

Tema 4: Ingeniería del conocimiento y metaintérpretes

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo
Francisco J. Martín Mateos

Dpto. de Ciencias de la Computación e Inteligencia Artificial

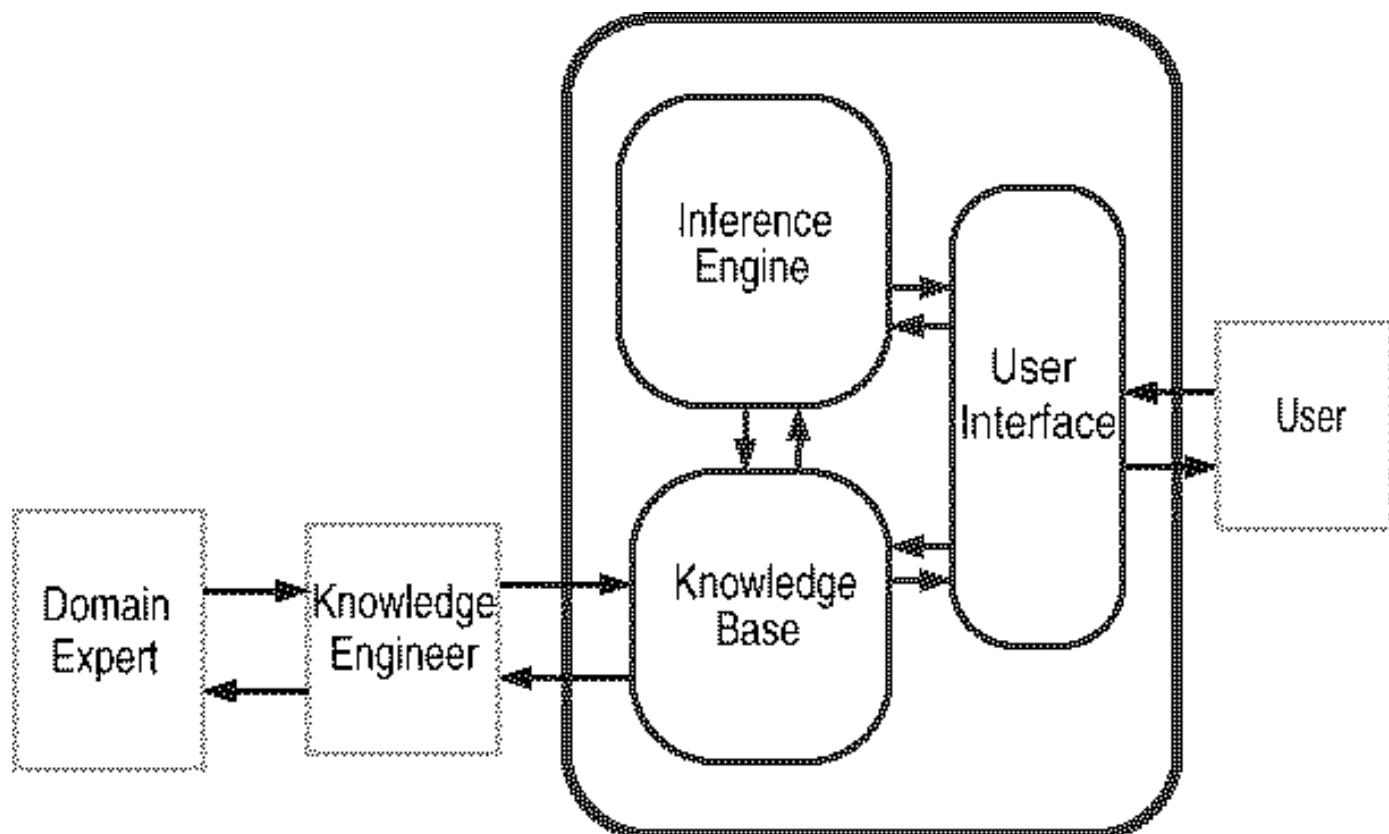
UNIVERSIDAD DE SEVILLA

Introducción

- **Contenido del tema**
 - **Arquitectura de los SBC (sistemas basados en el conocimiento)**
 - **Construcción de SBC mediante metaintérpretes:**
 - * Ejemplo de BC objeto
 - * Metaintérprete simple
 - * Metaintérprete con lenguaje ampliado
 - * Metaintérprete con profundidad acotada
 - **Herramientas para consulta al usuario**
 - **Herramientas para explicación al usuario:**
 - * Metaintérprete con CÓMO
 - * Metaintérprete con PORQUÉ
 - **Técnicas de depuración**
 - **Bibliografía:**
 - Poole–98 Cap. 6: “Knowledge engineering”

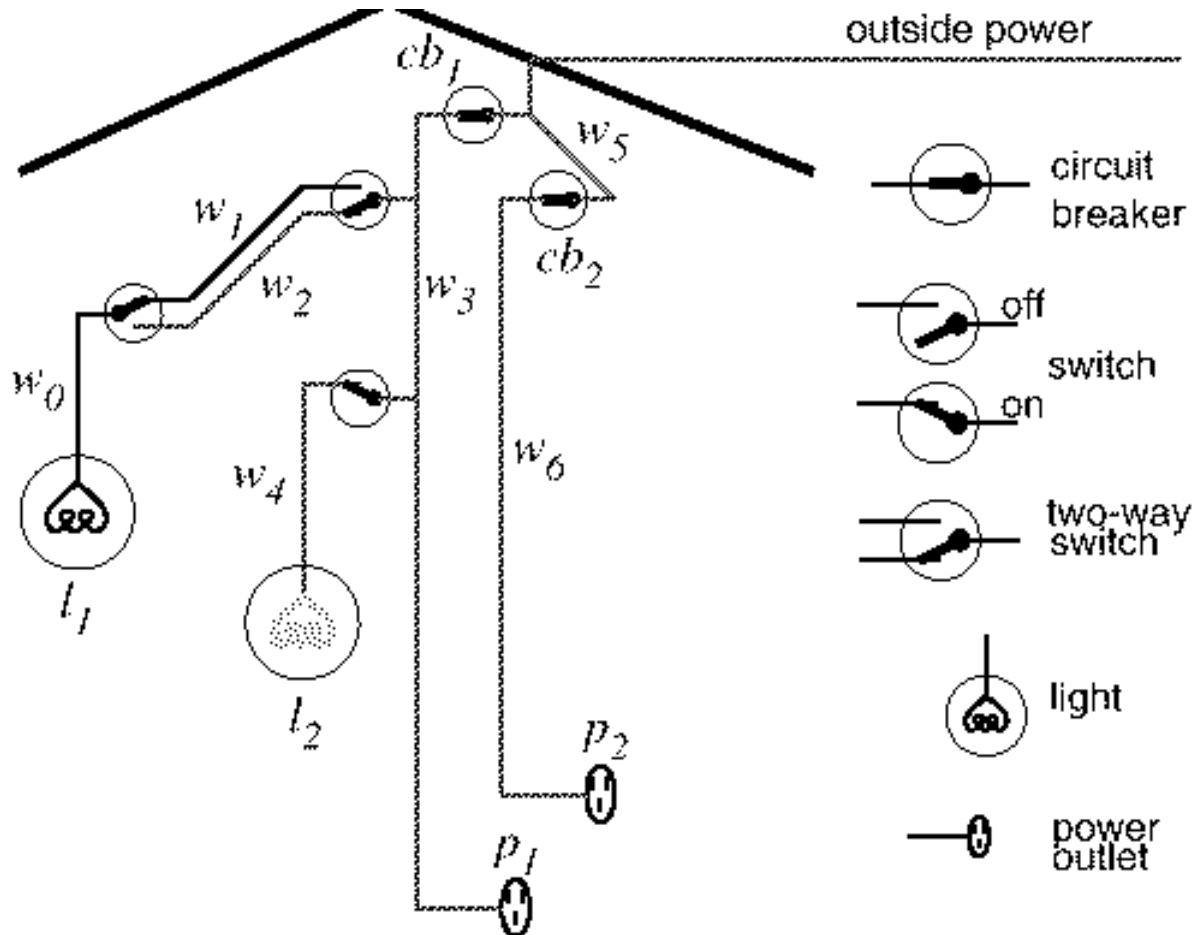
Arquitectura de los SBC

- Arquitectura de los SBC (Poole-98 p. 200)



Ejemplo de BC objeto

- El sistema eléctrico (Poole-98 p. 16)



Ejemplo de BC objeto

- **Ejemplo de BC objeto: i_electrica.pl**

- **Operadores**

```
:- op(1100, xfx, <-).  
:- op(1000, xfy, &).
```

- **luz(?L) se verifica si L es una luz**

```
luz(l1) <- verdad.  
luz(l2) <- verdad.
```

- **abajo(?I) se verifica si el interruptor I está hacia abajo**

```
abajo(i1) <- verdad.
```

- **arriba(?I) se verifica si el interruptor I está hacia arriba**

```
arriba(i2) <- verdad.  
arriba(i3) <- verdad.
```

- **esta_bien(?X) se verifica si la luz (o cortacircuito) X está bien.**

```
esta_bien(l1) <- verdad.  
esta_bien(l2) <- verdad.  
esta_bien(cc1) <- verdad.  
esta_bien(cc2) <- verdad.
```

Ejemplo de BC objeto

- `conectado(?D1, ?D2)` se verifica si los dispositivos D1 y D2 está conectados (de forma que puede fluir la corriente eléctrica de D2 a D1)

```
conectado(l1,c0) <- verdad.  
conectado(c0,c1) <- arriba(i2).  
conectado(c0,c2) <- abajo(i2).  
conectado(c1,c3) <- arriba(i1).  
conectado(c2,c3) <- abajo(i1).  
conectado(l2,c4) <- verdad.  
conectado(c4,c3) <- arriba(i3).  
conectado(e1,c3) <- verdad.  
conectado(c3,c5) <- esta_bien(cc1).  
conectado(e2,c6) <- verdad.  
conectado(c6,c5) <- esta_bien(cc2).  
conectado(c5,entrada) <- verdad.
```

- `tiene_corriente(?D)` se verifica si el dispositivo D tiene corriente

```
tiene_corriente(D) <-  
  conectado(D,D1) &  
  tiene_corriente(D1).  
tiene_corriente(entrada) <- verdad.
```

- `esta_encendida(?L)` se verifica si la luz L está encendida

```
esta_encendida(L) <-  
  luz(L) &  
  esta_bien(L) &  
  tiene_corriente(L).
```

Metaintérprete simple

- Sesión

```
?- prueba(esta_encendida(X)).  
X = 12 ;  
No
```

- Metaintérprete simple

- prueba(+0) se verifica si el objetivo 0 se puede demostrar a partir de la BC objeto

```
prueba(verdad).  
prueba((A & B)) :-  
    prueba(A),  
    prueba(B).  
prueba(A) :-  
    (A <- B),  
    prueba(B).
```

Metaintérprete ampliado

- Ampliación del lenguaje base:

- Disyunciones: $A ; B$
- Predicados predefinidos: $is, <, \dots$

- Operadores

```
:- op(1100, xfx, <-).  
:- op(1000, xfy, [&,;]).
```

- Ejemplo de BC ampliada

```
vecino(X,Y) <- Y is X-1 ; Y is X+1.
```

- Sesión

```
?- prueba(vecino(2,3)).  
Yes  
?- prueba(vecino(3,2)).  
Yes
```


Metaintérprete ampliado

- Metaintérprete ampliado

- prueba(+0) se verifica si el objetivo 0 se puede demostrar a partir de la BC objeto (que puede contener disyunciones y predicados predefinidos)

```
prueba(verdad).
prueba((A & B)) :-
    prueba(A),
    prueba(B).
prueba((A ; B)) :-
    prueba(A).
prueba((A ; B)) :-
    prueba(B).
prueba(A) :-
    predefinido(A),
    A.
prueba(A) :-
    (A <- B),
    prueba(B).
```

- predefinido(+0) se verifica si 0 es un predicado predefinido

```
predefinido((X is Y)).
predefinido((X < Y)).
```

Metaintérprete con profundidad acotada

- Metaintérprete con profundidad acotada

- prueba_pa(+0,+N) es verdad si el objetivo 0 se puede demostrar con profundidad N como máximo

```
prueba_pa(verdad,_N).
prueba_pa((A & B),N) :-
    prueba_pa(A,N),
    prueba_pa(B,N).
prueba_pa(A,N) :-
    N >= 0,
    N1 is N-1,
    (A <- B),
    prueba_pa(B,N1).
```

- Ejemplo

```
numero(0) <- verdad.
numero(s(X)) <- numero(X).
```

```
lista(0,[]) <- verdad.
lista(s(X),[a|L]) <-
    numero(s(X)) &
    lista(X,L).
```

Metaintérprete con profundidad acotada

● Sesión

```
?- prueba_pa(numero(N), 3).
```

```
N = 0 ;
```

```
N = s(0) ;
```

```
N = s(s(0)) ;
```

```
N = s(s(s(0))) ;
```

```
No
```

```
?- prueba_pa(numero(s(s(0))), 1).
```

```
No
```

```
?- prueba_pa(numero(s(s(0))), 2).
```

```
Yes
```

```
?- prueba_pa(lista(N, L), 3).
```

```
N = 0          L = [] ;
```

```
N = s(0)      L = [a] ;
```

```
N = s(s(0))  L = [a, a] ;
```

```
No
```

Metaintérprete con profundidad acotada

- Segundo ejemplo

- Programa

```
hermano(X,Y) <- hermano(Y,X).  
hermano(b,a) <- verdad.
```

- Sesión

```
?- prueba_pa(hermano(a,X),1).  
X = b ;  
No
```

```
?- prueba_pa(hermano(X,Y),1).  
X = a Y = b ;  
X = b Y = a ;  
No
```

```
?- prueba_pa(hermano(a,X),2).  
X = b ;  
No
```

```
?- prueba_pa(hermano(a,X),3).  
X = b ;  
X = b ;  
No
```

Consulta al usuario

- **Planteamiento del problema: Aportación del conocimiento de los usuarios cuando:**
 - No conocen las interioridades del sistema
 - No son expertos en el dominio
 - No concocen qué información es importante
 - No conocen la sintaxis del sistema
 - Poseen información esencial del caso particular del problema
- **Funciones del sistema:**
 - Determinar qué información es importante
 - Preguntar al usuario sobre dicha información
- **Tipos de objetivos:**
 - No preguntable
 - Preguntable no-preguntado
 - Preguntado

Consulta al usuario

- Preguntas elementales:
 - Son objetivos básicos (sin variables)
 - Las respuestas son “sí” o “no”
 - Se plantean si son importantes, preguntables y no preguntadas
 - El sistema almacena la respuesta
- Ejemplo: Modificación de `i_electrica.pl`

```
preguntable(arriba(_)).  
preguntable(abajo(_)).
```

- Sesión

```
?- prueba_p(esta_encendida(L)).  
¿Es verdad arriba(i2)? (si/no)  
|: si.  
¿Es verdad arriba(i1)? (si/no)  
|: no.  
¿Es verdad abajo(i2)? (si/no)  
|: no.  
¿Es verdad arriba(i3)? (si/no)  
|: si.
```

```
L = 12 ;  
No
```

Consulta al usuario

```
?- listing(respuesta).  
respuesta(arriba(i2), si).  
respuesta(arriba(i1), no).  
respuesta(abajo(i2), no).  
respuesta(arriba(i3), si).  
Yes
```

```
?- retractall(respuesta(_,_)).  
Yes
```

```
?- prueba_p(esta_encendida(L)).  
¿Es verdad arriba(i2)? (si/no)  
|: si.  
¿Es verdad arriba(i1)? (si/no)  
|: si.  
L = 11 ;
```

```
¿Es verdad abajo(i2)? (si/no)  
|: no.  
¿Es verdad arriba(i3)? (si/no)  
|: si.  
L = 12 ;  
No
```

Consulta al usuario

- **Metaintérprete con preguntas**

- prueba_p(+0) se verifica si el objetivo 0 se puede demostrar a partir de la BC objeto y las respuestas del usuario

```
prueba_p(verdad).
prueba_p((A & B)) :-
    prueba_p(A),
    prueba_p(B).
prueba_p(G) :-
    preguntable(G),
    respuesta(G,si).
prueba_p(G) :-
    preguntable(G),
    no_preguntado(G),
    pregunta(G,Respuesta),
    assert(respuesta(G,Respuesta)),
    Respuesta=si.
prueba_p(A) :-
    (A <- B),
    prueba_p(B).
```

- respuesta(?0,?R) se verifica si la respuesta al objetivo 0 es R. [Se añade dinámicamente a la base de datos]

```
:- dynamic respuesta/2.
```


Consulta al usuario

- `no_preguntado(+0)` es verdad si el objetivo 0 no se ha preguntado

```
no_preguntado(0) :-  
    not(respuesta(0,_)).
```

- `pregunta(+0, -Respuesta)` pregunta 0 al usuario y éste responde la Respuesta

```
pregunta(0,Respuesta) :-  
    escribe_lista(['¿Es verdad ',0,'? (si/no)']),  
    read(Respuesta).
```

- `escribe_lista(+L)` escribe cada uno de los elementos de la lista L

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

Consulta al usuario

- Consulta sobre relaciones funcionales
 - Ejemplo: $\text{edad}(P, N)$
- Forma de la consulta:
 - Menús
 - Texto de formato libre
- Preguntas generales

Pregunta	¿Preguntable?	Respuesta
$p(X)$	Sí	$p(f(Z))$
$p(f(c))$	No	
$p(a)$	Sí	Sí
$p(X)$	Sí	No
$P(X)$	No	

Explicación

- Necesidad del sistema de justificar sus respuestas
- Uso en explicación y depuración
- Tipos de explicaciones:
 - Preguntar CÓMO se ha probado un objetivo
 - Preguntar PORQUÉ plantea una consulta
 - Preguntar PORQUÉ_NO se ha probado un objetivo
- Preguntas CÓMO
 - El usuario puede preguntar CÓMO ha probado el objetivo q
 - El sistema muestra la instancia de la regla usada
 $q :- p_1, \dots, p_n.$
 - El usuario puede preguntar CÓMO i para obtener la regla usada para probar p_i
- El comando CÓMO permite descender en el árbol de prueba

Metaintérprete con árbol de prueba

```
?- [meta_con_explicacion_arbol, i_electrica].
```

```
Yes
```

```
?- prueba_con_demostracion(esta_encendida(L),T).
```

```
L = l2
```

```
T = si(esta_encendida(l2),  
      (si(luz(l2), verdad) &  
        si(esta_bien(l2), verdad) &  
          si(tiene_corriente(l2),  
            (si(conectado(l2, c4), verdad) &  
              si(tiene_corriente(c4),  
                (si(conectado(c4, c3),  
                  si(arriba(i3), verdad)) &  
                    si(tiene_corriente(c3),  
                      (si(conectado(c3, c5),  
                        si(esta_bien(cc1), verdad)) &  
                          si(tiene_corriente(c5),  
                            (si(conectado(c5, entrada), ver  
                              si(tiene_corriente(entrada), ver
```

```
No
```

Metaintérprete con árbol de prueba

- Metaintérprete con árbol de prueba
- prueba_con_demostracion(+0, ?A) es verdad si A es un árbol de prueba del objetivo 0

```
prueba_con_demostracion(verdad, verdad).
prueba_con_demostracion((A & B), (AA & AB)) :-
    prueba_con_demostracion(A, AA),
    prueba_con_demostracion(B, AB).
prueba_con_demostracion(0, si(0, AB)) :-
    (0 <- B),
    prueba_con_demostracion(B, AB).
```

Metaintérprete con CÓMO

- Sesión

```
?- [meta_con_explicacion_como, i_electrica].  
Yes
```

```
?- prueba_con_como(esta_encendida(L)).
```

```
esta_encendida(l2) :-  
    1: luz(l2)  
    2: esta_bien(l2)  
    3: tiene_corriente(l2)
```

```
|: 3.
```

```
tiene_corriente(l2) :-  
    1: conectado(l2, c4)  
    2: tiene_corriente(c4)
```

```
|: 2.
```

```
tiene_corriente(c4) :-  
    1: conectado(c4, c3)  
    2: tiene_corriente(c3)
```

```
|: 1.
```

```
conectado(c4, c3) :-  
    1: arriba(i3)
```

```
|: 1.
```

```
arriba(i3) es un hecho
```

```
L = l2 ;
```

```
No
```

Metaintérprete con CÓMO

- Metaintérprete con CÓMO

- prueba_con_como(+O) significa probar el objetivo O a partir de la BC objeto y navegar por su árbol de prueba mediante preguntas CÓMO

```
prueba_con_como(O) :-  
    prueba_con_demostracion(O,A),  
    navega(A).
```

- navega(+A) significa que se está navegando en el árbol A

```
navega(si(A,verdad)) :-  
    escribe_lista([A,' es un hecho']).  
navega(si(A,B)) :-  
    B \== verdad,  
    escribe_lista([A,' :-']),  
    escribe_cuerpo(B,1,_),  
    read(Orden),  
    interpreta_orden(Orden,B).
```

- escribe_lista(+L) escribe cada uno de los elementos de la lista L

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

Metaintérprete con CÓMO

- `escribe_cuerpo(+B,+N1,?N2)` es verdad si B es un cuerpo que se va a escribir, N1 es el número de átomos antes de la llamada a B y N2 es el número de átomos después de la llamada a B

```
escribe_cuerpo(verdad,N,N).
escribe_cuerpo((A & B),N1,N3) :-
    escribe_cuerpo(A,N1,N2),
    escribe_cuerpo(B,N2,N3).
escribe_cuerpo(si(H,_),N,N1) :-
    escribe_lista([' ',N,': ',H]),
    N1 is N+1.
```

- `interpreta_orden(+Orden,+B)` interpreta la Orden sobre el cuerpo B

```
interpreta_orden(N,B) :-
    integer(N),
    nth(B,N,E),
    navega(E).
```

- `nth(+E,+N,?A)` es verdad si A es el N-ésimo elemento de la estructura E

```
nth(A,1,A) :-
    not((A = (_,_))).
nth((A&_),1,A).
nth((_&B),N,E) :-
    N>1,
    N1 is N-1,
    nth(B,N1,E).
```


Metaintérprete con PORQUÉ

- **Ejemplo: Modificación de i_electrica.pl**

```
preguntable(arriba(_)).  
preguntable(abajo(_)).
```

- **Sesión**

```
?- [meta_con_explicacion_porque, i_electrica_con_preguntas].  
Yes
```

```
?- prueba_con_porque(esta_encendida(L)).  
¿Es verdad arriba(i2)? (si/no/porque)  
|: porque.
```

Se usa en:

```
conectado(c0, c1) <-  
    arriba(i2).
```

```
¿Es verdad arriba(i2)? (si/no/porque)  
|: porque.
```

Se usa en:

```
tiene_corriente(c0) <-  
    conectado(c0, c1) &  
    tiene_corriente(c1).
```

```
¿Es verdad arriba(i2)? (si/no/porque)  
|: porque.
```

Se usa en:

```
tiene_corriente(l1) <-  
    conectado(l1, c0) &  
    tiene_corriente(c0).
```

Metaintérprete con PORQUÉ

¿Es verdad arriba(i2)? (si/no/porque)

|: porque.

Se usa en:

```
esta_encendida(l1) <-  
  luz(l1) &  
  esta_bien(l1) &  
  tiene_corriente(l1).
```

¿Es verdad arriba(i2)? (si/no/porque)

|: porque.

Porque esa fue su pregunta!

¿Es verdad arriba(i2)? (si/no/porque)

|: si.

¿Es verdad arriba(i1)? (si/no/porque)

|: porque.

Se usa en:

```
conectado(c1, c3) <-  
  arriba(i1).
```

¿Es verdad arriba(i1)? (si/no/porque)

|: no.

¿Es verdad abajo(i2)? (si/no/porque)

|: no.

¿Es verdad arriba(i3)? (si/no/porque)

|: porque.

Se usa en:

```
conectado(c4, c3) <-  
  arriba(i3).
```

¿Es verdad arriba(i3)? (si/no/porque)

|: si.

L = 12 ;

No

Metaintérprete con PORQUÉ

- Ejemplo 2

- Base de conocimiento

```
a <- a1 & a2 & a3.  
a1 <- a11 & a12.  
a11 <- verdad.  
a12 <- verdad.  
a3 <- a31 & a32.  
a31 <- verdad.  
a32 <- verdad.
```

```
preguntable(a2).
```

- Sesión

```
% prueba_cpa = prueba_con_porque_aux  
  
?- trace(prueba_cpa, [call,exit]).  
Yes
```

Metaintérprete con PORQUÉ

```
?- prueba_con_porque(a).
Call: 8) prueba_cpa(a, [])
Call: 9) prueba_cpa((a1&a2&a3), [(a<-a1&a2&a3)])
Call:10) prueba_cpa(a1, [(a<-a1&a2&a3)])
Call:11) prueba_cpa((a1&a12), [(a1<-a1&a12), (a<-a1&a2&a3)])
Call:12) prueba_cpa(a11, [(a1<-a1&a12), (a<-a1&a2&a3)])
Call:13) prueba_cpa(verdad, [(a11<-verdad), (a1<-a1&a12), (a<-a1&a2&a3)])
Exit:13) prueba_cpa(verdad, [(a11<-verdad), (a1<-a1&a12), (a<-a1&a2&a3)])
Exit:12) prueba_cpa(a11, [(a1<-a1&a12), (a<-a1&a2&a3)])
Call:12) prueba_cpa(a12, [(a1<-a1&a12), (a<-a1&a2&a3)])
Call:13) prueba_cpa(verdad, [(a12<-verdad), (a1<-a1&a12), (a<-a1&a2&a3)])
Exit:13) prueba_cpa(verdad, [(a12<-verdad), (a1<-a1&a12), (a<-a1&a2&a3)])
Exit:12) prueba_cpa(a12, [(a1<-a1&a12), (a<-a1&a2&a3)])
Exit:11) prueba_cpa((a1&a12), [(a1<-a1&a12), (a<-a1&a2&a3)])
Exit:10) prueba_cpa(a1, [(a<-a1&a2&a3)])
Call:10) prueba_cpa((a2&a3), [(a<-a1&a2&a3)])
Call:11) prueba_cpa(a2, [(a<-a1&a2&a3)])
```

Metaintérprete con PORQUÉ

¿Es verdad a2? (si/no/porque)

|: porque.

Se usa en:

a <-

a1 &

a2 &

a3.

¿Es verdad a2? (si/no/porque)

|: porque.

Porque esa fue su pregunta!

¿Es verdad a2? (si/no/porque)

|: si.

Metaintérprete con PORQUÉ

Exit:11) prueba_cpa(a2, [(a<-a1&a2&a3)])
Call:11) prueba_cpa(a3, [(a<-a1&a2&a3)])
Call:12) prueba_cpa((a31&a32), [(a3<-a31&a32), (a<-a1&a2&a3)])
Call:13) prueba_cpa(a31, [(a3<-a31&a32), (a<-a1&a2&a3)])
Call:14) prueba_cpa(verdad, [(a31<-verdad), (a3<-a31&a32), (a<-a1&a2&a3)])
Exit:14) prueba_cpa(verdad, [(a31<-verdad), (a3<-a31&a32), (a<-a1&a2&a3)])
Exit:13) prueba_cpa(a31, [(a3<-a31&a32), (a<-a1&a2&a3)])
Call:13) prueba_cpa(a32, [(a3<-a31&a32), (a<-a1&a2&a3)])
Call:14) prueba_cpa(verdad, [(a32<-verdad), (a3<-a31&a32), (a<-a1&a2&a3)])
Exit:14) prueba_cpa(verdad, [(a32<-verdad), (a3<-a31&a32), (a<-a1&a2&a3)])
Exit:13) prueba_cpa(a32, [(a3<-a31&a32), (a<-a1&a2&a3)])
Exit:12) prueba_cpa((a31&a32), [(a3<-a31&a32), (a<-a1&a2&a3)])
Exit:11) prueba_cpa(a3, [(a<-a1&a2&a3)])
Exit:10) prueba_cpa((a2&a3), [(a<-a1&a2&a3)])
Exit: 9) prueba_cpa((a1&a2&a3), [(a<-a1&a2&a3)])
Exit: 8) prueba_cpa(a, [])

Yes

Metaintérprete con PORQUÉ

● Metaintérprete con PORQUÉ

- `prueba_con_porque(+0)` significa probar el objetivo 0, con las respuestas del usuario, permitiéndole preguntar PORQUÉ se le plantean preguntas

```
prueba_con_porque(0) :-  
    prueba_con_porque_aux(0, []).
```

- `prueba_con_porque_aux(+0,+Antecedentes)` se verifica si 0 es probable con la lista de Antecedentes
[Pregunta al usuario y le permite preguntar PORQUÉ]

```
prueba_con_porque_aux(verdad,_).  
prueba_con_porque_aux((A & B), Antecedentes) :-  
    prueba_con_porque_aux(A,Antecedentes),  
    prueba_con_porque_aux(B,Antecedentes).  
prueba_con_porque_aux(0,_) :-  
    preguntable(0),  
    respuesta(0,si).  
prueba_con_porque_aux(0,Antecedentes) :-  
    preguntable(0),  
    no_preguntado(0),  
    pregunta(0,Respuesta,Antecedentes),  
    assert(respuesta(0,Respuesta)),  
    Respuesta=si.  
prueba_con_porque_aux(A,Antecedentes) :-  
    (A <- B),  
    prueba_con_porque_aux(B,[(A <- B)|Antecedentes]).
```

Metaintérprete con PORQUÉ

- `pregunta(+O,-Respuesta,+Antecedentes)` pregunta al usuario, en el contexto dado por los Antecedentes, la cuestión `O` y éste responde la Respuesta

```
pregunta(O,Respuesta,Antecedentes) :-  
    escribe_lista(['¿Es verdad ',O,'? (si/no/porque)']),  
    read(Replica),  
    interpreta(O,Respuesta,Replica,Antecedentes).
```

- `interpreta(+O,?Respuesta,+Replica,+Antecedentes)`

```
interpreta(_,Replica,Replica,_) :-  
    Replica \== porque.  
interpreta(O,Respuesta,porque,[Regla|Reglas]) :-  
    write('Se usa en:'), nl,  
    escribe_regla(Regla),  
    pregunta(O,Respuesta,Reglas).  
interpreta(O,Respuesta,porque,[]) :-  
    write('Porque esa fue su pregunta!'), nl,  
    pregunta(O,Respuesta,[]).
```

- `escribe_regla(Regla)`

```
escribe_regla((A <- B)) :-  
    escribe_lista([' ',A, ' <- ']),  
    escribe_cuerpo(B).
```


Metaintérprete con PORQUÉ

- `escribe_cuerpo(C)`

```
escribe_cuerpo((A & B)) :-  
    escribe_lista(['      ',A, ' &']),  
    escribe_cuerpo(B).  
escribe_cuerpo(A) :-  
    not(A = (_B & _C)),  
    escribe_lista(['      ',A, '.']).
```

- **Utilidad de saber porqué el sistema plantea una pregunta:**
 - Aumenta la confianza del usuario
 - Ayuda al ingeniero del conocimiento en la optimización de las preguntas realizadas por el sistema
 - Una pregunta irrelevante puede manifestar la existencia de problemas en el sistema
 - El usuario puede usar el sistema para aprender

Depuración de bases de conocimiento

- Tipos de errores no sintácticos
 - Respuestas incorrectas
 - Respuestas perdidas
 - Bucle infinito
 - Preguntas irrelevantes
- Depuración de respuestas incorrectas
 - Una respuesta incorrecta es una respuesta computada que es falsa en la interpretación deseada
 - Una respuesta incorrecta implica la existencia de una cláusula de la BC que es falsa en la interpretación deseada (o que el sistema de razonamiento es inadecuado)
 - Si q es falsa en la interpretación deseada, existe una demostración de q usando $q :- p_1, \dots, p_n$. Entonces
 1. alguna p_i es falsa (y se depura), o
 2. todas las p_i son verdaderas (y la cláusula es falsa)

Depuración de respuestas incorrectas

● Ejemplo

- BC errónea: en `i_electrica.pl` cambiar la cláusula `conectado(c1,c3) <- arriba(i1).` por la cláusula errónea `conectado(c1,c3) <- arriba(i3).`

- Depuración de respuesta errónea con el metaintérprete con CÓMO

```
?- [i_electrica_con_error_1, meta_con_explicacion_como].  
Yes
```

```
?- prueba_con_como(esta_encendida(L)).
```

```
esta_encendida(l1) :-
```

```
1: luz(l1)
```

```
2: esta_bien(l1)
```

```
3: tiene_corriente(l1)
```

```
|: 3.
```

```
tiene_corriente(l1) :-
```

```
1: conectado(l1, c0)
```

```
2: tiene_corriente(c0)
```

```
|: 2.
```

```
tiene_corriente(c0) :-
```

```
1: conectado(c0, c1)
```

```
2: tiene_corriente(c1)
```

```
|: 2.
```

```
tiene_corriente(c1) :-
```

```
1: conectado(c1, c3)
```

```
2: tiene_corriente(c3)
```

```
|: 1.
```

```
conectado(c1, c3) :-
```

```
1: arriba(i3)
```

Bibliografía

- Flach, P. *Simply Logical (Intelligent Reasoning by Example)* (John Wiley, 1994)
 - Cap. 3: “Logic programming and Prolog”
- Lucas, P. y Gaag, L.v.d. *Principles of Expert Systems* (Addison–Wesley, 1991).
 - Cap. 1: “Introduction”
 - Cap. 3: “Production rules and inference”
 - Cap. 4: “Tools for knowledge and inference inspection”
- Poole, D.; Mackworth, A. y Goebel, R. *Computational Intelligence (A Logical Approach)* (Oxford University Press, 1998)
 - Cap. 6: “Knowledge engineering”
- Shoham, Y. *Artificial Intelligence Techniques in Prolog* (Morgan Kaufmann, 1994)
 - Cap. 3 “Backward–chaining methods”
 - Cap. 4 “Other rule–based methods”