4. BÚSQUEDA

Existe un subconjunto de problemas donde cada problema se puede definir mediante:

- Un objetivo
- Un estado inicial
- Un conjunto de acciones posibles

Sobre este tipo de problemas se pueden aplicar las **técnicas de búsqueda**.

Para resolver un problema con técnicas de búsqueda es necesario definir:

- Una forma de representar el los estados inicial, final e intermedios.
- Un conjunto de reglas compuestas por una condición de aplicación y una operación.
- Una estrategia de control para decidir el orden de aplicación de las reglas.

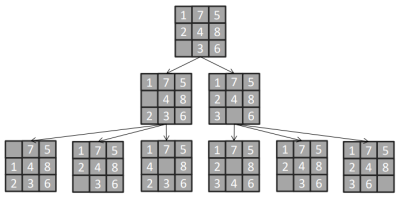
4.1 Concepto de búsqueda en IA

Búsqueda se entiende como un algoritmo en el que se busca un dato en un conjunto, donde el objetivo del proceso de búsqueda es **encontrar un cierto camino recorrido** hasta encontrar el dato, en otras palabras, el objetivo de un método de búsqueda es encontrar un estado (estado objetivo) en un espacio de estados.

No buscamos un resultado sino un recorrido. El objetivo es encontrar el camino de búsqueda.

El problema va a reflejarse en un conjunto de estados (iniciales y finales) y buscamos obtener el recorrido de inicio a fin.

- Se asocia la búsqueda al **espacio de estados**, para conocer qué secuencia de estados deben alcanzarse para finalmente lograr un estado objetivo bajo determinadas premisas.



Espacio de estados:

- Conjunto de todos los estados que pueden ser alcanzados aplicando operadores a partir del estado inicial.
- Conjunto de estados que pueden obtenerse si se aplicaran todos los operadores posibles a todos los estados que se fueran generando.

Es importante entender que el espacio de estados existe solo como una posibilidad, no está disponible ni se lo conoce.

- Cuando se genera el árbol se generan TODAS las posibilidades, teniendo en cuenta de **no generar un estado que ya está**.

- Un camino serán todos aquellos nodos que se visiten para llegar al nodo meta

Generalmente, la salida esperada de un método de búsqueda es el camino entre el estado inicial y el estado objetivo (camino de la solución).

Camino solución vs camino de búsqueda

El camino de la solución es la sucesión de estados, que comienza con el estado inicial y termina con el estado objetivo, donde cada estado puede ser generado aplicando una operación sobre el estado anterior. Es el camino “encontrado” por el método como solución al problema.

El camino de búsqueda es la sucesión de estados explorados por el método hasta encontrar la solución.

***Representación y Estructura de los procesos de Búsqueda**: Grafo, Árbol, Red*

Para la identificación del camino recorrido resultan más apropiados los métodos de búsqueda en árboles y grafos. El problema de búsqueda más complejo en su formulación es el de búsqueda en una red. Sin embargo una red puede reducirse a un árbol mediante la repetición de algunos nodos.

Características de los procesos de búsqueda

- Cabe la posibilidad de asociar un conjunto de estados a las diferentes situaciones en que se puede encontrar el objeto del dominio sobre el que se define el problema.
- Hay una serie de estados iniciales desde los que empieza el proceso de búsqueda.
- Existen ciertos operadores, tal que un operador aplicado sobre un estado produce otro estado.
- Existe al menos un **estado meta** o **estado solución**.
- Estos procesos de búsqueda se aplican en IA en:
    - Optimización → organización de actividades en una fábrica
    - Resolución de problemas
    - Juegos
    - Planificación → planificación de movimientos en un robot

4.2 Métodos de búsqueda

Objetivo → encontrar un estado (estado objetivo) en un espacio de estados.

Salida → camino entre el estado inicial y el estado objetivo (camino de la solución). En algunos casos el estado mismo objetivo es la salida esperada (cinemática inversa, 8 reinas)

Camino de la solución → sucesión de estados, que comienza con el estado inicial y termina con el estado objetivo, donde cada estado puede ser generado aplicando una operación sobre el estado anterior. Es el camino “encontrado” por el método como solución al problema.

Camino de búsqueda → sucesión de estados explorados por el método hasta encontrar la solución.

Tipos de métodos
- No informados: Primero en amplitud (FIFO), Primero en profundidad (LIFO)
- Informados (heurísticos): Escalada simple, Escalada por máxima pendiente, Enfriamiento simulado, Primero el mejor, A*

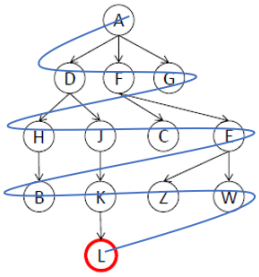
4.3 Métodos no informados

En este método, el estado inicial y el estado final se proporcionan al algoritmo de aprendizaje, comienza a buscar desde el estado inicial y continúa explorando cada estado hasta alcanzar la meta.

No dependen de información propia del problema, sino que proporcionan **métodos generales** para recorrer los árboles asociados al problema, por lo que se pueden aplicar en cualquier circunstancia.

Son algoritmos **exhaustivos**, en el peor de los casos, pueden acabar recorriendo todos los nodos.

Búsqueda Primero en Anchura/ Amplitud (BPA)



Para cada nodo se generan todas las posibles situaciones resultantes de la aplicación de todas las reglas adecuadas. Se continúa con este proceso hasta que alguna regla produce algún estado objetivo.

Partimos del nodo inicial y vamos analizando nivel a nivel

1. Crear una Pila (u otra estructura dinámica) y asignarle el estado inicial
2. Mientras la lista no esté vacía, hacer:
    a. Extraer el primer elemento de la lista
    b. Generar nuevos estados derivados del actual
    c. Explorar si los nuevos nodos son estado objetivo, si alguno lo es salir, en caso contrario añadir nuevos nodos a la lista y volver a 2.a

Cada vez que se visita un nodo se agregan sus hijos a la estructura dinámica, los mismos se agregan AL FINAL de la lista.

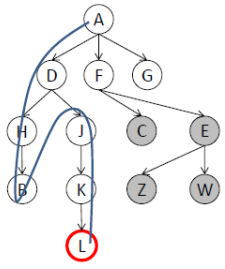
VENTAJAS:

- No queda atrapada explorando callejones sin salida.
- Si existe una solución, garantiza que se logre encontrarla.
- Si existen múltiples soluciones, se encuentra la solución mínima.
- *Es el único método que garantiza el camino (solución) más corto.*

DESVENTAJA:

- Requiere la generación y almacenamiento de todo el árbol con el consiguiente esfuerzo y capacidad de almacenamiento que esto implica.
- Mala performance (mucha memoria y poca velocidad, dada su explosión combinatoria).

Búsqueda Primero en Profundidad (BPP)



Continúa por una sola rama del árbol hasta encontrar una solución o hasta que se tome la decisión de terminar la búsqueda por esa dirección.

Terminar la búsqueda por una ruta tiene sentido cuando se ha llegado a un callejón sin salida, se produce un estado ya alcanzado o la ruta se alarga más de lo especificado en algún límite, por lo que se produce una vuelta atrás.

VENTAJAS:

- Necesita menos memoria ya que sólo se almacenan los nodos del camino que se sigue en ese instante.

- Si se tiene **suerte** con este método, se puede encontrar una solución sin tener que examinar gran parte del espacio de estados.
- Asegura hallar una solución si existe.

DESVENTAJAS:

- El proceso de búsqueda puede quedar atrapado en caminos sin salida como consecuencia de ciclos cerrados.
- No puede asegurarse que la solución encontrada sea la mejor, ya que en ramas aún no exploradas tal vez haya otras soluciones a menor profundidad.

Búsqueda Primero en Profundidad Acotado

Comienza generando rutas completas, manteniéndose la ruta más corta encontrada hasta ese momento. Deja de explorar una ruta tan pronto como su distancia total, hasta ese momento, sea mayor que la que se ha marcado como la ruta más corta. **Ramificación y acotación:**

- En lugar de generar todos los sucesores del estado actual, se genera un único sucesor.
- La elección de un nodo a ser visitado no implica que sea sacado de la lista.

Descarta los callejones sin salida (trabaja con superíndices que indican el padre del nodo), se eliminarán aquellos nodos que no tienen sucesor y no son nodo meta, y a su vez, todos aquellos nodos que sus hijos sean callejones sin salida.

4.4 Métodos informados o de búsqueda heurística

La estrategia de búsqueda informada en comparación con la búsqueda no informada tiene más información sobre la definición del problema y esta información del estado del problema, que se denomina heurística.

Para una regla heurística, no puede demostrarse teóricamente su validez, tampoco se puede comprobarla experimentalmente para todos los casos y por lo tanto carece de generalidad y de una garantía teórica de su resultado.

Una búsqueda informada intenta optimizar la solución repetidamente en función de la **función heurística**. Es posible que el método no siempre encuentre la mejor solución, ya que funciona de forma aproximada pero ofrece una solución aceptable en un tiempo razonable.

Heurística

- Para los métodos de búsqueda, es una técnica que **aumenta la eficiencia** del proceso, posiblemente sacrificando demandas de completitud.
- Es conocimiento (muchas veces intuitivo) que puede **guiar el proceso** de búsqueda.
    - El conocimiento del dominio del problema dirige el proceso de manera que sean exploradas en primer lugar aquellas trayectorias más prometedoras.
- Las heurísticas no garantizan la solución óptima, pero mejoran la calidad del proceso.
- La función heurística determina un grado de “bondad” para cada estado evaluado.
- El conocimiento heurístico generalmente se incorpora al proceso de búsqueda a través de una función heurística.

¿Por qué es necesario su uso?

- evita enredos
- da una buena solución
- profundiza la comprensión
- disminuye el uso de recursos
- no permite que se repitan los estados (no informados, si) → son grafos en vez de árboles

No hay callejones sin salida, ya que no hay caminos fijos, voy saltando según las heurísticas (para un problema de minimización se organizan de menor a mayor - al revés para maximización). *Si el nodo que viene tiene la misma heurística, se aplica el criterio deseado para acomodar siempre y cuando se respete en el resto de la búsqueda*.

Generación y prueba

1. Generar posible solución
2. Verificar si es una solución
3. Si no se halló solución, volver a comenzar
4. Si se halló solución, devolverla y terminar

La generación de la posible solución puede realizarse en formas diferentes, por ejemplo, aleatorio, sistemáticamente exhaustivo, sistemáticamente guiado por algún conocimiento, etc. Este algoritmo es un procedimiento de **BPP** ya que las soluciones deben generarse antes de que se comprueben.

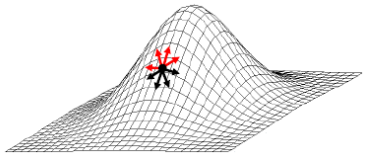
- Si es sistemático, y la solución existe, puede encontrarla, pero puede tardar más de lo admisible, dado que se transforma en una búsqueda exhaustiva.
- Al usar heurísticas, la solución puede encontrarse más rápidamente y si se combina con otras técnicas puede ser muy eficaz, dado que se restringe el espacio de búsqueda.

Escalada simple

La función de prueba se amplía con una función heurística que ayuda a evaluar los estados y proporciona una **estimación de lo cerca que nos encontramos del objetivo**, existe realimentación a partir del procedimiento de prueba, que se usa para ayudar al generador a decidirse por cual dirección moverse por el espacio de búsqueda.

La principal diferencia con generación y prueba es que se usa una función de evaluación que introduce conocimiento específico. Se debe definir con precisión qué es un estado mejor o peor.

Algoritmo:

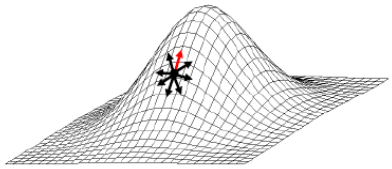


Si el estado actual es el objetivo, devolverlo y terminar. Sino, continuar. Partiendo del estado actual, se aplica uno de los operadores.

1. Si el estado generado, es el objetivo, finaliza la búsqueda.
2. Si el estado generado por el operador no es el objetivo, pero es mejor que el estado actual, el nuevo estado se convierte en el estado actual.
3. Si no es objetivo ni mejor, se aplica otro operador.

Se continúa con este ciclo hasta encontrar el estado objetivo.

Escalada por la máxima pendiente



Considera todos los movimientos posibles desde el estado actual y elige el mejor. Escalada por Máxima Pendiente tarda más en seleccionar un movimiento Escalada Simple necesita más movimientos para alcanzar una solución.

Algoritmo:

Si el estado actual es el objetivo, devolverlo y terminar, sino, aplicar al estado actual todos los operadores, para obtener los estados N1, N2, … Nk. Evaluar cada uno de estos:

1. Si el estado generado, es el objetivo, devolverlo y terminar.
2. Si no es el objetivo, pero es mejor que el estado actual, el nuevo estado se almacena como el "mejor hasta ahora" y evalúo el próximo estado Ni.
3. Si no es objetivo ni mejor, se continúa con el próximo Ni.

Se continúa con este ciclo hasta:

- Encontrar el estado objetivo
- Se da el caso que el "mejor hasta ahora" es el estado actual, en cuyo caso se informa la falla y devuelve el actual. → *Problema.*

Problema de los métodos escalada:

Existen casos en los que no se halla solución, dado que no se pueden generar estados mejores, esto ocurre cuando el estado actual está en un **máximo local, cresta o una meseta**, el método no puede avanzar. Hay formas de evitar estos problemas (pero no garantizan éxito):

- Vuelta atrás siguiendo un camino diferente (recomendado para máximos locales)
- Dar un gran salto (o varios pequeños) en alguna dirección (recomendado para mesetas)
- Movimientos en varias direcciones a la vez (recomendado para crestas)

Enfriamiento simulado

Con baja probabilidad, este método permite tomar "malas decisiones" para evitar mínimos locales y mesetas.

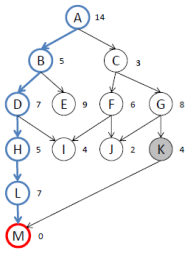
El objetivo es disminuir la probabilidad de caer en un máximo local, una meseta o una cresta. Por lo tanto se realiza una exploración con saltos amplios al principio, que se van reduciendo paulatinamente.

La búsqueda se transforma en un problema de optimización, porque pasamos de buscar un estado con ciertas propiedades generales a buscar un estado que tiene valor mínimo respecto a la función de energía considerada.

Vuelta atrás: Algunos métodos, como Primero el Mejor y A*, almacenan los estados menos prometedores para volver a ellos si no hay una mejor opción en el futuro:

Primero el mejor

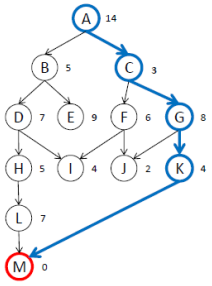
Combina las ventajas de BPA y BPP en un solo método, siguiendo un solo camino a la vez, y cambiándolo cuando alguna ruta parezca más prometedora. Se trabaja con la heurística asociada al nodo **f(n) = función heurística** → es la distancia estimada al estado objetivo.



- Utiliza una lista ABIERTA para guardar los estados a recorrer: se ordena según el valor de la heurística (primero el mejor)
- Se utiliza otra lista CERRADA para guardar los estados recorridos.
- No permite la creación de estados repetidos → utiliza un grafo.

Estrategia:

1. Se selecciona el nodo más prometedor, utilizando alguna función heurística apropiada.
2. Se expande el nodo elegido aplicando las reglas para generar a sus sucesores
3. Si algún sucesor es la solución, el proceso termina. Sino:
    a. Se añaden los nodos a la lista
    b. Se vuelve al punto inicial



*Búsqueda A**

Se modifica el anterior empleando una función heurística para estimar el costo desde un nodo (actual) hasta el nodo objetivo. Se implementa con un **grafo**.

Realiza una estimación de los méritos de cada uno de los nodos que se van generando, esto permite que **examine primero los caminos más prometedores**. (ventaja → encuentra caminos más cortos)

Se trabaja con la heurística asociada al nodo más el nivel en el que está asociado el nodo:

- f (n) = g (n) + h (n) → representa una estimación del coste necesario para alcanzar un estado objetivo por el camino que se ha seguido para generar el nodo actual.
- g (n) = costo del mejor camino encontrado hasta el momento desde la raíz hasta " n ".
- h (n) = costo de alcanzar el nodo meta desde el nodo actual

Se emplean dos listas de nodos:

- ABIERTOS: Nodos a los que se les aplicó la función heurística pero no han sido expandidos
- CERRADOS: Nodos que ya han sido expandidos

Los pasos son los siguientes:

1. Se toma el nodo más prometedor que se tenga en ese momento y que no se haya expandido
2. Se generan los sucesores del nodo elegido, se les aplica la función heurística y se les añade a la lista de nodos abiertos, verificando que dicho nodo no haya sido generado previamente

Algoritmo:

1. Comenzar con ABIERTOS (solo el inicial) hasta llegar a un objetivo o no queden ABIERTOS:
    a. Tomar el mejor nodo de ABIERTOS y generar sus sucesores, para cada sucesor:
        i. Si no se ha generado antes, añadirlo a ABIERTOS y almacenar a su padre
        ii. Si ya se ha generado antes
            1. Si el nuevo camino es mejor que el anterior → Cambiar al padre
            2. Actualizar el costo empleado para alcanzar el nodo y sus sucesores

5. Planificación

5.1 Planificación en IA

Los problemas de búsqueda de trayectorias y resolución de problemas serían en sí mismos casos de planificación. También, la planificación se considera una búsqueda en un espacio de estados con el agregado de que, lo que interesa, es el camino recorrido entre los estados alcanzados hasta llegar al estado objetivo.

En forma general cuando añadimos el problema de tener que planificar (y ejecutar) una secuencia de pasos (operaciones, cambios de estado, etc.) en un mundo no completamente previsible, nos ubicamos frente al problema de la Planificación en IA.

Para poder estudiar este problema en una forma sistemática, los primeros investigadores de la IA propusieron un grupo de "mundos" en los cuales podían proponer y probar mecanismos de planificación. El más famoso (y que utilizamos) es el "Mundo de los Bloques".

Clasificación:

- Numérica
- Lógica
    - Por pila de objetivos
    - No lineal por fijación de restricciones
    - Jerárquica. Se resuelven primero tareas de mayor complejidad sin considerar detalles
- Reactiva

El problema de la planificación

Se puede definir el proceso de resolución de un problema como una búsqueda en un espacio de estados donde:

- Cada estado se corresponde con una situación posible.
- La búsqueda comienza en una situación inicial.
- Se lleva a cabo una secuencia de operaciones permitidas hasta alcanzar una situación objetivo.

Entradas:

- Descripción del estado del mundo
- Descripción del Objetivo
- Conjunto de acciones

Salida:

Una secuencia de acciones que pueden ser aplicadas al estado, hasta alcanzar la descripción del estado final.

El método A* resuelve problemas del tipo mencionado, pero…

- La manipulación de la descripción completa de un estado es posible solo para problemas sencillos
- En problemas complejos es preferible trabajar por separado con partes pequeñas del problema y combinarlas soluciones parciales al final
  (Descomponer en subproblemas → muchas veces estos tienen dependencia entre ellos)

Existen dos formas importantes de descomponer:

- **Recalculo parcial de estados:** no se genera un nuevo estado cada vez que aplicamos una nueva regla, sino que se modifica lo que cambió:
  - modificar el estado en vez de tener un nuevo
  - manejamos un solo estado en vez de una lista
- **División del problema en subproblemas:** se deben manejar las dependencias entre ellos.

No tenemos información de los estados por los que pasamos. No existe lista abierta y cerrada.

Representación de estados y reglas

En planificación lógica los estados y reglas (operadores) se representan mediante predicados: **Pred_A(x,y,z)**

- Se pueden crear nuevos operadores
- Los predicados son “funciones con parámetros”
- La unión de varios predicados o un predicado (con AND ^) representan un **estado**

5.2 Mundo de los bloques

Clase de problemas frecuentemente utilizada para evaluar y desarrollar métodos de planificación.

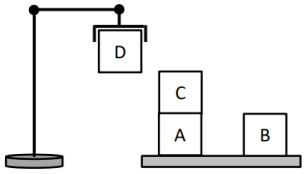
- Se adopta este mundo para hacer un estudio sistemático de los métodos y poder compararlos
- Existe una superficie plana sobre la que se sitúan los bloques
- Los bloques se pueden colocar unos sobre otros (para simplificar sólo uno sobre otro y no varios sobre otros)
- Un manipulador puede tomar y reubicar los bloques

Formado por:

1. Una superficie o mesa
2. Un conjunto de bloques apilables (un bloque puede estar encima de otro, un bloque a la vez)
3. Un brazo robot que mueve los bloques (agarra de un bloque) (Un manipulador puede tomar y reubicar los bloques)

Ejemplo de predicados:

- sobre(x,y)
- sobrelamesa(x)
- despejado(x)
- agarrado(x)
- brazolibre()



Ejemplo de definición de estados: sobrelamesa(A) ^ sobre(C, A) ^ sobrelamesa(B) ^ agarrado(D)

Operadores: Aplicar un operador es modificar el estado actual, creando uno nuevo.

Operadores del estilo STRIPS (Stanford Research Institute Problem Solver)

NOMBRE(parámetros)

- P: Pre condiciones → Se puede aplicar si los predicados de P existen en el estado actual.
- B: Borrar → Se borran los predicados de la lista B
- A: Agregar → Se añaden los predicados de la lista A

También puede representarse con solo dos líneas debajo (P y B se unen en una)

- APILAR(x,y)
  - P: despejado(y) ^ agarrado(x)
  - B: despejado(y) ^ agarrado(x)
  - A: brazolibre ^ sobre(x,y)
- DESAPILAR(x,y)
  - P: sobre(x,y) ^ despejado(x) ^ brazolibre()
  - B: sobre(x,y) ^ brazolibre()
  - A: agarrado(x) ^ despejado(y)
- TOMAR(x)
  - P: despejado(x) ^ sobrelamesa(x) ^ brazolibre()
  - B: sobrelamesa(x) ^ brazolibre()
  - A: agarrado(x)
- BAJAR(x)
  - P: agarrado(x)
  - B: agarrado(x)
  - A: sobrelamesa(x) ^ brazolibre()

Planificación como búsqueda en un espacio de estados

Dado que pasamos de un estado a otro mediante acciones –operadores- se observa que el problema de la planificación es básicamente un problema de búsqueda en un espacio de estados.

La lógica nos ofrece una manera de reducir nuestros pasos de búsqueda en planificación:

- Se deben representar los estados mediante predicados lo que permite pasar de uno a otro (en realidad saber que es posible pasar de uno a otro) aplicando reglas lógicas.
- La aplicación de la regla debería modificar el estado anterior (definido por un conjunto de predicados) definiendo un nuevo estado.
- Sin embargo, la regla dice que está permitido pasar de un estado particular a otro, pero no efectúa en un programa el cambio de predicados necesario para obtener el nuevo estado.

Entonces, para que en un programa el estado actual se modifique de acuerdo a la relación establecida en la regla es necesario aplicar procedimientos (operadores) que efectúen esa modificación. Con lo cual queda establecido que además de predicados y reglas se precisan operadores asociados a las reglas.

(Desapilar A B)

P: (Apilado A B) y (Despejado A)
B: (Apilado A B) y (Despejado A)
A: (Agarrado A) y (Despejado B)

(Apilar A B)

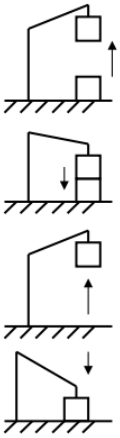
P: (Despejado B) y (Agarrado A)
B: (Despejado B) y (Agarrado A)
A: (Apilado A B)

(Levantar A)

P: (Despejado A) y (Sobrelamesa A)
B: (Despejado A) y (Sobrelamesa A)
A: (Agarrado A)

(Bajar A)

P: (Agarrado A)
B: (Agarrado A)
A: (Sobrelamesa A) y (Despejado A)



Operadores

Para poder pasar de un estado a otro, modificando predicados, usamos **operadores**, cada operador puede describirse mediante una lista de nuevos predicados. La aplicación modifica el estado actual:

- Se borran los predicados de la lista B
- Se añaden los predicados de la lista A

La condición de aplicación que está en la lista P. Se puede aplicar si los predicados de la lista P existen en el estado actual.

Planificación mediante la aplicación de reglas

Para evitar realizar la búsqueda explorando completamente el árbol (o grafo), se plantea para las descripciones simbólicas el empleo de reglas. Los pasos básicos para la aplicación de estas son:

1. Elegir el mejor operador aplicable → según función heurística.
2. Aplicar operador → regla para obtener nuevo estado.
3. Detectar si se ha llegado a una solución
4. Repetir

Planificación por pilas de objetivos

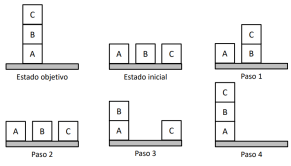
Se utiliza una **estructura de pila** donde se insertan predicados y operadores. Cada vez que se inserta un predicado compuesto, se deben apilar además los predicados atómicos. Los pasos son:

- Inicializar la pila con el estado objetivo (estado objetivo completo → base de la pila)
- Repetir hasta que la pila esté vacía:
    - Sacar el primer elemento.
        - Si es un operador, aplicarlo.
        - Si es un predicado verdadero, eliminarlo.
        - Si es un predicado atómico falso, apilar un operador que haga que se cumpla (ver lista A) y apilar su lista P → *lo reemplazamos por el operador que hace que este se cumpla y sus precondiciones*
        - Si es un operador compuesto falso, volver a insertarlo

Cuando un predicado no se cumple, tenemos que hacer que se cumpla → como? **con un operador**. Cuando hay dos opciones para agregar un operador podemos hacerlo aleatoriamente, o bien con alguna función heurística que nos ayude a tomar la decisión.

Cuando la pila está vacía es la condición de corte. Se toma el PLAN y se entrega al usuario, que describe lo que hay que hacer si queremos llegar al estado objetivo desde el estado inicial.

Anomalía de Sussman. Uno de los problemas más comunes, además de los bucles, que puede suceder al emplear el método es **incluir pasos innecesarios** en el plan. Para que no suceda hay que programar reglas o heurísticas más complejas e inteligentes que nos hagan elegir bien.



6. Sistemas Expertos y Programación Lógica

6.1 Sistemas Expertos

Los sistemas expertos (SE) resuelven problemas que normalmente son resueltos por **expertos humanos** (EH). Lo hacen mediante la aplicación de conocimiento de un dominio específico. *Se utilizan cuando no existe un método capaz de encontrar la solución, pero se cuenta con un "experto" que colabore con el desarrollo.*

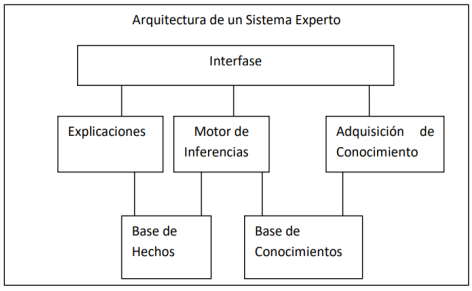
Para crear un SE, el conocimiento sobre el **dominio del problema** debe ser acotado y suficientemente estático (no debe cambiar).

- Pueden aplicarse a campos susceptibles de ser dominados por EH: diseño, detección y corrección de fallas, planificación de producción, diagnóstico médico, análisis financiero.
- No se pueden aplicar en áreas donde se necesite el sentido común (dominios no acotados), por ser muy extenso el dominio de conocimiento que debe tener el sistema.

Los SE deben resolver problemas por aplicación de conocimiento en un dominio específico (simil experto humano). Este conocimiento es adquirido a través de la intervención de expertos humanos, y almacenado en una **Base de Datos de Conocimiento** KDB. Por lo que, los SE necesitan**:**

- Acceder a una base de conocimiento sobre el dominio del problema.
- Uno o más mecanismos de razonamiento.
- Un mecanismo para explicar las decisiones → se espera de un SE que sea capaz de explicar la solución hallada en términos comprensibles para las personas.

El núcleo del SE es el **Motor de Inferencias** (compuesto por las reglas, se traduce el conocimiento heurístico del conocimiento). Este módulo puede interactuar con otros.



Reglas: se aplican sobre los símbolos definidos. Se obtienen de expertos humanos. Estas reglas de los expertos son consideradas heurísticas, dado que no es posible demostrar su validez general.

Heurístico → se obtiene una solución pero no necesariamente es la óptima, no se explora el conjunto de soluciones, solo se obtiene una.

Hechos: Los hechos son los predicados que se suponen verdaderos. No necesitan ser probados, son VERDADEROS. Es importante en nuestro estudio tener presentes los conceptos de Lógica Proposicional y Lógica de Predicados.

Algunas características del **tipo de problema resuelto** por un sistema experto son:

- Inexistencia de un algoritmo para hallar la solución
- El problema es resuelto satisfactoriamente por expertos humanos
- Posibilidad de contar con un experto humano para colaborar en el desarrollo
- El conocimiento experto del dominio debe ser relativamente "estático"

6.2 Elementos básicos del lenguaje lógico

La **Lógica de Primer Orden** es uno de los formalismos más utilizados para representar conocimiento en IA. La Lógica cuenta con un lenguaje formal mediante el cual es posible representar fórmulas, que permiten describir fragmentos del conocimiento y, además consta de un conjunto de reglas de inferencia que aplicadas a los axiomas, permiten derivar nuevo conocimiento.

El Alfabeto del Lenguaje de la Lógica de Primer Orden contiene dos tipos de símbolos:

a) **Símbolos lógicos**, entre los que se encuentran:
   i) símbolos de **constantes** proposicionales true y false,
   ii) símbolos de **operadores** prop. para negación, conjunción, disyunción e implicación
   iii) símbolos de operadores de **cuantificación** como cuantificador universal y existencial
   iv) símbolos **auxiliares** de escritura como corchetes [,], paréntesis (,) y coma;

b) **Símbolos no lógicos**:
   i) conjunto de símbolos constantes;
   ii) conjunto de símbolos de variables individuales;
   iii) conjunto de símbolos de funciones n-arias;
   iv) conjunto de símbolos de relaciones n-arias.

A partir de estos símbolos se construyen las expresiones válidas en el Lenguaje de Primer Orden: **los términos y las fórmulas.**

Las fórmulas atómicas o elementales son expresiones de la forma R(t1,...,tn) donde R es un símbolo de relación n-aria y t1,...,tn son términos

La aplicación del paradigma lógico requiere definir un lenguaje acotado y preciso, mediante el cual se expresen con precisión las reglas y sus resultados. Este lenguaje incluye en definitiva los siguientes componentes principales:

- **Átomos**: Son proposiciones simples. V (Verdadero), F (Falso) y símbolos en general incluyendo símbolos que representan sentencias.
- **Conectivos lógicos**: Conjunción, Disyunción, Implicación, Negación, doble implicación Sirven para formar preposiciones compuestas, a partir de preposiciones simples.
- **Sentencias**: Átomos o cláusulas formadas por la aplicación de conectivos a sentencias. Son pensamientos en los que se afirma algo y se expresan mediante enunciados u oraciones declarativas. Dichas proposiciones están unidas mediante conectivas lógicas
- **Cuantificador universal**: ∀ Significa que las variables son comunes a todos los elementos de la expresión.
- **Cuantificador existencial**: ∃ Se debe eliminar porque la cláusula debe cumplirse para todos los elementos. Las variables deben ser comunes a toda la expresión.
- **Símbolos de puntuación y delimitación** tales como: , () []

Se entiende por **silogismo** a una fórmula lógica con premisas seguidas de una conclusión, como ser, si q y p, entonces s. Los silogismos más empleados en SE y SI son:

1) Hipotético: p es q y q es s -> p es s, y
2) Disyuntivo: -p o q es decir p ∴q

6.3 Lógica de Predicados y Proposicional

La **lógica** es una disciplina que estudia métodos de formalización del conocimiento y formas válidas de inferencia. Existen dos niveles de abstracción:

- **Lógica de proposicional** - Lógica de predicados de orden cero (sin parámetros)
  Toma como elemento básico las frases declarativas simples o proposiciones, que pueden ser verdaderas o falsas y constituyen en sí mismas una unidad de comunicación de conocimiento. Se usan las formas clausales.

- **Lógica de predicados** - Lógica de predicados de primer orden
  Representa frases declarativas con mayor grado de detalle, considerando la estructura interna de las proposiciones. Toma como elementos básicos los objetos y sus relaciones:
  - Que se afirma (predicado o relación)
  - De quién se afirma (objeto)

Un predicado cuantificado es una proposición, es decir, cuando asigna valores a un predicado con variables, se puede convertir en una proposición. Predicados → palabras que establecen relaciones entre objetos.

La lógica de orden uno engloba a la lógica de orden cero. Es decir, que todo lo que se puede formalizar con lógica proposicional se puede formalizar en lógica de predicados, pero no al revés.

Representación de hechos simples en lógica

Fórmulas bien formadas: Proposición simple o compuesta que tiene sentido completo y cuyo valor de veracidad puede ser determinado. Las expresiones transformadas del lenguaje natural al lenguaje de la lógica se llaman fórmulas bien formadas (fbf).

La **lógica proposicional** proporciona un mecanismo para asignar valores de veracidad a la proposición compuesta, basado en los valores de veracidad de las proposiciones simples y en la naturaleza de los conectores lógicos involucrados. Ej: “Sócrates es un hombre” → SócratesHombre.

En lógica de predicados:

- Las fórmulas elementales, subatómicas o predicados son del tipo “hombre(Sócrates)”, donde “Sócrates” es el término o **argumento**.

- Cuando un predicado tiene dos o más argumentos, está indicando una relación entre esos dos o más objetos del mundo. → *odia(Marco, César)*
  - Si los argumentos son **constantes** (como “Marco”), la fórmula es un **hecho**.
  - Si los argumentos son **variables** (como x), la fórmula es una **regla.**
  - Los argumentos también pueden ser **funciones.** → vivo(Marco, ahora())

- Las fórmulas también pueden ser predicados conectados entre sí por conectivos
  leal(x, César) ∨ *odia(x, César)*
  - Si fórmulas incluyen entre los símbolos conectivos a la implicación son **sentencias**
    ∀*x: romano(x)* → *leal(x, César)* ∨ *odia(x, César)*

Funciones de Skolem: Cuando un cuantificador existencial está en el ámbito de un cuantificador universal, la variable cuantificada debe ser reemplazada con una función de Skolem de las variables cuantificadas universalmente.

Leyes de Morgan: Al invertir el valor de verdad de los símbolos de la expresión debe "invertirse" el conectivo. Estas leyes lógicas de aplicación en la teoría de los SE son:

- -(p ^ q) es equivalente a (-p v -q)
- -(p v q) es equivalente a (-p ^ -q)

Formas Clausales

Toda FBF o sentencia de lógica proposicional (calculo proposicional) puede expresarse en Forma Normalizada Conjuntiva (FNC), Forma Normalizada Disyuntiva (DNF) o como cláusula de Horn (CH).

Forma	**Ejemplo**
Forma Normalizada Conjuntiva (FNC) → Conjunción de disyunciones (o)	p ^ q ^ -s
Forma Normalizada Disyuntiva (FND) → Disyunción de conjunciones. (y)	p v -q v s
Clausulas de Horn (CH) → Conjunción de disyunciones con no más de un literal positivo (no más de una implicación)	-p v -q v s

Una **cláusula de Horn** es una conjunción de disyunciones con no más de un literal positivo. Entendiendo como un literal positivo a uno que no está negado. Por ejemplo:

- (-B1 v -B2 v B3) es una Cláusula Horn.
- (-A1 v A2) ^(-B1 v B3) es una Fórmula Horn. Sus componentes son cláusulas Horn.

Las CH representan la forma implicativa de otro modo, B → A ≡ -B v A y B1 ^ B2 → A ≡ -B1 v -B2 v A

Importancia de las Cláusulas de Horn:

- Si la expresión está en la forma de cláusula de Horn, se pueden obtener métodos de solución de menor complejidad.
- La **Clausula de Horn** tiene a su vez gran importancia en la teoría de los SE, ya que una Cláusula de Horn representa por sí sola una implicación lógica, q → p.
- Otra característica relevante es que los SE basados en CH conllevan una complejidad lineal en tanto que el resto en general se caracteriza por una complejidad de O(2^n) lo que los define como computacionalmente intratables o no polinomiales completos.

Independencia lógica en las Formas de Horn → Es necesario que no haya dependencia lógica entre los literales para que el método de resultados correctos

*La lógica proposicional se aventaja a la de Predicados en relación al **procesamiento de sus cláusulas** en computadores, ya que se puede representar mediante expresiones más sencillas y el procesamiento se reduce solo a establecer los valores verdaderos o falsos de sus componentes y posteriormente evaluar disyunciones y conjunciones.*

Satisfactibilidad (S):

- Es la capacidad de una fórmula o un conjunto de fórmulas para ser verdaderas.
- Para generar inferencias es necesario determinar la S (o no) del conjunto de fórmulas.
- En lógica proposicional → se debe determinar si existe alguna combinación de valores (verdadero/falso) para los literales que haga verdadero todo el conjunto.
- En lógica de predicados → además, es necesario encontrar una sustitución de variables que haga satisfacible el conjunto.

Reducción a forma clausal

Para aplicar el procedimiento de Resolución, las fbf se deben convertir en un conjunto de cláusulas, donde cada cláusula es una fbf en FNC que no contienen ninguna conectiva "∧". Los pasos relevantes de la reducción pueden resumirse como sigue:

1. Eliminar las implicaciones considerando que **a→ b = -a V b**
2. Reducir la aplicación de la negación (por de Morgan y doble negación) dado que

$$-(-p) = p$$
$$-(a \wedge b) = -a \vee -b$$
$$-(a \vee b) = -a \wedge -b$$

Leyes de Morgan

3. Eliminar los cuantificadores
   a. Si no hay "∃", se pueden eliminar los "∀"
   b. Los "∃" se eliminan incorporando funciones de Skolem

∀x: ∃y: menor(x, y)
⬇
∀x: menor(x, mayor_que(x))
⬇
menor(x, mayor_que(x))

4. Aplicar la propiedad distributiva y asociativa creando una cláusula separada para cada conjunto

$$(a \wedge b) \vee c = (a \vee c) \wedge (b \vee c)$$
$$(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c)$$
$$(a \vee b) \vee c = a \vee (b \vee c),\ (a \wedge b) \wedge c = a \wedge (b \wedge c)$$

5. Normalizar las variables que aparecen en las cláusulas generadas. Renombrar las variables para que no se repitan en distintas cláusulas.

Resolución → Principio de Robinson o Reducción al Absurdo.

La RESOLUCIÓN es un procedimiento cuya finalidad es evaluar la verdad o falsedad de una fórmula, se busca detectar la contradicción entre la cláusula a probar y un conjunto de cláusulas.

Obtiene demostraciones por **refutación**. Para probar la veracidad de una cláusula se intenta demostrar que su negación lleva a una contradicción con las proposiciones conocidas.

Es un proceso **iterativo simple**. En cada paso, se comparan (resuelven) dos cláusulas llamadas "cláusulas padre", produciendo una nueva cláusula (resolvente). La nueva cláusula representa la forma en la que las c. padre interaccionan entre ellas, es decir, es inferida a partir de las c. padre.

Para este método se definen las propiedades siguientes;

- Para las cláusulas P y –P el resolvente es la **cláusula vacía**.
- Si un conjunto de sentencias contiene ambos P y –P el **conjunto es insatisfacible**.
- Si ambos antecedentes son verdaderos luego el resolvente es verdadero.
- La **resolución verifica la refutación**, es decir: "Si el conjunto de cláusulas implica a P luego es contradictorio con el mismo conjunto que además contenga a –P"

A partir de lenguajes preexistentes se desarrolló un lenguaje orientado a la programación lógica: **Prolog** → permite hacer preguntas sobre objetos y relaciones del dominio, formulando como objetivos o metas que son evaluadas por el intérprete de Prolog.

El método de deducción utilizado por Prolog, se basa en el uso de una única regla de inferencia: **el Principio de Resolución**. Restringe el conjunto de cláusulas, lo que le permite llevar a cabo una prueba dirigida y, generalmente, con un universo de posibilidades explorable en tiempo de ejecución.

Método de Resolución en lógica proposicional

- La Resolución opera tomando cláusulas padre tales que cada una contenga el mismo literal.
- El literal común debe estar negado en una cláusula y sin negar en la otra.
- El resolvente se calcula como la disyunción de todos los literales de las dos cláusulas, excepto aquellos que se cancelan.
- Si la cláusula generada (resolvente) es la cláusula vacía, se ha llegado a una contradicción.

El algoritmo de resolución por refutación se puede sintetizar como una prueba de una proposición P con respecto a un conjunto de axiomas C. A partir de esta definición se establece el **algoritmo**:

1. Convertir todas la fbf de C a forma clausal.
2. Negar P y convertir a forma clausal. Añadir las cláusulas obtenidas al conjunto.
3. Hasta que se encuentre una contradicción o no se pueda seguir avanzando, repetir:
    a. Seleccionar dos cláusulas y llamarlas cláusulas padre.
    b. Calcular el resolvente
    c. Si el resolvente es la cláusula vacía, se encontró una contradicción (P es cierto). Si no, añadirlo al conjunto de cláusulas.

Resultado:

- Si se encuentra una contradicción, significa que lo que se quería probar era cierto.
- Si no se llega a la contradicción, no se puede sacar conclusiones. P puede ser cierto o falso.

Método de Resolución en lógica de predicados

- El método es similar al de lógica proposicional → Se agrega un proceso, la unificación.

La unificación se produce utilizando sustituciones. El objetivo es que, en dos referencias (literales) a un mismo predicado, los términos lleguen a ser idénticos.

Para unificar dos literales vamos recorriéndolos de izquierda a derecha. En primer lugar se comprueba si los predicados coinciden. Si es así, seguimos adelante; si no es que no son unificables. Si el predicado concuerda, comenzamos a comparar los argumentos. Si el primero de ellos coincide en ambos literales, continuamos con el siguiente y así hasta completar todos los argumentos.

Propagación Unitaria (UP) o la regla de un literal (OLR)

- Es una **técnica** para implementar el método de Resolución de forma más eficiente.
- Se transforma el conjunto de cláusulas obteniendo un conjunto equivalente de menor tamaño.
- El procedimiento se basa en las cláusulas unitarias (compuesta por un solo literal). Si se encuentra una cláusula unitaria formada por el literal L:
    - Toda cláusula que contenga L es removida (salvo por la unitaria)
    - En toda cláusula que contenga –L, este se elimina.

Algoritmo de Satisfacibilidad HornSat

- En lógica formal Satisfacibilidad Horn es el **problema** de decidir, si dado un conjunto de proposiciones en forma de cláusulas Horn es satisfacible.
- El problema de satisfacibilidad de Horn puede resolverse en **tiempo polinomial**. Un algoritmo de tiempo polinomial se basa en la regla de UP.
- También se puede resolver de una forma alternativa:
    - Todas las variables contenidas en una cláusula unitaria son seteadas al valor que satisface la cláusula unitaria. Todos los otros literales se setean a falso.
    - El resultado es el mínimo modelo de la formula de Horn.
    - UP tiene complejidad lineal sobre este conjunto.

Encadenamiento

- **Empleo de reglas explícitas del tipo si-entonces.** El modo más inmediato es aplicar reglas del tipo si-entonces, e instanciando los antecedentes de la misma para obtener una instancia válida para el consecuente.
  Consideremos entonces el caso de una regla aislada:
  Si <afirmación antecedente 1> es verdadera
  Si <afirmación antecedente 2> es verdadera
  …
  entonces<afirmación consecuente> es verdadera

En lugar de instanciar el consecuente para ver si es soportado por sus antecedentes se podrían trabajar a la inversa, esto da lugar a dos modelos fundamentales de sistemas expertos

- **SE con encadenamiento hacia delante**. El SE toma la serie de afirmaciones de entrada y ensaya todas las reglas disponibles, una y otra vez, incorporando nuevas afirmaciones, hasta que ninguna de las reglas pueda producir nuevas afirmaciones. Puede emplearse para verificar una hipótesis obteniendo una afirmación que concuerde con ella.

- **SE con encadenamiento hacia atrás.** Se comienza con una hipótesis y se trata de verificarla haciendo uso de las afirmaciones disponibles.

Redes Semánticas

Una de las formas de representación del conocimiento son las redes semánticas, las que nos proveen un modelo para explicar y almacenar el conocimiento referido a un dominio, a través de grafos que conectan a conceptos por su significado y su pertenencia a clases.

Ontología

- **Ontología** → Estudio de los tipos de objetos que pueblan la realidad, entidades, así como sus propiedades y relaciones.
- **Ontologías** → Vocabularios para personas y aplicaciones. que trabajan en un dominio.

Un *dominio* denota un área específica de interés a la cual pertenece el conocimiento almacenado.

Las máquinas carecen de las ontologías con las que nosotros contamos para entender el mundo y comunicarse entre ellas, por eso necesitan **ontologías explícitas**.

- Cuando dos sistemas de información intentan comunicarse, aparecen problemas semánticos que dificultan o imposibilitan la comunicación entre ellos.
- Con las ontologías los usuarios pueden organizar la información de manera que los agentes de software puedan interpretar el significado y así poder buscar e integrar los datos.

Dependiendo del grado de formalidad, las ontologías explícitas se clasifican en:

- Informales: lenguaje natural con ambigüedad
- Semi informales: forma estructurada del lenguaje natural
- Semi formales: Lenguajes estructurados como RDF Resource Description Framework
- Formales: Leng. lógico-matemático, sus símbolos se definen exactamente, sin ambigüedades

Formales y semi → permiten que las aplicaciones puedan usar las definiciones de los conceptos del dominio y sus relaciones.

En el campo de las tecnologías, estos elementos se encuentran en principio definidos como metalenguajes, se comenzó con el:

- **XML Extensible Markup Language** manejo de cosas. Lenguaje nacido para el intercambio de información. Aisla la semántica del formato, lenguajes de consulta como XPath y XQuery.
- **RDF Resource Description Framework** conocimiento de las cosas. Lenguaje basado en XML para describir recursos (ej. recurso → documento electrónico disponible en la web)
- **OWL Ontology Web Languag**e conocimiento de los mundos
- **SWRL Semantic Web Rule** Language para implementar procesos de razonamiento automático a partir de la información almacenada en las ontologías

Web semántica

- Es el futuro de la web buscando hacer la web un medio más colaborativo y entendible
- Es colocar datos en la web de una manera tal que las máquinas puedan entenderlos de una manera natural o convertirlos de esa forma.
- Una red de datos que pueden ser procesados directa o indirectamente por máquinas.

¿Por qué es necesaria la web semántica?

- Sobrecarga de información
- Sistemas cerrados
- Agregación de contenidos deficiente

¿Para qué sirve ?

- Toma de decisiones
- Desarrollo de negocios
- Compartición de información y descubrimiento de conocimiento
- Administración y automatización

7.1 Introducción al Lenguaje Lisp

LISP, acrónimo de list processing (procesamiento de listas), es un lenguaje de programación que fue diseñado para una fácil manipulación de cadenas de datos. En este:

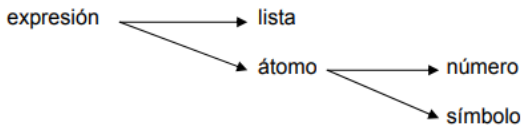
- Todos los componentes son expresados a través de información almacenada en listas.
- Se puede representar a cualquier componente en forma simbólica.

Componente = variables, constantes, sentencias de cualquier tipo, funciones, parámetros, datos, etc.

Características del lenguaje:

- **Notación prefija** → Primero escribimos el operador y luego los parámetros, *ya que siempre se evalúa primero el primer elemento de la lista*. Por ejemplo (+ 2 4).
- **Semántica entre ()** → Las listas se representan encerrándolas entre llaves (). Por este motivo un programa es una composición de listas, cada una circunscripta por un par de llaves.
- Índices arrancan en 0
- Tipo de dato del resultante depende de los tipos de datos de la operación
- Posibilidad de usar **Recursividad**

Componentes básicos del código LISP:



- Una **expresión** es una lista o un átomo.
- Un **átomo** es un símbolo o un número.
- Un **número** se puede escribir en cualquiera de las notaciones habituales.
- Un **símbolo** se representa por un nombre que está formado por caracteres alfanuméricos que no puedan interpretarse como números.
- Una **lista** es una sucesión de cero o más expresiones entre paréntesis. El blanco (no la coma) es el separador de expresiones dentro de una lista. Cada una de estas expresiones se denomina elemento de la lista.

Formas y el lazo de mayor nivel

Un código LISP está compuesto de formas; el intérprete Lisp lee una forma, la evalúa, e imprime el resultado. **Este procedimiento se denomina ciclo de lectura, evaluación, impresión (read eval-print loop).** En general, una forma es o un átomo (símbolo, entero, o cadena) o una lista.

- Si la forma es un átomo LISP lo evalúa inmediatamente.
- Si la forma es una lista, LISP trata su primer elemento como el nombre de una función; y evalúa los restantes elementos recursivamente, y luego llama a la función con los valores de los elementos restantes como argumentos.

Si LISP ve la forma (+ 3 4), trata a + como el nombre de una función. Luego evalúa 3 (obtiene 3) y 4 para obtener 4; finalmente llama a + con 3 y 4 como argumentos. La función + retorna (imprime) 7.

Cuando LISP evalúa una expresión, devuelve un valor (que será otra expresión) o señala un error:

- Un número se evalúa a sí mismo.
- Un símbolo se evalúa al valor que tiene asignado.
- Una lista se evalúa del siguiente modo:
    1) El primer elemento de la lista debe ser un símbolo “s” cuyo significado funcional “F” esté definido.
    2) Se evalúan los restantes elementos de la lista, obteniendo los valores v1,…,vn. Si el número o tipo de los valores obtenidos no es coherente con los argumentos requeridos por F, se produce un error.
    3) La lista se evalúa a F(v1,…,vn).

Evaluación de listas

Hay dos motivos importantes para considerar especialmente cómo se evalúan las listas:

- Como el código está compuesto de listas, la evaluación del mismo debe ser necesariamente una evaluación de listas.
- Lisp fue concebido inicialmente como intérprete. En este caso las listas pueden ser evaluadas independientemente una por una.

Por estos motivos existe una regla para evaluar las listas → **Las listas siempre se evalúan a partir de la evaluación del primer símbolo de la lista**. El evaluador Lisp siempre tratará de interpretar primero el primer elemento de la lista y en función de este decidirá cómo emplear los elementos que siguen (por ejemplo si deben ser tratados como los parámetros de una función).

7.2 Funciones

- Funciones de asignación.
    - Setq → asigna un valor
    - aref → referencia elementos de una lista
- Operadores
    - Operadores Aritméticos (+ - / *) → Retorna un número según las reglas de contagio: entero + racional = racional, racional + real = real, real + complejo = complejo.
    - Operadores Relacionales y Operadores de Igualdad
        - eq → para comparar símbolos. Dos símbolos son eq si y solo si son idénticos.
        - equal → para comparar listas.
- Funciones para el manejo de listas
    - Car → apunta al primer elemento de la lista
    - Cdr → apunta al resto de la lista
    - Append → concatena listas
    - Reverse → invierte elementos de la lista
    - Member → determina membresía y retorna el primer elemento encontrado igual
- Sentencias de Bifurcación: If, Cond y complemento Progn
- Definición de funciones: Defun
- Listas asociadas. Cons → asocia una palabra clave a un nodo.
- Iteración: do, dolist, loop (se usa con when y return)
- Ingreso / egreso de datos: print
- Funciones como argumentos: apply (opera el primer argumento sobre los restantes) y mapcar aplica n funciones sobre n listas.

8. Lógica difusa

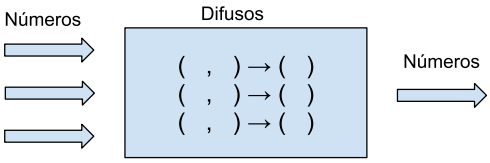
La teoría básica de los sistemas expertos está referida a la evaluación de **sistemas binarios**, es decir que solo pueden ser evaluados como verdaderos o falsos. En los problemas reales, el razonamiento considera **grados de certeza**.

Los razonamientos concernientes a problemas reales muestran que es posible considerar cierto grado de certeza o falsedad para una afirmación y no valores absolutos de verdad o falsedad.

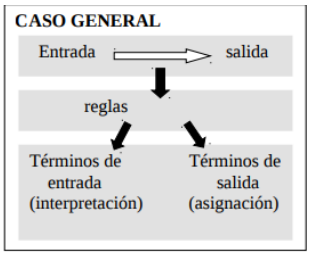
Para incorporar este concepto a los sistemas de inferencia y a los sistemas expertos se recurre a **modelos probabilísticos**, es decir se asignan valores de probabilidad a las sentencias y se propagan a través de las reglas. *Que un coeficiente de probabilidad se propague a través de una regla significa calcular qué p() se atribuye al consecuente en función de las p() de los antecedentes*.

Existen distintas formas de calcular cómo influyen las probabilidades de los antecedentes sobre los consecuentes. Esto da lugar a distintos esquemas de inferencia en IA, tales como Inferencia Probabilística, Inferencia Difusa, Inferencia Bayesiana y Propagación de Restricciones.

8.1 Lógica difusa



- Es un tipo de lógica **multivaluada**
- Utiliza conjuntos difusos y reglas de producción con la forma *SI antecedente ENTONCES consecuente*
- Aunque los valores utilizados varían entre 0 y 1, en el caso más general no se pueden tomar como probabilidades
- Se utilizan en campos como:
  - Control automático → Ej: si hace ciertos grados, agregar tanta agua a las plantas
  - Clasificación
  - Soporte de decisión
  - Sistemas expertos



Una visión general de un sistema de inferencia difusa

- La LD mapea un espacio de entrada en uno de salida
- El mecanismo principal es un sistema basado en reglas
- Las reglas se evalúan en paralelo
- Las reglas se refieren a variables y adjetivos que afectan las variables
- Antes de construir reglas, se definen los rango de las variables y la forma en la que las afectan los adjetivos

.

La idea tras un sistema de inferencia difusa es interpretar los valores de un vector de entrada y, basado en un conjunto de reglas, asignar valores al vector de salida

Reglas Si-entonces

Los conjuntos difusos y los operadores difusos son los verbos y sujetos de la lógica fuzzy (difusa). Debemos construir sentencias completas, las sentencias condicionales, reglas si-luego, son los primeros elementos que permiten sacar provecho de la lógica difusa. Una regla difusa toma la forma:

si x es A luego y es B

donde A y B son valores lingüísticos definidos por conjuntos difusos sobre los rangos (Universos de discurso) X e Y, respectivamente.

- La primera parte es el antecedente o premisa y la segunda es el consecuente o conclusión
- El antecedente es una **interpretación** que retorna un valor entre 0 y 1.
- El consecuente **asigna** un conjunto difuso a la variable de salida.

Aplicación de las reglas

- En lógica binaria → si el antecedente es verdadero, el consecuente es verdadero
- En lógica difusa → si el antecedente es verdadero en cierto grado, el consecuente es verdadero en el mismo grado 3A → 3B
- Los **antecedentes** pueden estar compuestos por múltiples partes. *"si el cielo está gris y el viento es fuerte y la presión es baja, entonces…"*

  En cuyo caso todas las partes del antecedente son calculadas simultáneamente y resueltas como un número simple empleado operadores lógicos (como un escalar),
- El **consecuente** también puede tener múltiples partes *"si la temperatura es baja luego la válvula de agua caliente está abierta y la válvula de agua fría está cerrada."*

  Todos los consecuentes son afectados igualmente por el resultado del antecedente.
- El consecuente especifica un conjunto difuso para la salida. La función de implicación modifica luego al conjunto difuso en el grado especificado por el antecedente.

Conjuntos difusos

Se definen conjuntos (difusos) para cada variable. Los conjuntos se definen mediante **funciones de membresía**.

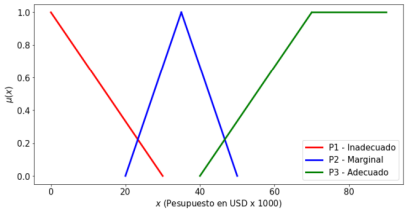
- Cada función indica el grado de pertenencia de la variable a dicho conjunto.
- Cada elemento puede pertenecer, en cierto grado, a más de un conjunto definido sobre la misma variable. *Las entradas pueden pertenecer a varios conjuntos difusos, con distintos grados de pertenencia.*
- Las funciones más comunes son:
  - Triangular
  - Trapezoidal
  - Gaussiana
  - Singleton

Presupuesto (para x en escala x 1000):

Inadecuado: $\mu_{P1}(x) = -\frac{x}{30} + 1 \; para \; 0 \leq x \leq 30$

Marginal: $\mu_{P2}(x) = \begin{cases} \frac{x}{15} - \frac{4}{3} \; para \; 20 \leq x \leq 35 \\ -\frac{x}{15} + \frac{50}{15} \; para \; 35 \leq x \leq 50 \end{cases}$

Adecuado: $\mu_{P3}(x) = \begin{cases} \frac{x}{30} - \frac{4}{3} \; para \; 40 \leq x \leq 70 \\ 1 \; para \; 70 \leq x \leq 90 \end{cases}$



Pasos de aplicación de las reglas

La interpretación de las reglas aplicando lógica difusa es un proceso de varias etapas

1) Fuzzificación
2) Aplicación de operadores difusos
3) Aplicación del método de implicación
4) Agregación
5) Defuzzificación

1. **Fuzzificación:**
   - *Evaluamos las entradas en las distintas funciones de membresía.*
   - Se determina el grado en el que cada entrada pertenece a cada uno de los conjuntos difusos.
   - Se evalúan las funciones de membresía.
   - La entrada es siempre un valor numérico limitado al universo del discurso de la variable de entrada y la salida es un grado difuso de pertenencia (siempre intervalo entre 0 y 1).

2. **Aplicación de operadores difusos**
   Una vez que las entradas han sido fuzzificadas, conocemos el grado en el cual cada parte del antecedente ha sido satisfecho para cada regla.

   - Si alguna regla tiene un antecedente compuesto, se deben aplicar los operadores para obtener un único número que representa el antecedente.
   - Cada operador difuso recibe uno o más valores y devuelve un **único valor de verdad**.

Operador	Método	Expresión
P y Q	Mínimo	mín(P, Q)
"	Producto	P * Q
"	Truncamiento	máx((P + Q - 1), 0)
P o Q	Máximo	máx(P, Q)
"	Amplificación	P + Q * (1 - P)
"	Adición	mín(P + Q, 1)
no P	Complemento	1 – P

3. **Aplicación del método de implicación**
   El método de implicación se define como la conformación del consecuente (un conjunto fuzzy) basado en el antecedente (un número). La entrada para la implicación es un número dado por el antecedente, y la salida es un conjunto fuzzy. La se obtiene para cada regla por separado. Algunos métodos podrían ser mín. (mínimo) que trunca el conjunto fuzzy, y prod. (producto) el que amplifica el conjunto de salida fuzzy.

   - Conforma el consecuente (conjunto difuso) para cada regla.
   - El resultado es proporcional al valor del antecedente.
   - Las reglas se evalúan individualmente para obtener el valor de la implicación. Luego deben asociarse los conjuntos de salida obtenidos para cada regla.

4. **Agregación**
    - Se unifican las salidas uniendo los procesos paralelos.
    - Se combinan las salidas en un único conjunto difuso.
    - La entrada del proceso es la lista de las salidas truncadas según el resultado de la implicación.
    - La salida es un único conjunto difuso para cada variable de salida.
    - Algunos métodos son: max (máximo), prob-or (or probabilística), y sum (la suma de cada conjunto de salida para cada regla).
    - El proceso de agregación consiste en formar un único conjunto de salida para todo el sistema a partir de los conjuntos de salida de cada regla.

5. **Defuzzificación**
    - La entrada para el proceso de defuzzificación es el conjunto difuso agregado (el conjunto difuso unificado de salida) y la salida es un número.
    - La defuzzificación tiene lugar en dos distintos pasos. Primero las funciones de pertenencia son escaladas de acuerdo a sus posibles valores, luego estas son usadas para calcular el centroide de los conjuntos difusos asociados.
    - El método más popular es el cálculo del centroide, el cual retorna el centro de un área bajo una curva.

$$CG = \frac{\sum_{x=a}^{b} \mu_A(x)x}{\sum_{x=a}^{b} \mu_A(x)}$$

        i. Numerador: fragmentos * riesgo
        ii. Denominador: cantidad de fragmentos * riesgo

7.2 Heurística y Lógica Difusa

En IA, se observa a veces cierta tendencia a asociar el concepto de heurística a los procedimientos de búsqueda. Sin embargo dado que su significado es general, puede obviamente aplicarse a otras áreas de la IA.

El caso de los Sistemas de Inferencia Difusa, hace particularmente evidente esta situación. Si nos detenemos a pensar en el ejemplo que se ha explicado en este capítulo, existen varios puntos oscuros en lo que hace a su validación a través de un método formal. Por ejemplo, El tipo de función de membresía de los adjetivos empleados en las reglas debe ser propuesta por el diseñador del sistema, los valores particulares también deben ser propuestos por el diseñador del sistema (digamos por ejemplo las pendientes y curvaturas particulares de cada curva), el método de defuzzificación ha de ser elegido por el diseñador también, y finalmente también los métodos de implicación a usar en un sistema.

Todo esto no solo muestra una aplicación particular de la heurística, sino que evidencia la fuerte naturaleza heurística de los Sistemas de Inferencia Difusa, lo que hace en algunos ámbitos, se cuestionen los fundamentos para el diseño y la construcción de estos sistemas. Por su parte los cultores de la Lógica Difusa contraponen a esta visión un tanto despectiva las aplicaciones exitosas logradas en diferentes ámbitos.

9. Complejidad e Inteligencia Artificial

La complejidad está asociada con los recursos de cómputo requeridos para resolver un problema, estos recursos son normalmente **tiempo** (que lleva correr un algoritmo) y **espacio** (que ocupa).

Procedimiento y algoritmo

- **Procedimiento:** todo conjunto finito y ordenado de instrucciones, que puedan ser ejecutadas por una persona o máquina, sin necesidad de conocimiento adicional a lo expresado en las sentencias.
    - Esto implica que un problema será considerado resuelto cuando se encuentre una forma mecánica de operar, a partir de sus datos hasta alcanzar un resultado.
    - Un procedimiento (efectivo o no) puede presentarse de muy diversas maneras: en su forma más general una secuencia mecánica de instrucciones es una **Máquina de Turing**, pero en el otro extremo una simple fórmula representa también esta noción de procedimiento.

- **Algoritmo:** procedimiento permite alcanzar la solución en un número finito de pasos.

Hay problemas para los cuales no es posible encontrar un procedimiento y otros problemas para los que existen procedimientos pero no algoritmos.

Concepto de complejidad

Un problema es complejo si su resolución requiere el empleo de un algoritmo complejo. Este último es a su vez complejo si su aplicación involucra cálculos complicados y/o su lógica subyacente es complicada. *En consecuencia, el concepto de complejidad no parece ser cuantificable o medible, sino más bien subjetivo*.

Se determina indirectamente la complejidad de un problema a partir de la medición de los recursos necesarios para resolverlo.

Por lo tanto, se trata de buscar algún indicador de la **complejidad intrínseca** de los problemas, procurando prescindir de la habilidad de la persona que deba resolverlos.

Para la definición de estas **métricas** es necesario establecer los siguientes tres aspectos:

- Identificar los parámetros primitivos que serán medidos para disponer de datos característicos del objeto estudiado.

- Definir las propias métricas, representadas por expresiones destinadas a determinar indicadores objetivos que representen niveles de complejidad.

- Establecer las condiciones en que se harán las mediciones, con el fin de asegurar que diferentes evaluaciones de complejidad tengan una referencia común y sean verdaderamente comparables.

Se determina indirectamente la complejidad de un problema a partir de la medición de los recursos necesarios para resolverlo. Se proponen como parámetros Tiempo y Espacio como los indicadores más representativos.

9.1 Complejidad Temporal T(n)

Se refiere a la cantidad de intervalos o unidades elementales que demanda completar la ejecución de un proceso, expresado en función del tamaño de los datos de entrada.

La complejidad temporal es la razón de crecimiento entre el indicador de tiempo y la dimensión de los datos, que representa el *parámetro medible*. Como resultado se hablara de complejidad temporal lineal, polinómica, logarítmica, exponencial, etc., según la naturaleza de la expresión que las vincula.

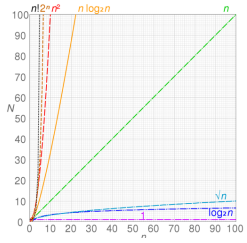
Por ejemplo, si las unidades de tiempo que demanda la solución de un problema de dimensión "n" es $T(n) = 3.n^3 + 6.n + 12$, se dirá que su complejidad temporal es polinómica, en este caso de grado 3.

El **límite en el crecimiento** de esta medida se denomina **complejidad temporal asintótica** y es finalmente lo que determina el tamaño del problema que puede ser resuelto con cierto algoritmo.

Notación Big O → Es el lenguaje utilizado para describir la complejidad de un algoritmo.

- También notación Bachmann–Landau
- Define la cota superior asintótica

Orden (función)	Descripción
$O(1)$	constante
$O(\log n)$	logarítmica
$O(n)$	lineal
$O(n^2)$	cuadrática
$O(n^3)$	cúbica
$O(a^n)$	exponencial
$O(n!)$	factorial



Diferencias en tiempos de procesamiento:

En la siguiente tabla se ilustra la complejidad temporal de ciertos algoritmos seleccionados:

Algoritmo	Orden de complejidad Temporal T(n)
A_1	n
A_2	n log n
A_3	n^2
A_4	n^3
A_5	2^n

Para compararlos, la siguiente tabla muestra los tamaños de los lotes de datos que pueden ser procesados por los diferentes algoritmos en esos mismos tres intervalos de tiempo. Se observa el impacto de las diferentes complejidades en la capacidad de cómputo:

Algoritmo	Máximo Tamaño de lote de datos "n"		
	1 seg	60 seg	3600 seg
A_1	1.000	60.000	3.600.000
A_2	140	4.893	200.000
A_3	31	244	1.897
A_4	10	39	153
A_5	9	15	21

Se puede notar que **disminuye la cantidad de datos a medida que aumenta la complejidad temporal**. Se busca determinar el incremento del tamaño de problema que puede ser resuelto por cada algoritmo en el mismo intervalo de tiempo de un segundo.

La tabla muestra estos incrementos y se representan con los símbolos “x” y “+” los casos de un factor o de un incremento constante, respectivamente. Consecuencia de recurrir a un medio de cómputo diez veces más rápido:

Algoritmo	Incremento en el tamaño “n” del problema que es procesable en 1 seg.
A_1	x 10,0
A_2	x 10,0 (aprox.)
A_3	x 3,16
A_4	x 2,15
A_5	+ 3,30

El incremento de la velocidad no es directamente proporcional a la velocidad de procesamiento de problemas de esa complejidad. La tabla anterior demuestra que con complejidades polinómicas de grado elevado, y es especial con complejidad de orden exponencial, el aumento de la capacidad de procesamiento tiene escaso impacto en la posibilidad de resolver problemas de mayor tamaño.

9.2 Complejidad Espacial E(n)

Es la medida de la complejidad que mide la cantidad de espacio de almacenamiento requerido para resolver un problema.

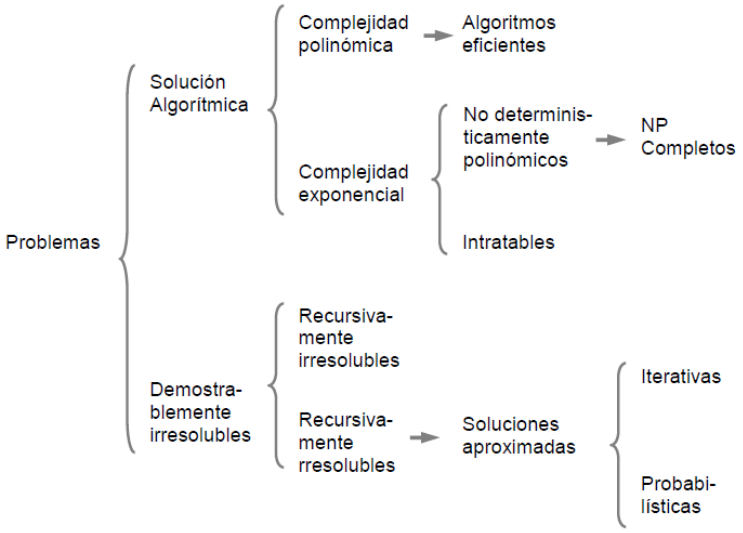
Esta complejidad espacial definirá la razón de crecimiento entre el espacio requerido y los datos del problema. Al aumentar la complejidad de un proceso aumenta también el espacio que demanda y se representa por E(n).

Al igual que en el caso de la complejidad temporal, el **límite en el crecimiento** de esta medida se denomina **complejidad espacial asintótica** y determinará finalmente el tamaño del problema que puede resolver cierto algoritmo.

9.3 Clasificación de problemas

Todos los problemas matemáticos imaginables pueden ser divididos en dos grupos: los que admiten un algoritmo para su solución y los demostrablemente irresolubles.

La dificultad que presentan los problemas está en el orden de la complejidad de su solución temporal o espacial, o directamente en no tener solución. En este último caso se recurre a las heurísticas, buscando soluciones aproximadas.



.

9.4 Complejidad e Inteligencia artificial

Complejidad temporal del entrenamiento de las redes neuronales artificiales

Se considerará aquí el entrenamiento de redes multicapa de Perceptrones por el método de backpropagation, quedando definida la arquitectura de la red a partir de los siguientes parámetros:

- m = número de capas
- N = número total de pesos en la red
- n1 = unidades de la capa de entrada
- nk = unidades de la capa oculta “k” (1 < k < m)
- nm = unidades de la capa de salida

Se define además P como el número de patrones de entrenamiento y M el número de ciclos o “épocas” de entrenamiento requerido. Con estos parámetros se calculará su complejidad temporal a partir de la estimación del número de operaciones requeridas y del criterio de costo uniforme.

$$T(P,M,m,n_1,..,n_m) = PM\left[\sum_{s=2}^{m-1} n_s(2n_{s+1} + 4n_{s-1} + 3) + n_m(4n_{m-1} + 5)\right] + M\sum_{s=2}^{m} n_s n_{s-1}$$

Como conclusión puede decirse que en general: **O(N3) < T(N) < O(N5)**

Es decir, la complejidad temporal del proceso de entrenamiento de redes multicapa de Perceptrones es de tipo polinómica, de grado entre 3 y 5. Se trata de un problema de solución algorítmica considerado eficiente.

Complejidad de los problemas de búsqueda en el espacio de estados

Se toman aquí como parámetros característicos del problema el factor de ramificación medio “R” y el nivel de profundidad alcanzado en el árbol de búsqueda “p”.

La complejidad temporal de todos los métodos tiene un límite exponencial. Más aún, la resolución de este tipo de problemas suele ser caracterizado como NP-completo, pero como mínimo, problema NP.

Como consecuencia, se justifica la necesidad de una **buena heurística** que restrinja en todo lo posible la zona del espacio de estados en que se desarrollará el proceso de búsqueda de la solución.

Método	Complejidad Temporal	Complejidad Espacial
Primero en anchura	R^p	R^p
Primero en profundidad	R^p	p.R
Descenso iterativo	R^p	p.R
Máxima pendiente	R^p	p.R
Primero el mejor	$\leq R^p$	$\leq R^p$
A*	$\leq R^p$	$\leq R^p$

La delimitación de la zona de trabajo en el espacio de estados es el efecto buscado con los algoritmos Primero el Mejor y A*, con los que pueden reducirse notablemente los indicadores de complejidad hasta convertirlos en algoritmos eficientes. Pero presentan la seria limitación de poder acabar degenerando en una búsqueda primero en anchura si la función heurística no es suficientemente expresiva de la condición del problema estudiado.