Tema 2: Representación del conocimiento basada en reglas

José A. Alonso Jiménez
Francisco Jesús Martín Mateos
José Luis Ruiz Reina

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Sistemas basados en el conocimiento

- **Sistemas basados en el conocimiento:**
  - **Programas que resuelven problemas usando un determinado dominio de conocimiento**
- **Terminología:**
  - **sistema basado en el conocimiento**
  - **sistema experto**
  - **sistema experto basado en el conocimiento**
- **Definiciones de sistemas expertos:**
  - **Un sistema experto es un programa que usa conocimiento y procedimientos de razonamiento para resolver problemas lo suficientemente difíciles como para necesitar de un experto para su solución (Feigenbaum, 1982)**
  - **Un sistema experto es un programa construido usando la tecnología de los sistemas expertos**
- **Sistemas basados en el conocimiento *vs* sistemas expertos:**
  - **SBC: conocimiento no necesariamente experto.**
  - **SE: conocimiento experto + interacción.**
  - **SE $\subseteq$ SBC.**

Características de los SBC

- **Ventajas:**
  - **Fácil acceso y disponibilidad de conocimiento (experto)**
  - **Coste reducido**
  - **Permanencia**
  - **Fiabilidad y rapidez**
  - **Respuestas no subjetivas**
  - **Tutores**
  - **Explicación del razonamiento**
  - **Competitivos con expertos humanos**
- **Se usan cuando el problema:**
  - **No se requiere "sentido común"**
  - **Se requiere razonamiento simbólico**
  - **No se resuelve con métodos "tradicionales"**
  - **Necesita de conocimiento experto**
  - **El coste compensa su uso**

Aplicaciones de SE

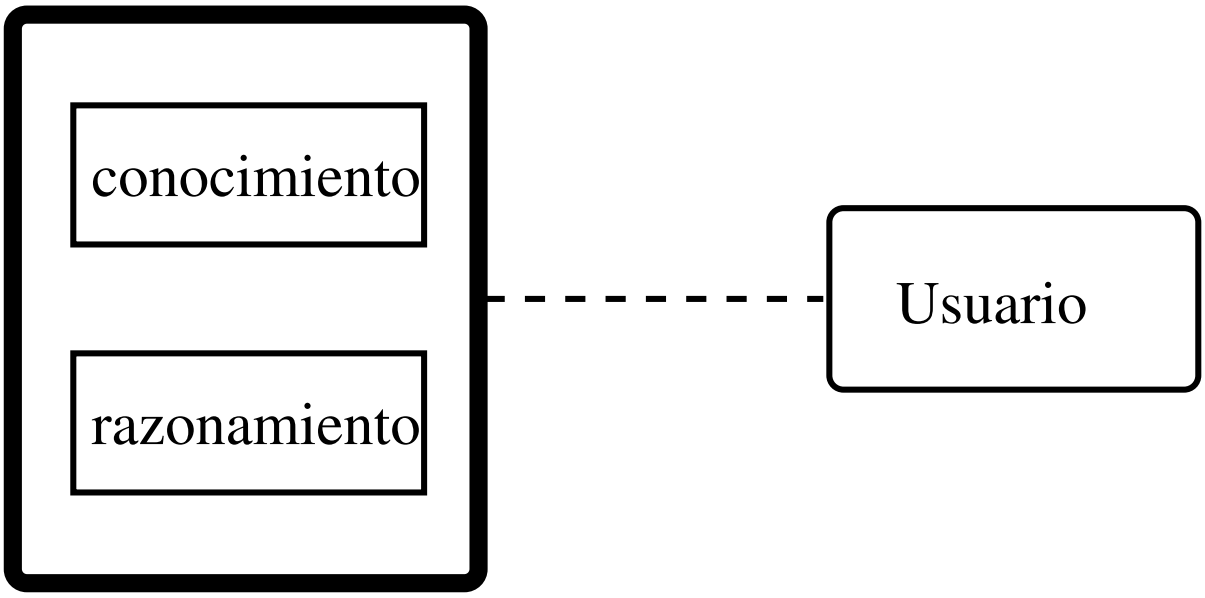
- **Tipos de aplicaciones con sistemas de producción:**
  - **Configuración**
  - **Diagnóstico**
  - **Enseñanza**
  - **Interpretación**
  - **Monitorización**
  - **Planificación**
  - **Predicción**
  - **Remedio**
  - **Control**
- **Campos de aplicación**
  - **Medicina: MYCIN, INTERNIST, CADUCEUS, CASNET**
  - **Geología: PROSPECTOR, Dipmeter Advisor**
  - **Informática: XCON**
  - **Matemáticas: MACSYMA**
  - **Educación: GUIDON, DEBUGGY**

Lenguajes de SBC

- **Lenguajes de SBC**
  - **LISP, PROLOG**
  - **EMYCIN**
  - **OPS5, ART, CLIPS**
  - **KEE**
- **PROLOG:**
  - **Programación Lógica**
  - **Razonamiento hacia atrás**
- **CLIPS**
  - **CLIPS = C Language Integrated Production Systems**
  - **Es un lenguaje basado en reglas de producción**
  - **Desarrollado en la NASA desde 1984**
  - **Escrito en C**
  - **La sintaxis es parecida a la de Lisp**
  - **Relacionado con OPS5 y ART**
  - **Objetivos: portabilidad, bajo coste y facilidad de integración**

Estructura de los SBC

- **Paradigma de los sistemas basados en conocimento (SBC):**
  - **Sistema basados en el conocimiento = Conocimiento + Razonamiento**
- **Estructura básica de los SBC**



- **Además:**
  - **Interacción con el usuario**
  - **Explicación de cómo y porqué**

Representación del conocimiento

- **Requisitos de los formalismos de representación del conocimiento:**
  - **potencia expresiva**
  - **facilidad de interpretación**
  - **eficiencia deductiva**
  - **posibilidad de explicación y justificación**
- **Principales formalismos de representación**
  - **lógica**
  - **reglas de producción**
  - **redes semánticas, marcos**
  - **lógicas de descripción**
  - **...**
- **Cada formalismo de representación usa un método de inferencia específico:**
  - **Resolución, SLD-resolución**
  - **Razonamiento hacia adelante y hacia atrás**
  - **Herencia**
  - **...**

Sistemas basados en reglas.

- **Introducción de los sistemas de producción**
  - **A. Newell y H.A. Simon** *Human problem solving* **(Prentice–Hall, 1972)**
- **Correspondencia entre sistemas de producción y memoria humana**
  - **Memoria de trabajo y memoria temporal**
  - **Base de conocimiento y memoria permanente**
- **S.E. basados en sistemas de producción**
  - **DENDRAL: S.E. para determinar estructuras moleculares (Buchanan, U. Stanford, 1964)**
  - **MYCIN: S.E. para diagnosticar meningitis e infecciones sanguíeas (Shortliffe, U. Stanford, 1972)**
  - **PROSPECTOR: S.E. para estudio de exploraciones geológicas (Duda, Stanford Research Institute, 1978)**
  - **R1 (XCON): SE para configurar equipos VAX (McDermott, CMU y DEC, 1978)**

Sistemas basados en reglas (II).

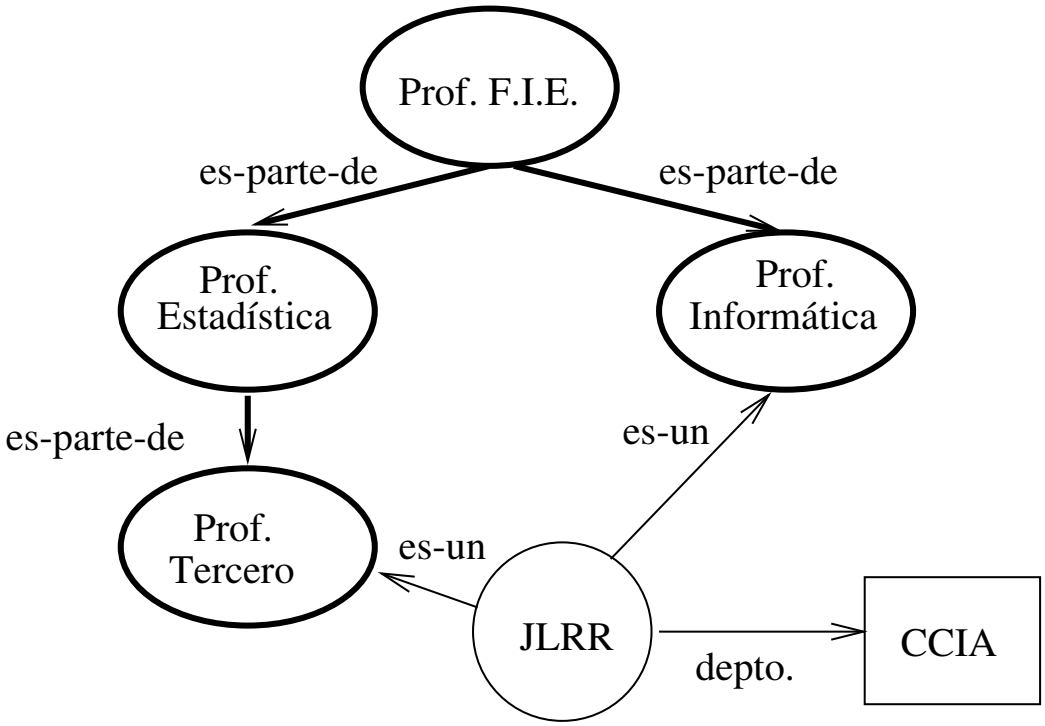
- **Ejemplos:**

```
SI
el problema no me sale Y
es la hora de consulta
ENTONCES
consultar al profesor

SI
la luz del semáforo es verde Y
no hay peatones cruzando
ENTONCES
continúa la marcha

SI
el programa Lisp no carga
ENTONCES
comprobar paréntesis
```

- **Inferencias:**
  - **Razonamiento hacia adelante (de abajo a arriba)**
  - **Razonamiento hacia atrás (de arriba a abajo)**

Redes semánticas.

- **Ejemplo:**



- **Inferencia:**
  - **Herencia**

Lógica:

- **Ejemplos:**
  - $\forall x(Perro(x) \rightarrow Animal(x))$
  - $\exists x(Animal(x) \wedge \neg Perro(x))$
  - $\forall x \exists y(padre(y, x))$
  - $\neg \forall x \exists y(padre(x, y))$
  - $\forall \epsilon \exists n_0 \forall n \geq n_0(|x_n - x_{n_0}| < \epsilon)$
- **Inferencia:**
  - **Deducción natural.**
  - **Cálculo de secuentes.**
  - **Regla de resolución.**
  - **Reescritura.**
- **Razonamiento automático:**
  - **Estudiado en las asignaturas "Lógica Informática" (2^o I.I.), "Lógica y Programación" (3^o I.I.) y "Razonamiento Automático" (5^o I.I.)**

Concimiento mediante reglas

- **En lo que sigue, estudiaremos con más detalle los sistemas basados en reglas:**
  - **Expresar determinados dominios con el formalismo de reglas**
  - **Razonamiento hacia adelante: CLIPS**
  - **Razonamiento hacia atrás: Prolog**
- **Programación en Prolog un SBC básico**
  - **Expresar las reglas como hechos Prolog**
  - **Razonamiento hacia atrás**
  - **Razonamiento hacia atrás con justificaciones**
  - **Preguntas ¿porqué?**
  - **Preguntas ¿cómo?**

Ejemplo:

```
 ?- se_deduce(opel_astra es gama X).

¿Es cierto opel_astra tiene 2000 c_cubicos?
Posibles respuestas: [si, no, porque] (seguidas de un punto): no.

¿Es cierto opel_astra tiene 1600 c_cubicos?
Posibles respuestas: [si, no, porque] (seguidas de un punto): porque.
=== Porque:
La regla r7 dice que probando:
opel_astra tiene 1600 c_cubicos
se tiene:
opel_astra tiene cilindrada media

Repito:
¿Es cierto opel_astra tiene 1600 c_cubicos?
Posibles respuestas: [si, no, porque] (seguidas de un punto): si.

****** Se ha encontrado respuesta afirmativa:
------ Opciones:
------ 1) Ver la prueba completa y continuar.
------ 2) Navegar dentro de la prueba.
------ 3) Continuar.
Posibles respuestas: [1, 2, 3] (seguidas de un punto): 1.

****** Prueba encontrada:
      por usted, sabemos que opel_astra tiene 1600 c_cubicos,
    luego, segun r7 se concluye que opel_astra tiene cilindrada media,
  luego, segun r2 se concluye que opel_astra tiene velocidad media,
  y
    por f3, sabemos que opel_astra no_tiene frenos abs,
    y
    por f2, sabemos que opel_astra tiene airbag,
  luego, segun r10 se concluye que opel_astra tiene seguridad media,
luego, segun r1 se concluye que opel_astra es gama media,

X = media ;

¿Es cierto opel_astra tiene 1400 c_cubicos?
Posibles respuestas: [si, no, porque] (seguidas de un punto): no.
No
```

Reglas y conocimiento

- **Reglas: formalismo mas común de representar el conocimiento en un SBC**
  - **Reglas `si ... entonces ...`**
  - **Sinónimo: reglas de *producción***
- **Interpretaciones:**
  - `Si condición P entonces conclusión C`
  - `Si ocurre S entonces acción A`
- **Ventajas del formalismo de reglas:**
  - **Modulares**
  - **Conocimiento incremental**
  - **Conocimiento modificable**
  - **Separación entre control y conocimiento**
  - **Permiten explicaciones al usuario**
  - **Preguntas ¿cómo? y ¿por qué?**

Ejemplos de reglas

• Regla en MYCIN

```
Si
 1. el reactivo toma el color azul
 2. la morfología del organismo es alargada
 3. el paciente es un posible receptor
Entonces
 existe una evidencia (0.7) de que la infección
 proviene de organismos pseudomonas.
```

• Regla en AL/X

```
Si
 la presión en V01 ha alcanzadola presión de
    apertura de la válvula auxiliar
 y
 la válvula auxiliar de V01 se ha abierto
Entonces
 la válvula auxiliar de V01 se ha abierto prematuramente
 [N=0.001, S=2000]
```

Ejemplos de reglas

- **Regla en CLIPS:**

```
(defrule DETECCION::historia-cambia
  (ciclo ?ciclo)
  ?historia <- (historia-sensor
                 (sensor ?id)
                 (estado ?estado)
                 (ciclo-comienzo ?ini)
                 (ciclo-final ?fin&:(= ?fin (- ?ciclo 1))))
  (estado-sensor (sensor ?id) (estado ?nuevo&~?estado)
                 (valor ?valor) (ciclo ?ciclo))
  =>
  (modify ?historia (estado ?nuevo) (valor ?valor)
                    (ciclo-comienzo ?ciclo)
                    (ciclo-final ?ciclo)))
```

- **En general, las reglas se extraen de:**
  - **Expertos humanos**
  - **Literatura especializada**
- **Extracción del conocimiento:**
  - **Una de las fases fundamentales en la construcción de SBC**

Razonamiento usando reglas

- Razonamiento = Motor de inferencia
- Dos maneras de razonar:
  - Hacia adelante (*forward chaining*)
  - Hacia atrás (*backward chaining*)
- Razonamiento hacia delante:
  - A partir de los hechos, buscar las conclusiones
  - Desde los datos hacia los objetivos
- Razonamiento hacia atrás:
  - Dadas las conclusiones que se buscan, buscar los hechos que las derivan
  - Desde los objetivos hacia los datos

Razonamiento hacia adelante

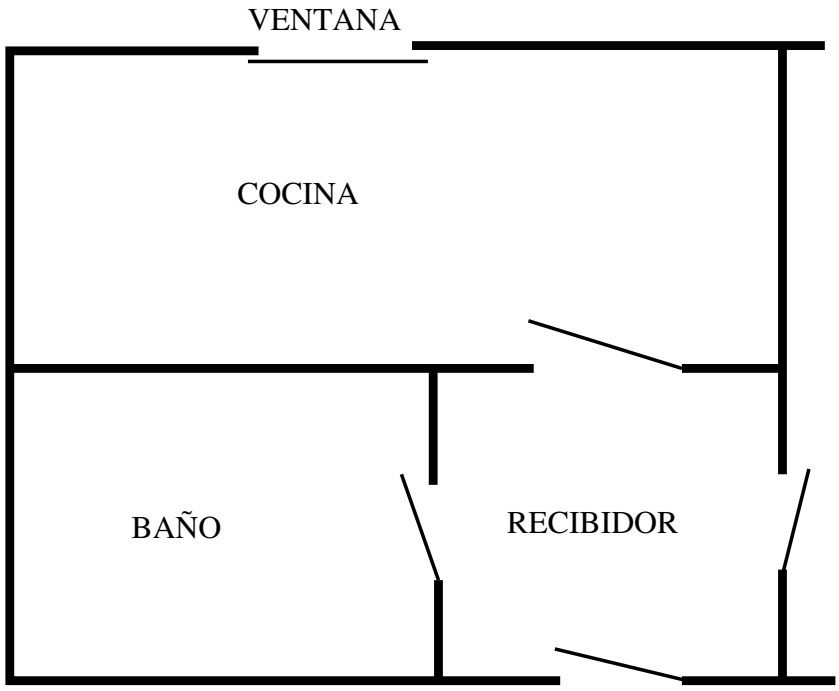
- **Componentes en un sistema de razonamiento hacia adelante:**
  - **Base de hechos (*memoria de trabajo*)**
  - **Base de reglas: *condiciones* (comprobando presencia o ausencia de hechos) y *acciones* (añadiendo o quitando hechos)**
  - **Un algoritmo para calcular eficientemente las reglas cuyas condiciones se satisfacen en cada momento (*activaciones en la agenda*)**
  - **Un método para decidir en cada momento qué regla de las activadas debe actuar (*resolución de conflictos*)**
- **Ciclo de ejecución: activación → resolución de conflictos → disparo → modificación de la base de hechos**
- **RETE: un algoritmo para calcular las activaciones eficientemente**
  - **Clave: en cada ciclo, las modificaciones a la agenda son pequeñas**
- **Como ya hemos visto, CLIPS implementa un modelo de razonamiento con reglas, usando deducción hacia adelante**
  - **Por tanto, CLIPS es un lenguaje muy adecuado para diseñar SBC que usen razonamiento con reglas hacia adelante**

Razonamiento hacia atrás

- A partir de una pregunta (*objetivo*) a la base de conocimiento, comprobar si ésta se deduce de la misma
- Los objetivos se deducen si:
  - Directamente a partir de los hechos, o bien
  - dado un objetivo y una regla que lo tenga como conclusión, las condiciones de la reglas se convierten en los nuevos subobjetivos
- Todo ello combinado con un mecanismo de *unificación*, para tratar la presencia de variables en los objetivos y en las reglas
- Es el tipo de razonamiento básico que existe en el lenguaje Prolog
  - Por tanto, Prolog es un lenguaje muy adecuado para diseñar SBC que usen razonamiento con reglas hacia atrás
- En lo que sigue veremos un SBC básico implementado en Prolog

Un ejemplo "de juguete": fuga de agua

- **Situación:**



- **Descripción:**
  - Si la cocina está seca y el recibidor mojado entonces la fuga de agua está en el baño.
  - Si el recibidor está mojado y el baño está seco entonces el problema está en la cocina.
  - Si la ventana está cerrada o no llueve entonces no entra agua del exterior.
  - Si el problema está en la cocina y no entra agua del exterior, la fuga de agua está en la cocina.

- **Evidencias:**
  - El recibidor está mojado.
  - El baño está seco.
  - La ventana está cerrada.

Ejemplo: fuga de agua

- **Representación directa en Prolog:**

```
fuga_en_bagno :- recibidor_mojado, cocina_seca.
problema_en_cocina :- recibidor_mojado, bagno_seco.
no_agua_exterior :- ventana_cerrada; no_llueve.
fuga_en_cocina :- problema_en_cocina, no_agua_exterior.

recibidor_mojado.
bagno_seco.
ventana_cerrada.
```

- **Pregunta en Prolog:**

```
 ?- fuga_en_cocina.
Yes
```

- **Desventajas:**
  - **El usuario puede no estar familiarizado con la sintaxis de Prolog**
  - **La *base de conocimiento* debe de distinguirse del programa Prolog que implemente el SBC (*motor de inferencia*)**

Sintaxis de reglas y hechos

- **Representación en Prolog:**

```
% La representacion interna de una regla debe ser:
%
%       #
%     /   \
%    /     \
%  Id    entonces
%          / \
%         /   \
%        si   Afirmacion
%         |
%        Cond

%  Cond puede ser una combinacion, usando 'o' o 'y',
%   de afirmaciones (la disyuncion con prioridad menor).

% Los hechos se deben representar internamente como:
%
%           hecho
%           /  \
%          /    \
%         Id    Afirmacion

:- op(875, xfx, hecho).
:- op(875, xfx, #).
:- op(825, fx, si).
:- op(850, xfx, entonces).
:- op(800, xfy, o). % Asociatividad a la derecha
:- op(775, xfy, y). % Asociatividad a la derecha
```

Reglas y hechos en la fuga de agua

- **Base de conocimiento:**

```
r1 # si recibidor_mojado y
        cocina_seca
     entonces
        problema_en_cocina.
r2 # si recibidor_mojado y
        bagno_seco
     entonces
        problema_en_cocina.
r3 # si ventana_cerrada o
        no_llueve
     entonces
        no_agua_exterior.
r4 # si problema_en_cocina y
        no_agua_exterior
     entonces
        fuga_en_cocina.

f1 hecho recibidor_mojado.
f2 hecho bagno_seco.
f3 hecho ventana_cerrada.
```

- **Las reglas y hechos de la base de conocimiento son hechos Prolog.**
- **Operadores Prolog para acercar la expresión de las reglas al lenguaje natural.**

Ejemplo: animales.pl (I)

```
% ========== Bratko, cap. 15

% ----- Operadores adicionales:
:- op(750,xfx,[es,tiene,come,vuela,pone,da,nada]).
:- op(500,fx,[un,una]).

% ----- Reglas:
r1 # si Animal tiene pelo
        o
        Animal da leche
     entonces
        Animal es mamifero.

r2 # si Animal tiene plumas
        o
        Animal vuela bien
        y
        Animal pone huevos
     entonces
        Animal es un pajaro.

r3 # si Animal es mamifero
        y
        (Animal come carne
         o
         Animal tiene 'dientes afilados'
         y
         Animal tiene garras
         y
         Animal tiene 'ojos oblicuos')
      entonces
         Animal es carnivoro.
```

Ejemplo: animales.pl(II)

```
r4 # si Animal es carnivoro
        y
        Animal tiene 'color pardo'
        y
        Animal tiene 'manchas oscuras'
     entonces
        Animal es un guepardo.

r5 # si Animal es carnivoro
        y
        Animal tiene 'color pardo'
        y
        Animal tiene 'rayas negras'
     entonces
        Animal es un tigre.

r6 # si Animal es un pajaro
        y
        Animal vuela mal
        y
        Animal nada bien
     entonces
        Animal es un pinguino.

r7 # si Animal es un pajaro
        y
        Animal vuela 'muy bien'
     entonces
        Animal es una gaviota.
```

Ejemplo: animales.pl (III)

```
%--------- Hechos:

f1 hecho oblongo tiene pelo.
f2 hecho oblongo come carne.
f2 hecho oblongo tiene 'color pardo'.
f3 hecho oblongo tiene 'rayas negras'.
f4 hecho rayo da leche.
f5 hecho rayo tiene 'dientes afilados'.
f6 hecho rayo tiene  garras.
f7 hecho rayo tiene 'ojos oblicuos'.
f8 hecho rayo tiene 'color pardo'.
f9 hecho rayo tiene 'manchas oscuras'.
```

- **Observaciones:**
  - **Uso de operadores adicionales, propios del dominio de conocimiento**
  - **Uso de variables en las reglas**

Ejemplo: coches.pl (I)

```
% ----- Operadores adicionales:
:- op(725,fx,[velocidad,seguridad,capacidad,gama,motor,
              frenos,cilindrada]).
:- op(725,xf,c_cubicos).
:- op(750,xfx,[es,tiene,no_tiene]).


% ----- Reglas:
r1 # si Coche tiene velocidad X y
        (Coche tiene seguridad X
         o
         Coche tiene capacidad X)
         o
        Coche tiene seguridad X y
        Coche tiene capacidad X
     entonces
        Coche es gama X.

r2 # si Coche tiene cilindrada X
     entonces
        Coche tiene velocidad X.

r3 # si Coche tiene motor inyeccion y
        Coche es mono-volumen
     entonces
        Coche tiene velocidad media.

r4 # si Coche tiene motor inyeccion y
        Coche es deportivo
     entonces
        Coche tiene velocidad alta.
```

Ejemplo: coches.pl (II)

```
r5 # si Coche es mono_volumen y
        Coche tiene motor diesel
     entonces
        Coche tiene velocidad baja.

r6 # si Coche tiene 2000 c_cubicos
     entonces
        Coche tiene cilindrada alta.

r7 # si Coche tiene 1600 c_cubicos
     entonces
        Coche tiene cilindrada media.


r8 # si Coche tiene 1400 c_cubicos
     entonces
        Coche tiene cilindrada baja.

r9 # si Coche tiene frenos abs y
        Coche tiene airbag
     entonces
        Coche tiene seguridad alta.

r10 # si Coche tiene frenos abs y
         Coche no_tiene airbag
         o
         Coche no_tiene frenos abs y
         Coche tiene airbag
       entonces
         Coche tiene seguridad media.
```

Ejemplo: coches.pl (III)

```
r11 # si Coche no_tiene frenos abs y
         Coche no_tiene airbag
      entonces
         Coche tiene seguridad baja.

r12 # si Coche es deportivo
      entonces
         Coche tiene capacidad baja.

r13 # si Coche es mono_volumen
      entonces
         Coche tiene capacidad alta.

r14 # si Coche es turismo
      entonces
         Coche tiene capacidad media.

%------ Hechos:

f1 hecho opel_astra tiene 1600 c_cubicos.
f2 hecho opel_astra tiene airbag.
f3 hecho opel_astra no_tiene frenos abs.
f4 hecho fiat_punto tiene 1400 c_cubicos.
f5 hecho fiat_punto no_tiene airbag.
f6 hecho fiat_punto no_tiene frenos abs.
f7 hecho renault_space es mono_volumen.
f8 hecho renault_space tiene motor diesel.
f9 hecho renault_space tiene frenos abs.
f10 hecho renault_space tiene airbag.
```

Razonamiento hacia atrás

- **Meta-intérpretes**
- **Programa Prolog:**

```
se_deduce(P) :-
        _ hecho P.

se_deduce(P) :-
        _ # si C entonces P,
        se_deduce(C).

se_deduce(P1 y P2) :-
        se_deduce(P1),
        se_deduce(P2).

se_deduce(P1 o _) :-
        se_deduce(P1).

se_deduce(_ o P2) :-
        se_deduce(P2).
```

- **Ejemplo:**

```
?-  se_deduce(fuga_en_cocina).

Yes
?- se_deduce(fuga_en_bagno).

No
```

Hacia adelante *vs* hacia atrás

- **Deducción = búsqueda en espacios de estados:**

```
Datos --> .... --> Objetivos
Evidencias --> .... --> Hipótesis
Observaciones --> .... --> Justificaciones
Síntomas --> ... --> Diagnóstico
```

- **La dirección de la búsqueda determina el tipo de razonamiento:**
  - **Hacia adelante: de los hechos hacia las conclusiones**
  - **Hacia atrás: de los objetivos hacia los hechos**
- **Problemas adecuados para razonar hacia adelante:**
  - **Monitorización y control**
  - **Problemas dirigidos por los datos**
  - **Sin necesidad de explicación**
- **Problemas adecuados para razonar hacia atrás:**
  - **Diagnóstico**
  - **Problemas dirigidos por los objetivos**
  - **Interacción/Explicación al usuario**

Razonamiento con incertidumbre

- **El conocimiento puede no ser *categórico***
  - **Reglas y hechos tienen un grado de certeza**
  - **Ejemplos:**

```
- Si el recibidor está mojado y el baño está seco,
  entonces existe probabilidad 0.9 de que el
  problema esté en la cocina.
- Creo que ha llovido, con probabilidad 0.8.
```

- **Reglas y hechos afectados de un factor de *certeza***
- **Modificación de la sintaxis:**

```
:- op(860, xfx, :). % factor de certeza.
:- op(875, xfx, hecho).
:- op(875, xfx, #).
:- op(825, fx, si).
:- op(850, xfx, entonces).
:- op(800, xfy, o).
:- op(775, xfy, y).
```

- **Ejemplo**

:

```
r2 # si recibidor_mojado y
        bagno_seco
     entonces
        problema_en_cocina : 0.9 .

f4 hecho no_llueve : 0.8.
```

Fuga de agua con incertidumbre

- **Base de conocimiento:**

```
r1 # si recibidor_mojado y
        cocina_seca
     entonces
        fuga_en_bagno : 0.8.
r2 # si recibidor_mojado y
        bagno_seco
     entonces
        problema_en_cocina : 0.9 .
r3 # si ventana_cerrada o
        no_llueve
     entonces
        no_agua_exterior : 1.
r4 # si problema_en_cocina y
        no_agua_exterior
     entonces
        fuga_en_cocina : 0.8 .

f1 hecho recibidor_mojado : 1.
f2 hecho bagno_seco : 1.
f3 hecho ventana_cerrada : 0.
f4 hecho no_llueve : 0.8.
f5 hecho cocina_seca : 0.
```

Razonamiento con incertidumbre

- **Manejo de probabilidades**

  Cada combinación lógica de proposiciones P, tiene asignado un coeficiente de certeza, c(P):
  - Los hechos, valores asignados en la BC.
  - Si P2 se obtiene mediante la regla
    _ si P1 entonces P2 : C
    entonces c(P2) = c(P1) * C
  - c(P1 y P2) = min(c(P1),c(P2))
  - c(P1 o P2) = max(c(P1),c(P2))

- **Razonamiento hacia atrás con probabilidad**

```
certeza(P,C) :-
        _ hecho P : C.

certeza(P,C) :-
        _ # si Cond entonces P : C1,
        certeza(Cond,C2),
        C is C1 * C2.

certeza(P1 y P2,C) :-
        certeza(P1,C1),
        certeza(P2,C2),
        C is min(C1,C2).

certeza(P1 o P2,C) :-
        certeza(P1,C1),
        certeza(P2,C2),
        C is max(C1,C2).
```

- **Ejemplo:**

```
?- certeza(fuga_en_cocina,C).
C = 0.64
?- certeza(fuga_en_bagno,C).
C = 0
```

Incertidumbre: observaciones.

- **Simplificaciones:**
  - **Distintas reglas no deben servir para deducir lo mismo**
  - **Independencia de las observaciones**
- **La realidad es más compleja**
- **Dos aproximaciones:**
  - **Modelar el razonamiento usando teoría de la probabilidad**

    ```
    * Ventajas: deducciones correctas.
    * Desventajas: demasiado complejo.
    ```

  - **Simplificar el tratamiento probabilístico**

    ```
    * Ventajas: funciona en la práctica.
    * Desventajas: posibles razonamientos incorrectos.
    ```

- **Existe numerosas técnicas para manejar el razonamiento inexacto y aproximado, que no trataremos aquí**

Implementación Prolog de un SBC

- **En lo que sigue, veremos la construcción en Prolog de un SBC *básico***
- **Un *armazón (shell)* de sistemas expertos**
- **Independiente del dominio de conocimiento**
- **Características a implementar (incrementalmente):**
  - **Razonamiento hacia atrás**
  - **Justificaciones a las respuestas (pruebas)**
  - **Acotación de profundidad y objetivos retardables**
  - **preguntas al usuario**
  - **Preguntas ¿cómo?**
  - **Preguntas ¿porqué?**
  - **Navegación por la prueba**
- **Características que <u>no</u> implementaremos:**
  - **Preguntas ¿quién?**
  - **Negación**
  - **Incertidumbre**
  - **Razonamiento hacia adelante**

Razonamiento hacia atrás

- **Recordar: fichero `sbc1.pl`**

```
:- op(875, xfx, hecho).
:- op(875, xfx, #).
:- op(825, fx, si).
:- op(850, xfx, entonces).
:- op(800, xfy, o).
:- op(775, xfy, y).

se_deduce(P) :-
        _ hecho P.

se_deduce(P) :-
        _ # si C entonces P,
        se_deduce(C).

se_deduce(P1 y P2) :-
        se_deduce(P1),
        se_deduce(P2).

se_deduce(P1 o _) :-
        se_deduce(P1).

se_deduce(_ o P2) :-
        se_deduce(P2).
```

Sesión

```
1 ?- [sbc1].
sbc1 compiled, 0.00 sec, 1,736 bytes.
...
2 ?- [coches].
coches compiled, 0.00 sec, 4,900 bytes.
...
3 ?- se_deduce(Que_coche es gama media).

Que_coche = opel_astra ;
...
4 ?- se_deduce(renault_space es gama X).

X = alta ;
...
5 ?- [animal].
...
Yes
6 ?- se_deduce(Quien es un tigre).

Quien = oblongo ;
...
7 ?- se_deduce(rayo es un Que).

Que = guepardo ;
...
8 ?- se_deduce(rayo es una gaviota).

No
```

Búsqueda con profundidad acotada

- **Fichero** `sbc2.pl`

```
% ...
se_deduce_acotado(P,_) :- _ hecho P.

se_deduce_acotado(P,N) :-
        N > 0, N1 is N - 1,
        _ # si C entonces P,
        se_deduce_acotado(C,N1).

se_deduce_acotado(P1 y P2,N) :-
        se_deduce_acotado(P1,N),
        se_deduce_acotado(P2,N).

se_deduce_acotado(P1 o _,N) :-
        se_deduce_acotado(P1,N).

se_deduce_acotado(_ o P2,N) :-
        se_deduce_acotado(P2,N).
```

- **Sesión:**

```
1 ?- [sbc2].
...
2 ?- [coches].
...
3 ?- se_deduce_acotado(opel_astra es gama media,1).
No
4 ?- se_deduce_acotado(opel_astra es gama media,2).
No
5 ?- se_deduce_acotado(opel_astra es gama media,3).
Yes
```

Objetivos retardables

- **Retardar objetivos:**
  - **Objetivos cuya respuesta no se busca**
  - **La respuesta devuelta es cierta siempre que sean ciertos los objetivos que se han dejado pendientes**
  - **Se obtiene información de lo que se necesita para que algo sea cierto**
  - **Puede servir para construir nuevas reglas**
- **Fichero** `sb3.pl`

```
% ...
se_deduce_r(P,R,R) :-  _ hecho P.

se_deduce_r(P,R1,R2) :-
        _ # si C entonces P,
        se_deduce_r(C,R1,R2).

se_deduce_r(P1 y P2,R1,R3) :-
        se_deduce_r(P1,R1,R2),
        se_deduce_r(P2,R2,R3).

se_deduce_r(P1 o _, R1,R2) :- se_deduce_r(P1,R1,R2).

se_deduce_r(_ o P2,R1,R2) :- se_deduce_r(P2,R1,R2).

se_deduce_r(P,R,[P|R]) :- retardable(P).
```

Sesión

```
% Ejemplo:
1 ?- [sbc3].
...
2 ?- [coches].
...
3 ?- se_deduce_r(X es gama media,[],L).
X = opel_astra
L = [opel_astra tiene 1600 c_cubicos] ;
...
4 ?- se_deduce_r(renault_space es gama media,[],L).
No
5 ?- se_deduce_r(X es gama Y,[],L).
X = renault_space
Y = alta
L = [renault_space tiene 2000 c_cubicos] ;

X = renault_space
Y = alta
L = [renault_space tiene 2000 c_cubicos] ;

X = opel_astra
Y = media
L = [opel_astra tiene 1600 c_cubicos] ;

X = fiat_punto
Y = baja
L = [fiat_punto tiene 1400 c_cubicos] ;

X = renault_space
Y = alta
L = [] ;
...
```

Respuestas con justificación o prueba

- **Estructura de las pruebas:**

```
P es cierto esta_afirmado_por F
P es cierto es_consecuencia_de R porque Prueba
Prueba1 y Prueba2
```

- **Operadores:**

```
:- op(850, xfx, es_consecuencia_de).
:- op(850, xfx, esta_afirmado_por).
:- op(800, xfx, porque).
:- op(750, xfx, es).
```

- **Fichero** `sbc4.pl`

```
% ....
se_deduce(P, P es cierto esta_afirmado_por F) :-
        F hecho P.

se_deduce(P, P es cierto es_consecuencia_de R porque Pr) :-
        R # si C entonces P,
        se_deduce(C, Pr).

se_deduce(P1 y P2, Prueba1 y Prueba2) :-
        se_deduce(P1, Prueba1),
        se_deduce(P2, Prueba2).

se_deduce(P1 o _, Prueba1) :-
        se_deduce(P1, Prueba1).

se_deduce(_ o P2, Prueba2) :-
        se_deduce(P2, Prueba2).
```

Escritura de pruebas

- **Fichero** `sbc4.pl`

```
se_deduce(P) :-
      se_deduce(P,Prueba),
      escribe_prueba(Prueba).


escribe_prueba(Prueba) :-
        nl,write_ln('****** Prueba encontrada:'),nl,
        escribe_prueba_aux(0,Prueba).


escribe_prueba_aux(N,A es cierto
                       es_consecuencia_de B
                       porque Pr):-
        M is N+2,
        escribe_prueba_aux(M,Pr),tab(N),
        escribe_lista(['luego, segun ',B,
                       ' se concluye que ',A,',']),nl.

escribe_prueba_aux(N,A y B):-
        escribe_prueba_aux(N,A),tab(N),write_ln('y'),
        escribe_prueba_aux(N,B).

escribe_prueba_aux(N,P es cierto esta_afirmado_por F):-
        tab(N),escribe_lista(['por ',F,',
                              sabemos que ',P,',']),nl.

escribe_lista([]).
escribe_lista([X|L]) :- write(X), escribe_lista(L).
```

Sesión

```
6 ?- [sbc4].
...
Yes
7 ?- [animal].
...
8 ?- se_deduce(X es un tigre).
****** Prueba encontrada:
      por f1, sabemos que oblongo tiene pelo,
    luego, segun r1 se concluye que oblongo es mamifero,
    y
    por f2, sabemos que oblongo come carne,
  luego, segun r3 se concluye que oblongo es carnivoro,
  y
  por f2, sabemos que oblongo tiene color pardo,
  y
  por f3, sabemos que oblongo tiene rayas negras,
luego, segun r5 se concluye que oblongo es un tigre,
X = oblongo ;
...
9 ?- [coches].
...
10 ?- se_deduce(renault_space es gama X).
****** Prueba encontrada:
    por f9, sabemos que renault_space tiene frenos abs,
    y
    por f10, sabemos que renault_space tiene airbag,
  luego, segun r9 se concluye que renault_space tiene seguridad alta,
  y
    por f7, sabemos que renault_space es mono_volumen,
  luego, segun r13 se concluye que renault_space tiene capacidad alta,
luego, segun r1 se concluye que renault_space es gama alta,
X = alta ;
...
```

Interacción con el usuario

- **Interacción**
  - **Preguntas del usuario al sistema**
  - **Preguntas del sistema al usuario**
- **Predicado `recoge_respuesta`:**

```
recoge_respuesta(R,L) :-
        escribe_lista(['Posibles respuestas: ',L,
                       ' (seguidas de un punto): ']),
        read(R),member(R,L),!.
recoge_respuesta(R,L) :-
        nl,write('Respuesta no valida.'),nl,
        recoge_respuesta(R,L).
```

- **Sesión:**

```
13 ?- recoge_respuesta(R,[si,no]).
Posibles respuestas:[si, no](seguidas de un punto):ni.

Respuesta no valida.
Posibles respuestas:[si, no](seguidas de un punto):so.

Respuesta no valida.
Posibles respuestas:[si, no](seguidas de un punto):si.

R = si ;
No
```

Hechos preguntables

- **Requiriendo información al usuario:**
  - **En general, toda la información no se proporciona de antemano**
  - **Un hecho es *preguntable* si el sistema se lo puede preguntar al usario, en lugar de deducirlo**
- **Predicado `preguntable`**
- **Cambio en las bases de conocimiento:**
  - **Modificación en `coches.pl`**

```
preguntable(_ tiene _ c_cubicos).

% f2 y f4 NO se dan de entrada.
```

  - **Modificación en `animal.pl`**

```
preguntable(_ vuela _).
preguntable(_ come _).
preguntable(_ tiene pelo).
preguntable(_ tiene 'color pardo').
preguntable(_ tiene 'rayas negras').
preguntable(_ tiene garras).

% f1, f2 f3, f4 y f6 NO se dan de entrada.
```

- **Un nuevo tipo de *prueba*:**

```
P es cierto esta_afirmado_por usted
```

SBC con hechos preguntables: sbc5.pl

```
:- dynamic respuesta/2.
...
se_deduce(P, P es cierto esta_afirmado_por F) :-
        F hecho P.

se_deduce(P,P es cierto es_consecuencia_de R porque Pr):-
        R # si C entonces P,
        se_deduce(C, Pr).

se_deduce(P1 y P2, Prueba1 y Prueba2) :-
        se_deduce(P1, Prueba1),
        se_deduce(P2, Prueba2).

se_deduce(P1 o _, Prueba1) :-
        se_deduce(P1, Prueba1).

se_deduce(_ o P2, Prueba2) :-
        se_deduce(P2, Prueba2).

se_deduce(P,P es cierto esta_afirmado_por usted) :-
        preguntable(P),
        respuesta(P,R),!,R=si.

se_deduce(P, P es cierto esta_afirmado_por usted) :-
        preguntable(P),
        pregunta(P,R),
        assert(respuesta(P,R)),R=si.

pregunta(P,R) :-
        nl,escribe_lista(['¿Es cierto ',P,'?']),nl,
        recoge_respuesta(R,[si,no]).
...
```

Preguntas ¿porqué?

- **Son útiles para el usuario:**
  - **Se gana en confianza**
  - **A posteriori, permite depurar las reglas**
  - **Preguntas irrelevantes**
  - **Aprendizaje del usuario**
- **¿Por qué se pregunta al usuario ? Cadena de razonamiento:**

```
El sistema pregunta ¿opel_astra tiene 1600 c_cubicos?
y esto se usa, por r7, para probar 'cilindrada media'
y esto se usa, por r2, para probar 'velocidad media'
y esto se usa, por r1, para probar 'gama media'
y esa es precisamente la pregunta original del usuario
```

- **Traza: lista de reglas conectando el objetivo actual con la pregunta del usuario**
- **Modificación al SBC:**
  - `se_deduce` **debe llevar un argumento más con la traza**
  - **sucesivas preguntas** `¿porque?` **muestran una regla de la traza**

Sesión

```
22 ?- se_deduce(opel_astra es gama media).
¿Es cierto opel_astra tiene 1600 c_cubicos?
Posibles respuestas: [si, no, porque] (seguidas de un punto): porque.
=== Porque:
La regla r7 dice que probando:
opel_astra tiene 1600 c_cubicos
se tiene: opel_astra tiene cilindrada media
**** Repito:
¿Es cierto opel_astra tiene 1600 c_cubicos?
Posibles respuestas: [si, no, porque] (seguidas de un punto): porque.
=== Porque:
La regla r2 dice que probando:
opel_astra tiene cilindrada media
se tiene: opel_astra tiene velocidad media
**** Repito:
¿Es cierto opel_astra tiene 1600 c_cubicos?
Posibles respuestas: [si, no, porque] (seguidas de un punto): porque.
=== Porque:
La regla r1 dice que probando:
 opel_astra tiene velocidad media y
 (opel_astra tiene seguridad media o opel_astra tiene capacidad media)o
 opel_astra tiene seguridad media y opel_astra tiene capacidad media
se tiene: opel_astra es gama media
**** Repito:
¿Es cierto opel_astra tiene 1600 c_cubicos?
Posibles respuestas: [si, no, porque] (seguidas de un punto): porque.
Porque esa fue su pregunta!!
Repito:
¿Es cierto opel_astra tiene 1600 c_cubicos?
Posibles respuestas: [si, no, porque] (seguidas de un punto): si.

****** Prueba encontrada:....
```

SBC con ¿por qué?: sbc6.pl (I)

```
....
se_deduce(P) :-
      se_deduce(P,[],Prueba),nl,nl,
      escribe_prueba(Prueba).

se_deduce(P,_,P es cierto esta_afirmado_por F) :-
        F hecho P.

se_deduce(P,T,P es cierto es_consecuencia_de R porque Pr):-
        R # si C entonces P,
        se_deduce(C,[R # si C entonces P|T],Pr).


se_deduce(P1 y P2,T,Prueba1 y Prueba2) :-
        se_deduce(P1,T,Prueba1),
        se_deduce(P2,T,Prueba2).


se_deduce(P1 o _, T,Prueba1) :-
        se_deduce(P1,T,Prueba1).

se_deduce(_ o P2, T,Prueba2) :-
        se_deduce(P2,T,Prueba2).

se_deduce(P,_,P es cierto esta_afirmado_por usted) :-
        preguntable(P),
        respuesta(P,R),  % sólo habra si o no.
        !,R=si.

se_deduce(P,T,P es cierto esta_afirmado_por usted) :-
        preguntable(P),
        pregunta(P,T,R),
        assert(respuesta(P,R)),R=si.
...
```

SBC con ¿por qué?: sbc6.pl (II)

```
...
pregunta(P,T,Respuesta) :-
        nl,escribe_lista(['¿Es cierto ',P,'?']),nl,
        recoge_respuesta(Leido,[si,no,porque]),
        gestiona_respuesta_porque(P,T,Respuesta,Leido).


gestiona_respuesta_porque(_,_,si,si).

gestiona_respuesta_porque(_,_,no,no).

gestiona_respuesta_porque(P,[R|Reglas],Respuesta,porque) :-
        escribe_regla(R),nl,write('Repito:'),
        pregunta(P,Reglas,Respuesta).

gestiona_respuesta_porque(P,[],Respuesta,porque) :-
        write_ln('Porque esa fue su pregunta!!'),
        write('Repito: '),
        pregunta(P,[],Respuesta).


escribe_regla(R # si C entonces P) :-
        nl,write('=== Porque:'),nl,
        escribe_lista(['La regla ',R,
                       ' dice que probando:']),
        nl,write_ln(C), % Se puede mejorar
        write('se tiene: '),nl,write(P),nl.
...
```

Árboles de prueba

- **Prueba de** `opel_astra es gama media`**:**

```
gama media
 R1 |
    |-- velocidad media
    |      R2    |
    |            |-- cilindrada media
    |                  R7     |
    |                         |-- 1600 c_cubicos
    |                                 U
    |-- seguridad media
           R10   |
                 |-- no_tiene frenos_abs
                 |        F3
                 |-- tiene airbag
                          F2
```

- **Sesión:**

```
...
Por r1, opel_astra es gama media es consecuencia de:
   1: opel_astra tiene velocidad media
   2: opel_astra tiene seguridad media
Posibles respuestas: [seguir, 1, 2] (seguidas de un punto): 1.
Por r2, opel_astra tiene velocidad media es consecuencia de:
   1: opel_astra tiene cilindrada media
Posibles respuestas: [seguir, arriba, 1] (seguidas de un punto): 1.
Por r7, opel_astra tiene cilindrada media es consecuencia de:
   1: opel_astra tiene 1600 c_cubicos
Posibles respuestas: [seguir, arriba, 1] (seguidas de un punto): 1.
opel_astra tiene 1600 c_cubicos esta afirmado por usted.
Posibles respuestas: [seguir, arriba] (seguidas de un punto): arriba.
Por r7, opel_astra tiene cilindrada media es consecuencia de:
   1: opel_astra tiene 1600 c_cubicos
....
```

Navegación por la prueba

```
navega(P es cierto esta_afirmado_por usted,Nivel) :-
   !,
   escribe_lista([P,' esta afirmado por usted.']),
   nl,
   opciones_no_numericas(Nivel,Opciones),
   recoge_respuesta(Opcion,Opciones),
   gestiona_respuesta_navega(Opcion,Nivel,_,_).

navega(P es cierto esta_afirmado_por F,Nivel) :-
   !,
   escribe_lista([P,' esta afirmado por el hecho ',F]),
   nl,
   opciones_no_numericas(Nivel,Opciones),
   recoge_respuesta(Opcion,Opciones),
   gestiona_respuesta_navega(Opcion,Nivel,_,_).

navega(A es cierto es_consecuencia_de B porque Pruebas,
           Nivel) :-
   !,
   escribe_lista(['Por ',B,',',A,'es consecuencia de:']),
   nl,
   escribe_cuerpo(Pruebas,1,Max),
   Max1 is Max - 1,
   findall(N,between(1,Max1,N),L),
   opciones_no_numericas(Nivel,M),
   append(M,L,Opciones),
   recoge_respuesta(Opcion,Opciones),
   (A es cierto es_consecuencia_de B porque Pruebas)=Aux,
   gestiona_respuesta_navega(Opcion,Nivel,Pruebas,Aux).

opciones_no_numericas(0,[seguir]):-!.
opciones_no_numericas(_,[seguir,arriba]).
```

Navegación por la prueba (II)

- **Respuesta del usuario:**

```
gestiona_respuesta_navega(arriba,_,_,_):-!.
gestiona_respuesta_navega(seguir,_,_,_):-!,fail.
gestiona_respuesta_navega(N,Nivel,Pruebas,P) :-
        n_esima_prueba(Pruebas,N,Pn),
        Nivel1 is Nivel+1,
        navega(Pn,Nivel1),
        navega(P,Nivel).
```

- **Funciones auxiliares:**

```
escribe_cuerpo(P1 y P2,N1,N3) :-
       !, escribe_cuerpo(P1,N1,N2), escribe_cuerpo(P2,N2,N3).

escribe_cuerpo(P,N,N1) :-
       tab(3), escribe_lista([N,': ']), escribe_lo_probado(P),nl,
       N1 is N+1.

escribe_lo_probado(P es cierto esta_afirmado_por _):- write(P).
escribe_lo_probado(P es cierto es_consecuencia_de _):- write(P).


n_esima_prueba((P1 y P2) y P3,N,P) :- !,n_esima_prueba(P1 y P2 y P3,N,P).
n_esima_prueba(P1 y _,1,P1):-!.
n_esima_prueba(_ y P2,N,P) :- !, N1 is N-1, n_esima_prueba(P2,N1,P).
n_esima_prueba(P,1,P).
```

Ejemplo de sesión (I)

```
1 ?- [sbc].
...
2 ?- [animal].
...
3 ?- se_deduce(oblongo es un Que).

¿Es cierto oblongo vuela bien?
Posibles respuestas: [si, no, porque] (seguidas de un punto): no.

¿Es cierto oblongo tiene pelo?
Posibles respuestas: [si, no, porque] (seguidas de un punto): si.

¿Es cierto oblongo come carne?
Posibles respuestas: [si, no, porque] (seguidas de un punto): porque.

=== Porque:
La regla r3 dice que probando:
oblongo es mamifero y (oblongo come carne o
oblongo tiene dientes afilados y oblongo tiene garras
y oblongo tiene ojos oblicuos)
se tiene:
oblongo es carnivoro
Repito:
¿Es cierto oblongo come carne?
Posibles respuestas: [si, no, porque] (seguidas de un punto): porque.

=== Porque:
La regla r4 dice que probando:
oblongo es carnivoro y oblongo tiene color pardo y
  oblongo tiene manchas oscuras
se tiene:
oblongo es un guepardo
```

Ejemplo de sesión (II)

```
Repito:
¿Es cierto oblongo come carne?
Posibles respuestas: [si, no, porque] (seguidas de un punto): porque.
Porque esa fue su pregunta!!
Repito:
¿Es cierto oblongo come carne?
Posibles respuestas: [si, no, porque] (seguidas de un punto): si.

¿Es cierto oblongo tiene color pardo?
Posibles respuestas: [si, no, porque] (seguidas de un punto): si.

¿Es cierto oblongo tiene rayas negras?
Posibles respuestas: [si, no, porque] (seguidas de un punto): si.
```

Ejemplo de sesión (III)

```
****** Se ha encontrado respuesta afirmativa:
------ Opciones:
------ 1) Ver la prueba completa y continuar.
------ 2) Navegar dentro de la prueba.
------ 3) Continuar.
Posibles respuestas: [1, 2, 3] (seguidas de un punto): 2.
Por r5, oblongo es un tigre es consecuencia de:
   1: oblongo es carnivoro
   2: oblongo tiene color pardo
   3: oblongo tiene rayas negras
Posibles respuestas: [seguir, 1, 2, 3] (seguidas de un punto): 3.
oblongo tiene rayas negras esta afirmado por usted.
Posibles respuestas: [seguir, arriba] (seguidas de un punto): 1.

Respuesta no valida.
Posibles respuestas: [seguir, arriba] (seguidas de un punto): arriba.
Por r5, oblongo es un tigre es consecuencia de:
   1: oblongo es carnivoro
   2: oblongo tiene color pardo
   3: oblongo tiene rayas negras
Posibles respuestas: [seguir, 1, 2, 3] (seguidas de un punto): 1.
Por r3, oblongo es carnivoro es consecuencia de:
   1: oblongo es mamifero
   2: oblongo come carne
Posibles respuestas: [seguir, arriba, 1, 2] (seguidas de un punto): 1.
Por r1, oblongo es mamifero es consecuencia de:
   1: oblongo tiene pelo
Posibles respuestas: [seguir, arriba, 1] (seguidas de un punto): 1.
oblongo tiene pelo esta afirmado por usted.
Posibles respuestas: [seguir, arriba] (seguidas de un punto): seguir.

No
```

Limitaciones y posibles ampliaciones

- **Preguntas ¿quién?**

```
3 ?- se_deduce(Quien es gama media).

¿Es cierto _G264 tiene 1600 c_cubicos?
Posibles respuestas:[si,no,porque](seguidas de un punto):
```

- **Distintas maneras de deducir lo mismo**
- **Reglas con incertidumbre**
  - **Razonamiento probabilístico**
- **Otros métodos de representar el conocimiento y de razonar**
  - **Marcos y redes semánticas**
  - **Razonamiento hacia adelante, combinación atrás-adelante**
- **Adquisición del conocimiento**
- **Aprendizaje**

Bibliografía

- Bratko, I. *Prolog Programming for Artificial Intelligence (2nd ed.)* (Addison–Wesley, 1990)
  - Cap. 15: "An expert system shell"
- Giarratano, J.C. y Riley, G. *Sistemas expertos: Principios y programación (3 ed.)* (International Thomson Editores, 2001)
  - Cap. 1: "Introducción a los sistemas expertos"
  - Cap. 2: "La representación del conocimiento"
  - Cap. 3: "Métodos de inferencia"
- Lucas, P. y Gaag L.v.d. *Principles of expert systems* (Addison–Wesley, 1991)
  - Cap. 1: "Introduction"
  - Cap. 3: "Production rules and inference"
- Poole, D; Mackworth, A. y Goebel, R. *Computational intelligence: A logical Approach* (Oxford University press, 1998)
  - Cap 5: "Knowledge engineering"

Apéndice: sbc.pl (I)

```
% ========================================
% sbc.pl
% Un pequeño sistema basado en el conocimiento
% ========================================

%%%%%%%%%% Representacion del conocimiento
% La representacion interna de una regla debe ser:
%
%       #
%     /   \
%    /     \
%  Id    entonces
%          / \
%         /   \
%        si   Afirmacion
%         |
%        Cond
%  Cond puede ser una combinacion, usando 'o' o 'y', de hechos (la
%  disyuncion con prioridad mayor).
% Ademas los hechos se deben representar internamente como:
%
%           hecho
%           /  \
%          /    \
%         Id    Afirmacion
:- op(875, xfx, hecho).
:- op(875, xfx, #).
:- op(825, fx, si).
:- op(850, xfx, entonces).
:- op(800, xfy, o). % Asociatividad a la derecha
:- op(775, xfy, y). % Asociatividad a la derecha
```

Apéndice: sbc.pl (II)

```
%%%%%%%%%%%%%%%%%%% Representación de las pruebas
% P es cierto esta_afirmado_por H (H es un número de hecho o usted).
% P es_consecuencia_de R porque Prueba.

:- op(850, xfx, es_consecuencia_de).
:- op(850, xfx, esta_afirmado_por).
:- op(800, xfx, porque).
:- op(750, xfx, es).

:- dynamic respuesta/2.

%%%%%%%%%%%% Motor de inferencia

se_deduce(P) :-
        se_deduce(P,[],Prueba),
        menu,
        recoge_respuesta(Leida,[1,2,3]),
        (Leida=1,escribe_prueba(Prueba);
         Leida=2,navega(Prueba,0);
         Leida=3).

se_deduce(P,_,P es cierto esta_afirmado_por F) :-
        F hecho P.

se_deduce(P,T,P es cierto es_consecuencia_de R porque Prueba ) :-
        R # si C entonces P,
        se_deduce(C,[R # si C entonces P|T],Prueba).


se_deduce(P1 y P2,T,Prueba1 y Prueba2) :-
        se_deduce(P1,T,Prueba1),
        se_deduce(P2,T,Prueba2).
```

Apéndice: sbc.pl(III)

```
se_deduce(P1 o _, T,Prueba1) :- se_deduce(P1,T,Prueba1).

se_deduce(_ o P2, T,Prueba2) :- se_deduce(P2,T,Prueba2).

se_deduce(P,_,P es cierto esta_afirmado_por usted) :-
        preguntable(P), respuesta(P,R),  % sólo habra si o no.
        !,R=si.

se_deduce(P,T,P es cierto esta_afirmado_por usted) :-
        preguntable(P), pregunta(P,T,R), assert(respuesta(P,R)),
        R=si.

%%%%%%%%%%%% Menú final
menu :-
        nl,nl,write_ln('****** Se ha encontrado respuesta afirmativa:'),
        write_ln('------ Opciones:'),
        write_ln('------ 1) Ver la prueba completa y continuar.'),
        write_ln('------ 2) Navegar dentro de la prueba.'),
        write_ln('------ 3) Continuar. ').

%%%%%%%%%%% Auxiliares
recoge_respuesta(R,L) :-
        write('Posibles respuestas: '),write(L),
        write(' (seguidas de un punto): '),
        read(R),member(R,L),!.
recoge_respuesta(R,L) :-
        nl,write_ln('Respuesta no valida.'),
        recoge_respuesta(R,L).

escribe_lista([]).
escribe_lista([X|L]) :- write(X), escribe_lista(L).
```

Apéndice: sbc.pl (IV)

```
%%%%%%%%%%% Escritura de la prueba
escribe_prueba(Prueba) :-
        nl,write_ln('****** Prueba encontrada:'),nl,
        escribe_prueba_aux(0,Prueba).

escribe_prueba_aux(N,A es cierto es_consecuencia_de B porque Prueba):-
        M is N+2,
        escribe_prueba_aux(M,Prueba),tab(N),
        escribe_lista(['luego, segun ',B,' se concluye que ',A,',']),nl.

escribe_prueba_aux(N,A y B):-
        escribe_prueba_aux(N,A),tab(N),write_ln('y'),
        escribe_prueba_aux(N,B).

escribe_prueba_aux(N,P es cierto esta_afirmado_por F):-
        tab(N),escribe_lista(['por ',F,', sabemos que ',P,',']),nl.

%%%%%%%%%%%% Preguntas al usuario. Preguntas porque
pregunta(P,T,Respuesta) :-
        nl,escribe_lista(['¿Es cierto ',P,'?']),nl,
        recoge_respuesta(Leido,[si,no,porque]),
        gestiona_respuesta_porque(P,T,Respuesta,Leido).

gestiona_respuesta_porque(_,_,si,si).
gestiona_respuesta_porque(_,_,no,no).
gestiona_respuesta_porque(P,[R|Reglas],Respuesta,porque) :-
        escribe_regla(R),nl,write('Repito:'),
        pregunta(P,Reglas,Respuesta).
gestiona_respuesta_porque(P,[],Respuesta,porque) :-
        write_ln('Porque esa fue su pregunta!!'),write('Repito: '),
        pregunta(P,[],Respuesta).
```

Apéndice: sbc.pl (V)

```
escribe_regla(R # si C entonces P) :-
        nl,write_ln('=== Porque:'),
        escribe_lista(['La regla ',R,' dice que probando:']),nl,
        write_ln(C),
        % escribe_condicion(C), cuando se mejore el de abajo.
        write_ln('se tiene: '),write_ln(P).

%%%%%%%%%%%%% Navegación por el árbol.
navega(P es cierto esta_afirmado_por usted,Nivel) :-
        !,
        escribe_lista([P,' esta afirmado por usted.']),nl,
        opciones_no_numericas(Nivel,Opciones),
        recoge_respuesta(Opcion,Opciones),
        gestiona_respuesta_navega(Opcion,Nivel,_,_).

navega(P es cierto esta_afirmado_por F,Nivel) :-
        !,escribe_lista([P,' esta afirmado por el hecho ',F]),nl,
        opciones_no_numericas(Nivel,Opciones),
        recoge_respuesta(Opcion,Opciones),
        gestiona_respuesta_navega(Opcion,Nivel,_,_).

navega(A es cierto es_consecuencia_de B porque Pruebas,Nivel) :-
        !,
        escribe_lista(['Por ',B,', ',A,' es consecuencia de:']),nl,
        escribe_cuerpo(Pruebas,1,Max),
        Max1 is Max - 1,
        findall(N,between(1,Max1,N),L),
        opciones_no_numericas(Nivel,M),
        append(M,L,Opciones),
        recoge_respuesta(Opcion,Opciones),
        (A es cierto es_consecuencia_de B porque Pruebas) = Aux,
        gestiona_respuesta_navega(Opcion,Nivel,Pruebas,Aux).
```

Apéndice: sbc.pl (VI)

```
gestiona_respuesta_navega(arriba,_,_,_):-!.
gestiona_respuesta_navega(seguir,_,_,_):-!,fail.
gestiona_respuesta_navega(N,Nivel,Pruebas,P) :-
        n_esima_prueba(Pruebas,N,Pn),
        Nivel1 is Nivel+1,
        navega(Pn,Nivel1),
        navega(P,Nivel).

opciones_no_numericas(0,[seguir]):-!.
opciones_no_numericas(_,[seguir,arriba]).

escribe_cuerpo(P1 y P2,N1,N3) :-
        !,
        escribe_cuerpo(P1,N1,N2),
        escribe_cuerpo(P2,N2,N3).

escribe_cuerpo(P,N,N1) :-
        tab(3),
        escribe_lista([N,': ']),
        escribe_lo_probado(P),nl,
        N1 is N+1.


escribe_lo_probado(P es cierto esta_afirmado_por _):- write(P).
escribe_lo_probado(P es cierto es_consecuencia_de _) :- write(P).

n_esima_prueba((P1 y P2) y P3,N,P) :- !,n_esima_prueba(P1 y P2 y P3,N,P).
n_esima_prueba(P1 y _,1,P1):-!.
n_esima_prueba(_ y P2,N,P) :-!, N1 is N-1, n_esima_prueba(P2,N1,P).
n_esima_prueba(P,1,P).
```