Meta intérpretes (PROLOG)

Ingeniería de Conocimiento

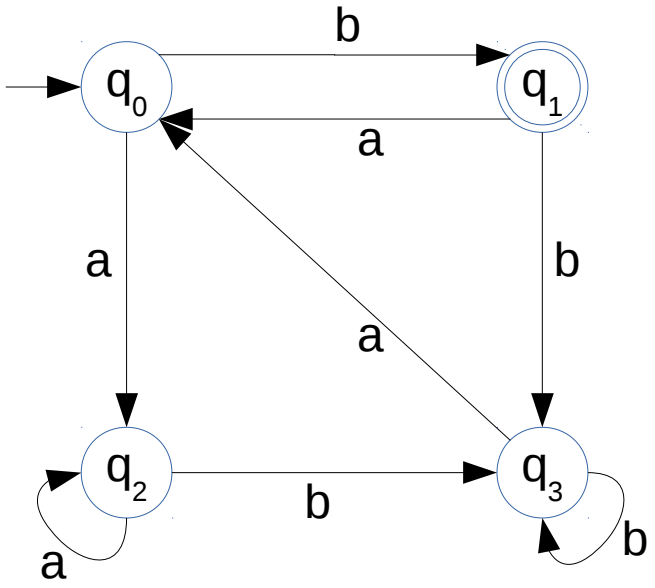
3º Grado de Ingeniería Informática

Intérpretes

- Un metaprograma:
  - Programa que utiliza otro programa como entrada.
  - El hecho de PROLOG ser un intérprete le confiere una ventaja para esta funcionalidad
  - Ejemplos hay muchos. A nivel teórico destaca:
    - Máquina **Universal** de Turing
    - Como entrada recibe la codificación de otra máquina de **Turing**.
    - Esto fue el germen del modelo **Von Neumann**,
    - Lo que llevó a la máquina de **propósito general**

Intérprete de máquinas de estados

- Ejemplo:
  - Máquina de Moore
  - Construir la tabla de transición.
  - Base de conocimiento de un programa Prolog
  - Calcular la función de transición generalizada en Prolog:



$$F(q_i, abbab) = q_f$$

Autómata de Pila

- Reconocer cadenas:

$$L=\{a^n b^n\}$$

$$\delta(q_0, a, z)=(q_0, az)$$
$$\delta(q_0, a, a)=(q_0, aa)$$
$$\delta(q_0, b, a)=(q_1, \lambda)$$
$$\delta(q_1, b, a)=(q_1, \lambda)$$
$$\delta(q_1, \lambda, z)=(q_f, z)$$

$$f(q_0, aabb, z)=(q_f, \lambda, z)$$

– **Ejercicio:**

- Construir la base de conocimiento en Prolog
- Calcular la función de transición f.
- Plantear un predicado “acepta” de una cadena del lenguaje L

```
mueve(q0, a, [z], q0, [a|z]).
mueve(q0, a, [a|H], q0, [a|[a|H]]).
mueve(q0, b, [a|H], q1, H).
mueve(q1, b, [a|H], q1, H).
mueve(q1,[],[z],qf,[z]).


transita(q1,[],z,qf,[z]):-!.
transita(Qi, [X|Y], R, Qf, T):- X \=[],
           mueve(Qi, X, R, P, S), transita(P, Y, S, Qf, T).

acepta(X,Resultado) :- transita(q0,X,[z],Q,_), Q=qf,
                       Resultado is 1, !.
```

Máquinas de Turing

- Sería factible su implementación en Prolog:
  - La transición obedecería a:
    - Estado actual
    - Carácter al que apunta el cabezal
  - Provocaría:
    - Estado siguiente
    - Escritura en la posición del cabezal
    - Movimiento de éste: L (izda.) o R (dcha.)
  - Implicaría construir predicados para manejar una **lista** como una **cinta**. Hay alternativas.

Meta-intérprete

- **Intérprete** de un lenguaje escrito en el propio lenguaje.
- Esta idea podría llevar a plantear la creación de **lenguajes de programación**, incluso, su propio entorno integrado.
- En Prolog esto es factible, puesto que se pueden formular los problemas bajo un enfoque de **programación lógica**.

Meta intérprete más sencillo

```
solve(A):-A.
```

- No tiene interés, puesto que no aporta nada.
- Se trata con los meta intérpretes, poder modificar el **cómputo** o la regla de **búsqueda**

Meta intérprete *vanilla*

- Disponible en:

  http://artint.info/index.html

- En síntesis es:

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

vanilla – Lectura Declatariva

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

- La meta vacía es cierta.
- La meta conjuntiva (A, B) es cierta, si A es cierta y B es cierta.
- La meta A es cierta, si existe una clausula A:-B y B es cierta

vanilla – Lectura Operacional

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

- La meta vacía está resuelta.
- Para resolver la meta (A,B), primero resolver la meta A y después la B. (Regla de cómputo).
- Para resolver la meta A, buscar una claúsula cuya cabeza unifique A y resolver el cuerpo usando la regla de búsqueda de Prolog.

vanilla – Versión mejorada

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

- Estaría limitándose a PROLOG “puro” = programación lógica:
  - Sin modificación de la reevalución: corte, fail, repeat.
  - Sin negación por fallos.
  - Sin asociación de procedimiento: predicados predefinidos

Ejemplo: Propagación de señal

```
valor(w1, 1).
conectado(w2, w1).
conectado(w3, w2).
valor(W,X):-conectado(W,V), valor(V,X)
```

- Ejecutar paso a paso la consulta:
  - `valor(W,X).`
- Añadir el metaintérprete vanilla y ejecutar paso a paso:
  - `solve(valor(W,X)).`

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.

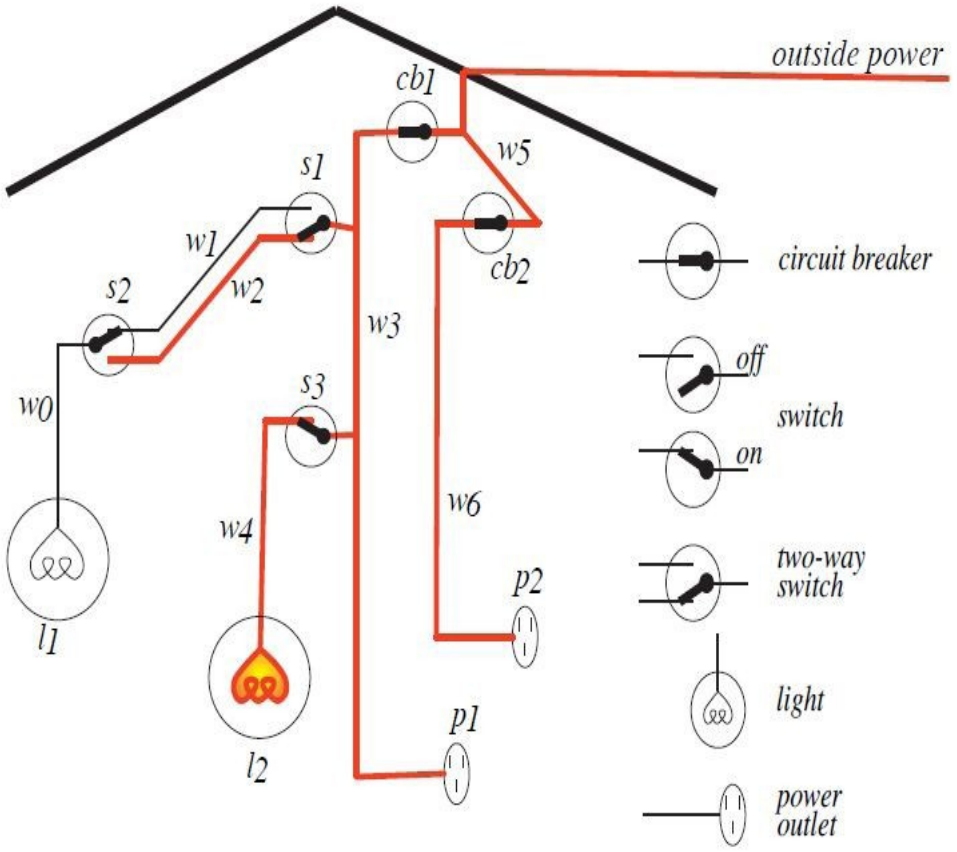
Ejercicio: modificación del meta intérprete

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

solve(true):-!.
solve((A & B)) :-!, solve(A), solve(B).
¿LA ULTIMA CLÁUSULA?
```

- Completar.

Asistente al diagnóstico (dominio)



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

Modelar el 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).
```

Modelar el dominio

- El cable externo tiene tensión:

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

- L1 es una bombilla:

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

- El interruptor s1 está abierto:

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

- El interruptor s2 está abierto:

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

Modelar el 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 que modela el ejemplo de asistente al diagnóstico 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(_).
```

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 (entrega):

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

```
?- solve_traza_nivel(valor(w3,X)).
0 valor(w3,_G374)
   1 conectado(w3,w2)
   1 valor(w2,_G374)
      2 conectado(w2,w1)
      2 valor(w1,1)
X = 1 ;
      2 valor(w1,_G374)
false.
```

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