.

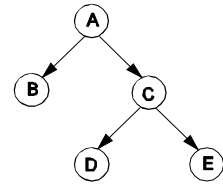


Grafo dirigido acíclico (GDA): es aquel que no contiene ciclos.

Poliárbol:  
- GDA conexo que no contiene bucles.  
- Cada nodo puede tener más de un padre.



- Árbol:
- Grafo que no contiene ciclos ni bucles.
  - Puede ser dirigido o no dirigido.
  - **Cada nodo tiene como máximo un padre.**
  - (Es un caso particular de poliárbol).



Grafo múltiplemente conexo: si hay algún camino cerrado (ciclo o bucle).

### DEFINICIONES:

- Nodos:
- Elementos en el espacio de estados.
  - Representan situaciones válidas en el dominio.
  - Nodo meta: nodo terminal que satisface las condiciones de objetivo.

Expansión de un nodo: - Obtención de todos sus posibles sucesores.

- Para ello se aplican todos los operadores disponibles.

Nodo o estado cerrado: ya ha sido expandido.

Nodo o estado abierto: todavía queda por aplicarle algún operador.

Coste de un arco:

- Refleja el tiempo requerido para aplicar un operador a un estado, durante el proceso de búsqueda.
- Por defecto puede valer 1.

Coste de un nodo:

- Tiempo consumido en alcanzar ese nodo, desde la raíz, a lo largo del mejor camino encontrado hasta un momento dado.

Factor de ramificación: - Número medio de descendientes de un nodo. O lo que es igual:

- Número medio de operadores que se pueden aplicar a un nodo.

Longitud de una trayectoria: - Número de nodos generados en un camino (número de operadores aplicados).

Profundidad:

- Longitud del camino más corto desde el estado inicial a una meta. O lo que es igual:
- Longitud de la secuencia de operadores más corta que resuelve el problema.
- Profundidad del nodo raíz: 0 (se han necesitado cero operadores para obtener el nodo raíz).

### 3.2.2. Representación de los problemas de búsqueda:

Hay dos esquemas generales que resuelven el problema de búsqueda:

- Esquema de producción o búsqueda en el espacio de estados.
- Esquema de reducción: descomposición del problema.

Espacio de representación o espacio del problema:

- Es el entorno en el que se desarrolla el proceso de búsqueda.
- Formado por:
  - Conjunto de estados.
  - Conjunto de operadores.
- Instancia del problema: cuando se indican cuál es el estado inicial y el final.

- Espacio de estados: espacio del problema cuando se usa el esquema de producción.
- Espacio de reducción: espacio del problema cuando se usa el esquema de reducción del problema.

#### Equiparación:

- Comprobación de la aplicabilidad de los operadores disponibles sobre un estado.
- No todos los operadores disponibles se pueden aplicar a todos los estados.
- El concepto de equiparación se usa sólo en el espacio de estados.

#### Estrategia de control (EC):

- Proceso encargado de seleccionar en cada momento el estado que será expandido y el operador que será aplicado.
- Debe conseguir que la solución obtenida sea óptima, teniendo en cuenta los recursos disponibles.

#### Instanciación de un operador:

- Forma concreta de aplicar ese operador a un mismo estado.
- Puede ocurrir que un operador se pueda aplicar de distintas maneras a un mismo estado.

#### Búsqueda necesaria cuando:

- Hay más de un operador aplicable.
- Hay más de una instanciación para un operador concreto.

En el espacio de estados, cuando se aplica un operador a un estado, produce un único estado nuevo.

En el espacio de reducción, cuando se aplica un operador a un estado, se pueden producir más de un estado nuevo.

#### Proceso de búsqueda:

- (1) Seleccionar los nodos del grafo de búsqueda generado.
- (2) Determinar qué reglas son aplicables al nodo elegido en (1).  
Pueden existir distintas formas de aplicación de alguna de las reglas (operadores).
- (3) Seleccionar, de entre las reglas obtenidas en (2), una regla.  
Aplicar esa regla.
- (4) Comprobar que el estado alcanzado tras la aplicación de (3) cumple las condiciones del estado meta.

El conjunto de estados que se obtiene con el proceso de búsqueda se llama grafo de estados: en el caso más sencillo será un árbol.

El espacio de estados contiene todos los estados posibles para un problema.

- Puede resultar infinito o, al menos, intratable. Ejemplo: ajedrez.
- Se trata de una medida implícita de la dimensión (complejidad) del problema.

El grafo de estados contiene sólo aquellos estados que han sido explorados durante el proceso de búsqueda.

- Hace explícita una parte del grafo implícito que supone el espacio de estados.
- Es una medida explícita del proceso realizado.

Complejidad de un cálculo: Cantidad de recursos necesarios para efectuarlo.

- Complejidad temporal:
  - Tiempo necesario para hacer un cálculo.
  - Número de operaciones realizadas.
- Complejidad espacial:
  - Almacenamiento requerido.
  - Cantidad de datos que es necesario recordar en un momento dado.

Base de datos de trabajo (BD): Está formada por el estado inicial y todos los que vayan obteniéndose a partir de él.

### 3.2.3. Cómo limitar el espacio de búsqueda:

Explosión combinatoria: - Cuando, en un espacio de búsqueda, el número de nodos crece exponencialmente.  
- Con técnicas de búsqueda exhaustiva el problema es intratable.

Conocimientos heurísticos: - Reflejan información que depende del dominio.  
- Ayudan a dirigir la búsqueda.  
- Pueden evitar la explosión combinatoria (según qué problema).  
- Producen búsquedas eficientes.

Búsqueda ciega: Genera estados y luego comprueba si cumplen o no las condiciones del objetivo.

Búsqueda heurística: Se basa en el espacio de estados estudiado hasta ese momento, para añadir información que pueda restringir drásticamente la búsqueda.

## 3.3. Búsqueda en integración simbólica.

### 3.3.1. Descripción general del sistema:

A partir de ejemplos de integración propuestos, el sistema debe aprender y refinar las heurísticas que indican cuándo un operador debe aplicarse a un estado de un problema en la búsqueda de la solución.

Para resolver un problema de búsqueda es necesario:

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

Cada operador consta de:

- Dominio de estados: · Son aquellos estados a los que se les puede aplicar dicho operador.
  - Es el espacio de aplicabilidad.
  - Son los estados equiparables con ese operador.
- Regla de reescritura.
- Rango de estados que pueden producirse por aplicación de ese operador.

### 3.3.2. Lenguaje de descripciones:

Definición en extenso del espacio de aplicabilidad: incluye todos los estados a los que se pueda aplicar el operador en cuestión.

Para comprobar de forma eficiente si un determinado estado puede ser objeto de la transformación indicada por un determinado operador, lo que hay que hacer es verificar si la expresión de dicho estado tiene una forma semejante a la de ese operador. Para hacerlo se necesitan:

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

### 3.3.3. Lenguaje de operadores:

En este apartado se describe un ejemplo concreto de lenguaje de operadores. Basta con leerlo rápidamente, ya que es una mera concreción de las partes de un operador, como se describe al final del apartado 3.3.1. de este resumen (página 106).

### 3.3.4. Equiparación de descripciones:

Cuando la precondition asociada a un operador es más general que la descripción de un estado, entonces ese operador se puede aplicar a ese estado.

### 3.3.5. Solucionador:

Es la parte del sistema que realiza la búsqueda de la solución del problema planteado.

El problema se representa con un grafo de búsqueda, que en cada momento contiene el conjunto de todos los nodos que ya se han alcanzado partiendo del nodo raíz.

El procedimiento *SOLUCIONADOR* recibe como argumentos:

- Problema, expresado en el lenguaje adecuado.
- Recursos\_ asignados para su solución. Si se superan, es como si el problema no tuviese solución.

Eficacia: capacidad de realizar una tarea.

Eficiencia: realizar una tarea con el menor coste posible.

La estrategia de control (EC) que se utilice debe tener como objetivo principal la eficiencia.

Recordemos que cada enlace del grafo lleva asociado un coste de cálculo.

Ese coste es un factor de medida necesario para dirigir correctamente el proceso de búsqueda.



Requerimientos que debe tener una EC:

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

### 3.4. Búsqueda sin información del dominio.

También se llaman técnicas de búsqueda ciega, sistemática, exhaustiva u objetiva.

Los principios que cumplen estas estrategias de control son:

- No dejar sin explorar ningún nodo (al menos a priori).
- No explorar más de una vez el mismo nodo.

Nos encontramos básicamente 4 técnicas:

- Búsqueda en amplitud.
- Búsqueda en profundidad.
- Búsqueda con retroceso.
- Otros métodos derivados.

#### 3.4.1. Búsqueda en amplitud:

##### · Descripción:

Con esta EC no se genera ningún nodo de nivel N hasta que se hayan obtenido y explorado todos los nodos pertenecientes al nivel N-1.

Se debe crear una lista de “nodos por expandir”: uso de una cola (FIFO).

##### · Procedimiento Búsqueda en amplitud:

1. Crear una lista de nodos llamada *ABIERTA*.

Inicializar esa lista con un único nodo raíz.

A ese nodo se le habrá asignado el estado inicial del problema planteado.

2. Hasta que (*ABIERTA* esté vacía) o (se encuentre una meta) hacer:

2.1 Extraer el primer nodo de *ABIERTA*. Llamarlo *m*.

2.2 Expandir *m* (crear todos sus sucesores). Forma de hacerlo:

Para (cada operador aplicable) y (cada forma de aplicación) hacer:

(a) Aplicar el operador a *m*.

Se obtiene así un nuevo estado *w*.

Crear un “puntero” que permita saber que el antecesor inmediato de *w* es *m*.

(b) Si (el nuevo estado generado es “meta”) entonces:

- Devolver dicho estado.
- Salir del proceso iterativo inicializado en 2.2.

(c) Si no:

Incluir el nuevo estado al final de *ABIERTA*.

Una vez completado este proceso para todos los sucesores de  $m$  (si no se ha encontrado antes una meta) se continúa el proceso iterativo en el paso 2.

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

· Complejidad:

Temporal: depende de

- Factor de ramificación: supongamos que el número medio de sucesores es  $n$ .

- Profundidad de la solución: supongamos que se alcanza en el nivel  $p$ .

Entonces, el tiempo empleado será  $1+n+n^2+n^3+\dots+n^p$  (hay que recorrer todos los nodos de cada nivel).

Puesto que  $p$  suele ser un valor grande, la complejidad temporal será **del orden  $O(n^p)$** .

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

· Análisis:

Ventajas:

- Este método garantiza que encuentra una solución (si existe).

- Si hay varias soluciones, encuentra la de menor coste (óptima), puesto que encuentra siempre la que requiere menor número de pasos desde la raíz.

Inconvenientes:

- Requiere demasiado espacio de almacenamiento: con casos reales suele ser inviabile.

- En ocasiones requiere expandir muchos nodos inútilmente: si la solución está a poca profundidad, pero hay mucha ramificación para alcanzarla.

### 3.4.2. Búsqueda en profundidad:

· Descripción:

En cada nivel se selecciona un único nodo para ser expandido. Por esto, en cada momento sólo se estará considerando un único camino o rama del árbol.

El funcionamiento responde al modelo de una pila (estructura LIFO).

El elemento de la cima de la pila (último que entró), que es el más profundo que se ha explorado en esa rama, es el único que se considera.

Si una rama puede ser infinita o excesivamente larga, se añade una condición para abandonarla: el límite de profundidad (lp).

Aunque sólo se considera un camino en cada momento, deben obtenerse todos los sucesores de cada nodo explorado, igual que en la búsqueda en amplitud.

Sin embargo, se eliminan aquellos nodos que se consideran “vía muerta”.

· Procedimiento Búsqueda en profundidad:

Crear\_pila(*ABIERTA*)

Añadir\_nodo(*ABIERTA*, raíz)

Hasta que (se encuentre una meta) o (se devuelva fallo) hacer

Si (*ABIERTA* = vacía) entonces

Devolver fallo.

FinSi

$m \leftarrow$  Extraer\_nodo(*ABIERTA*)

Si (Profundidad( $m$ ) < lp) entonces

{Expandir  $m$ . Modo de hacerlo:}

Para (cada operador aplicable) y (cada forma de aplicación) hacer

Aplicar el operador a  $m$

Se obtiene así un nuevo estado  $w$

Crear un puntero que permita saber que el antecesor inmediato de  $w$  es  $m$

Si (el sucesor  $w$  es meta) entonces

Devolver  $w$  {solución}

FinSi

Si (sucesor NO está en un “callejón sin salida”) entonces

Añadir\_nodo(*ABIERTA*,  $w$ )

FinSi

FinPara

FinSi

FinHasta

- El orden en el que se aplican los operadores, para obtener los sucesores de  $m$ , es completamente arbitrario. Esto refleja el carácter “no informado” de este procedimiento.
- El camino completo de la solución se obtiene recorriendo los punteros desde el nodo meta, siguiendo sus antepasados, hasta el nodo raíz.
- En el caso de grafos, para evitar duplicados, hay que comprobar si el nuevo nodo generado ya existía en el grafo de búsqueda. Para reducir la complejidad, sólo se evitan los duplicados en la rama que se está explorando.

· Complejidad:

Temporal: Dado que en las estrategias de búsqueda la complejidad se calcula en el caso peor, ocurre que la complejidad es idéntica a la de la búsqueda en amplitud:  **$O(n^n)$** . Esto es así porque se generarán (en el caso peor) los mismos nodos, aunque en diferente orden.

Espacial: Sólo es necesario guardar el camino que se está explorando en ese momento, y además los sucesores del nodo que está siendo expandido. Siendo

- Factor de ramificación:  $n$
- Profundidad de la solución:  $p$

La complejidad espacial es del orden  $O(n + p)$

· Análisis:

Ventajas:

- Reducido valor de su complejidad espacial.
- La eficiencia del método aumenta cuando hay múltiples soluciones posibles.

Inconvenientes:

- Complejidad temporal.
- Dificultad en establecer un límite de exploración o profundidad ( $lp$ ) adecuado.

### 3.4.3. Búsqueda con retroceso:

· Descripción:

Su principal diferencia, respecto a los dos métodos anteriores, es que ahora sólo se considera un único sucesor por cada nodo, en cada iteración. Así se restringe el espacio de estados considerado.

También en este caso se eliminan del espacio de búsqueda los nodos que son “vías muertas”.

Concretando:

- Se realiza un recorrido en profundidad, pero ahora, cada vez que se selecciona un nodo, si no es una meta ni un “callejón sin salida”, sólo se genera uno de sus sucesores.
- El camino sigue avanzando por ese sucesor.
- Si se encuentra el “callejón sin salida” se retrocede hasta el primer antepasado del que todavía partan caminos no explorados.
- Allí se genera un nuevo sucesor y se continúa avanzando por él de la misma forma.

· Procedimiento Búsqueda con retroceso:

Crear\_pila(*ABIERTA*)

Añadir\_nodo(*ABIERTA*, raíz)

Hasta que (se encuentre una meta) o (se devuelva fallo) hacer

    Si (*ABIERTA* = vacía) entonces

        Devolver fallo

    Fin\_Si

$m \leftarrow$  Extraer\_nodo(*ABIERTA*)   {Desapilar}

    Si (Profundidad( $m$ ) <  $lp$ ) y ( $m$  tiene sucesores sin examinar) entonces

        Aplicar a  $m$  el operador que corresponda. \*

        Se obtiene así un nuevo estado  $w$

        Crear un puntero que permita saber que el antecesor inmediato de  $w$  es  $m$

        Marcar esa rama como explorada.

```

    Si (el sucesor  $w$  es meta) entonces
        Devolver  $w$  {solución}
    Fin_Si
    Si ( $w$  NO está en un “callejón sin salida”) entonces
        Añadir_nodo(ABIERTA,  $w$ )
    Fin_Si
Fin_Si
Fin_Hasta

```

\* Aplicar el operador que corresponda implica que no pueden repetirse operadores en un mismo nodo. Para evitar repeticiones, se pueden adjuntar a  $m$  los operadores que se le van aplicando en cada momento.

· Complejidad:

Temporal: **Orden  $O(n^n)$**

Espacial: Sólo es necesario recordar el camino recorrido hasta un momento dado. Por lo tanto, la complejidad espacial es del **orden  $O(p)$** .

· Análisis:

Ventajas:

- Necesita muy poco espacio de almacenamiento.
- No se genera ninguna rama del árbol de búsqueda que se encuentre después del camino de la solución.

Inconvenientes:

- Resulta complicado establecer el límite de exploración  $l_p$ .
- Al no establecer ningún orden previo de recorrido de los sucesores de un nodo, la eficiencia temporal está limitada por su carácter fortuito.

Otra forma de volver sobre los pasos ya realizados es usar el procedimiento general llamado Búsqueda con Retroceso Dependiente y Dirigido. Consiste en establecer cuándo dos pasos son contradictorios, retrocediéndose entonces hasta el nivel en que pueda ser reemplazada la elección que causó la contradicción.

Hay que señalar que, ni la búsqueda en profundidad, ni la búsqueda con retroceso, están diseñadas para hallar la solución de menor coste (óptima).

### 3.4.4. Otros métodos derivados:

#### A) Búsqueda en profundidad progresiva:

Se trata de una búsqueda en profundidad por niveles, de forma que el límite de exploración aumenta en una unidad por cada nivel alcanzado.

Comienza haciéndose un recorrido en profundidad en el que sólo se incluyen el nodo raíz y los sucesores del

primer nivel.

Si no se encuentra la solución entre esos nodos, se procede a realizar una nueva búsqueda en profundidad. Ahora se incluyen el nodo raíz, los nodos del primer nivel y los nodos del segundo nivel.

Si tampoco se encuentra la solución, se hace una nueva búsqueda en profundidad, incluyendo también los nodos del tercer nivel, y así sucesivamente.

Esta estrategia es la más eficiente dentro de las “no informadas” y que son capaces de encontrar la solución de menor coste (óptima).

Complejidad temporal:  $O(n^n)$

Complejidad espacial:  $O(p)$

#### B) Búsqueda bidireccional (en amplitud):

##### Razonamiento dirigido por los datos o encadenamiento hacia adelante:

- El proceso de búsqueda parte de los datos suministrados (planteamiento del problema) y avanza en el camino de búsqueda hacia un estado meta.
- Espacio de representación: espacio de estados.

##### Razonamiento dirigido por las metas o encadenamiento hacia atrás:

- 1) - Al aplicar un operador al problema, se produce exactamente 1 nuevo estado, que será un problema más sencillo, por lo general.
  - Espacio de estados.
- 1) - Al aplicar un operador el problema se divide en varios subproblemas más simples.
  - Espacio de reducción.

##### · Descripción de la búsqueda bidireccional en amplitud:

Este procedimiento sólo se puede usar cuando se conoce el enunciado del problema y además la solución exacta del mismo, y lo que se buscan son los pasos que permiten llegar del enunciado a la solución.

Necesita dos grafos de búsqueda diferentes.

Los operadores tienen que ser invertibles, o al menos debe ser posible el recorrido hacia atrás.

Se realiza simultáneamente la búsqueda de la meta partiendo del enunciado inicial y la búsqueda del estado inicial a partir de un estado solución.

El proceso termina cuando ambas búsquedas confluyan en algún estado.

Para que la búsqueda bidireccional funcione, al menos una de las dos búsquedas debe ser en amplitud. Aquí se describe con ambas búsquedas en amplitud.

##### · Procedimiento Búsqueda Bidireccional:

1. Inicializar dos grafos de búsqueda: A1 y A2.
  - En A1 el nodo raíz es el estado inicial del problema.
  - En A2 el nodo raíz es el estado meta de ese problema.
2. Inicializar el límite de exploración:  $lp \leftarrow 0$ .

2.1 Actualizar el límite de exploración:  $lp \leftarrow lp + 1$  (el incremento puede ser mayor que 1).

3. Continuar con la exploración de A1 hasta el nivel  $lp$ .

4. Continuar con la exploración de A2 hasta el nivel  $lp$ .

5. Comprobar si alguno de los nuevos estados generados en 3 y en 4 coinciden entre sí.

Si coinciden, entonces devolver la trayectoria de la solución completa (de A1 más A2).

Si no, volver al paso 2.1.

· Complejidad:

Temporal: Dado que cada uno de los procesos de búsqueda sólo tiene que recorrer la mitad del camino, la complejidad temporal es del **orden  $O(n^{p/2})$** .

Espacial:  **$O(n^{p/2})$** .

· Análisis:

Ventajas:

- Reducción de la complejidad temporal y espacial, en comparación con otras técnicas de búsqueda exhaustiva.

Inconvenientes:

- El número de iteraciones antes de cambiar el sentido de la búsqueda es un parámetro crítico: dificultad para ajustar  $lp$ .

- No hay criterio para ordenar la selección de nodos meta a lo largo del proceso.

### 3.5. Reducción del problema.

En el uso del razonamiento dirigido por metas puede darse el caso de que un operador genere más de un problema nuevo. En realidad, en este caso, generaría varios subproblemas, cada uno de ellos más sencillo que el original.

Estos problemas se representan con el esquema reducción del problema.

Cuando un problema puede descomponerse, existirán dos tipos de operadores:

- Operadores que permiten la descomposición.
- Operadores “normales”.

Los distintos subproblemas pueden ser tratados de forma independiente (y simultánea).


La forma de representación usada para resolver estos problemas son, según el problema:

- Grafos Y/O.
- Árboles Y/O.

Árbol Y/O:

- Raíz: contiene el problema completo que intenta resolverse.
- Nodos intermedios: 

Pueden ser:	· Subproblemas en que puede dividirse el problema inicial.
Se dibujan huecos.	· Submetas que deben alcanzarse.

- Nodos terminales:	Pueden ser:	<ul style="list-style-type: none"> <li>· Problema básico (resoluble directamente). Se etiqueta RESUELTO.</li> <li>· Subproblema que no se puede descomponer. Etiqueta IRRESOLUBLE.</li> </ul>
		Se dibujan rellenos.
- Enlaces:	Enlaces “O”	<ul style="list-style-type: none"> <li>· El problema padre puede resolverse de varias formas <u>alternativas</u>.</li> <li>· Todos los enlaces usados hasta ahora eran de este tipo.</li> </ul>
	Enlaces “Y”	<ul style="list-style-type: none"> <li>· El problema padre se descompone en varios <u>subproblemas</u>.</li> <li>· Se deben resolver <u>todos</u> los subproblemas para que el padre quede resuelto.</li> <li>· Se representan mediante una línea arqueada que conecta todos los nodos implicados.</li> </ul>
		
- Solución:		<ul style="list-style-type: none"> <li>· No es un simple camino a lo largo del grafo (árbol).</li> </ul> <p>Se trata de un grafo (árbol) llamado <u>grafo solución</u> (árbol solución).</p> <ul style="list-style-type: none"> <li>· La meta no será un único nodo (como hasta ahora) sino que puede estar formada por varios nodos terminales (si están unidos por un enlace Y).</li> <li>· Ramas discontinuas: no pertenecen al árbol solución.</li> </ul>

#### Aplicaciones de los grafos Y/O:

- Juegos, sobre todo cuando hay dos adversarios:
  - Un oponente elige una de las jugadas posibles.
  - Después deben considerarse todas las jugadas posibles del otro adversario.
- Cuando la solución es un conjunto no ordenado de acciones (pero deben hacerse todas).
- Razonamiento lógico.
- Planificación por etapas, donde no importe el orden en que se ejecuten las mismas.

### 3.6. Algoritmo general de búsqueda en grafos.

Recordemos que, a diferencia de los árboles, en los grafos un nodo puede tener varios antecesores.

Este algoritmo usa dos listas: ABIERTA y CERRADA.

Lista ABIERTA: permite reanudar, en cualquier momento, un camino que había sido abandonado.

Lista CERRADA: almacena los nodos elegidos a lo largo de todo el grafo de búsqueda.

#### · Procedimiento general Búsqueda en grafos:

1. Crear un grafo de exploración G que consista en un único nodo.
- Ese nodo debe contener la descripción del problema: es el estado inicial.
- Crear una lista de nodos llamada *ABIERTA* e inicializarla con dicho nodo.
2. Crear una lista de nodos llamada *CERRADA*. Al principio estará vacía.



```

5      3. Hasta que (se encuentre una meta) o (se devuelva fallo) hacer
6          Si (ABIERTA = vacía) entonces
7              Devolver fallo
8          Fin_Si
9           $m \leftarrow \text{Extraer\_nodo}(\textit{ABIERTA})$ 
10         Añadir_nodo(CERRADA,  $m$ )
11         {Expandir  $m$ . Pasos para hacerlo: }
12              $M \leftarrow$  conjunto de todos los sucesores de  $m$  que no sean antecesores de  $m$ 
13             Introducir todos los elementos de  $M$  en el grafo  $G$ , como sucesores de  $m$ 

14         Si (ningún miembro de  $M$  es meta) entonces
15             Poner un puntero hacia  $m$  desde los nuevos nodos generados {elementos de  $M$ }
16             Incluir los elementos de  $M$  en ABIERTA
17             Para (cada nodo de  $M$  que ya estuviese en ABIERTA o en CERRADA) hacer
18                 ¿Redirigir su puntero hacia  $m$ ?
19             Fin_Para
20             Para (cada nodo  $v$  de  $M$  que ya estuviese en CERRADA) hacer
21                 Para (cada descendiente de  $v$ ) hacer
22                     ¿Redirigir su puntero?
23                 Fin_Para
24             Fin_Para
25         Si_no
26             Devolver solución {El camino se obtiene recorriendo los punteros de sus antepasados}
27         Fin_Si
28         {Final de la expansión de  $m$ }
29         Reordenar ABIERTA con algún criterio previamente establecido.
30     Fin_Hasta

```

· Observaciones:

- La condición “Si (*ABIERTA* = vacía) entonces...” , de la línea 6, garantiza que ningún nodo es antecesor de sí mismo.

Lograr esta condición conlleva un coste muy elevado, pues el grafo de búsqueda puede ser muy grande.

- El criterio que reasigna los punteros (bucles “Para” que comienzan en las líneas 17 y 20 respectivamente) debe intentar que señalen el mejor camino encontrado hasta el momento.

- Nodos abiertos: son los nodos frontera no expandidos.

Nodos cerrados: son todos los nodos expandidos, independientemente de que hayan tenido sucesores o no.

- Si un camino “cerrado” cambia el puntero de su antecesor (bucle “Para” que comienza en la línea 20), el

camino menos costoso puede ser parte del camino menos costoso de alguno de sus sucesores.

- El criterio de ordenación de los nodos en ABIERTA (línea 29) depende del problema concreto que se está intentando resolver.
- La idea fundamental de este método de búsqueda es que resulta importante poder saber si un nuevo estado ya había sido generado y expandido previamente. Así se evita repetir la exploración de algunos caminos. Por eso se usan dos listas.

### 3.7. Discusión.

1. Los elementos básicos presentados en este capítulo NO son suficientes para abordar problemas complejos (reales).

El formalismo de representación limita y determina, en gran parte, los posibles resultados.

2. En este tema se ha considerado siempre que todos los operadores tenían el mismo coste de aplicación. Esta circunstancia sólo suele ocurrir en los problemas “de laboratorio”.

El coste que tenga la aplicación de cada operador puede ser determinante para decidir cuál es el camino de búsqueda más adecuado.

Esta situación refleja la influencia del dominio de aplicación en la resolución de problemas.

### PROBLEMAS:

En el repaso, hacer los tres problemas del libro.

# TEMA 4

## BÚSQUEDA HEURÍSTICA.

### 4.1. Elementos implicados.

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

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

Se distinguen dos dominios de los cuales se puede extraer conocimiento:

- Dominio del observador (DO): La información de este dominio ayuda a resolver dos tipos de problemas:
  - Problemas de solución parcialmente conocida: la experiencia ha establecido parámetros que ayudan a resolver muchos casos (no todos).
  - Problemas intratables de solución conocida: aquí se usa algún criterio obtenido de una simplificación del problema, que sí sea tratable. De nuevo influye la experiencia del experto.
- Dominio propio (DP): La búsqueda heurística se va a aplicar en problemas de los que se sabe que su complejidad temporal es de  $O(a^n)$ , es decir, exponencial. Son los problemas en los que se produce la “explosión combinatoria”.

Los métodos heurísticos no garantizan encontrar la solución óptima, pero obtienen buenas aproximaciones con mucho menos esfuerzo que las búsquedas sin información del dominio.

#### 4.1.1. Conocimiento de control:

Es el conocimiento usado para controlar el funcionamiento del sistema.

Las búsquedas heurísticas pretenden ahorrar tiempo y/o espacio de almacenamiento, pero no eliminando partes del espacio de búsqueda, sino guiando la búsqueda en ese espacio para que sea más eficiente.

Hay dos tipos de técnicas de control:

- Reglas.
- Funciones de evaluación.

#### · Reglas de control heurístico:

Hay dos tipos de conocimiento de control:

##### 1) Conocimiento dependiente del dominio.

Se usa para:

- A) Seleccionar alguna de las situaciones alcanzadas.
- B) Seleccionar la siguiente acción aplicable a la situación seleccionada.

## 2) Conocimiento general sobre el funcionamiento del solucionador.

Este conocimiento no depende del dominio de aplicación.

Ejemplo: un operador es útil si al aplicarlo resuelve el problema.

Se trata de las “metarreglas”, que se verán en el tema 6.

Existen dos modelos de reglas de control heurístico. Cada uno se obtiene traduciendo uno de los tipos de conocimiento de control mencionados.

Las reglas se tratarán en el tema 6.

### · Funciones de evaluación heurística (fev):

Proporcionan un valor numérico (n) que refleja en qué grado se considera prometedor un estado, en la búsqueda de la solución.

Tienen la forma:  $f(\text{estado}_i) = n_i$

Estas funciones miden, de algún modo, la distancia que separa al estado actual del estado meta. Pero hay que tener en cuenta que se trata de una estimación.

Normalmente se aplican varias restricciones a la aplicación de los operadores a los estados (no todos los operadores se pueden aplicar siempre a cualquier estado). Si se elimina alguna de estas restricciones se dice que se ha simplificado el modelo.

Este modelo simplificado es el que se emplea en las fev.

(Leer el ejemplo del 8-puzzle en la página 142 - es muy simple -).

Cuando se recorre un grafo de búsqueda se intenta siempre maximizar (o minimizar) la fev. El valor máximo (o mínimo) se obtendrá, como es lógico, cuando la fev se aplique sobre el nodo meta.

Usar fev muy exactas puede obligar a que sean muy complejas, y por tanto, costosas de calcular. Esto las haría perder ventajas, por lo que siempre se procura que sean lo más simples posible. Un medio de conseguirlo es usar modelos simplificados del problema.

### 4.1.2. Planteamiento del problema:

El planteamiento del problema se hace según el esquema de tarea genérica de búsqueda mostrado en el apartado 3.1 del tema anterior:

#### Dados

- Un conjunto de estados.
- Uno o más estados iniciales.
- Uno o más estados finales.
- Un conjunto de operadores.



#### Encontrar

Una secuencia de operadores que permita pasar del estado inicial al estado final.

La diferencia ahora está en el uso del conocimiento del dominio para guiar la selección de los operadores.

## 4.2. Una estrategia irrevocable.

### - Estrategias de búsqueda tentativas:

En cualquier momento del proceso de búsqueda puede abandonarse un camino de exploración para seguir estudiando otro camino más prometedor.

### - Estrategias de búsqueda irrevocables:

Primero se ordena el espacio de búsqueda mediante algún criterio.

Este criterio debe conseguir que nunca sea necesario reconsiderar una decisión ya tomada: nunca se puede abandonar el camino de exploración elegido.

### · Método del gradiente:

#### - También se llama búsqueda en escalada.

- La búsqueda hace su recorrido siguiendo los nodos en los que el valor de la función sea máximo. Esto significa que se busca siempre la máxima pendiente (gradiente).

- Se trata de un recorrido en profundidad, pero ahora se usa información para guiar la búsqueda.

#### - Procedimiento Método del gradiente:

```
1      {Denominar m al estado inicial del problema}
2      elegido ← m                                {Asignar m a una variable llamada elegido}
3      Hasta que (se encuentre una meta) o (se devuelva fallo) hacer
4          {Expandir m. Modo de hacerlo: }
5          Para (cada operador aplicable) y (cada forma de aplicación) hacer
6              nuevo ← Operador(m)                {Aplicar el operador a m. Se obtiene así un estado nuevo. }
7              Si (nuevo es meta) entonces
8                  Devolver nuevo                  {Solución. }
9                  {Se sale del bucle Para y del bucle Hasta, puesto que termina el procedimiento. }
10             Fin_Si
11             Si (f(nuevo) es mejor que f(elegido)) entonces
12                 elegido ← nuevo
13             Fin_Si
14         Fin_Para
15         Si (f(elegido) ≠ f(m)) entonces
16             m ← elegido
17         Si no
18             Devolver fallo
19         Fin_Si
20     Fin_Hasta
```

- En la línea 15 se comprueba si alguno de los sucesores del nodo expandido tiene un mejor valor.

Si eso ocurre, este será el nuevo nodo a expandir.

Esto debe cumplirse siempre, puesto que no hay posibilidad de reconsiderar decisiones.

- A veces conviene guardar en una lista CERRADA los nodos expandidos, para no volver a tomarlos.

- Campos de aplicación del método del gradiente:

Se usará este método de búsqueda siempre que se pueda encontrar una fev que vaya creciendo (o decreciendo) hasta el valor que tenga esa fev en el nodo meta.

- Teoría de juegos. Ejemplo sencillo: 8-puzzle.
- Búsqueda de máximos y mínimos locales en el campo del análisis numérico.
- Cuando en cualquier estado se pueda aplicar cualquier operador disponible.

Esta propiedad se llama conmutatividad.

- Análisis:

· Ventajas:

- Sencillez.
- No siempre se necesita recordar (almacenar) el camino recorrido. Puede ser suficiente con almacenar el estado *elegido* en cada momento.

· Inconvenientes:

- Depende completamente de que la fev esté bien definida:
  - Debe tener un valor que sea máximo (o mínimo) en un nodo meta.
  - Debe ser sencilla de calcular.
  - No debe tener máximos (o mínimos) locales.

Ya se ha dicho antes que para obtener una fev que cumpla los requisitos necesarios se usan modelos simplificados del problema real.

Para que se pueda crear un modelo simplificado, el problema tiene que ser descomponible.

Las fev que cumplen todos los requisitos se dice que son muy informativas o que están muy informadas.

### 4.3. Estrategias de exploración alternativas.

Muchas veces es muy difícil encontrar fev tan informadas como las necesarias para el método del gradiente.

Estas estrategias alternativas usan criterios que sólo aportan información parcial para guiar el proceso de búsqueda.

Por eso estas estrategias no son tan infalibles.

#### 4.3.1. Búsqueda Primero el Mejor:

- Este es el algoritmo general de búsqueda en grafos explicado en el apartado 3.6.
- Combina las ventajas de la búsqueda en amplitud y la búsqueda en profundidad.
- Si el camino actual se aleja de la meta, se abandona y se retoman caminos de exploración abandonados previamente.

Esta es su mayor diferencia respecto al método del gradiente.

- Usa una fev que mide (estima) la distancia a la meta.
- Resuelve el problema de las ‘mesetas’ o puntos de ‘estancamiento’: si aparecen, se abandona ese camino. Este problema no lo resuelve bien el método del gradiente (obliga a “afinar” mucho más la fev de forma que no se produzcan esos puntos).

Recordemos que el algoritmo de búsqueda general en grafos se basaba en saber si un nuevo estado había sido generado y expandido previamente, con lo que se evitaba repetir la exploración de algunos caminos.

Para conseguirlo usa dos listas (ABIERTA y CERRADA):

ABIERTA: almacena los nodos que han sido generados, pero no expandidos.

CERRADA: almacena los nodos de ABIERTA que han sido seleccionados para su expansión.

Dicha selección de nodos puede ser informada o no.

Cuando la selección es informada (usa una fev heurística) nos encontramos en el algoritmo de búsqueda “Primero el mejor”.

Si en algún momento de la búsqueda se quiere reanudar un camino abandonado previamente, sólo es necesario recurrir a ABIERTA.

La búsqueda “Primero el mejor” reordena la lista ABIERTA en cada paso del algoritmo.

Esta ordenación está basada en la asociación, a cualquier nodo del grafo, del valor que toma una fev aplicada al mismo.

Esta fev no tiene en cuenta el coste del mejor camino parcial encontrado en cada momento desde la raíz hasta el nodo considerado (en otros métodos de búsqueda sí se considera este coste).

En consecuencia, la fev consiste sólo en una estimación del menor coste, desde cada nodo, a un nodo meta.

Leer el ejemplo de la página 151.

#### · Análisis:

##### Ventajas:

- Soluciona el problema de los mínimos (o máximos) locales.
- Siempre encuentra el mínimo (o máximo) global.

##### Inconvenientes:

- No se considera el camino recorrido hasta el momento.

#### 4.3.1.b. Búsqueda en haz:

- Es una variación de la búsqueda “Primero el mejor”. No usa la lista ABIERTA.
- En vez de considerar, en cada paso, un único “mejor” estado para expandirlo, se consideran varios (un haz).
- Para seleccionar, de entre los estados generados, aquellos que serán expandidos en un momento dado, se pueden usar dos métodos:
  - 1) Permitir que sólo un número fijo de los nodos más prometedores en cada momento sean expandidos.
  - 2) Establecer un valor umbral de la fev, por debajo (o encima) del cual ningún nodo generado será expandido.
- Este método requiere menos recursos, por considerarse menos nodos en el espacio de búsqueda.

#### 4.3.2. Algoritmo A\*:

- Es el mejor método de búsqueda del tipo “Primero el mejor”.
- Siempre considera el camino recorrido hasta el momento.
- Hasta ahora la fev era la “distancia estimada desde el nodo actual hasta la meta”.

Aquí la fev es

$$f(n) = g(n) + h(n)$$

Donde:

$g(n)$ : coste real del mejor camino encontrado en un determinado momento desde la raíz hasta n (camino recorrido)

$h(n)$ : estimación del coste del mejor camino desde n hasta una meta.

$f(n)$ , por tanto, también es una estimación del coste total del mejor camino que vaya de la raíz hasta una meta, pasando por n.

#### · Procedimiento A\*:

Partimos del estado inicial del problema planteado.

Este estado lo almacenamos en un nodo que llamamos raíz ( $r$ ).

1. Crear una lista de nodos llamada *ABIERTA*.

Inicializar *ABIERTA* con un único nodo: el nodo raíz ( $r$ ).

$g(r) \leftarrow 0$       { Como no se ha recorrido aún ningún camino, su coste es 0. }

2. Crear una lista de nodos llamada *CERRADA*.      { Inicialmente vacía. }

3. Hasta que (se encuentre una meta) o (se devuelva fallo) hacer

    Si (*ABIERTA* está vacía) entonces

        Devolver fallo

    Si no

        {El menor elemento (según la fev) de *ABIERTA* se quita de *ABIERTA* y se añade a *CERRADA*.}

$m \leftarrow \text{fev\_mínimo}(\text{ABIERTA})$

$\text{ABIERTA} \leftarrow \text{ABIERTA} \setminus \{m\}$

$\text{CERRADA} \leftarrow \text{CERRADA} \cup \{m\}$

        Si ( $m$  es meta) entonces

            Devolver camino\_solución      {Se obtiene recorriendo los punteros de los antepasados.}

            Salir del bucle “hasta” del punto 3. { Es lógico, puesto que al Devolver se finaliza. }

        Si no

            Expandir  $m$       { Se generarán todos los sucesores de  $m$ . }

        Fin\_Si

        Para (cada sucesor  $n'$  de  $m$ ) hacer

            Crear puntero de  $n'$  a  $m$

$g(n') \leftarrow g(m) + c(m, n')$       { Donde  $c(m, n')$  es el coste de pasar de  $m$  a  $n'$ . }

            Si ( $n'$  está en *ABIERTA*) entonces



Llamar  $n$  al nodo encontrado en *ABIERTA*

Añadir  $n$  a los sucesores de  $m$ .

Si  $(g(n') < g(n))$  entonces

Redireccionar el puntero de  $n$  a  $m$ .

Cambiar el camino de menor coste encontrado desde la raíz a  $n$ .

$g(n) \leftarrow g(n')$

$f(n) \leftarrow g(n') + h(n)$

Fin\_Si

Si no si  $(n'$  está en *CERRADA*) entonces

Llamar  $n$  al nodo encontrado en *CERRADA*.

Si  $(g(n') < g(n))$  entonces

- Hacer recorrido en profundidad de los descendientes  $n_i$  de  $n'$  para propagar el nuevo menor coste  $g(n')$  a esos descendientes.
- También se actualizarán los valores de  $f$  correspondientes.
- El recorrido en profundidad empezará en  $n'$  tendrá en cuenta:
  - (a) Para (los nodos descendientes  $n_i$  cuyo puntero <sup>\*1</sup> conduzca hacia  $n'$ ) hacer
 

$g(n_i) \leftarrow g(n_i')$

$f(n_i) \leftarrow g(n_i') + h(n_i)$

Hacer

Seguir el recorrido del puntero de  $n_i$

Hasta que (se encuentre otro  $n_i$  que no tenga sucesores) o  $(g(n_i) = g(n_i'))$ <sup>\*2</sup>

Fin\_Para
  - (b) Para (los restantes  $n_i$  cuyo puntero NO conduce a  $n'$ ) hacer
 

Si  $(g(n_i') < g(n_i))$  entonces

Actualizar el puntero de  $n_i$  para que conduzca a  $n'$ . {  $n'$  ya apunta al mejor camino desde la raíz por el momento. }

Continuar la propagación.

Fin\_Si

Fin\_Para

Fin\_Si

Si no { Si  $n'$  no está en *ABIERTA* ni en *CERRADA*: }

Calcular  $h(n')$

$f(n') \leftarrow g(n') + h(n')$

$ABIERTA \leftarrow ABIERTA \cup \{n'\}$

Añadir  $n'$  a la lista de sucesores de  $m$

Fin\_Si

Fin\_Para

Fin\_Si

Fin\_Hasta

Aclaraciones respecto a este algoritmo:

\*<sup>1</sup>) Ese puntero debe apuntar siempre al mejor antecesor encontrado hasta ese momento.

\*<sup>2</sup>) Si  $g(n_i) = g(n_i')$ , se ha producido un ciclo y hay que terminar la propagación.

· Propiedades formales del algoritmo A\*:

1.- Es un método de búsqueda completo, incluso para grafos infinitos: si existe solución, la encuentra.

2.- Es admisibile: siempre encuentra la solución óptima, aunque sólo si se usa una h admisible.

Llamemos  $h^*(n)$  al coste real del camino óptimo desde n hasta un nodo meta.

$h(n)$  es la estimación heurística del coste de ese camino.

Entonces  $h(n)$  es admisible si

$$\forall n, h(n) \leq h^*(n) \quad (\text{También se dice que } h \text{ es } \underline{\text{mirorante}} \text{ de } h^*.)$$

Una función heurística  $h_1$  se dice que está más informada que otra  $h_2$  si

a)  $h_1$  y  $h_2$  son minorantes de  $h^*$ .

b)  $\forall n, h_1(n) \geq h_2(n)$  (Es decir, si  $h_1$  devuelve un valor más cercano al real.)

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

Si  $h(n) > h^*(n) \Rightarrow$  la solución encontrada tendrá mayor coste que la óptima (pero también es solución).

3.- Si  $\left| \begin{array}{l} n' \text{ es un nodo sucesor de } n \\ c(n, n') \text{ es el coste del camino óptimo entre } n \text{ y } n' \end{array} \right.$

entonces se dice que una función (en general, y por tanto se cumple con las funciones heurísticas) es monótona si nunca decrece:

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

Si la función heurística h es monótona, se puede asegurar que el algoritmo A\* encuentra siempre un camino óptimo para todos los nodos expandidos.

La consecuencia de esto es que deja de ser necesario revisar los valores de f a lo largo del camino de búsqueda expandido, con lo que disminuye el coste temporal.

4.- Entre todos los algoritmos que usan una función heurística monótona, y que encuentran una solución óptima, el algoritmo A\* expande el menor número de nodos.

· Análisis:

Ventajas:

El algoritmo A* es un método	<ul style="list-style-type: none"><li>- Completo,</li><li>- Admisibile (siempre que se use h admisible),</li><li>- Que encuentra la solución óptima para todos los nodos (siempre que se use h monótona).</li></ul>
------------------------------	---

#### Inconvenientes:

- Espacio necesario, por tener que usar ABIERTA.
- La propiedad de admisibilidad obliga a distinguir entre caminos casi iguales. Esto aumenta mucho el coste temporal del algoritmo. Es preferible buscar soluciones dentro de unos márgenes acotados de error.

4.3.2.b. Algoritmo A\* realizando la búsqueda en haz: Permite reducir el espacio de búsqueda.

4.3.2.c. Algoritmo A\* en profundidad progresiva (A\* - P):

Es una versión del algoritmo A\* que usa un planteamiento similar al del método de exploración progresiva.

Cada iteración es una búsqueda en profundidad completa, en la que se calcula si cualquier nodo supera el umbral de máximo coste. Si esto ocurre, se elimina esa rama y se vuelve al nodo generado más reciente.

No se necesita la lista ABIERTA, por lo que disminuye la complejidad espacial.

4.3.4. Método AO\* (también se llama YO\*):

Se trata de aplicar el algoritmo A\* a los grafos Y/O (hasta ahora sólo tratábamos con grafos O).

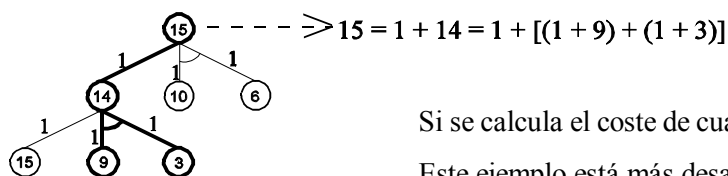
También encuentra una solución óptima, siempre que la función heurística  $h$  que estima la distancia desde un nodo al objetivo siga un criterio optimista (devuelva un valor que sea menor o igual que el real).

Recordemos que en los grafos Y/O hay enlaces de tipo Y, que conectan un nodo padre con los nodos subproblemas que hay que solucionar para que ese nodo se considere resuelto.

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

Se consideran subgrafos (subárboles) con respecto al grafo de búsqueda completo.

El coste de un grafo es la suma de los costes de los nodos que lo forman más los costes de los arcos que unen dichos nodos. Por defecto suponemos que todos los arcos tienen coste 1. En el siguiente ejemplo se ve el coste del árbol solución óptimo en un árbol Y/O:



Si se calcula el coste de cualquier otra solución se comprueba que sería mayor.

Este ejemplo está más desarrollado en la página 163: leerlo.

Hay que hacer hincapié en que ahora la solución no es un único nodo meta, sino que, con los grafos Y/O puede ser un conjunto de ellos.

En el proceso de búsqueda del grafo solución, cada vez que se genere un nodo o un grupo de nodos, el algoritmo debe comprobar si sus antecesores se han convertido en nodos resueltos. Esto se debe a que el valor inicial asignado a dichos antecesores era el resultado de la estimación de la función heurística  $h$ .

El algoritmo AO\* repite dos operaciones:

- Una hacia abajo, en la que encuentra el grafo solución parcial.

- Otra hacia arriba, de revisión de costes. También se forman nuevos subgrafos parciales menos costosos desde cada nodo (si procede).

En la siguiente descripción del procedimiento AO\*, la variable 'coste\_máximo' refleja el mayor valor que puede llegar a tener el coste del cálculo de una determinada solución.

· Procedimiento AO\*:

1. Crear un grafo  $G$ . { Al principio sólo está formado por el nodo raíz  $m$ . }

Estimar la distancia, desde  $m$  hasta la meta, mediante la función heurística  $h(m)$

2. Hasta que ( $m$  se considere 'resuelto') o ( $\text{coste}(m) > \text{coste\_máximo}$ ) hacer

{ Lógicamente, el grafo tiene nodos que forman los extremos; pues bien: }

- Seguir los enlaces que señalan el mejor grafo parcial de la solución (hasta el momento), hasta alcanzar los extremos.

- Escoger alguno de esos extremos y llamarlo  $n$ .

Expandir  $n$  generando todos sus sucesores  $s$ .

Si ( $n$  no tiene sucesores) entonces

$n \leftarrow \text{coste\_máximo}$

Fin\_Si

Para (cada sucesor  $s$ ) hacer

Si ( $s$  es un nodo terminal) entonces

Marcar  $s$  como 'resuelto'

$s \leftarrow 0$

Si no

$s \leftarrow h(s)$

Fin\_Si

Fin\_Para

Crear un conjunto  $S$  con los nodos  $n$ . { Conjunto de Sucesores. }

Hasta que ( $S = \text{vacío}$ ) hacer

$s' \leftarrow$  Elemento de  $S$  que no posea descendientes en  $S$ .

{ Actualizar el coste de  $s'$ : }

Para (cada enlace que parte de  $s'$ ) hacer

Calcular coste del enlace.

Fin\_Para

Marcar el enlace de menor coste.

Si (existiera una marca de otro enlace, saliente de  $s'$ , elegido anteriormente) entonces

Borrar la marca "antigua".

Fin\_Si

$h(s') \leftarrow$  Coste del enlace marcado.

$s' \leftarrow h(s')$

Si ( $s'$  = 'resuelto') o ( $h(s')$  ha cambiado) entonces

Añadir al conjunto  $S$  todos los antecesores de  $s'$ .

Continuar proceso recursivo de actualización de valores de los nodos.

{ Este proceso recursivo puede llegar hasta la raíz del grafo  $G$ . }

Fin\_Si

Fin\_Hasta

Fin\_Hasta

#### 4.4. Búsqueda con adversarios.

Se estudian las situaciones en que participan 2 (o más) jugadores, perfectamente informados:

- Conocen todas las reglas aplicables.
- Conocen la disposición de los elementos implicados (por ejemplo, situación de todas las piezas en un tablero).

En teoría de juegos, lo mejor es usar el esquema de reducción del problema.

Por eso la representación del problema se hace con grafos Y/O:

- Enlaces O: movimientos posibles en un instante dado.
- Enlaces Y: movimientos del adversario.

Los dos jugadores participan por turnos (no a la vez, como en una carrera).

Cada nivel del árbol representa un turno.

El resultado de cada movimiento sitúa a cada jugador en un nuevo conjunto de estados.

El árbol obtenido es una representación explícita de todas las posibles partidas del juego.

Cada partida está representada por una rama del árbol, desde la raíz hasta un nodo hoja.

Los nodos hoja representan 3 finales posibles:

- Ganar.
- Perder.
- Empatar.

Es imposible considerar todos los caminos por los que podría pasar el juego hasta la resolución de la partida: se produce el problema de la explosión combinatoria.

Por esto se establece un límite de profundidad en la exploración del árbol, a partir de la situación actual de la partida. A ese límite también se le llama frontera de exploración.

Desde el estado actual se estudian los posibles movimientos y las reacciones del contrario, sucesivamente, hasta alcanzar el límite de profundidad establecido.

A partir de ese límite se tiene que usar una función heurística para estimar cuál es la posibilidad de que esos nodos sean ganadores, perdedores, o de empate.

Así se estima el tipo de juego más probable que “cuelga” de cada nodo situado en el límite de exploración.

Existen dos formas de evaluar los nodos del árbol de búsqueda. El valor que se obtiene será la etiqueta para ese nodo:

- Etiquetado según MAX.
- Etiquetado MM valor.

a) Etiquetado según MAX:

A cada nodo se le asocia un valor que representa lo prometedor que es esa situación de la partida para el jugador MAX.

Esta consideración es independiente de si le toca “mover” a MAX o a MIN.

Por tanto:

- Desde un nodo padre MAX se elige la jugada que lleve hasta el hijo con mayor valor (MAX busca la jugada que resulte más prometedor para MAX).
- Desde un nodo padre MIN se elige la jugada que lleve hasta el hijo con menor valor (esto es porque MIN busca la jugada que resulte meno prometedor para MAX).

$$\text{MAX}(m) = \begin{cases} \text{fev}(m) & \text{si } m \text{ no tiene sucesores.} \\ \max_n \{ \text{MAX}(n) \} & \text{si } m \text{ es un nodo MAX.} \\ \min_n \{ \text{MAX}(n) \} & \text{si } m \text{ es un nodo MIN.} \end{cases} \quad \text{Donde } n \text{ es un sucesor de } m.$$

b) Etiquetado MM valor:

A cada nodo le asigna un valor que indica lo prometedor que es esa situación para el jugador que posee el turno de movimiento en ella.

Si la situación es prometedor para MAX tendrá valor positivo, y si lo es para MIN, negativo.

Para establecer el valor de un nodo (da igual que sea MAX o MIN):

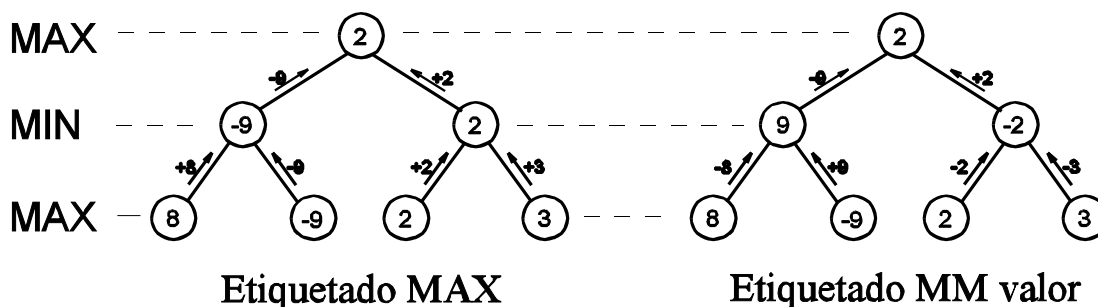
- 1º) Elegir el mínimo de los valores de los hijos.
- 2º) Cambiar de signo el resultado.

O bien (da exactamente igual, pero se suele hacer lo siguiente):

- 1º) Cambiar el signo de los valores de los hijos.
- 2º) Elegir el máximo.

$$\text{MM valor}(m) = \begin{cases} +\text{fev}(m) & \text{si } m \text{ no tiene sucesores y es un nodo MAX.} \\ -\text{fev}(m) & \text{si } m \text{ no tiene sucesores y es un nodo MIN.} \\ \max_n \{ -\text{MM valor}(n) \}, & \text{siendo } n \text{ un sucesor de } m. \text{ (Hecho según la segunda forma).} \end{cases}$$

\* Ejemplo:



- Metodo MINIMAX.
- Método de poda  $\alpha$ - $\beta$ : es una variación del MINIMAX que resulta más eficiente.

- Se comporta como un procedimiento de búsqueda con retroceso con una frontera de exploración calculada. Como se sabe, este tipo de búsqueda hace una exploración exhaustiva.
- MINIMAX se usa en situaciones en que el objetivo es ganar una partida en la que participan dos adversarios que realizan movimientos alternativos.
- La función heurística debe medir la ventaja relativa de realizar las posibles jugadas. Cuanto mejor esté diseñada esta función (cosa nada fácil), más eficaz será MINIMAX.
- La estrategia MINIMAX consiste (básicamente) en prever el mayor número de jugadas posibles, tanto propias como del adversario, y actuar en consecuencia.
- Siempre se considera que el oponente va a realizar la mejor de las opciones posibles para él, cuando le toque “mover”.
- Se recorre un árbol Y/O.
- Cada nivel del árbol tiene nodos de un tipo: MAX (protagonista) o MIN (oponente).
- El nodo raíz suele ser siempre MAX, y representa al jugador que mueve en primer lugar.
- El objetivo del jugador MAX es maximizar el valor de la función de evaluación heurística (f): elegir siempre el camino que le conduzca a un nodo frontera con mayor valor de fev.
- MIN hará siempre lo contrario: elegirá el valor menor para fev, pues éste será el peor valor posible para MAX, pero el mejor para él.
- En la siguiente descripción del algoritmo:

- Aparece C(jugador). Simplemente es una función que cambia o alterna entre los jugadores.
- La descripción está basada en el etiquetado MM valor.
- La llamada inicial a este procedimiento sería MINIMAX(nodo\_inicial, 0, MAX).
- Debe existir una variable global que marque el límite de profundidad a explorar (lp).

MINIMAX( $m, profundidad, jugador$ );      { *profundidad* representa el nivel que se está explorando ahora,  
no es el límite de profundidad. }

- Si no

2. { Generar los sucesores de  $m$ : }
- 2.1  $mejor \leftarrow \min_j \{ fev(j) \forall j \}$  { Asigna el menor valor soportado por la fev. }
- 2.2 Para (cada sucesor  $n$  de  $m$ ) hacer
  - $M(n) \leftarrow \text{MINIMAX}(n, \text{profundidad} + 1, C(\text{jugador}))$
  - $mejor \leftarrow \max(-M(n), mejor)$
 Fin\_Para
3. Si (se han analizado recursivamente todos los sucesores de un nivel *-profundidad-* ) entonces
  - Devolver  $mejor$
 Fin\_Si

· Complejidad:

Consideremos que el árbol a explorar tiene las siguientes características:

- Coste uniforme.
- Profundidad  $p$ .
- Factor de ramificación  $n$ .

Este árbol contiene  $n^p$  nodos terminales. El procedimiento MINIMAX deberá analizar todos esos nodos en el peor caso.

· Análisis:

Ventajas:

- Es un método general.
- Sencillez (de implementación).
- Permite representar y buscar ganadores en juegos con dos adversarios.
- El etiquetado de los nodos permite identificar fácilmente las estrategias posibles.

Inconvenientes:

- Es un método demasiado exhaustivo: poco eficiente por su complejidad ( $n^p$ ).

El método  $\alpha$ - $\beta$  sí permite que no se analicen todos los caminos cuando se dan ciertas condiciones.

Leer el ejemplo de la página 51 del libro de problemas para comprender este inconveniente.

#### 4.4.2. Estrategia de poda $\alpha$ - $\beta$ :

- Es una variación del método MINIMAX que permite ahorrarse el recorrido de caminos inútiles del árbol de búsqueda.
- Es el método más usado en los problemas de juegos.
- En cada llamada recursiva a un nodo hijo, pasa dos valores ( $\alpha$  y  $\beta$ ):

$\alpha$  marca la cota inferior.

$\beta$  marca la cota superior.



- Tales cotas indican el margen de valores de la fev que se van a ir buscando en la parte del árbol que queda por explorar.
- Si en algún momento  $\alpha \geq \beta$ , ya no tiene sentido seguir con la búsqueda (la cota inferior es mayor que la superior):
  - Si se está en un nodo MAX, se hace una poda  $\alpha$  (poda causada por  $\alpha$ ).
  - Si se está en un nodo MIN, se hace una poda  $\beta$  (poda causada por  $\beta$ ).
- Así se usa:
  - $\alpha$ : valor mínimo que debe tener un nodo MIN para que siga siendo explorado.
  - $\beta$ : valor máximo que debe tener un nodo MAX para que siga siendo explorado.
- Así se obtiene:
  - $\alpha$ : valor mayor encontrado hasta el momento de todos sus antecesores MAX.
  - $\beta$ : valor menor encontrado hasta el momento de todos sus antecesores MIN.
- En la primera llamada al procedimiento `alfabeta()`, al pasar los parámetros  $\alpha$  y  $\beta$ :
  - $\alpha = -\infty$ , o el menor valor que pueda devolver la fev.
  - $\beta = +\infty$ , o el mayor valor que pueda devolver la fev.
- En la siguiente descripción del procedimiento `alfabeta()`:
  - $m$ : nodo actual.
  - $n_i$ : nodos hijos del nodo actual, con  $i = 1..b$ .
  - En la implementación debe existir una variable global ' $lp$ ' que marque el límite de profundidad a explorar.
  - $C(jugador)$  es una función que cambia (alterna) de jugador.
  - Primera llamada al procedimiento:
 
$$\text{alfabeta}(\text{nodo\_inicial}, 0, \text{MAX}, \text{fev}_{\min}, \text{fev}_{\max})$$
  - Procedimiento de poda  $\alpha - \beta$ :
 
$$\text{alfabeta}(m, \text{profundidad}, \text{jugador}, \alpha, \beta);$$

Si ( $m$  no tiene sucesores) o ( $\text{profundidad} = lp$ ) entonces

Devolver  $\text{fev}(m)$

Si no

$i \leftarrow 1$  { Se empezará explorando el primer hijo de  $m$ :  $n_1$  }

Si ( $m$  es un nodo MAX) entonces

Mientras ( $i \leq b$ ) hacer { Explorar todos los hijos  $n_i$  de  $m$ : }

$\alpha \leftarrow \max[\alpha, \text{alfabeta}(n_i, \text{profundidad} + 1, C(jugador), \alpha, \beta)]$

Si ( $\alpha \geq \beta$ ) entonces

Devolver  $\beta$  { Por tanto, aquí se sale del bucle 'Mientras', lo que significa que no se exploran más hijos de  $m$ : se hace una poda  $\alpha$ . }

Fin\_Si

$i \leftarrow i + 1$

Fin\_Mientras

Devolver  $\alpha$

Si no { Si  $m$  es un nodo MIN: }

Mientras ( $i \leq b$ ) hacer { Explorar todos los hijos  $n_i$  de  $m$ : }

$\beta \leftarrow \min[\beta, \text{alfabeta}(n_i, \text{profundidad} + 1, C(\text{jugador}), \alpha, \beta)]$

Si ( $\alpha \geq \beta$ ) entonces

Devolver  $\alpha$  { Por lo tanto, aquí se sale del bucle 'Mientras', lo que significa que no se explorarán más hijos de  $m$ : se hace una poda  $\beta$ . }

Fin\_Si

$i \leftarrow i + 1$

Fin\_Mientras

Devolver  $\beta$

Fin\_Si

Fin\_Si

· Complejidad:

Caso mejor: Cuando el árbol de búsqueda esté perfectamente ordenado.

Esto ocurre si las ramas de más a la izquierda devuelven los mejores valores de  $\alpha$  y de  $\beta$ . Así se eliminan el resto de los caminos.

Caso peor: Habría que examinar todos los nodos terminales.

Entonces tendría la misma complejidad que el método MINIMAX:  $n^p$ .

· Análisis:

Ventajas:

- Eficiencia: Evita el recorrido exhaustivo del árbol de búsqueda.  
Reduce mucho el número de nodos expandidos.  
Permite afrontar problemas supuestamente intratables.

Inconvenientes:

- Dependencia excesiva del ordenamiento de los nodos del árbol.

- A continuación aparece una modificación del alfabeta() que hace un uso directo del etiquetado MM valor. Es muy parecido al procedimiento MINIMAX(), pero aquí se hacen podas:

alfabeta( $m, \text{profundidad}, \text{jugador}, \text{límite\_siguientes}, \text{límite\_actual}$ );

Si ( $m$  no tiene sucesores) o ( $\text{profundidad} = lp$ ) entonces

Si ( $\text{jugador} = \text{MAX}$ ) entonces

Devolver fev( $m$ )

Si no { Si  $\text{jugador} = \text{MIN}$  }

Devolver  $- \text{fev}(m)$

```

    Fin_Si
Si no
    { Generar los sucesores de  $m$ ;  $b$  será el número de sucesores encontrados: }
     $i \leftarrow 1$ 
    Mientras ( $i \leq b$ ) hacer          { Explorar todos los hijos  $n_i$  de  $m$ : }
         $valor \leftarrow \text{alfabeta}(n_i, \text{profundidad} + 1, C(\text{jugador}), -\text{límite\_actual}, -\text{límite\_siguientes})$ 
        { Observar que los 2 últimos parámetros han intercambiado sus posiciones y han cambiado de signo. }
         $\text{límite\_siguientes} \leftarrow \max[-valor, \text{límite\_siguientes}]$ 
        Si ( $\text{límite\_siguientes} \geq \text{límite\_actual}$ ) entonces
            Devolver  $\text{límite\_actual}$     { Puesto que aquí se sale del bucle ‘Mientras’, ya no se exploran más
                                         hijos de  $m$ : se ha hecho una poda. }
        Fin_Si
    Fin_Mientras
    Devolver  $\text{límite\_siguientes}$ 
Fin_Si

```

Llamada inicial:  $\text{alfabeta}(\text{estado\_inicial}, 0, \text{MAX}, \text{fev}_{\min}, \text{fev}_{\max})$  { Consideramos  $\text{fev}_{\min} = -\infty$  y  $\text{fev}_{\max} = +\infty$  }

## PROBLEMAS:

Como repaso “rápido” de todo el tema, hacer los siguientes ejercicios del libro de problemas:

- 2.1. Método de escalada. En vez de resolverlo como en el libro, es mucho más sencillo representar la traza en una tabla de tres columnas:

$m$	<i>elegido</i>	<i>nuevo</i>
$\vdots$	$\vdots$	$\vdots$

- 2.2.a. Método “Primero el mejor”.
- 2.4. Algoritmo A\*.
- 2.6. Algoritmo A\*, aplicado a un caso “real”: 8- puzzle. Cálculo de heurísticas.
- 2.8. Algoritmo AO\* (YO\*).
- 2.10. Etiquetado MAX. Poda  $\alpha, \beta$ .



## TEMA 5

### LÓGICA.

#### 5.2. Lógica de proposiciones.

Una fórmula cualquiera puede ser:

Válida ( $\vdash$ ): cierta en todas las interpretaciones (tautología).

Satisfacible: cierta en, al menos, una interpretación.

Insatisfacible: falsa en todas las interpretaciones (contradicción).

Regla de transformación básica: Modus Ponens.

Si S y S  $\rightarrow$  R son fórmulas válidas, entonces R también lo es.

$$\frac{\vdash S, \vdash S \rightarrow R}{\vdash R} \qquad \frac{S \quad S \rightarrow R}{R}$$

También se llama Regla de eliminación del condicional (RE  $\rightarrow$ ). Ejemplo:

Premisas	1. $p \rightarrow (\neg q \rightarrow r)$
	2. $\neg q$
	3. $p$
	4. RE $\rightarrow$ , 1, 3. Se obtiene $\neg q \rightarrow r$
	5. RE $\rightarrow$ , 2, 4. Se obtiene <b>r</b>

(RI  $\rightarrow$ ) Regla de introducción del condicional.

-También se llama Teorema de la deducción: si en un cálculo existe una demostración de una fórmula R (conclusión) a partir de una fórmula S (premisa), entonces también existe una demostración de  $S \rightarrow R$ .

- Por lo tanto,  $S \rightarrow R$  es una tautología.

- Generalización de esta regla:

Dadas una serie de premisas y una conclusión, la fórmula:  $P_1 \wedge P_2 \wedge \dots \wedge P_n \rightarrow C$  es una tautología.

#### Resolución por refutación:

- Suponer que la conclusión buscada es falsa y comprobar que  $(P_1 \wedge P_2 \wedge \dots \wedge P_n \rightarrow \neg C)$  es una contradicción.

- Si esto ocurre, entonces sabemos que C es cierta.

### Forma clausulada:

También se llama

Forma Normal de Skolem.
Forma Normal conjuntiva.

Cláusula: disyunción de variables proposicionales, negadas o sin negar.

Ejemplo:  $(\neg p \vee q) \wedge (\neg q \vee s \vee r) \wedge \neg s \wedge \neg r$

Donde cada una de las partes subrayadas es una cláusula.

### Regla de resolución:

- Se parte de dos cláusulas (generatrices o premisas).
- En ambas cláusulas debe aparecer la misma variable:

· En una negada.
· En la otra sin negar.
- Se obtiene una sola cláusula (resolvente o conclusión).
- El resolvente estará formado por la disyunción del resto de las variables.
- Ejemplo en el que se eliminará la variable q:

P1:  $\neg p \vee q$

P2:  $\neg q \vee s \vee r$

C:  $\neg p \vee s \vee r$

### Propiedades básicas de un formalismo lógico:

- 1.- Consistente: no se puede demostrar simultáneamente una fórmula y su negación.
- 2.- Completo: cualquier fórmula válida es demostrable a partir de los axiomas y del conjunto de reglas de inferencia.
- 3.- Decidible: existe un procedimiento para comprobar si una fórmula es válida en el sistema.

La lógica de proposiciones cumple las tres propiedades. Por eso se dice que es un formalismo de representación robusto.

## 5.3. Lógica de predicados.

Distingue qué se afirma (propiedades o relaciones),  
y de quién se afirma (individuos).

Ejemplo:

P1: Los estudiantes de informática aman la lógica.

P2: Juan es estudiante de informática.

C: Juan ama la lógica.

En lógica de predicados se representa así:

P1:  $\forall x (\text{Estudiante\_infor}(x) \rightarrow \text{Ama\_la\_lógica}(x))$

P2: Estudiante\_infor(Juan)

C: Ama\_la\_lógica(Juan)

<u>Predicados</u>	Propiedades (Estudiante_infor), o relaciones entre individuos (Amigo, Ama).
-------------------	--

Los predicados llevan asociados <u>argumentos</u> o <u>términos</u> que pueden ser	Variables (x). Constantes (Juan). Funciones (máximo, sucesor, etc.)
--	---

Además emplean cuantificadores	$\forall$ (universal). $\exists$ (existencial o particular).
--------------------------------	---

Variables libres: no están afectadas por un cuantificador.

Variables ligadas: las afecta un cuantificador.

Ejemplo:

$\exists x (\text{Estudiante\_Infor}(x) \wedge \text{Amigo}(x, y))$

$x$ : ligada.  
 $y$ : libre.

Variable que queda ligada por ese cuantificador.

Predicados con un solo argumento: monádicos.

Predicados con varios argumentos: poliádicos (diádicos, triádicos, etc.).

Funciones: Pueden tener términos que sean variables o constantes.

No tienen un valor de verdad asociado.

Simplifican la notación.

Conversión entre cuantificadores:

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

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

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

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

Importante: Una expresión no es cierta ni falsa hasta que la particularicemos.

Para validar sentencias hay que referirse a un determinado universo de discurso.

Ejemplo:

Universo de discurso:  $\{a, b, c\}$

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

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

Recordemos que el cálculo de proposiciones tenía como propiedad la decidibilidad.

Sin embargo, el cálculo de predicados es semidecidible:

Sólo puede probar teoremas (fórmulas obtenidas a partir de los axiomas y las reglas de inferencia del cálculo). Pero ese “fallo” no es importante en IA, pues aquí no es obligatorio que los métodos usados para resolver los problemas de IA sean óptimos.

### Reglas usadas en el cálculo de predicados:

Además de las reglas usadas en el cálculo de proposiciones, se emplean éstas:

1.- Regla de eliminación del cuantificador universal (RE  $\forall$ ):

$$\frac{\forall x_1, x_2, \dots, x_n \quad P(x_1, x_2, \dots, x_n)}{P(a_1, a_2, \dots, a_n)}$$

2.- Regla de introducción del cuantificador universal (RI  $\forall$ ):

$$\frac{P(a_1, a_2, \dots, a_n)}{\forall x_1, x_2, \dots, x_n \quad P(x_1, x_2, \dots, x_n)}$$

Para que sea válida,  $a_1, a_2, \dots, a_n$  deben ser individuos cualesquiera y no unos concretos.

3.- Regla de eliminación del cuantificador existencial (RE  $\exists$ ):

$$\frac{\exists x_1, x_2, \dots, x_n \quad P(x_1, x_2, \dots, x_n)}{P(a_1, a_2, \dots, a_n)}$$

Al contrario que en RI  $\forall$ , ahora los parámetros  $a_1, a_2, \dots, a_n$  deben ser individuos concretos, no cualquiera.

4.- Regla de introducción del cuantificador existencial (RI  $\exists$ ):

$$\frac{P(a_1, a_2, \dots, a_n)}{\exists x_1, x_2, \dots, x_n \quad P(x_1, x_2, \dots, x_n)}$$

- El cuantificador universal no implica necesariamente la noción de existencia:

- “Todos los ingenieros son prácticos”. Aquí sí se supone que existen ingenieros.
- “Todo el que piense lo contrario será considerado desertor”. Aquí NO implica existencia de personas “que piensen lo contrario”.

- El símbolo ‘ $\vee$ ’ tiene carácter inclusivo. Por eso la siguiente frase:

“Todos los seres humanos son mujeres o son hombres”

debe representarse así:

- $\forall x (\text{Humano}(x) \rightarrow ((\text{Mujer}(x) \wedge \neg \text{Hombre}(x)) \vee (\text{Hombre}(x) \wedge \neg \text{Mujer}(x))))$
- $\forall x (\text{Humano}(x) \rightarrow ((\text{Mujer}(x) \vee \text{Hombre}(x)) \wedge \neg (\text{Mujer}(x) \wedge \text{Hombre}(x))))$

- Dado que para validar una expresión hay que particularizar las variables que contenga, cuantas menos variables, más eficiente (computacionalmente) será su validación.

Sin embargo, si ahorramos variables, puede ocurrir que la expresión resulte poco clara.

Granularidad de un enunciado es su capacidad expresiva o nivel de detalle.



### 5.3.1. Ampliaciones de la lógica de predicados:

#### - Lógica de predicados de orden superior:

Permite cuantificar los nombres de predicado y las funciones.

Ejemplo:

“Todos los mamíferos poseen una propiedad en común.”

$$\exists P \forall x (\text{Mamífero}(x) \rightarrow P(x))$$

#### - Lógica de predicados con identidad:

Se establece una nueva forma “ $x = y$ ” en el cálculo de predicados, que permite expresar la relación de identidad.

Ejemplos:

a) “Hay al menos 2 porteros en la selección de fútbol”. (Puede haber 2 o más)

$$\exists x \exists y (\text{Portero}(x) \wedge \text{Portero}(y) \wedge (x \neq y))$$

b) “Hay a lo sumo un capitán en el equipo”. (Puede haber 1 o ninguno)

$$\forall x \forall y ((\text{Capitán}(x) \wedge \text{Capitán}(y)) \rightarrow (x = y))$$

c) “Hay exactamente un entrenador”.

$$\exists x (\text{Entrenador}(x) \wedge \forall y (\text{Entrenador}(y) \rightarrow (x = y)))$$

↑

↑

Existe uno

No hay más de uno

## 5.4. Deducción automática: Resolución:

La forma general de la regla de resolución permite realizar la deducción automática, tanto en la lógica de proposiciones como en la de predicados.

Puede ocurrir que este método no termine, en algunos casos.

### 5.4.1. Forma Clausulada:

Para aplicar el método de resolución, primero debe conseguirse que todas las premisas y la conclusión estén en forma clausulada:

- Todos los cuantificadores están a la cabeza de la fórmula.
- Sólo existen cuantificadores universales ( $\forall$ ).
- El resto de la fórmula
  - Se llama matriz.
  - No contiene cuantificadores.
  - Está formado por la conjunción de fórmulas:  
Cada fórmula es una disyunción de fórmulas atómicas, negadas o sin negar.

La fórmula clausular facilita su representación en el ordenador.

- Procedimiento de conversión de sentencias en forma clausulada (páginas 200 ... 203 del libro de texto):

1.- Eliminación de '→' y '↔'

$$(P \rightarrow S) \leftrightarrow (\neg P \vee S) \quad ; \quad (P \leftrightarrow S) \leftrightarrow (P \rightarrow S) \wedge (S \rightarrow P)$$

2.- Eliminación de '¬' de las fórmulas compuestas:

- Usar leyes de De Morgan.
- Usar ley de doble negación.
- Suprimir negaciones en las fórmulas afectadas por cuantificadores. Ejemplo:  $\forall x \neg P(x) \leftrightarrow \neg \exists x P(x)$

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

Se pueden aplicar estas dos leyes:

- Ley de distribución del  $\forall$  por la  $\wedge$ :  $\forall x(R(x) \wedge P(x)) \leftrightarrow \forall x R(x) \wedge \forall x P(x)$
- Ley de distribución del  $\exists$  por la  $\vee$ :  $\exists x(R(x) \vee P(x)) \leftrightarrow \exists x R(x) \vee \exists x P(x)$

4.- Colocar todos los cuantificadores a la cabeza de la fórmula.

Usar estas 8 reglas: 1ª)  $R \vee \forall x P(x) \leftrightarrow \forall x (R \vee P(x))$

$$2ª) \quad R(x) \vee \forall y P(y) \leftrightarrow \forall y (R(x) \vee P(y))$$

(Las otras 6 reglas resultan de sustituir, en estas dos, el signo  $\forall$  por el signo  $\exists$ . Y el signo  $\vee$  por  $\wedge$ )

5.- Eliminar los cuantificadores existenciales. Hay 2 casos:

A) Si  $\exists$  no tiene  $\forall$  a su izquierda, se elimina  $\exists$  y se introduce una constante de Skolem.

Ejemplo:

$$\exists x \forall y \forall z \forall u ((R(x) \vee P(x, u)) \wedge S(x, y, z)) \quad \text{se transforma en:}$$

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

El nombre de esta constante no puede coincidir con el de otra que ya hubiera.

B) Si  $\exists$  tiene uno o varios  $\forall$  a su izquierda:

- Se elimina  $\exists$ .
- Se introduce una función de Skolem.

Ejemplos:

$$\forall x \exists y (Hombre(x) \rightarrow Padre(x, y)) \quad \text{se transforma en} \quad \forall x (Hombre(x) \rightarrow Padre(x, f(x)))$$

$$\forall y \forall z \exists x (R(x) \vee P(x, y, z)) \quad \text{se transforma en} \quad \forall y \forall z (R(g(y, z)) \vee P(g(y, z), y, z))$$

6.- Eliminar todos los  $\forall$ .

7.- Convertir la fórmula en conjunción de disyunciones. Usar:

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

8.- Considerar cada una de las disyunciones del paso 7 como una cláusula aparte. alguna de ellas puede estar formada por un único literal.

9.- Diferenciar el nombre de las variables entre las distintas cláusulas surgidas en el paso 8: cambiar los nombres de las variables si hace falta.

No debe haber dos cláusulas que hagan referencia al mismo nombre de variable.

#### 5.4.2. Unificación:

Equiparación de fórmulas: Consiste en comparar predicados que tengan el mismo nombre y comprobar si sus términos (argumentos) se pueden unificar.

Para que dos términos sean unificables debe existir una sustitución que los haga idénticos.

Partimos de predicados en forma normal clausulada (FNC): Variables cuantificadas universalmente, aunque ya no aparezca el símbolo  $\forall$ .

Por tanto, las variables podrán sustituirse por constantes, funciones, u otras variables.

Ejemplo: las dos expresiones siguientes pueden unificarse a través de la sustitución  $s_1$ :

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

La sustitución  $s_1$  debe interpretarse así: “Donde haya una  $u$ , poner  $f(x)$  y donde haya una  $v$ , poner  $b$ ”.

Composición o producto de sustituciones: no cumple la propiedad conmutativa.

No es lo mismo hacer una sustitución y luego otra distinta que hacer estas mismas sustituciones en el orden contrario:  $s_1 s_2 \neq s_2 s_1$

Se debe hacer la unificación con el unificador de máxima generalidad (UMG), también llamado unificador mínimo: cualquier otro unificador que se pueda aplicar a esa fórmula se podrá obtener a partir del UMG, por medio de alguna otra sustitución.

Leer el ejemplo que aparece al final de la página 206 para comprender la composición de sustituciones y la obtención del UMG (son muy sencillas).

#### 5.4.3. Método general de resolución:

1. Se convierten las premisas y la negación de la conclusión, en su forma clausulada.
2. Se seleccionan sucesivamente parejas de literales, que estén en cláusulas distintas, con el mismo nombre (uno negado y otro sin negar).
3. Se comparan (mediante su UMG), y si existe la unificación, se crea una cláusula nueva, formada por la disyunción del resto de los literales de las generatrices.
4. Se añade la nueva cláusula a las ya existentes y se repite el proceso, hasta conseguir  $\lambda$ .

El proceso de resolución puede tratarse como una búsqueda en un espacio de cláusulas.

En cada momento se tiene un grafo de cláusulas explícito.

El objetivo de la búsqueda en ese grafo es encontrar la cláusula vacía ( $\lambda$ ).

Hay dos grupos generales de estrategias para hacer esta búsqueda:

- Intentar limitar el número de cláusulas que se consideran en el espacio de búsqueda.
- Intentar dirigir el proceso de búsqueda (esto se ha demostrado insuficiente).

Leer el ejemplo de la página 211: muestra dos o tres cuestiones interesantes para simplificar el proceso de búsqueda.

(Muy sencillo).

## 5.5. Extensiones de la lógica clásica.

### 5.5.1. Lógica modal:

Permite expresar matices (modalidades) de la verdad o falsedad asociadas a un enunciado.

Ahora se pueden manejar los conceptos de necesidad y posibilidad.

Se añaden los siguientes símbolos a la lógica clásica:

$\Box P$	“es <u>necesario</u> que P”	Necesari $\Box$	} (Asociación nemotécnica)
$\Diamond P$	“es <u>posible</u> que P”	Posibl $\Diamond$	

Los operadores  $\Box$  y  $\Diamond$  tienen el mismo nivel que  $\neg$ .

Equivalencia entre los dos operadores:

$$\Box \neg P \leftrightarrow \neg \Diamond P$$

$$\Diamond P \leftrightarrow \neg \Box \neg P$$

Nueva definición de la implicación:  $R \Rightarrow S \equiv \neg \Diamond (R \wedge \neg S)$

De R se sigue necesariamente la conclusión S,

es lo mismo que:

No es posible que se cumpla R y (a la vez) no se cumpla S.

Nuevas reglas de inferencia utilizadas:

$$1) \neg \Diamond P \leftrightarrow \Box \neg P$$

$$2) \neg \Box P \leftrightarrow \Diamond \neg P$$

$$3) \Box P \rightarrow P \quad (\text{Si “P” es necesario, entonces es cierto en cualquier posible mundo, incluido el real.})$$

$$4) \Diamond P \rightarrow P \quad (\text{Si “P” es posible, entonces es cierto en algún mundo. A este mundo se le asignará un nombre que no se haya especificado antes.})$$

Necesidad y posibilidad son dos “modos de verdad”.

“ $\Box A$ ” es un concepto más fuerte que “A es cierto”.

“ $\Diamond A$ ” es un concepto más débil que “A es cierto”.

Mundo: describe cómo las cosas podrían ser o haber sido.

De entre todos los mundos posibles en lógica modal, el mundo real describe como son las cosas realmente.

Enunciado verdadero: es cierto en el mundo real.

Enunciado necesario: es cierto en cualquier mundo.

Enunciado posible: es cierto en algún mundo posible (puede ser cierto o no en el mundo real).

Representación:

W: P	“P es verdad en el mundo W.”
W2 supóngase : P	“Se supone que P es verdad en el mundo W2.”
$(A \supset B)$	Se trata de la implicación “Si se cumple A, entonces se cumple B.”
$\therefore$	Representa “Se infiere” o “Se deduce”. Precede a las conclusiones.

Para que se considere que se ha producido una contradicción (durante un proceso de inferencia), tal contradicción debe darse en el mismo mundo.

W1: A		Contradicción	W1: A		No es contradicción, por darse en mundos distintos
W1: $\neg A$			W2: $\neg A$		

### 5.5.2. Lógica difusa:

Repasar las páginas 113 a 119 del libro de problemas.

### 5.5.3. Lógicas no monótonas:

La lógica clásica debe cumplir estas 3 propiedades para que “funcione”:

1. Es completa: Toda fórmula válida (tautología) debe ser demostrable con los axiomas y reglas (es decir, con las leyes) disponibles.
2. Es tratable: El cálculo necesario para manejar las leyes y premisas en los procesos de inferencia no debe ser muy complejo.
3. Es consistente: Si, a partir de unas premisas, se obtiene una conclusión, al añadir nuevas premisas debe seguir obteniéndose esa conclusión.

Las lógicas no monótonas NO son consistentes: hay veces en que obtenemos conclusiones a partir de información (premisas) parcial. Cuando conseguimos más información, tal vez modifiquemos o eliminemos esas primeras conclusiones.

## **PROBLEMAS:**

Como repaso “rápido” de todo el tema, hacer los siguientes ejercicios del libro de problemas:

Relacionados con la lógica modal:

Ejemplo de la página 112 del libro de problemas, y/o alguno de los problemas 3.11, 3.12, 3.13.

Relacionado con la lógica difusa:

3.21.

Relacionado con las lógicas no monótonas:

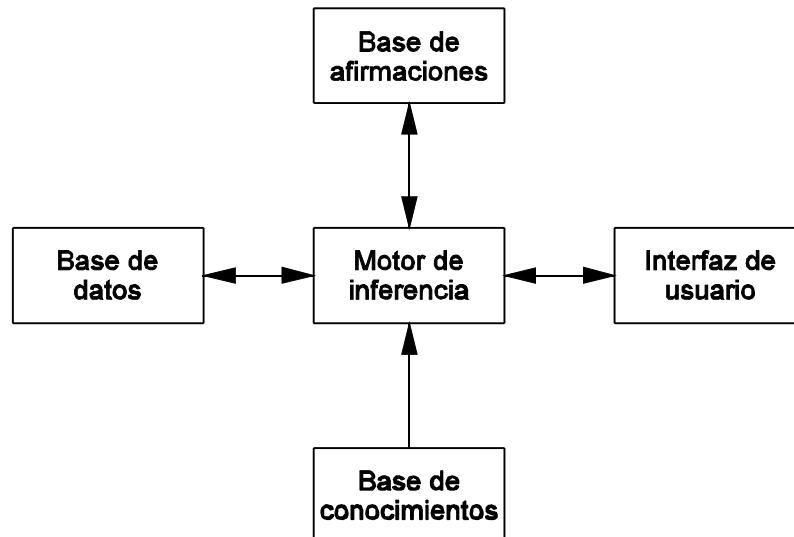
3.23.

## TEMA 6

### REGLAS.

#### 6.1. Componentes básicos de los SBR.

- Un sistema basado en reglas (SBR) consta de 5 elementos básicos:



- |                                 |   |
|---------------------------------|---|
| · <u>Motor de inferencia:</u>   | <p>A veces se llama “<u>intérprete de reglas</u>”.</p> <p>Es el elemento central del sistema.</p> <p>Coordina la información de los demás elementos.</p> <p>Envía los resultados de la inferencia al lugar oportuno.</p>  |
| · <u>Base de conocimientos:</u> | <p><u>Contiene las reglas.</u></p> <p>A veces también contiene las afirmaciones <u>iniciales</u> (si las hay).</p> <p>Es específica del dominio: medicina, geología, etc.</p>   |
| · <u>Base de afirmaciones:</u>  | <p>También se llama “<u>memoria de trabajo</u>”, o “<u>base de hechos</u>”.</p> <div style="border-left: 1px solid black; padding-left: 10px;"> <p>Contiene</p> <ul style="list-style-type: none"> <li>- Las afirmaciones iniciales que haya en la base de conocimientos.</li> <li>- Afirmaciones que se extraen de la base de datos.</li> <li>- Afirmaciones que el usuario introduce como datos.</li> <li>- Afirmaciones que el motor de inferencia infiere a partir de las demás afirmaciones al aplicar las reglas.</li> </ul> </div> |
| · <u>Interfaz de usuario:</u>   | <p>Solicita al usuario la información necesaria.</p> <p>Muestra los resultados de la inferencia.</p> <p>Ofrece explicaciones sobre cómo y por qué está funcionando el sistema.</p>  |
| · <u>Base de datos:</u>         | <p>Almacena información puntual.</p> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div>Ejemplo:</div> <div>El paciente tiene 36 años.</div> <div>Pesa 75 Kg.</div> </div> <div style="display: flex; justify-content: space-between; margin-top: 10px;"> <div></div> <div>Mide 1,73 m.</div> <div>Padeció fiebre reumática.</div> </div>  |

- En los sistemas expertos convencionales, el motor de inferencia no modifica la base de conocimientos.

Por este motivo, en el diagrama, la flecha que relaciona estos dos bloques es de un solo sentido.

Si el sistema fuese más avanzado y tuviese capacidad de aprendizaje, entonces esa flecha tendría que ser de doble sentido.

- El diagrama dibujado muestra la estructura genérica de un SBR. Pero puede ocurrir que no se necesite la base de datos, o que haya un módulo que recoja información de sensores y controle actuadores, o que no exista interfaz de usuario (control no supervisado).

- El motor de inferencia es un programa de ordenador independiente del dominio de aplicación del sistema.

Por tanto, se puede sustituir una base conocimientos por otra, siempre que ambas tengan la misma sintaxis.

## 6.2. Estructura de las reglas.

### 6.2.1. Antecedente y consecuente:

- Las reglas pueden escribirse:  $A \rightarrow C$  (si A, entonces C)

- Antecedente: · También se llama “parte izquierda”.

· Contiene las cláusulas que deben cumplirse para que la regla pueda ejecutarse o evaluarse.

- Consecuente: · También se llama “parte derecha”.

· En interpretación declarativa: indica las conclusiones que se obtienen de las premisas.

· En interpretación imperativa: indica las acciones que debe realizar el sistema cuando ejecuta la regla.

- Elementos que pueden aparecer en una regla:

· Hipótesis: | Ejemplos: hielo\_en\_la\_carretera, diabetes, hay\_becarios, etc.

| Cada hipótesis tendrá asociado un valor de verdad en cada momento.

· Dato: | Ejemplos: nivel\_de\_gasolina, dedicación, edad\_del\_paciente.

| Cada dato puede tomar cierto tipo de valores: numéricos, verdadero/falso, encendido/apagado, etc.

| Cuando se usan marcos, los datos pueden corresponder a los campos de las instancias. Por ejemplo:

Pedro\_García, edad  
└──┬──┘  
Instancia Campo

· Relación de comparación: | Se establece entre dos datos, o  
| entre un dato y un valor.

| Cada relación de comparación es cierta o falsa en un momento dado.

| Ejemplos: “nivel\_de\_gasolina = 8”

| “temperatura\_exterior > 0”

· Relación de pertenencia: | Se establece entre una instancia y un marco.

| Ejemplo: “Pedro\_García ES paciente”

| “x ES y” se considera cierta cuando x es una instancia del marco y.



- Cláusula:
 

Puede ser	1) Una <u>hipótesis</u> . 2) Una <u>relación</u> (de comparación o de pertenencia). 3) O la <u>negación, conjunción o disyunción de otras cláusulas</u> .
-----------	---

- Según el consecuente pueden darse 3 tipos de acciones:

- Afirmar.
- Retratar.
- Actuar: el sistema enviará una orden a los actuadores con los que está conectado.

- Ejemplo de regla:

SI temperatura_interior > temperatura_deseada  temperatura_deseada > temperatura_exterior	Antecedente: precedido de la palabra clave SI.
ENTONCES ACTUAR encender_ventilador  AFIRMAR ventilador=encendido	Consecuente: la palabra clave AFIRMAR puede no aparecer.

Entre las distintas condiciones (en el antecedente), y entre las distintas acciones (en el consecuente), se supone que hay una conjunción Y implícita.

### 6.2.2. Uso de variables en las reglas:

Con lo visto hasta ahora, habría que definir una regla particular para cada individuo o cada elemento concreto. Esto haría las reglas impracticables.

Por eso se hace imprescindible introducir variables en las reglas.

Ejemplo: “Todos los catedráticos son doctores.”

En lógica de predicados:  $\forall x (\text{catedrático}(x) \rightarrow \text{doctor}(x))$

Con reglas:

(A)	SI ?x ES catedrático
	ENTONCES ?x ES doctor
(B)	SI ?x.categoría = catedrático
	ENTONCES ?x.titulación = doctor

### 6.2.3. Comentarios adicionales:

- Datos univaluados: sólo admiten un valor. Si se les asigna un valor nuevo se elimina el anterior.

Ejemplo: “Edad = 28”

Datos multivaluados: admiten más de un valor a la vez.

Ejemplo: “Pedro\_García.Enfermedad\_anterior = infarto\_de\_miocardio”

no eliminará la afirmación siguiente, si ya estaba almacenada:

“Pedro\_García.Enfermedad\_anterior = apendicitis”

- Los SBR comerciales pueden indicar la falsedad de las afirmaciones de forma distinta. Unos lo indican explícitamente y otros, por ejemplo, consideran que una proposición que no esté en la base de afirmaciones, y que no pueda deducirse de la información disponible, es falsa (axioma del mundo cerrado).

## 6.3. Inferencia.

### 6.3.1. Comparación de patrones:

- Patrón:

- Cláusula con variables.
- Equivale a un conjunto de afirmaciones.
- Ejemplo:

“?x ES profesor”	representa	“Pedro_García ES profesor”,
		“Antonio_Pérez Es profesor”,
		etc.

- Para que se ejecute una regla, debe cumplirse su antecedente.

Para que se cumpla el antecedente deben cumplirse, a la vez, todas las cláusulas que lo compongan.

- Casos posibles:

#### 1) Cláusula sin variables:

- Si contiene una hipótesis, el antecedente se cumple cuando en la base de afirmaciones hay una afirmación que coincide con ella.
- Si contiene una relación, se cumplirá cuando los valores de los datos se ajusten a esa relación.
- Ejemplo: SI hielo\_en\_la\_carretera  
velocidad > 70  
ENTONCES recomendación = reducir\_la\_velocidad

#### 2) Una variable aparece por primera vez en esa regla:

- Primero se asigna un valor a dicha variable.
- Si el patrón obtenido coincide con uno de los elementos existentes en la base de afirmaciones, entonces la cláusula se cumple.
- Ejemplo: La base de afirmaciones (BA) contiene: “Pedro\_García ES catedrático”.

Y además tenemos esta regla:

SI ?x ES catedrático

ENTONCES director\_dpto\_es\_catedrático

Con esta situación, la siguiente asignación consigue que la regla se ejecute: “?x ≡ Pedro\_García”

Si en la BA no hubiese afirmaciones de ese tipo, diríamos que *la regla ha fracasado*.

3) Una variable aparece por segunda vez (y sucesivas) en una regla:

- Necesariamente, esa variable ya tiene asignado un valor: se asignó la primera vez que apareció.
- Ejemplo: SI ?x ES profesor

?x.área\_de\_conocimiento = ccia

ENTONCES ?x.puede\_enseñar = teoría\_de\_automatas

Si en la BA existe la afirmación “Antonio\_Pérez ES profesor”, la primera cláusula (primera aparición de la variable) se satisface con la asignación “?x ≡ Antonio\_Pérez”.

La segunda cláusula se satisface si “Antonio\_Pérez.área\_de\_conocimiento” es ccia.

En este caso la regla se ejecuta, y la afirmación “Antonio\_Pérez.puede\_enseñar = teoría\_de\_automatas” pasará a la BA.

- Podría darse el caso de tener más de una asignación posible para la variable.

Entonces, la regla se ejecutará una vez por cada asignación posible.

- Estas son las operaciones que hace el motor de inferencia para saber si una regla puede ejecutarse.

### 6.3.2. Tipos de encadenamiento:

- El motor de inferencia debe determinar cuál de las reglas existentes debe ser examinada.

Este proceso de búsqueda y selección de reglas puede ser de dos tipos:

- Encadenamiento hacia adelante o basado en datos:

- \* Se da cuando la información introducida en el sistema hace que se ejecute una regla, y la conclusión obtenida permite que se ejecuten otras reglas.
- \* Este encadenamiento ejecutará todas las reglas posibles en función de la información disponible.
- \* No es muy eficiente.

- Encadenamiento hacia atrás o basado en objetivos:

- \* Consiste en **buscar** una regla que permita obtener una determinada conclusión (objetivo).
- \* Suele solicitar al usuario la información que no ha podido deducir, para así continuar el proceso.
- \* Lleva implícito un proceso de búsqueda, por lo que es más específico.
- \* Al ser más específico, resulta más eficiente.

- Algunas herramientas sólo permiten encadenamiento hacia adelante, otras sólo hacia atrás, y otras ambos.

### 6.3.3. Dependencia reversible e irreversible:

- Una ventaja de los SBR es que permiten el razonamiento no monótono:

se pueden retractar afirmaciones anteriores como consecuencia de nuevas inferencias.

- Hay que definir qué ocurre si una afirmación que se retracta ya había sido utilizada para obtener otras conclusiones.

Para esto, al definir una regla, hay que indicar el tipo de dependencia:

· Dependencia reversible:

Si en un momento dado se cumple la premisa, el consecuente pasa a la BA.

Si se retracta la afirmación del antecedente, debe retractarse también el consecuente.

Ejemplo: “SI bombilla\_encendida  
ENTONCES habitación\_iluminada”

· Dependencia irreversible:

Cuando sucede que, si se retracta la afirmación del antecedente, la afirmación del consecuente debe permanecer en la BA una vez que ha llegado allí.

Ejemplo: “SI bombilla\_encendida  
ENTONCES película\_velada”

Evidentemente, una vez que se haya velado la película (por estar la bombilla encendida), continuará velada (esta afirmación debe seguir en la BA) aunque retractemos la primera afirmación.

- El SBR debe registrar, junto a cada afirmación, la forma en que ha sido obtenida.

De esta forma, en función del tipo de reglas que hayan intervenido en su obtención, se sabrá si la afirmación debe ser mantenida en la BA o eliminada de la misma.

## 6.4. Control del razonamiento.

- Este control determina qué regla ejecutar primero cuando hay varias disponibles.

- El control del razonamiento es necesario por:

1) Contenido de la inferencia: a veces las conclusiones pueden depender del orden en que se apliquen las reglas.

Deben activarse antes las reglas más específicas que las generales.

2) Eficiencia: elegir la regla adecuada conducirá rápidamente a una solución.

3) Diálogo generado: el sistema no debe preguntar al usuario la información que puede deducir por sí mismo.

El orden en que se solicite la información debe ser razonable y no “aleatorio”.

### 6.4.1. Mecanismos sencillos para controlar el razonamiento:

A) Ordenar las reglas en la base de conocimientos.

- Se colocarán en primer lugar aquellas que deseamos se examinen primero.

- Es poco elegante.

- En una base de conocimientos grande, el mantenimiento (añadir o eliminar reglas) puede resultar complicado, pudiéndose obtener un flujo de razonamiento imprevisto.

- Sólo es aplicable en sistemas simples, que almacenen las reglas en una lista.

B) Ordenar las cláusulas cuidadosamente dentro de cada regla.

- Sólo es aplicable en sistemas que usen encadenamiento hacia atrás.

- Una vez que las cláusulas están ordenadas, si falla una de ellas, las demás cláusulas que aparezcan después, en la misma regla, no necesitan ser comprobadas.
- Se deben colocar primero aquellas cláusulas que tengan más probabilidades de fallar, con lo que se optimiza la búsqueda.

C) Añadir nuevas cláusulas relacionadas con el punto de inferencia en que nos encontremos.

- Válido para ambos tipos de encadenamiento.
- Permite dividir la resolución del problema en varias etapas.

#### 6.4.2. Control de las agendas:

- Recordemos que una instanciación de una regla es una asignación de variables que satisface esa regla.

- Agenda: · Lista de instancias.

· Puede tener forma de pila o de cola.

- Las herramientas más avanzadas realizan una búsqueda de todas las reglas que están listas para ser ejecutadas y las guardan en una agenda.

- Algunas herramientas permiten tener varias agendas, y escoger un tipo de estructura (pila o cola) para cada una de ellas.

- En principio, una pila o una cola establece el orden en que entran y salen sus elementos.

Este orden puede alterarse asignando una prioridad a cada regla.

Cuando una instancia de una regla entra en una agenda, se sitúa en ella en función del orden que indique su prioridad.

#### 6.4.2.b. Conjuntos de reglas:

- Permiten activar / desactivar, en bloque, a grupos de reglas.

- Estructuran la base de conocimientos, mejorando la eficiencia.

#### 6.4.3. Metarreglas:

- Son reglas destinadas a razonar sobre otras reglas.

- Lo que se pretende es estimar qué reglas pueden ser más relevantes para el problema en cuestión, de manera que sean examinadas primero.

- Hay metarreglas dependientes del dominio, si en ellas aparecen conceptos propios de un campo concreto. Si no aparece ese tipo de conceptos, serán independientes del dominio.

- Las metarreglas son uno de los medios utilizados para guiar las búsquedas heurísticas de reglas, de manera que el encadenamiento de esas reglas resulte lo más eficiente posible.

#### 6.4.4. Otros métodos de control del razonamiento:

- Un sistema basado en reglas debe tener dos características:
  - Sensibilidad: Debe procesar con prontitud la nueva información que recibe.
  - Estabilidad: La llegada de información poco relevante no debería cambiar excesivamente la línea de razonamiento.
- Como ambas cosas se contraponen, hay que buscar mecanismos que logren un compromiso entre ellas.
  1. Refractariedad: Se impide que una regla se ejecute repetidamente si no se ha introducido nueva información.
  2. Actualidad: Examinar primero aquellas reglas que se apoyan en la información más reciente.
  3. Especificidad: Cuanta más información tiene en cuenta una regla, más específica es.  
Se trata, por tanto, de aplicar primero las reglas más específicas.  
Si nuestra herramienta no soporta la especificidad de las reglas, deberemos recurrir a otros mecanismos, como la asignación de prioridades.

#### 6.5. Explicación del razonamiento.

- El primer sistema experto capaz de explicar su razonamiento fue MYCIN.
  - Realiza inferencia mediante encadenamiento hacia atrás: cuando le falta un dato y no puede deducirlo a partir de otros, pregunta al usuario.
  - El usuario puede pedir explicaciones: WHY (¿por qué ese dato?) y HOW (¿cómo ha obtenido la conclusión?).
- Los sistemas actuales funcionan casi igual: se ha avanzado poco en la explicación del razonamiento.
- Debido a que la forma de explicar el razonamiento al estilo de MYCIN está muy ligada al encadenamiento de reglas:
  - a) Es muy útil para el diseñador del sistema experto.
  - b) Resulta insuficiente y artificiosa para el usuario medio.

#### 6.6. Tratamiento de la incertidumbre.

- En el mundo real existe mucha incertidumbre:
  - Información | imprecisa,  
                  | incompleta,  
                  | errónea.
  - Modelo | incompleto  
           | inexacto.
  - etc.
- Los dos métodos principales para abordar este problema mediante reglas son
  - Factores de certeza.
  - Lógica difusa.

### 6.6.1. Factores de certeza de MYCIN:

- Supongamos una regla sencilla: SI  $e$  ENTONCES  $h$
- Factor de certeza,  $FC(h, e)$ : indica la fiabilidad con que podemos aceptar la hipótesis  $h$  en el caso de tener la evidencia  $e$ .
- Los factores de certeza, en MYCIN, se obtuvieron a partir de estimaciones subjetivas de los médicos participantes en el proyecto.
- Si hay varias reglas que aportan evidencia a favor o en contra de una hipótesis, se necesita una fórmula que combine los factores de certeza correspondientes.
- El modelo de los factores de certeza tuvo muchas críticas, pues podía producir resultados contrarios al sentido común y a la teoría de la probabilidad.

### 6.6.2. Lógica difusa en SBR:

Ejemplos:

1º) Aquí aparecen tres conceptos difusos (subrayados):

SI curva ES muy cerrada

velocidad ES alta

ENTONCES ACCIÓN frenar moderadamente

2º) Esta regla está cualificada globalmente por un modificador difuso (subrayado):

Generalmente (SI  $?x$  ES deportista\_alta\_competición

ENTONCES  $?x$  padece hipertrofia\_v\_i)

## 6.7. Valoración.

### 6.7.1. Comparación con los programas basados en comandos:

- Programación imperativa: Está basada en comandos.  
Se dice lo que debe hacer el ordenador.
- Programación declarativa: Se indica al ordenador cuál es el conocimiento que debe aplicar.
- Ventajas de la programación basada en reglas frente a la basada en comandos:

1) Flexibilidad:

- Los comandos tienen carácter secuencial: el flujo de la ejecución es muy rígido.
- Las reglas tienen carácter modular: pueden ser añadidas o quitadas de la base de conocimientos sin preocuparse por el orden.

2) Dado que en los SBR el uso de variables está basado en la comparación de patrones, cuando una variable aparece en el consecuente, se pueden obtener conclusiones diferentes.

3) Los SBR permiten dependencia reversible e irreversible.

Un comando IF-THEN nunca podrá deshacer una acción, aunque deje de cumplirse la condición que contenía.

4) Capacidades avanzadas de los SBR:

- Explicar su razonamiento.
- Reorganizar el conocimiento que posee.
- Aprender de sus errores.

5) Tratamiento de la incertidumbre:

Los comandos IF-THEN no admiten grados de verdad intermedios. Las reglas sí.

- Inconvenientes de los SBR frente a la programación basada en comandos:

1) Eficiencia de los programas basados en comandos, puesto que no necesitan búsquedas, sino que recurren a algoritmos (mucho más rápidos).

2) Economía de recursos:

Un SBR, por muy sencillo que sea, siempre necesita un motor de inferencia completo, lo que obliga a consumir mucha memoria.

3) Las tareas de bajo nivel siempre son más eficientes si se hacen con comandos.

- NO es posible, ni aconsejable, prescindir de la programación algorítmica.

#### 6.7.2. Comparación con la lógica de predicados:

- Las reglas no admiten el cuantificador existencial ( $\exists$ ).
- Esto reduce su expresividad.
- Sin embargo, las reglas son más eficientes (consumen menos recursos de espacio y tiempo).
- Las reglas pueden tratar la incertidumbre, y se acercan a las lógicas modales y al razonamiento no monótono.

#### 6.7.3. Crítica de los SBR:

- En los sistemas expertos reales se aprecia que las reglas no son absolutamente modulares: al añadir nuevas reglas los resultados pueden ser imprevisibles.

- Además hay que tener mucho cuidado al ordenar las cláusulas dentro de las reglas. Esto implica que debe tenerse muy claro el flujo de la inferencia. De aquí que no resulten tan distintos de la programación imperativa como parece a priori.

- Por otra parte, el tratamiento de la incertidumbre que hacen los SBR es muy discutido.



## 6.8. Expresividad y tratabilidad.

Generalmente, al aumentar la expresividad, suele disminuir la tratabilidad, y viceversa.

La expresividad es necesaria para plantear un problema.

La tratabilidad es necesaria para resolverlo.

Los SBR son más expresivos que la lógica de proposiciones, pero menos expresivos que la lógica de predicados.

## PROBLEMAS:

Como repaso “rápido” de todo el tema, hacer los siguientes ejercicios del libro de problemas:

4.1: Inferencia con encadenamiento hacia adelante y hacia atrás.

4.3: Encadenamiento hacia atrás usando árboles Y/O para representarlo.

4.4: Uso de variables.

4.9: Instanciación de reglas.



# TEMA 7

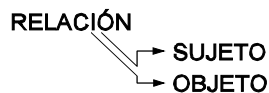
## REDES ASOCIATIVAS.

(Leer el índice del libro de texto para este tema. Da una visión global y muy esquemática, que aquí es muy conveniente)

### 7.1. Grafos relacionales.

#### 7.1.1. Modelo de memoria semántica, de Quillian:

- Intenta representar el significado de las palabras de forma parecida a como aparecen en un diccionario.
- Fue pensado para la lengua inglesa y es muy dependiente del idioma.
- Cada definición se encuadra en un plano.
- La palabra a definir: · Se encierra en un óvalo.
  - Se llama nodo-tipo.
- El resto de las palabras: · Constituyen la definición.
  - Se llaman nodos-réplica.
- Se utilizan 6 tipos de enlaces para unir unas palabras con otras:
  - 1.- Subclase: une un nodo-tipo con la clase a la que pertenece.
  - 2.- Modificación: une dos nodos-réplica. El segundo modifica el alcance del primero.
  - 3.- Disyunción (O).
  - 4.- Conjunción (Y).
  - 5.- Propiedad: indica que un sujeto guarda relación con un objeto.



- 6.- Referencia al tipo: es como los hiperenlaces. Van de un nodo-réplica a un nodo-tipo igual, pero en otro plano.
- Variables: · Representan conceptos que aparecen en otra parte del mismo plano.
    - Podrían sustituirse por enlaces que apuntaran a esos elementos. Pero esto resultaría menos claro.
  - Ver ejemplo en la página siguiente.
  - El programa de Quillian permitía comparar 2 palabras (comparar es un modo de inferencia).

Ejemplo:

Comparar: Planta, Vivo/a.

1ª intersección: Vivo/a.

- Una Planta-1 es una estructura viva.

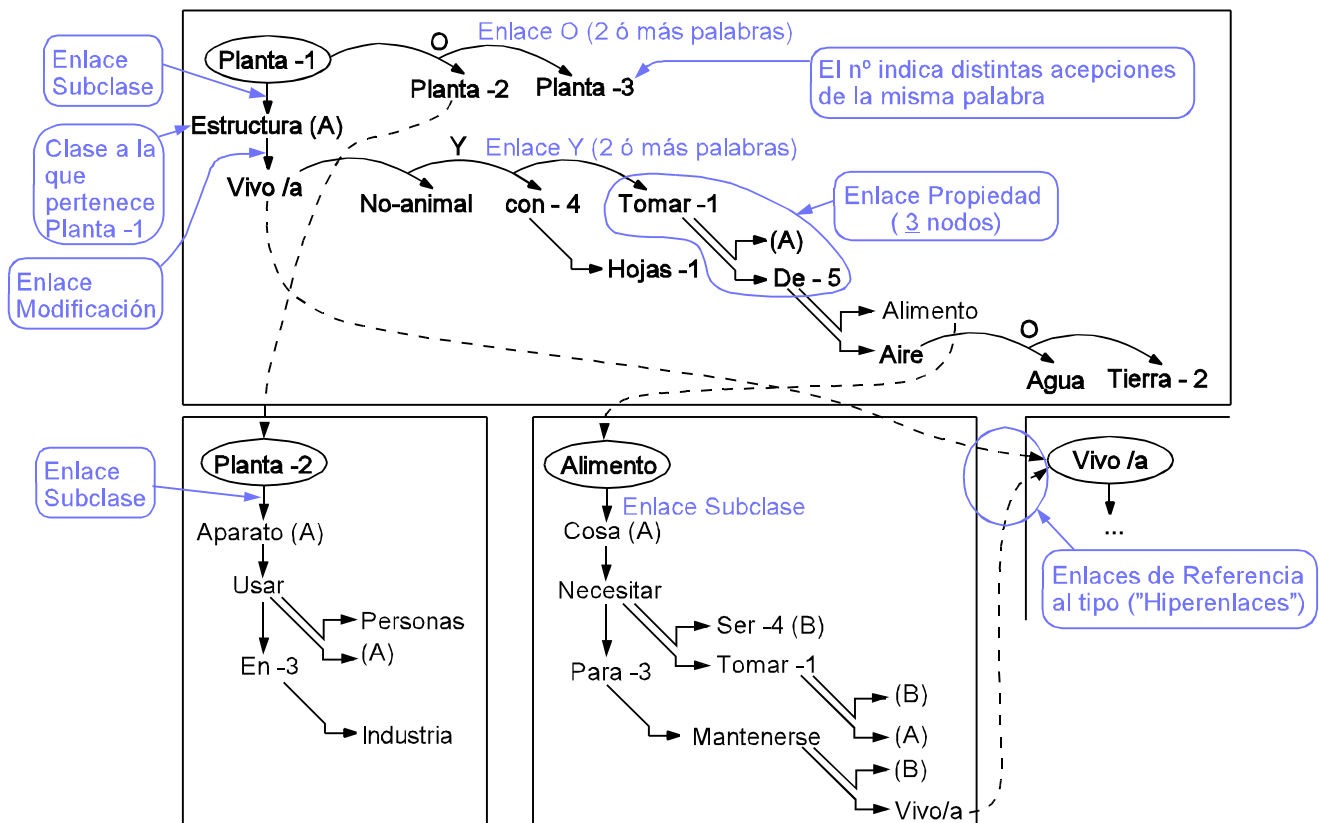
2ª intersección: Vivo/a.

- Una Planta-1 es una estructura viva que toma-1 alimento del aire, agua o de la tierra-2. Este alimento es una cosa que un ser-4 necesita tomar-1 para mantenerse vivo/a.

El primer camino (1ª intersección) consta de 3 enlaces en el grafo de la página siguiente.

El segundo camino tiene 12 enlaces, de los cuales, los dos primeros coinciden con el camino anterior.

Para seguir estos caminos hay que fijarse en los enlaces de referencia al tipo (“hiperenlaces”).



**Ejemplo de uso del modelo de memoria semántica de Quillian**

### 7.1.2. Sistema Scholar, de Carbonell:

- Era un sistema de enseñanza asistida por ordenador.
- Su base de conocimiento estaba estructurada en forma de red.
- El motor de inferencia (que generaba y respondía preguntas) era independiente del dominio representado en la red: servía para enseñar sobre materias distintas.

- Distingue entre los nodos que representan conceptos (o clases), p.e. PAÍS, y los nodos que representan ejemplos particulares (instancias), p.e. BRASIL.

- Define cada nodo con 2 elementos:
  - Clase a la que pertenece.
  - Lista de propiedades.

- Cada propiedad consta de:
  - Atributo.
  - Valor.

- NO suele considerarse como una red semántica.

### 7.1.3. Grafos de dependencia conceptual, de Schank:

- Schank no pretendía representar palabras, sino conceptos.

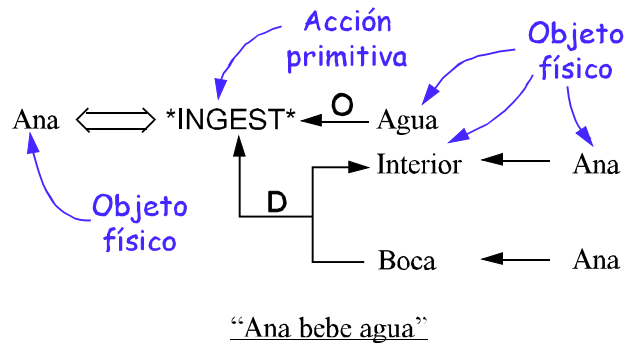
- Intenta representar cualquier frase con un número limitado de **primitivas**.
  - 6 categorías conceptuales.
  - 16 reglas sintácticas.
  - Varias acciones primitivas (podría ser suficiente con 12).
- El uso de un número limitado de primitivas, permite que el sistema sea eficiente.
- Categorías conceptuales:
  - Objeto físico (cosa o ser vivo).
  - Atributo del objeto físico.
  - Acción.
  - Atributo de la acción.
  - Tiempo.
  - Localización.
- Acciones primitivas:
  - PTRANS Transferencia física (cambiar de lugar un objeto).
  - ATRANS Transferencia abstracta (cambiar el poder, o la propiedad).
  - MTRANS Transferencia mental (decir, contar, comunicar, etc.).
  - PROPEL Empujar.
  - MOVE Mover un miembro de un animal.
  - INGEST Ingerir.
  - DO Hacer algo que no está determinado explícitamente.
  - Etc.
- Reglas sintácticas: indican el tipo de relación que hay entre los elementos de una frase.

Algunas de estas reglas son:

- 1) Objeto físico  $\longleftrightarrow$  Acción : Un actor actúa (p.e. Juan bebe).
- 2) Objeto físico  $\longleftrightarrow$  Atributo de objeto : Un objeto posee un atributo (p.e. casa grande).
- 3) Acción  $\xleftarrow{O}$  Objeto físico : Objeto de una acción (p.e. beber agua).
- 4) Acción  $\xleftarrow{R}$ 
  - Objeto físico
  - Objeto físico
 : Recepción de una acción (p.e. Juan transfiere (algo ) a Pedro).
- 5) Acción  $\xleftarrow{D}$ 
  - Objeto físico
  - Objeto físico
 : Dirección de un objeto en acción.
- 6)
 
$$\begin{array}{c} X \\ \uparrow \\ \uparrow \\ \uparrow \\ Y \end{array}$$
 : X ha causado Y.
- 7) Objeto 1  $\xleftarrow{\quad}$  Objeto 2 : Objeto 2 es parte de Objeto 1, o bien Objeto 2 es poseedor de Objeto 1.

- Modificadores de enlace (algunos de ellos):
  - p (pasado).
  - c (condicional).
  - f (futuro).
  - / (negación).
  - <sin modificadores> (presente).
  - ? (pregunta).

Ejemplo:



- Ventajas de usar primitivas:
    - Determinan unívocamente la representación del conocimiento.
    - Permiten construir un intérprete para realizar inferencias.
- Para esto es imprescindible que el número de primitivas sea limitado.

- Tipos de inferencias posibles con los grafos de dependencia conceptual de Schank:

- Establecer las condiciones:

Por ejemplo, de "Juan comió un filete" se infiere que:

- A) Juan existía.
- B) El filete existía.
- C) Juan y el filete estuvieron en contacto físico alguna vez.

- Hallar las causas que motivaron algo:

Por ejemplo, de "Juan pidió el libro a María" se infiere que Juan quería leer ese libro.

- Deducir el resultado de las acciones:

De "Juan comió un filete" se infiere que el filete dejó de existir.

- El modelo de Schank supone un avance respecto al de Quillian.

- Inconvenientes de usar un conjunto limitado de primitivas:

- Las primitivas no tienen porqué ser universales. Algunas frases significan cosas distintas según el idioma.
- Los grafos conceptuales requieren una descripción demasiado detallada de las acciones:  
Frases muy sencillas producen grafos complicados.
- ¿Cuántas y cuáles deben ser las primitivas?
- No siempre se debería centrar la representación en torno al verbo.

#### 7.1.4. Problemas de los grafos relacionales:

- Los grafos relacionales no son capaces de representar la lógica de predicados de primer orden:

- Pueden representar el cuantificador existencial: "Juan tiene un amigo".
- Pero NO pueden representar el cuantificador universal: "Todo hombre tiene un amigo".

- Resulta difícil, o imposible, tratar la interacción entre más de dos proposiciones:

"Cuando es de noche y hay niebla, resulta peligroso conducir." (Cada verbo subrayado es una proposición).

## 7.2. Redes proposicionales.

- Las redes relacionales sólo permiten representar las relaciones que haya dentro de una frase: sujeto, verbo, objeto directo, etc.

- Las redes proposicionales pueden representar, además, relaciones entre frases.

- Un nodo de las redes proposicionales puede representar:

- Una proposición.
- Una frase entera.
- Un párrafo.
- Una historia completa.

- Estas redes pueden representar toda la lógica de predicados de primer orden.

- Dos casos concretos de redes proposicionales son:

- Redes de Shapiro.
- Grafos de Sowa.

En ambos se puede realizar una representación gráfica (grafos) o una representación lineal.

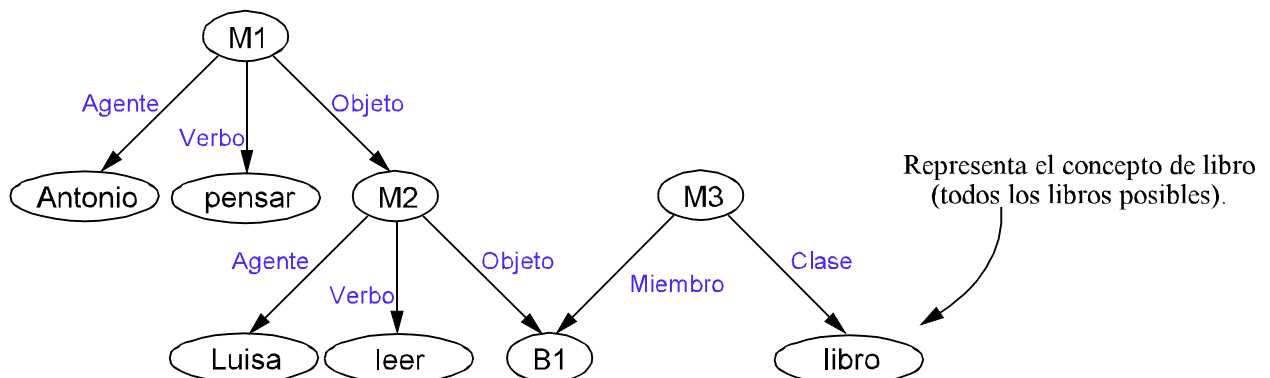
En la representación lineal:

- Nodos: entre corchetes.
- Etiquetas de los arcos: entre paréntesis.
- Ventajas:
  - a) Ocupa menos espacio que los grafos.
  - b) Es más fácil de manejar por un programa de ordenador.

### 7.2.1. Redes de Shapiro:

Ejemplo: “Antonio piensa que Luisa está leyendo un libro.”

- Representación gráfica:



El nodo B1 puede pareceros que sobra. Pero si se dibujase tan sólo una flecha desde M2 hasta libro, se estaría indicando que Luisa lee todos los libros existentes. De esta forma se “extrae” un *Miembro* de la *Clase* libro.

- Representación lineal:

[Pensar] -

——> (AGT) ——> [PERS: Antonio]

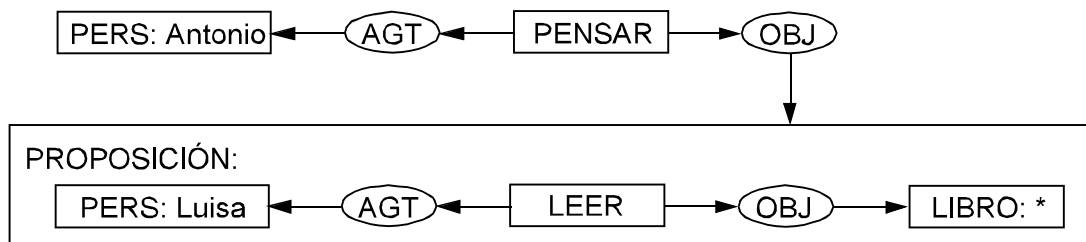
——> (OBJ) ——> [PROPOSICIÓN: [leer] -

——> (AGT) ——> [PERS: Luisa]

——> (OBJ) ——> [libro]]

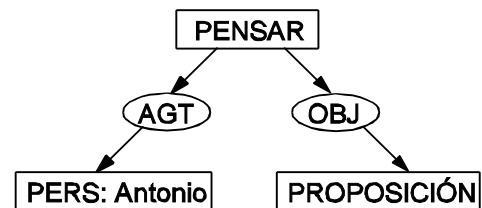
### 7.2.2. Representación mediante Grafos de Sowa:

Ejemplo: “Antonio piensa que Luisa está leyendo un libro.”



- Para ver la semejanza con la representación de Shapiro:  $\implies$

- PROPOSICIÓN se corresponde con el nodo M2 de Shapiro.
- Si encerramos todo esto en un rectángulo, obtendríamos una proposición, que se correspondería con el nodo M1



- Sowa indica explícitamente la clase a la que pertenece:

PERS: Antonio      Antonio pertenece a la clase PERSONA.

LIBRO: \*      No se conoce el libro concreto, pero pertenece a la clase LIBRO.

El asterisco se corresponde con ‘ $\exists$ ’: Existe un libro.

En la práctica el asterisco suele omitirse. Esto es lo que ocurre con los verbos (PENSAR, LEER).

- Leer los ejemplos de las páginas 277 y 278: expresión de distintas frases.

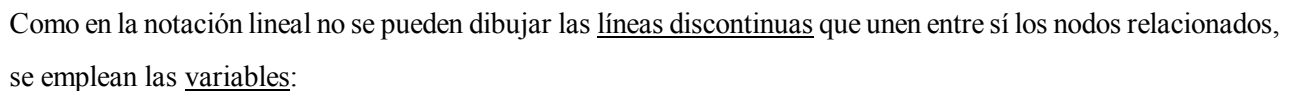
- La representación lineal es idéntica para las redes de Shapiro y los Grafos de Sowa.

- Uso del signo #:

- “El libro” se representa por “LIBRO: # ”.
  - El intérprete que convierte el lenguaje natural en un grafo debe averiguar cuál es el libro en cuestión.
  - En “PERS: # YO” deberá averiguarse quién pronuncia la frase.
  - En “DIA: # mañana” deberá averiguarse qué día es hoy.
- “un libro” se representa por “LIBRO: \* ” o bien “LIBRO”. No hace falta identificar ese libro.



- Ejemplo: “Quien tiene un coche, lo utiliza.”



(SI) —→ [PROPOSICIÓN: [POSEER] -  
(AGT) —→ [PERS: \*x]  
(OBJ) —→ [COCHE: \*y] ]  
(ENT) —→ [PROPOSICIÓN: [UTILIZAR] -  
(AGT) —→ [T: \*x]  
(OBJ) —→ [T: \*y] ] ]

$$\forall x \forall y ((\text{COCHE}(y) \wedge \text{POSEE}(x, y)) \rightarrow \text{UTILIZA}(x, y))$$

- 7.7

### 7.2.3. Inferencia en grafos de Sowa:

- Se realiza mediante 4 operaciones básicas:

- Restricción: Concreta más la información contenida en un grafo.

Puesto que añade información al grafo inicial, a veces se pierde veracidad en el grafo resultante.

- Generalización: Es la operación contraria a la restricción.

No añade información nueva, por lo que se mantiene la veracidad.

- Unión: Une dos grafos en uno solo.

- Simplificación: Se aplica siempre después de la Unión.

Simplifica el grafo resultante.

- Ver el ejemplo de la página 281.

A) [PERS] <— (AGT) <— [BEBER] —> (OBJ) —> [AGUA]

B) [PERS: Marta] <— (AGT) <— [BEBER] —> (OBJ) —> [AGUA]

C) [PERS: Marta] <— (AGT) <— [BEBER] —> (INSTR) —> [VASO]

D) [BEBER] -

(AGT) —> [PERS: Marta]

(OBJ) —> [AGUA]

(INSTR) —> [VASO]

El paso de A) a B) se ha hecho mediante Restricción (se ha añadido información).

La representación D) se ha logrado mediante la Unión y Simplificación de B) y C).

## 7.3. Redes de clasificación.

### 7.3.1. Extensión e Intensión:

- Significado Intensional de un término: es la definición de ese concepto.

Ejemplos: · Hombre: animal racional.

· Número primo: número natural mayor que 1 y sólo divisible por él mismo y la unidad.

- Significado Extensional de un término: es el conjunto de los elementos a los que se les puede aplicar ese concepto dentro de un universo determinado.

Ejemplos: · Hombre: habría que escribir el conjunto formado por todos los seres humanos de la Tierra.

· Número primo: {2, 3, 5, 7, ...}

### 7.3.2. Jerarquía de conceptos:

- Para representarla se usan grafos dirigidos acíclicos.

· Los nodos representan los conceptos de los distintos niveles.

· Los arcos unen los nodos de niveles distintos, que están relacionados entre sí. Un arco trazado de A a B indicará que el concepto A es más general que el B. Además, A aparecerá en un nivel superior a B.

· El concepto de nivel superior engloba a todos los demás. Se representa por *T*.

- En estos grafos no sólo aparecen objetos físicos, sino que pueden aparecer nodos que representen proposiciones completas. También los verbos se clasifican.

En resumen: cada nodo de los grafos de Sowa pertenece a un concepto en las redes de clasificación.

### 7.3.3. Mecanismos de herencia:

- La herencia es un tipo de inferencia.

Consiste en que un concepto reciba (herede) las propiedades de sus antepasados en la red jerárquica.

- Herencia estricta: Todos los conceptos que son descendientes de A, poseen obligatoriamente las mismas propiedades de A.

- Herencia por defecto: · Se supone que los descendientes de A poseen las propiedades de A, mientras no se indique lo contrario.

· Razonamiento no monótono: nueva información puede cambiar la conclusión.

- Ejemplo: el concepto “ser\_vivo” implica respirar. De aquí se infiere que el concepto “perro” también respira.

### 7.3.4. Sistemas taxonómicos / Sistemas asertivos:

- Conocimiento asertivo: · Sólo contiene afirmaciones (aserciones) concretas: “Juan tiene un libro”.

· No incluye definiciones de conceptos.

· No realiza clasificaciones jerárquicas.

- Conocimiento taxonómico: · Basado en la clasificación jerárquica de conceptos.

· Además define tales conceptos.

- Hay dos grupos de redes semánticas, en función del tipo de conocimiento al que den mayor importancia.

No obstante, los dos grupos de redes pueden incluir ambos tipos de conocimiento, aunque generalmente de forma separada: · Red de clasificación.

· Sistema para el tratamiento de proposiciones (aserciones).

- Los grafos relacionales, las redes proposicionales y las redes de clasificación, están orientadas principalmente a la comprensión y representación del lenguaje natural.

## 7.4. Redes causales.

- Se orientan, sobre todo, a problemas de diagnóstico: · Medicina.

· Diagnóstico de averías en aparatos.

- Nodos: se corresponden con variables (edad, sexo, fiebre, etc.).

- Arcos: se corresponden con relaciones de influencia, sobre todo de causalidad.

(X influye en el valor de Y, X causa (produce) Y).

#### 7.4.1. El sistema experto CASNET:

- Fue el primer sistema experto basado en una red causal.
- Servía de apoyo a los médicos en el diagnóstico y tratamiento del glaucoma.
- Permitía:
  - Razonamiento temporal: capacidad para seguir la evolución de la enfermedad.
  - Diagnóstico múltiple: un diagnóstico no ha de excluir a los otros, obligatoriamente.
  - Explicación de sus conclusiones: justificación de cómo y por qué obtiene los resultados.
- Estructura los nodos en 3 niveles (3 planos):
  - Observaciones: síntomas, signos y pruebas de laboratorio.
  - Estados patofisiológicos: alteraciones del funcionamiento normal.
  - Estados de enfermedad: enfermedades clasificadas en árbol.
- Posee varios tipos de enlaces, según en qué nivel estén.
- Es capaz de dar recomendaciones terapéuticas.

#### 7.4.2. Redes bayesianas (RB):

##### 7.4.2.1. Antecedentes:

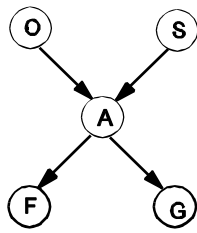
- En los problemas de diagnóstico se parte de ciertos hallazgos.  
Esos hallazgos dan lugar a evidencias.  
El objetivo es encontrar la hipótesis más **probable** a partir de las evidencias.
- Los primeros sistemas de diagnóstico, basados en métodos probabilísticos, utilizaban el método clásico de diagnóstico probabilista.  
Este es un método imposible de manejar debido al enorme número de valores de probabilidad que necesita.  
Para simplificarlo y hacerlo tratable, se introducen dos hipótesis:
  - Los diagnósticos son:
    - A) Exclusivos: dos de ellos no pueden ser ciertos a la vez.
    - B) Exhaustivos: no hay otro diagnóstico posible fuera de ellos.
  - Independencia condicional: para cada diagnóstico, la probabilidad de encontrar un hallazgo es independiente de que se hayan encontrado otros.
- Estas hipótesis NO se ajustan a la realidad.
- Por este motivo aparecieron las redes bayesianas:
  - Mantienen las ventajas del método probabilista clásico: basado en teoría matemática.
  - Desechan la hipótesis de diagnósticos exclusivos y exhaustivos.
  - La independencia condicional se aplica sólo a los efectos inmediatos de una alteración.
  - Consiguen:
    - A) Modularidad: facilita el cálculo de la probabilidad.
    - B) El problema resulta tratable.
    - C) Se ajustan a la realidad.

##### 7.4.2.2. El sistema experto DIAVAL:

- Es el primer sistema experto basesiano español.
- Ayuda a los médicos en el diagnóstico de enfermedades del corazón.

#### 7.4.2.3. Definición de red bayesiana:

- Ejemplo: red bayesiana para el paludismo.



O: País de origen del paciente.

S: Grupo sanguíneo del paciente.

A: El paciente padece paludismo.

F: El paciente tiene fiebre.

G: El paciente tiene gota gruesa.

Factores que influyen en el paludismo.

Síntomas del paludismo.

- Dado que las redes bayesianas son redes causales, las flechas indican, por ejemplo, que “O causa A”, o que “A causa F”.
- Por otro lado, en las redes bayesianas siempre hay que trabajar con probabilidades. Así, hay que expresar “O causa A, con una probabilidad p”.
- Las redes bayesianas cumplen una propiedad matemática: separación direccional.

La probabilidad de una variable X, una vez determinados los valores de sus padres, es independiente de los demás nodos-variables que no sean descendientes de X.

O expresado de otra forma: la probabilidad de una variable sólo depende de la probabilidad de sus padres y de sus hijos directos.

- Los nodos que no tienen padres reciben una probabilidad a priori (no depende de otras variables).

Los demás nodos tendrán una probabilidad que estará condicionada por los valores de sus padres.

- En el ejemplo, esto se representa así:

P(o)  
P(s)  
P(a | o, s)  
P(f | a)  
P(g | a)

$P(f | a, o, s, g) = P(f | a)$  (f sólo depende de a, y es independiente de o, s, g).

La probabilidad global de la red es el producto de las probabilidades parciales:

$$P(o, s, a, f, g) = P(o) \cdot P(s) \cdot P(a | o, s) \cdot P(f | a) \cdot P(g | a)$$

- Red bayesiana:
  - Es un **grafo** acíclico conexo.
  - Además, cada una de sus variables (nodos) tiene un valor de **probabilidad**.
  - La probabilidad de las variables cumple la **separación condicional**.

#### 7.4.2.4. Inferencia en redes bayesianas:

- Consiste en:
  - fijar en la red el valor de las variables conocidas, y después,
  - calcular la probabilidad de las variables cuyo valor es desconocido.
- En redes pequeñas se puede hacer con cálculos sencillos.

- En redes grandes se usan algoritmos:
  - Basados en paso de mensajes numéricos entre nodos, a través de sus enlaces.
  - Más eficientes.
  - El cálculo de la probabilidad se puede hacer de forma distribuida:  
Se asocia cada nodo a un procesador físico.

#### 7.4.3. Ventajas y limitaciones de las redes causales:

##### - Ventajas:

- Capacidad de **explicar** la cadena de causas que va desde la enfermedad diagnosticada, hasta los efectos observados.
- Pueden realizar 3 tipos de razonamiento:
  - 1) Abductivo: Busca la causa que explica los efectos observados.  
Razonamiento “hacia arriba”.
  - 2) Deductivo: Va desde las causas hacia los efectos.  
Razonamiento “hacia abajo”.
  - 3) Intercasual: La confirmación de una causa reduce la sospecha sobre otras.  
La evidencia disponible puede descartar todas las hipótesis menos una.  
Razonamiento “en horizontal”.

Estos tres tipos de razonamiento suelen usarse al mismo tiempo.

##### - Inconveniente:

- Limitación en su rango de aplicaciones:  
Rendimiento muy bueno en problemas de diagnóstico.  
Pero no son válidas para planificación, control, o diseño. En estos casos es mejor usar reglas.

#### 7.4.3.1. Ventajas e inconvenientes de las redes bayesianas:

- Ventajas:
  - Las mismas que cualquier red causal, y además:
  - Combinan las ventajas de los métodos probabilistas y los causales.
- Inconvenientes:
  - El mismo que cualquier red causal, y además:
  - Necesitan muchas probabilidades numéricas objetivas no disponibles, lo que lleva a utilizar estimaciones subjetivas.
  - Cuando aparecen bucles en la red, los cálculos se complican muchísimo.

## ESQUEMA RESUMEN:

Redes asociativas:	· Grafos relacionales	A) Memoria semántica de Quillian.
		B) Grafos de dependencia conceptual, de Schank.
	· Redes proposicionales	A) Redes de Shapiro.
		B) Grafos de Sowa.
	· Redes de clasificación: jerarquía y herencia.	
	· Redes causales.	
	Redes bayesianas.	

## PROBLEMAS:

Ejercicios recomendados, del libro de problemas, para un repaso rápido:

5.1: Red semántica de Quillian.

5.3: Grafos de dependencia conceptual, de Schank.

5.4: Grafos de dependencia conceptual, de Schank.

5.6: Red de Shapiro y Grafo de Sowa.

5.9: Notación lineal en redes proposicionales (Shapiro y Sowa).

5.10: Notación lineal en redes proposicionales (Shapiro y Sowa). Inferencia.

5.12: Red bayesiana.





# TEMA 8

## MARCOS Y GUIONES.

### 8.1. Concepto de marco.

#### 8.1.1. La propuesta de Minsky:

- El nombre original de los marcos era frame: se puede traducir de varias formas:

- Situación.
- Armazón o estructura básica usada para construir algo.
- Conjunto de circunstancias que rodean un suceso.
- Objetos en que se centra la atención de una cámara de cine.

En realidad, todos estos significados, juntos, definen la idea de marco.

- Los marcos están pensados principalmente para tareas de reconocimiento:

- Comprensión del lenguaje natural.
- Visión artificial.

- Un marco está formado por un conjunto de campos (slots).

Un campo puede ser otro marco.

- Ejemplo: Marco *casa*.

Algunos de sus campos serán habitaciones.

Cada habitación es, a su vez, un marco:

Marco *dormitorio*: contiene *cama* , *mesa de noche*, etc. como campos.

Marco *cocina*: contiene *nevera*, *fregadero*, etc. como campos.

Etcétera.

- Los marcos posibilitan el reconocimiento descendente o basado en expectativas.

Si estamos en una casa que no conocemos y encontramos una cama, al coincidir con un campo del marco *dormitorio* identificaremos la habitación de que se trata.

Los demás campos de este marco nos hacen pensar que en la misma habitación puede haber una *mesa de noche*, un *armario ropero*, etc.

- Se puede definir una misma entidad desde distintos puntos de vista.

Un hombre puede, a la vez, ser profesor, padre de familia, cliente de un banco, etc.

Para cada uno de esos aspectos se definirá un marco con unos campos concretos.

- Dado que Minsky sólo aportó ideas sobre los marcos, han aparecido muchas formas distintas de representar marcos y de utilizarlos.

#### 8.1.2. El sistema experto P.I.P. (Present Illness Program):

- Se usa para ayudar en el diagnóstico de enfermedades.

- Cada enfermedad se representa con un marco.
- Un campo identifica el marco, otro lo clasifica, otros representan síntomas, otros representan posibles diagnósticos, etc.
- Cada marco está en uno de 4 estados posibles:
  - Durmiente.
  - Semiactivo.
  - Activo.
  - Aceptado.
- El diagnóstico comienza con la recogida de información.
  - Inicialmente todos los marcos están durmientes.
  - Cuando se introduce en el programa un dato (síntoma) que esté contenido en uno o varios marcos, tales marcos se activan. Así pasan a considerarse como candidatos para el diagnóstico.
  - Los marcos que se encuentren relacionados con un marco activo, pasan a ser semiactivos. Así podrán ser evaluados cuando se reciba nueva información.

Esta forma de relacionar marcos evita la explosión del conjunto de hipótesis al crear un espacio de búsqueda limitado.

  - Ahora el programa evaluará las hipótesis generadas, para encontrar el marco (diagnóstico) que mejor explique la evidencia disponible.
  - Cuando la evidencia acumulada supera un umbral, el marco correspondiente pasa a ser aceptado.
  - Si todavía no se logra un diagnóstico, se continúa recopilando información, pero de un modo que es orientado por los marcos activos. Este proceso es cíclico hasta alcanzar un diagnóstico.

### 8.1.3. El lenguaje KRL (Knowledge Representation Language):

- Se quería que fuera la base para resolver problemas de IA.
- Objetivos básicos:
  - A) Representar el conocimiento de forma estructurada.
  - B) Eficiencia: incluye explícitamente información, para no tener que deducirla.
- Usa un mecanismo de razonamiento basado en la comparación o ajuste (Matching).
- Métodos para controlar el razonamiento:
  - Asignación de procedimientos a los marcos o sus campos.
  - Ajuste de la profundidad de procesamiento.
  - Gestión de agendas.
  - Niveles de prioridad.
  - Etcétera.
- Detecta errores “de programación”.
- Objetivos básicos del lenguaje KRL:
  - A) Representar el conocimiento de forma estructurada:
    - La información se agrupa en objetos o unidades.

· Los campos de una unidad pueden tener valores por defecto:

(1) Procedentes de la herencia de propiedades.

(2) Procedentes de prototipos.

· Un prototipo: \* Es una unidad.

\* Representa un elemento típico de una clase.

\* No se corresponde con un objeto concreto.

B) Eficiencia: incluso a costa de incluir información redundante.

- El mecanismo de razonamiento llamado matching compara los campos de un objeto conocido, con los de un marco.  
Cuantos más campos coincidan, mayor será la evidencia de que ese objeto pertenece a la clase definida por tal marco.

Se puede ajustar la distinta importancia que cada atributo tiene en la identificación de un objeto.

Esta comparación es flexible, ya que puede bastar un ajuste parcial. Recordemos que en la comparación de reglas, debían cumplirse todas las premisas para que la regla se ejecutara.

- Dentro de los métodos de control del razonamiento, destaca la asignación de procedimientos a los marcos o a sus campos.

Estos procedimientos pueden ser de dos tipos:

(1) Sirvientes: activados por el marco para alcanzar cierto objetivo.

(2) Demonios: se activan automáticamente en ciertas circunstancias.

- Otro método de control de razonamiento es el ajuste de la profundidad de procesamiento. Consiste en que el usuario puede determinar cuántos marcos van a ser activados.

#### 8.1.4. Herramientas basadas en marcos:

- La mayoría de las herramientas actuales permiten combinar marcos y reglas.

- Las herramientas actuales no suelen diferenciar entre marcos activos e inactivos.

- Actualmente los marcos se organizan de forma jerárquica: uso de la herencia.

- Se tiende a usar los marcos para construir sistemas expertos, en vez de orientarlos al reconocimiento de imágenes y del lenguaje natural.

## 8.2. Inferencia mediante marcos.

- Se empieza creando una red de marcos e instancias.

Marcos: · Representan conceptos.

· Heredan los campos de todos sus antepasados en la red.

Instancias: · Representan elementos dentro de las clases.

· Pueden pertenecer a un solo marco, o a varios.

· Heredan los campos y valores de todos los marcos a los que pertenece.

- En algunas herramientas se permite la herencia bidireccional: el valor asignado a una instancia puede convertirse en valor por defecto del marco al que pertenece.

### 8.2.1. Facetas:

- Definen las propiedades de un campo:

- Valor por defecto: \* Valor que toma el campo mientras no se indique otra cosa.
  - \* Se asigna al marco y lo heredan las instancias que se creen a partir de ese marco.
- Multivaluado: \* Indica si el campo es univaluado o multivaluado.
- Restricciones: \* Limitan el conjunto de valores que puede recibir un campo.
- Certeza: \* Credibilidad de valor asignado al campo.
- Facetas de interfaz: \* Texto, preguntas y mensajes.
  - \* Se destinan a la interacción con el usuario.
  - \* Aparecerán en la interfaz cuando sea necesario.
- Demonios: tienen especial importancia, por lo que se tratan en el apartado siguiente.
- Facetas definidas por el usuario (programador).

### 8.2.2. Demonios:

- Son funciones o procedimientos que se invocan automáticamente al realizar ciertas operaciones sobre el valor de un campo.

- Se usan para: · Actualizar los campos que dependen de otros.

- Actualizar la presentación gráfica del interfaz.
- Buscar en una base de datos la información requerida.
- Controlar dispositivos externos.
- Etcétera.

- Un campo puede tener asociados distintos tipos de demonios.

- Cada tipo de demonio se activa ante un suceso concreto.

- Tipos de demonios:

- De necesidad: se ejecuta cuando se necesita saber el valor de ese campo, pero no hay ningún valor asignado.
- De acceso: se ejecuta cada vez que se acceda a ese campo, aunque ya tenga asignado un valor.
- De asignación: se ejecuta cada vez que se asigna un valor a ese campo.
- De modificación: se ejecuta cuando varía el valor de ese campo.
- De borrado: se ejecuta al eliminar el valor de ese campo.

### 8.2.3. Puntos de vista:

- Es un mecanismo de control del razonamiento.

- Algunas herramientas permiten que un campo de una instancia posea varios valores, correspondientes a distintos puntos de vista.

### 8.3. Guiones.

#### 8.3.1. Planteamiento del problema:

Muchas veces una palabra o frase sólo adquiere su significado cuando se examina el contexto o situación en que es pronunciada.

- Este es el objetivo de los guiones: reconocer situaciones, para lograr la comprensión del lenguaje natural.
- Igual que un marco tiene campos que permiten reconocer objetos, un guión posee escenas para reconocer situaciones.
- Un guión puede considerarse como un tipo particular de marco:
  - Cada campo corresponde a un suceso.
  - Los campos-sucesos forman una secuencia.
- Así que un guión es una estructura de conocimiento que contiene una secuencia de acciones.
- Las acciones están unidas entre sí por una relación de causalidad: la realización de una acción permite que ocurra la siguiente.

#### 8.3.2. Representación del conocimiento:

- Un guión completo se compone de los siguientes elementos:
  - Escenas: \* Sucesos descritos en el guión.
    - \* Los sucesos están enlazados causalmente en forma de secuencia.
  - Roles y objetos: \* Corresponden a las personas y las cosas que intervienen.
    - \* Incluyen restricciones: qué personas u objetos pueden ser asignados a las variables.
  - Cabeceras: \* Activan el guión en el momento oportuno.
    - \* Una cabecera da nombre al guión.
    - \* Otras representan: Condiciones.
- Instrumentos.
- Lugares.
- Los guiones son una mezcla entre los grafos de dependencia conceptual y los marcos.

#### 8.3.3. Inferencia mediante guiones:

- Para comprobar que el sistema ha “comprendido” un texto escrito, se pueden hacer dos cosas:
  - A) Formularle preguntas concretas y ver si responde adecuadamente.
  - B) Pedirle que repita el mismo texto con otras palabras.

- Proceso de inferencia:

- 1.- **Seleccionar** el guión que mejor explica o se adapta a la historia que se analiza.

Si una palabra del texto analizado aparece en alguna cabecera de un guión (bien en la que le da nombre o en otras), ese guión se activa.

Normalmente se activan varios guiones, por lo que después habrá que seleccionarlos (o rechazarlos).

- 2.- Asignar las variables: se identifican los roles, objetos y lugares que intervienen en la historia.

Se hace por comparación de patrones para **reconocer** cada elemento.

Esto produce un guión “instanciado”.

- 3.- El guión “instanciado” permite obtener información que no aparecía explícitamente en la historia escrita: **predicción**.

Esto es la inferencia.

- Un problema, en español, es que muchas veces omitimos el sujeto, pero la máquina debe averiguar cuál es. Para ello puede buscarse la concordancia de género y número. Si no es suficiente, hay que acudir a la información contenida en los guiones activados.

- Otro tipo de inferencia: dar por supuesto que han ocurrido los demás sucesos que aparecen en el guión, aunque no estén de forma explícita en la historia escrita.

- Hay dos métodos para comprender un texto mediante guiones:

- 1.- Ascendente:
  - Analizar palabra por palabra.
  - Activa los marcos que mejor explican la información que va apareciendo.
- 2.- Descendente:
  - Selecciona un marco.
  - La información del texto que se ajusta a ese marco es asimilada, y el resto se ignora.
  - Se usa cuando sólo se quiere obtener la información más relevante del texto.
  - Es menos sensible a detalles que puedan perturbar la interpretación.
  - Puede despreciar información importante, sólo porque no se ajusta al guión seleccionado, lo cual es un inconveniente.

Normalmente se combinan el reconocimiento ascendente y el descendente.

#### 8.3.4. Ventajas, inconvenientes y extensiones de los guiones:

- Ventajas: Los guiones se desarrollaron como una extensión de los grafos de dependencia conceptual, de Schank.

Por lo tanto, representan el conocimiento sin depender de ningún idioma.

Las escenas serán idénticas, independientemente del idioma.

Los resultados de la inferencia se representan con grafos de dependencia conceptual, por lo que también son independientes del idioma.

Esto es muy útil en programas de traducción automática.

- Inconvenientes:
  - Rigidez del mecanismo de representación, ya que cada guión representa una secuencia fija de acciones.

- No es posible compartir información entre guiones.
- Incapacidad para expresar motivaciones e intenciones: ¿Qué motiva a alguien a hacer algo?, ¿Con qué intención hace tal cosa?.
- Extensiones: Paquetes de organización de memoria (MOPs):
  - También contienen una secuencia de escenas.
  - Pueden establecerse enlaces entre los distintos MOPs.  
Esto permite que compartan información.
  - Muy usados en aprendizaje y razonamiento basado en casos (RBC).

## 8.4. Comentarios.

### 8.4.1. Paradigmas en la utilización de marcos:

- Hay dos formas de usar los marcos:
  - A) Patrones para la comparación:
    - Partimos de cierta información, que puede ser parcial.  
Si esta información se ajusta a un marco, entonces, a partir de los demás datos que haya en ese marco, podemos deducir la presencia de otros elementos.
    - Así usan los marcos el sistema experto PIP y los guiones.
    - El razonamiento tiene 3 fases: Activación, reconocimiento y Predicción.
  - B) Almacenes de información:
    - Se distingue entre clases e instancias.
    - La organización se hace en forma de red jerárquica: uso de la herencia.
    - Todo esto permite que la información se guarde de manera estructurada.
    - Permite el razonamiento por defecto.
- El lenguaje KRL aprovecha ambos modelos.
- La mayor parte de las herramientas actuales sólo se ajustan al segundo modelo, lo que las aleja de la idea original de marcos que tenía Minsky.

### 8.4.2. Valoración:

- Los marcos representan el conocimiento de forma más estructurada que las reglas.  
La lógica tradicional dispersa el conocimiento entre las proposiciones, sin estructurarlo.  
A mayor estructuración:
  - Más eficiente es el razonamiento.
  - Más fácil resulta mantener la base de conocimientos.
- Los marcos permiten formas de razonamiento que no admite la lógica:
  - Razonamiento por defecto.

- Reconocimiento de entidades y situaciones: los marcos son más eficaces incluso que las reglas, pues pueden reconocerlas aunque sólo se disponga de información parcial.
- La lógica realiza muchas más inferencias de las deseadas, ya que no pueden discernir entre lo fundamental y lo superfluo en un problema dado.
- En la vida real, muchas veces, hay que realizar inferencias partiendo de datos inciertos e incompletos, incluso con el riesgo de estar equivocándonos. Pero de otra forma no podríamos deducir nada.  
La lógica no permite esto, porque sólo deduce aquello que es absolutamente cierto.
- No obstante cada método tiene sus virtudes y sus defectos, y cada uno se adapta mejor a unas necesidades.

## **PROBLEMAS:**

Ejercicios, del libro de problemas, recomendados para un repaso rápido:

- 6.1: Marcos. Organizar noticias.
- 6.2: Marcos. Resumir noticias.
- 6.4: Marcos y lógica de predicados: representación de una red semántica.
- 6.5: Estructurar información con marcos.
- 6.7: Marcos. Estructurar información. Demonios.
- 6.8.B: Marcos. Uso de los “puntos de vista”.
- 6.12: Guiones.
- 6.13: Guiones.



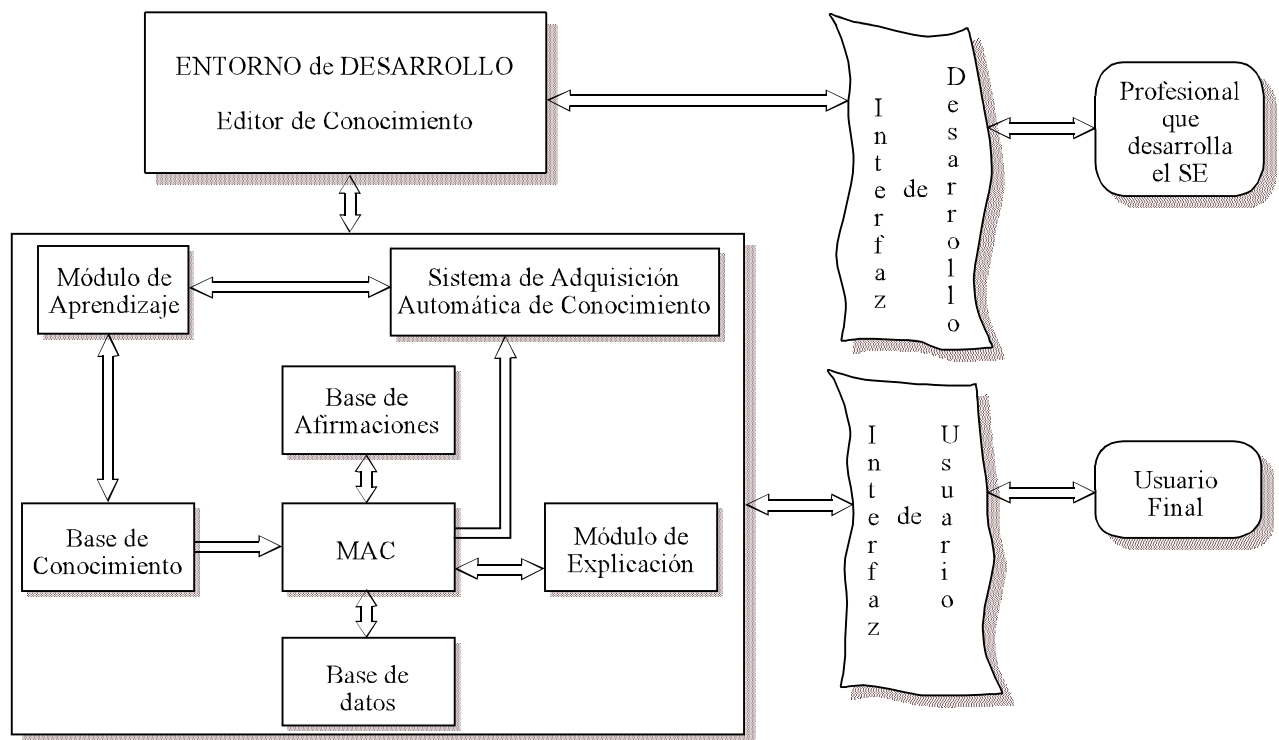
## TEMA 9

### SISTEMAS EXPERTOS.

#### 9.1. Concepto de S.E.

- Sistema experto (SE): programa de ordenador que codifica un modelo del conocimiento de un experto humano en un campo reducido.
- Características básicas de un SE:
  - Separación entre: \* Base de conocimientos.
    - \* Mecanismos de uso de esos conocimientos en inferencia.
  - Competencia en su campo.
  - Dominio reducido.
  - Capacidad de explicación.
  - Flexibilidad en el diálogo.
  - Tratamiento de la incertidumbre.
- Los SE son la forma más común de sistema basado en conocimiento (SBC).
- Los SE de primera generación eran sistemas basados en reglas.

##### 9.1.1. Estructura básica de un SE:



- Elementos básicos de un SE (coinciden totalmente con los de los sistemas basados en reglas vistos en el Tema 6):

1) Base de conocimientos (BC):

- Está compuesta por un conjunto de reglas, marcos o ambas cosas. También puede incluir una red neuronal.
- Contiene la representación modular y computable del conocimiento del experto humano en ese campo.

2) Mecanismos de aplicación del conocimiento (MAC):

- Si se usan reglas, este “bloque” se encarga de seleccionar las que deben ejecutarse en cada momento.
- Si se usan marcos, la inferencia se basará en la herencia de propiedades.

3) Base de afirmaciones o de hechos ciertos (BA):

- Es una memoria de trabajo (o memoria temporal).
- Aquí, el MAC:
  - \* Almacena las conclusiones transitorias que va obteniendo.
  - \* Busca las premisas que le permitirán obtener nuevas conclusiones.

4) Base de datos (BD):

- Es una base de datos convencional.
- Almacena información adicional propia de cada aplicación, que no se incluye en la BA ni en la BC.
- Se usa para almacenar y recuperar de forma eficiente dicha información.

5) Interfaz de usuario:

- Debe ser potente y cómodo de manejar.
- De la interfaz depende, muchas veces, la aceptación o rechazo de un programa.

- Existen, además, otros 3 módulos muy complejos de implementar y que sólo se encuentran en los Sistemas Expertos más avanzados:

1) Módulo de explicación.

2) Módulo de aprendizaje.

3) Módulo de adquisición automática de conocimiento.

### 9.1.2. Características de un Sistema Experto:

1.- Competencia en su campo: se trata de que sea experto.

- Debe resolver los problemas con la eficiencia y calidad de un experto humano.

2.- Dominio reducido: especialización.

- Para ser experto en algo hay que especializarse en esa materia.

3.- Capacidad de explicación:

- Debe ser capaz de explicar:
  - \* Qué método ha aplicado para resolver el problema.
  - \* Por qué ha aplicado ese método.
- Los resultados obtenidos hasta ahora son muy modestos, debido a la gran complejidad que supone.

4.- Tratamiento de la incertidumbre:

- En la mayor parte de las aplicaciones reales surge la incertidumbre.

5.- Flexibilidad en el diálogo:

- Es deseable que:
  - \* Acepte información en cualquier momento.
  - \* Realice preguntas adaptándose a cada situación particular.

#### 6.- Representación explícita del conocimiento:

- Precisamente, los Sistemas Expertos se han ideado para que posean una representación del conocimiento realmente explícita.

#### 9.1.3. Ventajas y limitaciones:

##### - Ventajas de los SE frente a los especialistas humanos:

- 1.- Permanencia: un experto humano puede morir o cambiar de empresa.
- 2.- Duplicación: un SE puede duplicarse muchas veces para usarlo en muchos lugares distintos a la vez.
- 3.- Fiabilidad: un SE no se ve influenciado por emociones, prejuicios, fatiga, etc.
- 4.- Rapidez: cuando un experto humano se encuentra con un caso poco frecuente, tiene que consultar fuentes de información adicionales (catálogos, informes anteriores, etc.), “perdiendo” mucho tiempo. Un ordenador no requiere ese tiempo extra.
- 5.- Bajo coste: crear un SE resulta caro, pero después se duplica, y el coste global se reduce muchísimo.

##### - Limitaciones de los SE:

- 1.- No tienen sentido común.
- 2.- Flexibilidad: los SE son muy rígidos (no pueden salirse de lo programado).
- 3.- Están lejos de trabajar en lenguaje natural.
- 4.- No tienen experiencia sensorial, o es muy limitada, como la visión artificial.
- 5.- Perspectiva global: un humano distingue rápida y fácilmente lo importante de lo superfluo al tratar cualquier tema.
- 6.- Falta de capacidad de aprendizaje.
- 7.- Los SE no pueden manejar conocimiento que no esté estructurado.
- 8.- No pueden realizar funciones genuinamente humanas, por ejemplo la creatividad.

- No obstante, se entiende que lo principal debe ser hacer que los Sistemas Basados en Conocimiento sean una extensión de la inteligencia natural.

## 9.2. Escenarios y funciones.

- Se han desarrollado SE para resolver problemas científico-técnicos en casi todos los campos.

El campo pionero y dominante es el de la medicina.

- Antes de desarrollar un SE hay que hacer un estudio de viabilidad.

- Dominio limitado.
- La tarea del experto humano se puede describir de forma:
  - \* Clara.
  - \* Completa.
  - \* Inequívoca.

- En todos los campos se realizan las mismas actividades básicas. Por eso se consideran las Tareas Genéricas (TG).
- Las bases de datos ayudan cuando hay que manejar grandes cantidades de datos, pero poco conocimiento.  
En el lado contrario están los SE: ayudan cuando hay que manejar pocos datos, pero aplicando muchísimo conocimiento.
- En la práctica, lo que se hace es integrar ambas cosas.
- Para poder usar un SE con eficacia, todo el entorno debe estar completamente informatizado. Por ejemplo, en un hospital: historias clínicas, gestión de pacientes, procedimientos diagnósticos, etc.

### 9.3. Tareas genéricas (TG).

#### 9.3.1. Catálogo de TG:

- Las tareas genéricas se pueden clasificar en 3 grupos:

- Análisis: Su objetivo es conocer el comportamiento de un sistema.
  - A) Identificación:
    - \* Monitorización de variables.
    - \* Diagnóstico de fallos del sistema.
    - \* Clasificación de tales fallos.
  - B) Predicción:
    - \* Del comportamiento del sistema.
    - \* De valores, usando aproximaciones.
- Modificación: Se pretende cambiar algún aspecto del sistema.
  - A) Reparación.
  - B) Control (se modifica algo para variar o corregir el comportamiento del sistema).
  - C) Supervisión.
  - D) Aprendizaje (al hacer que el sistema aprenda, lo estamos modificando).
- Síntesis: El objetivo es construir un sistema a partir de especificaciones, componentes y técnicas.
  - A) Diseño.
  - B) Configuración.
  - C) Planificación.
  - D) Modelado.

- Esta clasificación no es rígida, ya que pueden aparecer TG mezcla de varias de éstas.