Extensiones de Meta intérpretes (PROLOG)

Ingeniería de Conocimiento

3º Grado de Ingeniería Informática

vanilla con predicados predefinidos

```
builtin(A is B).   builtin(A > B).       builtin(A < B).
builtin(A = B).    builtin(A =:= B).     builtin(A =< B).
builtin(A >= B).   builtin(functor(T, F, N)).
builtin(read(X)).  builtin(write(X)).

solve(true):- !.
solve((A,B)) :-!, solve(A), solve(B).
solve(A):- builtin(A), !, A.
solve(A) :- clause(A, B), solve(B).
```

- Ejecutar paso a paso:

```
? solve(write('¡¡¡Esto
    funciona!!!')).
```

Extensión vanilla pruebas

```
builtin(A is B).   builtin(A > B).       builtin(A < B).
builtin(A = B).    builtin(A =:= B).     builtin(A =< B).
builtin(A >= B).   builtin(functor(T, F, N)).
builtin(read(X)).  builtin(write(X)).

solve(true,true) :- !.
solve((A, B), (ProofA, ProofB)) :-!, solve(A, ProofA),
                                     solve(B, ProofB).
solve(A, (A:-builtin)):- builtin(A), !, A.
solve(A, (A:-Proof)) :- clause(A, B), solve(B, Proof).
```

- Ejecutar paso a paso:

```
1 ?- solve(valor(w1,X),Prueba).

2 ?- solve(valor(w2,X),Prueba).

3 ?- solve(valor(w3,X),Prueba).
```

Modificación del Lenguaje Base

- **Lenguaje Base** son las expresiones manejadas por el meta intérprete.
- **Metalenguaje**: lenguaje del intérprete
- En los ejemplos precedentes:
  - **Cláusulas** definidas.
  - **Predicados** predefinidos e interpretados “como” en Prolog
- Modificar el lenguaje base:
  - **Separar** cláusulas definidas de los predicados predefinidos
  - Se usará la llamada “**Sintactic sugaring**”, esto es, una sintáctica más cercana al hombre, pero a la vez más “verbosa”.
  - Se aplicará a lo “no aportado” por Prolog hasta el momento.

Modificación ejemplo “propagación de señal”

```
true ---> valor(w1, 1).
true ---> conectado(w2, w1).
true ---> conectado(w3, w2).
conectado(W,V) & valor(V,X) ---> valor(W,X).
```

- Se necesita definir los nuevos operadores:

```
:op(40, xfy, &).

:op(50, xfy, --->).
```

- Ejercicio: buscar en la ayuda de Prolog su significado.

Ejercicio: modificación del meta intérprete

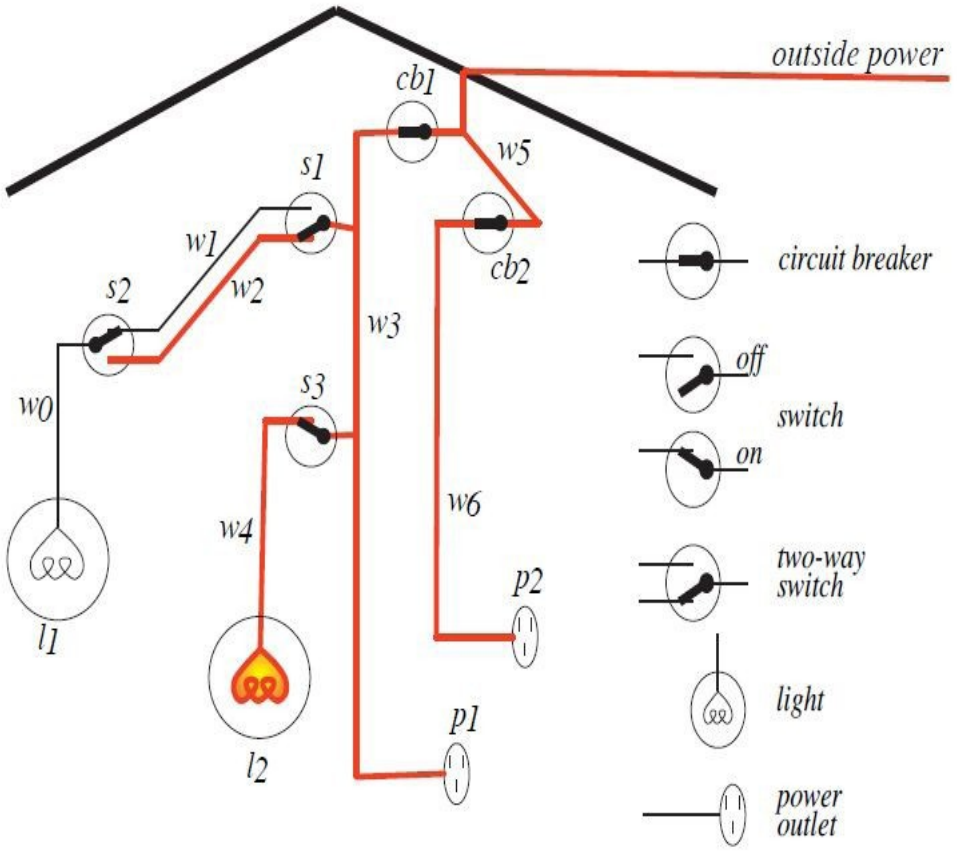
```
:-op(40, xfy, &).
:-op(50, xfy, --->).

solve(true):-!.
solve((A & B)) :-!, solve(A), solve(B).
solve(A) :- !, (B ---> A), solve(B).
```

- Ejercicio:
- Añadir la base de conocimiento de “conectado.pl” pero con el lenguaje modificado.
- Ejecutar con el nuevo metaintéprete el equivalente a

  ? solve(conectado(W,X)).

Asistente al diagnóstico (dominio)



David Poole, Alan Mackworth. Artificial Intelligence: Foundations of Computational Agents, Cambridge University Press, 2010

Modelo del dominio en el lenguaje base

- Si una bombilla (light) funciona correctamente (ok) y le llega tensión (live), entonces se enciende (lit).

```
light(L)& ok(L) & live(L)---> lit(L).
```

- Si un cable está conectado a otro (connected_to), al que le llega tensión (live), entonces tiene tensión (live):

```
connected_to(W,W1) & live(W1)---> live(W).
```

Modelo del dominio

- El cable externo tiene tensión:

```
true ---> live(outside).
```

- L1 es una bombilla:

```
true ---> light(l1).
```

- El interruptor s1 está arriba:

```
true ---> down(s1).
```

- El interruptor s2 está abajo:

```
true ---> up(s2).
```

Modelo del dominio

- Si el interruptor s2 está abierto y funciona correctamente, entonces el cable w0 está conectado al cable w1:

```
up(s2) & ok(s2) ---> connected_to(w0,w1).
```

- Si el diferencial cb2 funciona correctamente, entonces el cable w6 está conectado al cable w5:

```
ok(cb2) ---> connected_to(w6,w5).
```

- El enchufe p2 está conectado al cable w6:

```
true ---> connected_to(p2,w6).
```

Ejercicio:

- Completar la base de conocimiento del modelo del ejemplo de asistente al diagnóstico de la instalación eléctrica propuesto por Poole y Mackworth.
- Hay que tomar como base la situación reflejada en el esquema, es decir:

```
true ---> live(outside).
true ---> down(s1).
true ---> up(s2).
true ---> up(s3).
true ---> ok(_).
```

- Comprobar su funcionamiento con el encendido de las bombillas o el estado de los enchufes, por ejemplo.

Meta-intérprete con traza

```
solve_traza(true):-!.
solve_traza((A, B)) :-!, solve_traza(A), solve_traza(B).
solve_traza(A):- !, write('Call: '), write(A), nl,
                clause(A,B), solve_traza(B),
                write('Exit: '), write(A), nl.
```

- Ejercicio:
  - aplicarlo a ejemplo inicial para obtener la traza de:

```
?- solve_traza(valor(X,Y)).
```

Ejercicio:

- Modificar el meta-intérprete anterior para que se obtenga esto:

```
?- solve_traza_nivel(valor(w3,X)).
     1 conectado(w3,w2)
          2 conectado(w2,w1)
          2 valor(w1,1)
     1 valor(w2,1)
0 valor(w3,1)
X = 1 ;
false.
```

- Hay un predicado predefinido para el manejo de tabuladores (tab). Consultar el manual.