3. Sistemas Basados en Conocimiento

3.1 Introducción y Definición

A diferencia de los sistemas de "búsqueda en espacio de estados" (vistos anteriormente), donde se busca una solución explorando estados ciegamente o con heurísticas, los SBC se basan en **conocimiento explícito** para razonar.

Diferencias Clave: Búsqueda vs. Conocimiento.

Característica	Búsqueda en Espacios de Estados	Sistemas Basados en Conocimiento
Base	Estados y operadores	Base de Conocimiento (Reglas/Lógica)
Método	Algoritmo de búsqueda (exploración)	Método de razonamiento (inferencia)
Datos Iniciales	Estado inicial	Hechos conocidos
Resultado	Camino o estado solución	Respuesta a una consulta / Diagnóstico

3.2 Sistemas Expertos (SE)

Son el tipo más representativo de SBC. Se definen como sistemas con un nivel de competencia equivalente o superior a un experto humano en un dominio concreto.

¿Cuándo usar un Sistema Experto?
No sirven para todo. Se recomiendan cuando:

- El problema es complejo y de un dominio especializado.
- **No existe un algoritmo tradicional** para resolverlo.
- Existen expertos humanos capaces de articular ese conocimiento.

Historia y Ejemplos

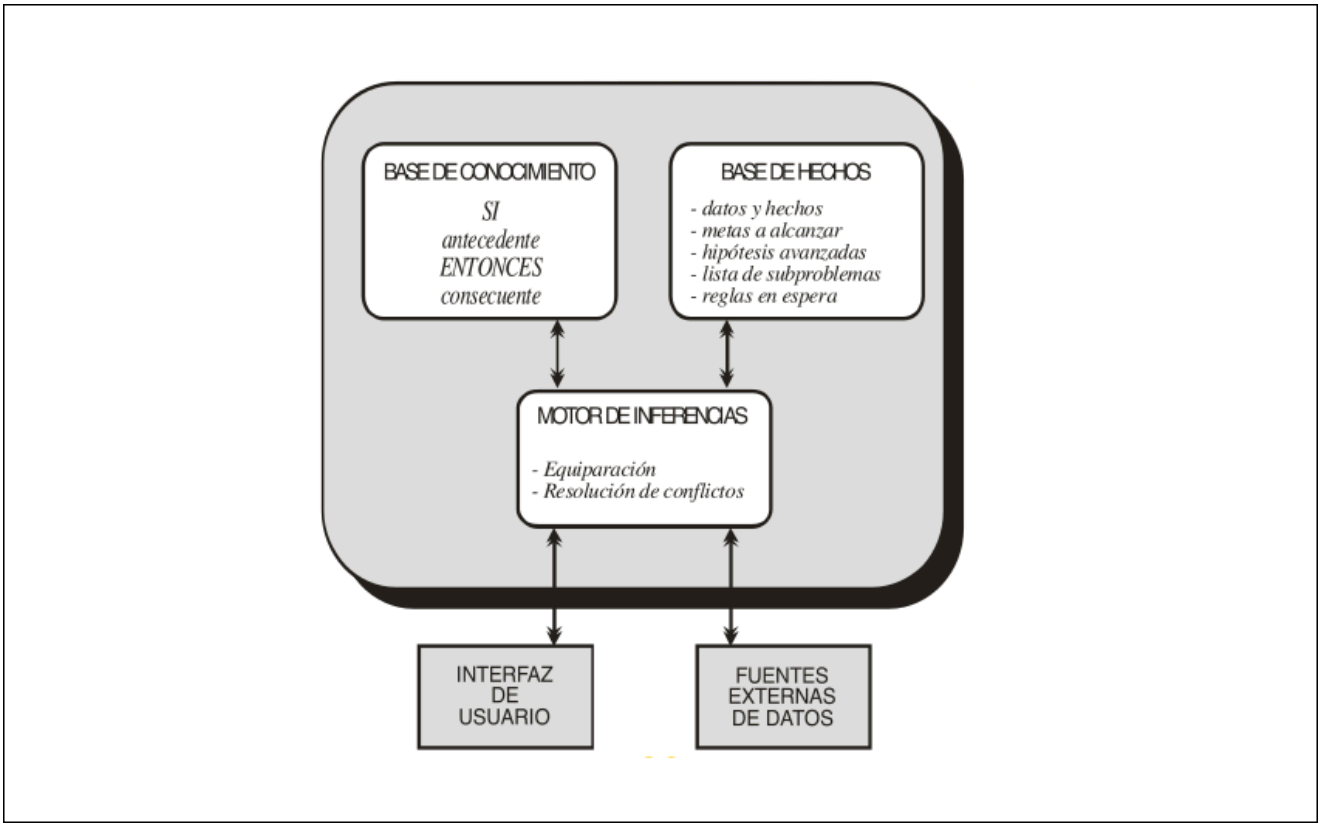
- **Dendral (1965)**: Identificación de compuestos químicos.
- **Mycin (años 70)**: Diagnóstico de infecciones bacterianas.

3.3 Arquitectura de un Sistema Experto

Para que una máquina "razone", necesita separar lo que *sabe* (conocimiento) de lo que *está pasando* (hechos) y de *cómo pensar* (motor).

Componentes Principales:

1. **Base de Conocimiento (BC)**: Es estática y general del dominio. Contiene las reglas ("Si X entonces Y").
2. **Base de Hechos (BH)**: Es dinámica y específica del caso actual. Contiene los datos del problema y lo que se va descubriendo.
3. **Motor de Inferencia**: Es el "cerebro". Aplica el conocimiento a los hechos para deducir nuevas verdades. Realiza tareas de **equiparación** (matching) y **resolución de conflictos**.



Hechos	Conocimiento
Específicos, relacionados con el problema concreto que se intenta resolver.	General, relacionado con el dominio en el que se opera y el tipo de problemas.
Dinámicos, a partir de unos hechos iniciales pueden aparecer otros distintos.	Relativamente estático, aunque puede cambiar si se añade nuevo conocimiento.
Aumenta durante la resolución del problema.	Normalmente no aumenta durante la resolución del problema.
Necesidad de almacenamiento y recuperación eficiente.	Necesidad de razonamiento eficiente.
Se busca que los hechos obtenidos directamente del problema sean precisos y exactos.	Puede ser impreciso e incierto; por lo tanto, también los hechos inferidos pueden serlo.

3.4 Representación del Conocimiento

¿Cómo guardamos el conocimiento en la máquina?

Lógica Formal (La base matemática)

Es la forma más rigurosa de representar relaciones.

- **Lógica Proposicional**: Usa afirmaciones verdaderas/falsas y operadores (AND, OR, NOT, Implicación). Se basa en reglas de inferencia como el **Modus Ponens** (Si $p \rightarrow q$ y tengo p, entonces q) o **Modus Tollens**.
- **Lógica de Predicados**: Más potente. Introduce cuantificadores ("Para todo", "Existe"), constantes (objetos) y predicados (relaciones como `es-padre(x,y)`).

Reglas de Producción y Sistemas Basados en Reglas (SBR)

Es la estructura más utilizada en los Sistemas Expertos. Cuando representamos el conocimiento de esta forma, hablamos de **Sistemas Basados en Reglas**.

- **Unidad básica**: La Regla de Producción.
  - Formato: `SI <condición/situación> ENTONCES <acción>`.
- **Dinámica del SBR**:
  - La parte `SI` busca coincidencias en la **Base de Hechos**.
  - La parte `ENTONCES` modifica la memoria: añade nuevos hechos, borra antiguos o ejecuta acciones .
- **Ejemplo**:
  - *Regla*: SI "coche no arranca" Y "batería < 10V" ENTONCES "cambiar batería".

> Lo importante de la sección 3.4 es esto, lo otro lo tiene el tipo tirao por la presentación sin ningún tipo de sentido.

Redes Semánticas

Representación gráfica mediante grafos.

- Nodos = Conceptos/Objetos (ej. "Juan", "Persona").
- Arcos = Relaciones o herencia (ej. "es un", "tipo de"). Permite heredar propiedades (si Juan es Persona, y Persona tiene altura, Juan tiene altura)13131313.

3.5 Estrategias de Razonamiento (El Motor en acción)

Una vez tenemos reglas y hechos, ¿cómo los procesa el motor? Existen dos estrategias opuestas:

Encadenamiento Hacia Adelante (Forward Chaining)

- **Filosofía**: Guiado por los **datos** (Data-driven).
- **Proceso**: Partimos de los hechos conocidos → buscamos reglas que se cumplan → ejecutamos acciones (disparamos reglas) → obtenemos nuevos hechos hasta llegar a un objetivo o no poder seguir .
- **Ejemplo (Coches)**: Sé que "el coche no arranca" y "batería < 10V" → Deduzco "cambiar batería".
- **Algoritmo**: Ciclo de **Equiparar** (ver qué reglas se cumplen) → **Resolver** (elegir una del conjunto conflicto) → **Aplicar** (ejecutarla).

Encadenamiento Hacia Atrás (Backward Chaining)

- **Filosofía**: Guiado por los **objetivos** (Goal-driven).
- **Proceso**: Partimos de una **hipótesis** (¿Tiene gripe?) → buscamos reglas que concluyan eso → verificamos si se cumplen sus premisas (antecedentes).
- **Estructura**: Se suele modelar como un **Grafo AND/OR**. Para probar una meta, debo probar sus sub-metas.
- **Ejemplo (Médico)**: ¿Tiene infección? (Meta) → Para ello necesito saber si tiene fiebre y dolor (Sub-metas) → Verifico temperatura.

3.6 Comparativa: SE vs. Programación Convencional

Es fundamental entender que programar un Sistema Experto no es programar un algoritmo clásico.

Programación Convencional	Sistemas Expertos
Imperativa (Cómo hacerlo)	**Declarativa** (Qué se sabe)
Modificación por re-programación de código	Modificación añadiendo/quitando reglas en la base de conocimiento
Solución algorítmica (precisa, única)	Solución inferida (puede ser incierta/probabilística)
Guiada por flujo de control	Guiada por el Motor de Inferencia

Tendencias Modernas: LLMs y "Chain of Thought":

- Los Grandes Modelos de Lenguaje (LLMs) pueden realizar razonamientos lógicos (silogismos), aunque a veces fallan si las premisas del mundo real entran en conflicto con la lógica formal (ej. "todos los pájaros vuelan" vs "pingüinos").
- **Chain-of-Thought (CoT)**: Técnica de *prompting* donde se pide al modelo que "piense paso a paso". Esto mejora drásticamente la capacidad de resolver problemas matemáticos o lógicos complejos en comparación con pedir solo la respuesta final.

3.7 CLIPS (C Language Integrated Production System)

> **Nota**
>
> Esto no debería de caer en el final, pero más vale prevenir que curar

¿Qué es CLIPS?

Es una herramienta de software diseñada para desarrollar **Sistemas Expertos** y sistemas basados en reglas. Funciona mediante un paradigma de **programación declarativa**: tú dices *qué* sabes (hechos) y *qué* hacer si ocurren ciertas cosas (reglas), y el motor decide *cuándo* hacerlo.

Arquitectura: Los 3 Pilares

Para entender cualquier código en CLIPS, debes distinguir estos tres elementos:

1. **Memoria de Trabajo (Working Memory)**: Es la "pizarra" donde están escritos los datos actuales (Hechos). Es volátil (cambia mientras el programa corre).
2. **Base de Conocimiento (Knowledge Base)**: Donde están guardadas las Reglas (Lógica estática).
3. **Motor de Inferencia**: El "cerebro" que compara constantemente los Hechos con las Reglas para ver cuál se puede activar.

3Los Hechos (Datos)

Representan el estado del mundo. En el examen te pondrán código y tendrás que identificar qué tipo de hecho es.

A. Tipos de Hechos

1. **Vectores Ordenados (Ordered Facts)**:
   - Son listas simples. El significado depende de la posición.
   - *Ejemplo*: `(Pedro 45 V)` → Debes saber de memoria que el primero es nombre, el segundo edad, etc. Es rígido y propenso a errores.
2. **Registros (Deftemplates)**:
   - Estructuras con campos nombrados (slots). Son flexibles y claros.
   - Si no das valor a un campo, se pone `nil` (nulo).
   - *Sintaxis*:
     Fragmento de código

```
(deftemplate Persona
    (field Nombre)
    (field Edad))

; Uso:
(assert (Persona (Nombre Juan) (Edad 30)))
```

B. Gestión de Hechos (Comandos Clave)

Estos comandos modifican la Memoria de Trabajo. Es vital entender qué hace cada uno para seguir la traza del programa:

Comando	Acción	Detalle Importante para Examen
`assert`	Añade un hecho.	CLIPS le asigna un ID único (ej. `f-1`).
`retract`	Borra un hecho.	Se usa el ID: `(retract 1)`.
`modify`	Modifica campos.	**Ojo**: En realidad borra el hecho viejo y crea uno nuevo con **nuevo ID**.
`duplicate`	Copia un hecho.	Crea un clon (nuevo ID) cambiando solo lo que indiques.

Las Reglas (Lógica)

Estructura fundamental: SI (Condiciones) ENTONCES (Acciones).

En CLIPS se separan por la flecha ⇒.

```
(defrule nombre-regla
   ; PARTE IZQUIERDA (LHS - Antecedente)
   (Condicion 1)
   (Condicion 2)
   =>
   ; PARTE DERECHA (RHS - Consecuente)
   (Accion 1)
)
```

Conceptos de Lectura de Reglas

1. **Matching (Emparejamiento)**: Para que una regla se active, **TODOS** los patrones del LHS deben cumplirse a la vez (es un **AND** implícito).

2. **Variables (`?nombre`)**:

- Sirven para capturar valores de un hecho y usarlos.
- *Ejemplo*: `(temperatura ?t)` → "Busca un hecho temperatura y guarda su valor en la variable `?t`".

3. **Restricciones**:
   - `?t&:(> ?t 39)` → Léelo como: "La variable `t` **Y** (`&`) tal que `t` sea mayor que 39".
4. **Comodines (`$?`)**:
   - Indica "cero o más campos". Útil cuando no sabes la longitud exacta de un vector.

Ciclo de Ejecución (El Motor)

En el examen te pedirán seguir la traza (qué pasa paso a paso).

1. `(clear)`: Borra TODO (reglas y hechos). Deja el sistema en blanco.
2. `(reset)`:
   - Borra los hechos actuales.
   - Restaura los hechos iniciales (definidos con `deffacts` si los hubiera).
   - Deja el sistema listo para empezar.
3. `(run)`:
   - Arranca el motor.
   - El motor mira los hechos → Busca reglas que coincidan → Ejecuta las acciones.
   - **Importante**: Si una acción (`assert`) añade un hecho nuevo, el motor se para, reevalúa todas las reglas con el nuevo dato, y sigue.
4. `(facts)`: Muestra la lista actual de hechos en memoria.

Ejemplo Práctico de Traza (Lectura de Código)

Si te dan este código, así es como debes "leerlo" mentalmente:

Código:

```
(defrule regla-fiebre
   (temperatura ?t&:(> ?t 39))  ; Condición 1: T > 39
   =>
   (assert (fiebre))            ; Acción: Añadir hecho fiebre
)

(defrule regla-infeccion
   (fiebre)                     ; Condición 1
   (garganta_inflamada)         ; Condición 2
   =>
   (assert (infeccion_bacteriana)) ; Acción
)
```

Situación (Hechos iniciales):

1. `(assert (temperatura 40))`
2. `(assert (garganta_inflamada))`

Traza Lógica (Lo que ocurre al hacer `run`):

1. El motor ve `temperatura 40`.
2. Comprueba `regla-fiebre`: ¿40 > 39? **Sí**. → **Se activa**.
3. **Ejecución**: Se añade el hecho `(fiebre)` a la memoria.
4. El motor ve el nuevo hecho `(fiebre)`.
5. Comprueba `regla-infeccion`:
   - ¿Existe `(fiebre)`? **Sí** (acaba de crearse).
   - ¿Existe `(garganta_inflamada)`? **Sí** (estaba al principio).
6. **Ejecución**: Se añade el hecho `(infeccion_bacteriana)`.
7. No hay más reglas que se cumplan. Fin.

Resultado final (`facts`):

- `f-0 (temperatura 40)`
- `f-1 (garganta_inflamada)`
- `f-2 (fiebre)`
- `f-3 (infeccion_bacteriana)`