Programación declarativa (2004–05)

Tema 9: Metaprogramación

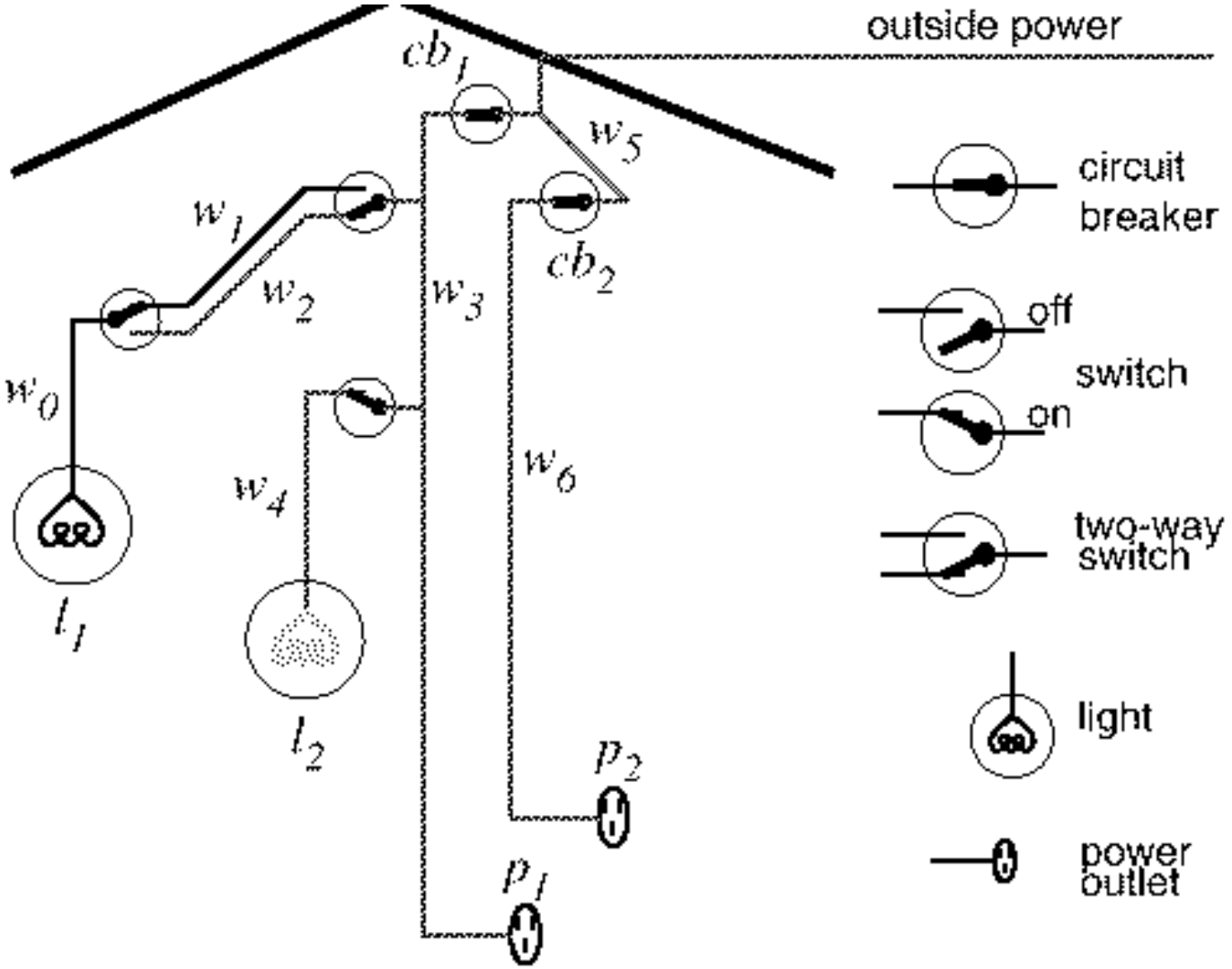
José A. Alonso Jiménez

Dpto. Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Ejemplo de BC objeto

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



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.
```

- `está_bien(?X)` se verifica si la luz (o cortacircuito) `X` está bien.

```
está_bien(l1) <- verdad.
está_bien(l2) <- verdad.
está_bien(cc1) <- verdad.
está_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) <- está_bien(cc1).
conectado(e2,c6) <- verdad.
conectado(c6,c5) <- está_bien(cc2).
conectado(c5,entrada) <- verdad.
```

Ejemplo de BC objeto

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

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

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

```
está_encendida(L) <-
    luz(L) &
    está_bien(L) &
    tiene_corriente(L).
```

- Sesión con la BC y el metaintérprete simple:

```
?- prueba(está_encendida(X)).
X = l2 ;
No
```

Metaintérprete simple

- Metaintérprete simple:
  - `prueba(+O)` se verifica si el objetivo `O` se puede demostrar a parti
    de la BC objeto.

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

Metaintéprete ampliado

- Ampliación del lenguaje base:
  - Disyunciones: `A ; B`
  - Predicados predefinidos: `is`, `<`, ...
- 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
```

Metaintéprete ampliado

- `prueba(+O)` se verifica si el objetivo `O` 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(+O)` se verifica si `O` es un predicado predefinido.

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

Metaintérprete con profundidad acotada

- Ejemplo:
  - Programa objeto:

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

  - Sesión:

```
?- prueba(hermano(X,Y)).
ERROR: Out of local stack

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

Metaintérprete con profundidad acotada

- `prueba_pa(+O,+N)` es verdad si el objetivo `O` se puede demostrar co 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).
```

Metaintérprete con preguntas

- Ejemplo: Modificación de `i_electrica.pl`

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

- Sesión

```
?- prueba_p(está_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 = l2 ;
No
```

Metaintéprete con preguntas

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

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

Metaintérprete con preguntas

- `prueba_p(+O)` se verifica si el objetivo `O` 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).
```

Metaintéprete con preguntas

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

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

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

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

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

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

Metaintéprete con preguntas

- `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 árbol de prueba

```
?- [meta_con_explicacion_arbol, i_electrica].
?- prueba_con_demostracion(está_encendida(L),T).
L = l2
T = si(está_encendida(l2),
       (si(luz(l2), verdad) &
        si(está_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(está_bien(cc1), verdad))
                    si(tiene_corriente(c5),
                       (si(conectado(c5, entrada),
                       si(tiene_corriente(entrada),
```

Metaintérprete con árbol de prueba

- `prueba_con_demostración((+O,?A)` es verdad si `A` es un árbol de prueba del objetivo `O`

```
prueba_con_demostración((verdad,verdad).
prueba_con_demostración(((A & B),(PA & PB)) :-
    prueba_con_demostración((A,PA),
    prueba_con_demostración((B,PB).
prueba_con_demostración((O,si(O,PB)) :-
    (O <- B),
    prueba_con_demostración((B,PB).
```

Metaintérprete con explicación

- Sesión

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

?- prueba_con_explicación(está_encendida(L)).
está_encendida(l2) :-
   1: luz(l2)
   2: está_bien(l2)
   3: tiene_corriente(l2)
|: 3.
tiene_corriente(l2) :-
   1: conectado(l2, c4)
   2: tiene_corriente(c4)
|: 2.
```

Metaintérprete con explicación

- Sesión

```
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 explicación

- `prueba_con_explicación(+O)` significa probar el objetivo `O` a parti de la BC objeto y navegar por su árbol de prueba mediante preguntas.

```
prueba_con_explicación(O) :-
    prueba_con_demostración((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).
```

Metaintérprete con explicación

- `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).
```

- `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.
```

Metaintérprete con explicación

- `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).
```

Bibliografía

- Poole, D.; Mackworth, A. y Goebel, R. *Computational Intelligence (A Logical Approach)* (Oxford University Press, 1998)
  - Cap. 5: “Knowledge engineering”