Tema 6: Programación lógica de segundo orden

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo

Dpto. de Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Modificación de la B.C.

- **Predicados `assert` y `retract`**
  - **`assert(+Term)` inserta un hecho o una cláusula en la base de conocimientos. `Term` es insertado como última cláusula del predicado correspondiente.**
  - **`retract(+Term)` elimina la primera cláusula de la base de conocimientos que unifica con `Term`**
  - **Ejemplos**

```
?- hace_frio.
[WARNING: Undefined predicate: 'hace_frio/0']
No
?- assert(hace_frio).
Yes
?- hace_frio.
Yes
?- retract(hace_frio).
Yes
?- hace_frio.
No
```

Modificación de la B.C.

- **El predicado `listing`**
  - **`listing(+Pred)` lista las cláusulas en cuya cabeza aparece el predicado `Pred`**
  - **Ejemplos**

```
?- listing(select).
select([A|B], A, B).
select([A|B], C, [A|D]) :- select(B, C, D).
Yes
?- assert( (gana(X,Y) :- rapido(X), lento(Y))).
X = _G445
Y = _G446
Yes
?- listing(gana).
gana(A, B) :-
    rapido(A),
    lento(B).
Yes
?- assert(rapido(juan)),assert(lento(jose)),
   assert(lento(luis)).
Yes
?- gana(X,Y).
X = juan  Y = jose ;
X = juan  Y = luis ;
No
?- retract(lento(X)).
X = jose ;
X = luis ;
No
?- gana(X,Y).
No
```

Modificación de la B.C.

- **Los predicados `asserta` y `assertz`**
  - `asserta(+Term)` **equivale a** `assert/1`**, pero** `Term` **es insertado como primera cláusula del predicado correspondiente**
  - `assertz(+Term)` **equivale a** `assert/1`
  - **Ejemplos**

```
?- assert(p(a)), assertz(p(b)), asserta(p(c)).
Yes
?- p(X).
X = c ;
X = a ;
X = b ;
No
?- listing(p).
p(c).
p(a).
p(b).
Yes
```

Modificación de la B.C.

- **Los predicados `retractall` y `abolish`**
  - `retractall(+Head)` **elimina de la base de conocimientos todas las cláusulas cuya cabeza unifica con** `Head`
  - `abolish(+SimbPred/+Aridad)` **elimina de la base de conocimientos todas las cláusulas que en su cabeza aparece el símbolo de predicado** `SimbPred/Aridad`
  - `abolish(+SimbPred, +Aridad)` **es equivalente a** `abolish(+SimbPred/+Aridad)`
  - **Ejemplo**

```
?- assert(p(a)), assert(p(b)).
Yes
?- retractall(p(_)).
Yes
?- p(a).
No
?- assert(p(a)), assert(p(b)).
Yes
?- abolish(p/1).
Yes
?- p(a).
[WARNING: Undefined predicate: ‘p/1’]
No
```

Modificación de la B.C.

```
?- assert(f(a,b)).
Yes
?- f(X,Y).
X = a    Y = b ;
?- asserta(f(a,a)),assertz(f(b,b)).
Yes
?- f(X,Y).
X = a   Y = a ;
X = a   Y = b ;
X = b   Y = b ;
?- listing(f).
f(a, a).
f(a, b).
f(b, b).
?- retract(f(_,a)).
Yes
?- listing(f).
f(a, b).
f(b, b).
?- assert(f(b,a)).
Yes
?- listing(f).
f(a, b).
f(b, b).
f(b, a).
?- retractall(f(b,_)).
Yes
?- listing(f).
f(a, b).
?- abolish(f/2).
Yes
?- listing(f).
[WARNING: No predicates for ‘f’]
No
```

Modificación de la B.C.

- **Multiplicaciones (`tabla.pl`)**
  - **`crea_tabla` añade los hechos `producto(X,Y,Z)` donde `X` e `Y` son números de 0 a 9 y `Z` es el producto de `X` e `Y`.**

```
crea_tabla :-
   L = [0,1,2,3,4,5,6,7,8,9],
   member(X,L),
   member(Y,L),
   Z is X*Y,
   assert(producto(X,Y,Z)),
   fail.
crea_tabla.
```

  - **Sesión:**

```
?- crea_tabla.
Yes
?- listing(producto).
producto(0, 0, 0).  producto(0, 1, 0).   ...
... producto(9, 8, 72). producto(9, 9, 81).
Yes
```

  - **Determinar las descomposiciones de 6 en producto de dos números.**

```
?- producto(A,B,6).
A = 1     A = 2     A = 3     A = 6
B = 6 ;   B = 3 ;   B = 2 ;   B = 1 ;
No
```

Modificación dinámica de la BC

- **Programa**

```
:- dynamic r/2.

relacionados(X,Y) :-
   assert(r(X,Y)).

no_relacionados(X,Y) :-
   retract(r(X,Y)).
```

- **Sesión**

```
?- r(X,Y).
No
?- relacionados(a,b).
Yes
?- r(X,Y).
X = a
Y = b ;
No
?- no_relacionados(a,b).
Yes
?- r(a,b).
No
?- relacionados(a,b), relacionados(c,d).
Yes
?- r(X,Y).
X = a
Y = b ;
X = c
Y = d ;
No
```

Todas las soluciones

- **El predicado `findall`**

```
?- assert(clase(a,voc)),
   assert(clase(b,con)),
   assert(clase(e,voc)),
   assert(clase(c,con)).
Yes

?- findall(X,clase(X,voc),L).
X = _G331
L = [a, e]
Yes

?- findall(_X,clase(_X,voc),L).
L = [a, e]
Yes

?- findall(_X,clase(_X,_Clase),L).
L = [a, b, e, c]
Yes

?- findall(X,clase(X,vocal),L).
X = _G355
L = []
Yes

?- findall(X,(member(X,[c,b,c]),member(X,[c,b,a])),L).
X = _G373
L = [c, b, c]
Yes

?- findall(X,(member(X,[c,b,c]),member(X,[1,2,3])),L).
X = _G373
L = []
Yes
```

Todas las soluciones

- **El predicado** `bagof`

```
?- bagof(X,clase(X,voc),L).
X = _G331
L = [a, e]
Yes

?- bagof(X,clase(X,Clase),L).
X = _G343   Clase = voc   L = [a, e] ;
X = _G343   Clase = con   L = [b, c] ;
No

% L = {X: (existe Y)[clase(X,Y)]}
?- bagof(X,Y^clase(X,Y),L).
X = _G379   Y = _G380   L = [a, b, e, c] ;
No

?- bagof(_X,_Y^clase(_X,_Y),L).
L = [a, b, e, c] ;
No

?- bagof(letra(_X),_Y^clase(_X,_Y),L).
L = [letra(a), letra(b), letra(e), letra(c)]
Yes

?- bagof(X,clase(X,vocal),L).
No

?- bagof(X,(member(X,[c,b,c]),member(X,[c,b,a])),L).
X = _G361
L = [c, b, c] ;
No

?- bagof(X,(member(X,[c,b,c]),member(X,[1,2,3])),L).
No
```

Todas las soluciones

- **El predicado** `setof`

```
?- setof(X,clase(X,voc),L).
X = _G331
L = [a, e]
Yes

?- setof(X,clase(X,Clase),L).
X = _G343   Clase = voc   L = [a, e] ;
X = _G343   Clase = con   L = [b, c] ;
No

% L = {X: (existe Y)[clase(X,Y)]}
?- setof(X,Y^clase(X,Y),L).
X = _G379   Y = _G380   L = [a, b, c, e] ;
No

?- setof(_X,_Y^clase(_X,_Y),L).
L = [a, b, c, e] ;
No

?- setof(letra(_X),_Y^clase(_X,_Y),L).
L = [letra(a), letra(b), letra(c), letra(e)]
Yes

?- setof(X,clase(X,vocal),L).
No

?- setof(X,(member(X,[c,b,c]),member(X,[c,b,a])),L).
X = _G361
L = [b, c]
Yes

?- setof(X,(member(X,[c,b,c]),member(X,[1,2,3])),L).
No
```

Todas las soluciones

- **Operaciones conjuntistas**
  - `interseccion(S,T,U)` **se verifica si** `U` **es la intersección de** `S` **y** `T`**. Por ejemplo,**

```
?- interseccion([1,4,2],[2,3,4],U).
U = [2,4]
```

```
interseccion(S,T,U) :-
   setof(X, (member(X,S), member(X,T)), U).
```

  - `diferencia(S,T,U)` **se verifica si** `U` **es la diferencia de los conjuntos de** `S` **y** `T`**. Por ejemplo,**

```
?- diferencia([5,1,2],[2,3,4],U).
U = [1,5]
```

```
diferencia(S,T,U) :-
   setof(X, (member(X,S), not(member(X,T))), U).
```

  - `n_union(S,T,U)` **se verifica si** `U` **es la unión de** `S` **y** `T`**. Por ejemplo,**

```
?- n_union([1,2,4],[2,3,4],U).
U = [1,2,3,4]
```

```
n_union(S,T,U) :-
   setof(X, (member(X,S); member(X,T)), U).
```

Todas las soluciones

- **partes(X,L) se verifica si L es el conjunto de las partes de X. Por ejemplo,**

```
?- partes([a,b],L).
L = [[], [a], [a, b], [b]]
```

```
partes(X,L) :-
   setof(Y,subconjunto(Y,X),L).

subconjunto([],[]).
subconjunto([X|L1],[X|L2]) :-
   subconjunto(L1,L2).
subconjunto(L1,[_|L2]) :-
   subconjunto(L1,L2).
```

Todas las soluciones

- **Definición de `findall`**
  - **Definición**

```
n_findall(X,Objetivo,_Lista_de_X):-
   Objetivo,
   assert(almacena(X)),
   fail.
n_findall(_X,_Objetivo,Lista_de_X):-
   assert(almacena(fin)),
   recoge(Lista_de_X).

recoge(L):-
   retract(almacena(X)),
   !,
   recoge_aux(X,L).

recoge_aux(fin,[]):- !.
recoge_aux(X,[X|L]):-
   recoge(L).
```

  - **Sesión**

```
?- assert(p(a)), assert(p(b)).
Yes
?- listing(p).
p(a).
p(b).
Yes
?- n_findall(X,p(X),L).
X = _G163
L = [a, b]
Yes
```

Predicados de segundo orden

- **El predicado `apply`**
  - **`n_apply(+Term,+Lista)` se verifica si es demostrable `Term` después de aumentar el número de sus argumentos con los elementos de `Lista`**
  - **Ejemplo**

```
?- plus(1,2,X).
X = 3
Yes
?- n_apply(plus,[1,2,X]).
X = 3 ;
No
?- n_apply(plus(1),[2,X]).
X = 3
Yes
?- n_apply(plus(1,2),[X]).
X = 3
Yes
?- n_apply(append([1,2]),[X,[1,2,3,4,5]]).
X = [3, 4, 5] ;
No
```

  - **Programa `n_apply(+Term,+List)`**

```
n_apply(Term,List):-
   Term =.. [Pred|Arg1],
   append(Arg1,List,Arg2),
   Atomo =.. [Pred|Arg2],
   Atomo.
```

  - **El predicado predefinido `apply`**

Predicados de segundo orden

- **Patrones aplicativos y `maplist`**
  - **`padre(X,P)` se verifica si `P` es el padre de `X`**

```
padre(beatriz,andres).
padre(david,  carlos).
padre(elisa,  ernesto).
```

  - **`madre(X,M)` se verifica si `M` es la madre de `X`**

```
madre(beatriz,maria).
madre(david,  eva).
madre(elisa,  carmen).
```

  - **`padres(L1,L2)` se verifica si cada elemento de `L2` es el padre del correspondiente elemento de `L1`. Por ejemplo,**

```
?- padres([beatriz,david,elisa],L).
L = [andres, carlos, ernesto]
Yes
```

```
padres([],[]).
padres([X|R1],[P|R2]):-
   padre(X,P),
   padres(R1,R2).
```

Predicados de segundo orden

- **madres(L1,L2) se verifica si cada elemento de L2 es la madre del correspondiente elemento de L1. Por ejemplo,**

```
?- madres([beatriz,david,elisa],L).
L = [maria, eva, carmen]
Yes
```

```
madres([],[]).
madres([X|R1],[M|R2]):-
   madre(X,M),
   madres(R1,R2).
```

- **Preguntas con maplist**

```
?- maplist(padre,[beatriz,david,elisa],L).
L = [andres, carlos, ernesto]
Yes

?- maplist(madre,[beatriz,david,elisa],L).
L = [maria, eva, carmen]
Yes
```

- **Definición de maplist**

```
n_maplist(_,[],[]).
n_maplist(R,[X1|L1],[X2|L2]) :-
   apply(R,[X1,X2]),
   n_maplist(R,L1,L2).
```

Contenido

- **Programación lógica de segundo orden:**
  1. **Modificación de la base de conocimiento:**
     (a) **Relaciones para ampliar B.C.:** `assert`, `asserta` **y** `assertz`.
     (b) **Relación para consultar la B.C.:** `listing`.
     (c) **Relaciones para reducir la BC:** `retract`, `retractall` **y** `abolish`.
     (d) **Ejemplo: producto mediante tabla de multiplicar.**
  2. **Modificación dinámica de la BC:**
     (a) **La declaración** `dynamic`.
  3. **Todas las soluciones:**
     (a) **Las relaciones** `findall`, `bagof` **y** `setof`.
     (b) **Ejemplo: operaciones conjuntistas: intersección, diferencia, unión, partes.**
     (c) **Definición de** `findall`.
  4. **Predicados de segundo orden:**
     (a) **El predicado** `apply`.
     (b) **El predicado** `maplist`.

Bibliografía

- Bratko, I. *Prolog Programming for Artificial Intelligence (3th ed.)* (Addison–Wesley, 2001)
  - Cap. 7: "More Built–in Procedures"
- Van Le, T. *Techniques of Prolog Programming* (John Wiley, 1993)
  - Cap. 6: "Advanced programming techniques and data structures"
- Flach, P. *Simply Logical (Intelligent Reasoning by Example)* (John Wiley, 1994)
  - Cap. 3: "Logic programming and Prolog".
- Clocksin, W.F. y Mellish, C.S. *Programming in Prolog (Fourth Edition)* (Springer Verlag, 1994)
  - Cap. 6: "Built–in Predicates"