Tema 11: Aplicaciones de RA con OTTER

José A. Alonso Jiménez
Miguel A. Gutiérrez Naranjo

Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla

El factorial mediante resolución

- **Entrada**

```
set(prolog_style_variables).
set(binary_res).

list(usable).
fact(0,1).
-fact((X - 1),Y) | fact(X,(X * Y)).

-p(fact(X,Y)) | -fact(X,Y) | $ans(Y).
end_of_list.

list(demodulators).
1-1 = 0.   2-1 = 1.   3-1 = 2.  4-1 = 3.
end_of_list.

list(sos).
p(fact(4,Y)).
end_of_list.
```

El factorial mediante resolución

- **Demostración**

```
---------------- PROOF ----------------
1 [] fact(0,1).
2 [] -fact(X-1,Y)|fact(X,X*Y).
3 [] -p(fact(X,Y))| -fact(X,Y)|$ans(Y).
4 [] 1-1=0.
5 [] 2-1=1.
6 [] 3-1=2.
7 [] 4-1=3.
8 [] p(fact(4,Y)).
9 [binary,8.1,3.1] -fact(4,A)|$ans(A).
10 [binary,9.1,2.2,demod,7] $ans(4*A)| -fact(3,A).
11 [binary,10.1,2.2,demod,6] $ans(4*3*A)| -fact(2,A).
12 [binary,11.1,2.2,demod,5] $ans(4*3*2*A)| -fact(1,A).
13 [binary,12.1,2.2,demod,4] $ans(4*3*2*1*A)| -fact(0,A).
14 [binary,13.1,1.1] $ans(4*3*2*1*1).
------------ end of proof -------------
```

El factorial con producto evaluable

```
set(prolog_style_variables).
set(binary_res).

list(usable).
fact(0,1).
-fact((X - 1),Y) | fact(X,$PROD(X,Y)).
-p(fact(X,Y)) | -fact(X,Y) | resp(Y).
end_of_list.

list(demodulators).
1-1 = 0.   2-1 = 1.   3-1 = 2.  4-1 = 3.
end_of_list.

list(sos).
p(fact(4,Y)).
end_of_list.

list(passive).
-resp(X) | $ans(X).
end_of_list.
```

El factorial con producto evaluable

- **Prueba**

```
1 [] fact(0,1).
2 [] -fact(X-1,Y)|fact(X,$PROD(X,Y)).
3 [] -p(fact(X,Y))| -fact(X,Y)|resp(Y).
4 [] 1-1=0.
5 [] 2-1=1.
6 [] 3-1=2.
7 [] 4-1=3.
8 [] p(fact(4,Y)).
9 [] -resp(Y)|$ans(Y).
10 [binary,8.1,3.1] -fact(4,A)|resp(A).
11 [binary,10.1,2.2,demod,7] resp($PROD(4,A))| -fact(3,A).
12 [binary,11.2,2.2,demod,6] resp($PROD(4,$PROD(3,A)))| -fact(2,A).
13 [binary,12.2,2.2,demod,5] resp($PROD(4,$PROD(3,$PROD(2,A))))| -fact(1,A).
14 [binary,13.2,2.2,demod,4]
   resp($PROD(4,$PROD(3,$PROD(2,$PROD(1,A)))))| -fact(0,A).
16 [binary,14.2,1.1,demod] resp(24).
17 [binary,16.1,9.1] $ans(24).
```

El factorial con producto evaluable (II)

```
set(prolog_style_variables).
set(binary_res).
make_evaluable(_*_, $PROD(_,_)).

list(usable).
fact(0,1).
-fact((X - 1),Y) | fact(X,$PROD(X,Y)).
-p(fact(X,Y)) | -fact(X,Y) | resp(Y).
end_of_list.

list(demodulators).
1-1 = 0.   2-1 = 1.   3-1 = 2.  4-1 = 3.
end_of_list.

list(sos).
p(fact(4,Y)).
end_of_list.

list(passive).
-resp(X) | $ans(X).
end_of_list.
```

El factorial con producto evaluable (II)

- **Prueba**

```
1 [] fact(0,1).
2 [] -fact(X-1,Y)|fact(X,X*Y).
3 [] -p(fact(X,Y))| -fact(X,Y)|resp(Y).
4 [] 1-1=0.
5 [] 2-1=1.
6 [] 3-1=2.
7 [] 4-1=3.
8 [] p(fact(4,Y)).
9 [] -resp(X)|$ans(X).
10 [binary,8.1,3.1] -fact(4,A)|resp(A).
11 [binary,10.1,2.2,demod,7] resp(4*A)| -fact(3,A).
12 [binary,11.2,2.2,demod,6] resp(4*3*A)| -fact(2,A).
13 [binary,12.2,2.2,demod,5] resp(4*3*2*A)| -fact(1,A).
14 [binary,13.2,2.2,demod,4] resp(4*3*2*1*A)| -fact(0,A).
16 [binary,14.2,1.1,demod] resp(24).
17 [binary,16.1,9.1] $ans(24).
```

El factorial con producto y resta evaluables

- **Entrada**

```
set(prolog_style_variables).
make_evaluable(_*_, $PROD(_,_)).
make_evaluable(_-_, $DIFF(_,_)).
set(binary_res).

list(usable).
fact(0,1).
-fact(X-1,Y) | fact(X,X*Y).
-p(fact(X,Y)) | -fact(X,Y) | resp(Y).
end_of_list.

list(sos).
p(fact(4,Y)).
end_of_list.

list(passive).
-resp(X) | $ans(X).
end_of_list.
```

El factorial con producto y resta evaluables

- **Prueba**

```
1 [] fact(0,1).
2 [] -fact(X-1,Y)|fact(X,X*Y).
3 [] -p(fact(X,Y))| -fact(X,Y)|resp(Y).
4 [] p(fact(4,Y)).
5 [] -resp(X)|$ans(X).
6 [binary,4.1,3.1] -fact(4,A)|resp(A).
7 [binary,6.1,2.2,demod] resp(4*A)| -fact(3,A).
8 [binary,7.2,2.2,demod] resp(4*3*A)| -fact(2,A).
9 [binary,8.2,2.2,demod] resp(4*3*2*A)| -fact(1,A).
10 [binary,9.2,2.2,demod] resp(4*3*2*1*A)| -fact(0,A).
12 [binary,10.2,1.1,demod] resp(24).
13 [binary,12.1,5.1] $ans(24).
```

El factorial mediante demodulación

- **Entrada**

```
set(prolog_style_variables).
make_evaluable(_*_, $PROD(_,_)).
make_evaluable(_-_, $DIFF(_,_)).
make_evaluable(_>_, $GT(_,_)).
set(binary_res).

list(demodulators).
fact(0) = 1.
X>0 -> fact(X) = X*fact(X-1).
end_of_list.

list(usable).
-p(X) | $ans(factorial,X,fact(X)).
end_of_list.

list(sos).
p(4).
end_of_list.
```

El factorial mediante demodulación

- **Prueba**

```
1 [] fact(0)=1.
2 [] X>0->fact(X)=X*fact(X-1).
3 [] -p(X)|$ans(factorial,X,fact(X)).
4 [] p(4).
5 [binary,4.1,3.1,demod,2,2,2,2,1] $ans(factorial,4,24).
```

El factorial mediante IF

- **Entrada**

```
set(prolog_style_variables).
make_evaluable(_*_, $PROD(_,_)).
make_evaluable(_-_, $DIFF(_,_)).
make_evaluable(_==_, $EQ(_,_)).
set(binary_res).

list(demodulators).
fact(X) = $IF(X==0, 1, X*fact(X-1)).
end_of_list.

list(usable).
-p(X) | $ans(factorial,X,fact(X)).
end_of_list.

list(sos).
p(4).
end_of_list.
```

El factorial mediante IF

- **Prueba**

```
1 [] fact(X)=$IF(X==0,1,X*fact(X-1)).
2 [] -p(X)|$ans(factorial,X,fact(X)).
3 [] p(5).
4 [binary,3.1,2.1,demod,1,1,1,1,1] $ans(factorial,4,24).
```

Máximo común divisor

```
set(prolog_style_variables).
set(binary_res).
make_evaluable(_-_, $DIFF(_,_)).
make_evaluable(_<_, $LT(_,_)).
assign(max_proofs,-1).

list(demodulators).
mcd(0,X) = X.
mcd(X,0) = X.
mcd(X,X) = X.
X<Y -> mcd(X,Y) = mcd(X,Y-X).
Y<X -> mcd(X,Y) = mcd(X-Y,Y).
end_of_list.

list(usable).
-p(X,Y) | $ans(mcd,X,Y,mcd(X,Y)).
end_of_list.

list(sos).
p(12,15).    p(12,13).    p(2,0).
end_of_list.
```

Máximo común divisor

```
---------------- PROOF ----------------
3 [] mcd(X,X)=X.
4 [] X<Y->mcd(X,Y)=mcd(X,Y-X).
5 [] Y<X->mcd(X,Y)=mcd(X-Y,Y).
6 [] -p(X,Y)|$ans(mcd,X,Y,mcd(X,Y)).
7 [] p(12,15).
10 [binary,7.1,6.1,demod,4,5,5,5,3] $ans(mcd,12,15,3).

---------------- PROOF ----------------
3 [] mcd(X,X)=X.
4 [] X<Y->mcd(X,Y)=mcd(X,Y-X).
5 [] Y<X->mcd(X,Y)=mcd(X-Y,Y).
6 [] -p(X,Y)|$ans(mcd,X,Y,mcd(X,Y)).
8 [] p(12,13).
11 [binary,8.1,6.1,demod,4,5,5,5,5,5,5,5,5,5,5,5,3] $ans(mcd,12,13,1).

---------------- PROOF ----------------
2 [] mcd(X,0)=X.
6 [] -p(X,Y)|$ans(mcd,X,Y,mcd(X,Y)).
9 [] p(2,0).
12 [binary,9.1,6.1,demod,2] $ans(mcd,2,0,2).
```

Máximo común divisor

```
set(prolog_style_variables).
make_evaluable(_-_, $DIFF(_,_)).
make_evaluable(_<_, $LT(_,_)).
make_evaluable(_==_, $EQ(_,_)).
set(binary_res).

list(demodulators).
mcd(X,Y) = $IF(X==0, Y,
                     $IF(Y==0, X,
                               $IF(X<Y, mcd(X,Y-X),
                                        mcd(Y,X-Y)))).
end_of_list.

list(usable).
-p(X,Y) | $ans(mcd,X,Y,mcd(X,Y)).
end_of_list.

list(sos).
p(12,15).   p(12,7).   p(2,0).
end_of_list.
```

Máximo común divisor

```
list(demodulators).
1 [] mcd(X,Y)=$IF(X==0,Y,$IF(Y==0,X,$IF(X<Y,mcd(X,Y-X),mcd(Y,X-Y)))).
end_of_list.

list(usable).
2 [] -p(X,Y)|$ans(mcd,X,Y,mcd(X,Y)).
end_of_list.

list(sos).
3 [] p(12,15).
4 [] p(12,7).
5 [] p(2,0).
end_of_list.

-----> EMPTY CLAUSE 6 [binary,3.1,2.1,demod,1,1,1,1,1,1] $ans(mcd,12,15,3).
-----> EMPTY CLAUSE 7 [binary,4.1,2.1,demod,1,1,1,1,1,1,1] $ans(mcd,12,7,1).
-----> EMPTY CLAUSE 8 [binary,5.1,2.1,demod,1] $ans(mcd,2,0,2).
```

Listas: La relación de pertenencia

```
set(prolog_style_variables).
set(binary_res).
assign(max_proofs,-1).

list(demodulators).
pertenece(X,[]) = $F.
$ID(X,Y) -> pertenece(X,[Y|L]) = $T.
$LNE(X,Y) -> pertenece(X,[Y|L]) = pertenece(X,L).
end_of_list.

list(usable).
-p(pertenece(X,Y)) | -pertenece(X,Y) | $ans(pert,X,Y).
-p(pertenece(X,Y)) |  pertenece(X,Y) | $ans(no_pert,X,Y).
end_of_list.

list(sos).
p(pertenece(a,[])).          p(pertenece(a,[a,b,c])).
p(pertenece(b,[a,b,c])).     p(pertenece(d,[a,b,c])).
end_of_list.
```

Listas: La relación de pertenencia

```
1 [] pertenece(X,[])=$F.
5 [] -p(pertenece(X,Y))|pertenece(X,Y)|$ans(no_pert,X,Y).
6 [] p(pertenece(a,[])).
10 [binary,6.1,5.1,demod,1] $ans(no_pert,a,[]).
---------------- PROOF ----------------
2 [] $ID(X,Y)->pertenece(X,[Y|L])=$T.
4 [] -p(pertenece(X,Y))| -pertenece(X,Y)|$ans(pert,X,Y).
7 [] p(pertenece(a,[a,b,c])).
12 [binary,7.1,4.1,demod,2] $ans(pert,a,[a,b,c]).
---------------- PROOF ----------------
2 [] $ID(X,Y)->pertenece(X,[Y|L])=$T.
3 [] $LNE(X,Y)->pertenece(X,[Y|L])=pertenece(X,L).
4 [] -p(pertenece(X,Y))| -pertenece(X,Y)|$ans(pert,X,Y).
8 [] p(pertenece(b,[a,b,c])).
14 [binary,8.1,4.1,demod,3,2] $ans(pert,b,[a,b,c]).
---------------- PROOF ----------------
1 [] pertenece(X,[])=$F.
3 [] $LNE(X,Y)->pertenece(X,[Y|L])=pertenece(X,L).
5 [] -p(pertenece(X,Y))|pertenece(X,Y)|$ans(no_pert,X,Y).
9 [] p(pertenece(d,[a,b,c])).
15 [binary,9.1,5.1,demod,3,3,3,1] $ans(no_pert,d,[a,b,c]).
```

Listas: La relación de pertenencia

```
set(prolog_style_variables).
set(binary_res).
assign(max_proofs,-1).

list(demodulators).
pertenece(X,[])    = $F.
pertenece(X,[Y|L]) = $IF($ID(X,Y), $T,
                                   pertenece(X,L)).
end_of_list.

list(usable).
-p(pertenece(X,Y)) | -pertenece(X,Y) | $ans(pert,X,Y).
-p(pertenece(X,Y)) |  pertenece(X,Y) | $ans(no_pert,X,Y).
end_of_list.

list(sos).
p(pertenece(a,[])).        p(pertenece(a,[a,b,c])).
p(pertenece(b,[a,b,c])).   p(pertenece(d,[a,b,c])).
end_of_list.
```

Listas: La relación de pertenencia

```
1 [] pertenece(X,[])=$F.
4 [] -p(pertenece(X,Y))|pertenece(X,Y)|$ans(no_pert,X,Y).
5 [] p(pertenece(a,[])).
9 [binary,5.1,4.1,demod,1] $ans(no_pert,a,[]).
---------------- PROOF ----------------
2 [] pertenece(X,[Y|L])=$IF($ID(X,Y),$T,pertenece(X,L)).
3 [] -p(pertenece(X,Y))| -pertenece(X,Y)|$ans(pert,X,Y).
6 [] p(pertenece(a,[a,b,c])).
10 [binary,6.1,3.1,demod,2] $ans(pert,a,[a,b,c]).
---------------- PROOF ----------------
2 [] pertenece(X,[Y|L])=$IF($ID(X,Y),$T,pertenece(X,L)).
3 [] -p(pertenece(X,Y))| -pertenece(X,Y)|$ans(pert,X,Y).
7 [] p(pertenece(b,[a,b,c])).
11 [binary,7.1,3.1,demod,2,2] $ans(pert,b,[a,b,c]).
---------------- PROOF ----------------
1 [] pertenece(X,[])=$F.
2 [] pertenece(X,[Y|L])=$IF($ID(X,Y),$T,pertenece(X,L)).
4 [] -p(pertenece(X,Y))|pertenece(X,Y)|$ans(no_pert,X,Y).
8 [] p(pertenece(d,[a,b,c])).
12 [binary,8.1,4.1,demod,2,2,2,1] $ans(no_pert,d,[a,b,c]).
```

Concatenación de listas

```
set(prolog_style_variables).
set(binary_res).
assign(max_proofs,-1).

list(demodulators).
concatenacion([],L)      = L.
concatenacion([X|L1],L2) = [X|concatenacion(L1,L2)].
end_of_list.

list(usable).
-p(concatenacion(X,Y)) | $ans(concatenacion,X,Y,concatenacion(X,Y)).
end_of_list.

list(sos).
p(concatenacion([b,e],[a,b,c,d])).
end_of_list.
```

Concatenación de listas

```
---------------- PROOF ----------------
1 [] concatenacion([],L)=L.
2 [] concatenacion([X|L1],L2)=[X|concatenacion(L1,L2)].
3 [] -p(concatenacion(X,Y))|$ans(concatenacion,X,Y,concatenacion(X,Y)).
4 [] p(concatenacion([b,e],[a,b,c,d])).
5 [binary,4.1,3.1,demod,2,2,1]
     $ans(concatenacion,[b,e],[a,b,c,d],[b,e,a,b,c,d]).
------------ end of proof -------------
```

Inversión de listas

```
set(prolog_style_variables).
set(binary_res).
assign(max_proofs,-1).

list(demodulators).
inversa(L)              = inversa_aux(L,[]).
inversa_aux([], L)      = L.
inversa_aux([X|L1],L2) = inversa_aux(L1,[X|L2]).
end_of_list.

list(usable).
-p(inversa(X)) | $ans(inversa,X,inversa(X)).
end_of_list.

list(sos).
p(inversa([a,b,c])).
end_of_list.
```

Inversión de listas

```
---------------- PROOF ----------------
1 [] inversa(L)=inversa_aux(L,[]).
2 [] inversa_aux([],L)=L.
3 [] inversa_aux([X|L1],L2)=inversa_aux(L1,[X|L2]).
4 [] -p(inversa(X))|$ans(inversa,X,inversa(X)).
5 [] p(inversa([a,b,c])).
6 [binary,5.1,4.1,demod,1,3,3,3,2] $ans(inversa,[a,b,c],[c,b,a]).
------------ end of proof -------------
```

Operaciones conjuntistas

```
set(prolog_style_variables).
make_evaluable(_&_, $AND(_,_)).
set(binary_res).
assign(max_proofs,-1).

list(demodulators).
pertenece(X,[])    = $F.
pertenece(X,[Y|L]) = $IF($ID(X,Y), $T,
                                   pertenece(X,L)).

subconjunto([],L)      = $T.
subconjunto([X|L1],L2) = (pertenece(X,L2) & subconjunto(L1,L2)).

interseccion([],L)      = [].
interseccion([X|L1],L2) = $IF(pertenece(X,L2), [X|interseccion(L1,L2)],
                                                interseccion(L1,L2)).

union([],L)      = L.
union([X|L1],L2) = $IF(pertenece(X,L2), union(L1,L2),
                                        [X|union(L1,L2)]).
end_of_list.
```

Operaciones conjuntistas

```
list(usable).
-p(subconjunto(X,Y)) | -subconjunto(X,Y) | $ans(subc,X,Y).
-p(subconjunto(X,Y)) |  subconjunto(X,Y) | $ans(no_subc,X,Y).
-p(interseccion(X,Y)) | $ans(interseccion,X,Y,interseccion(X,Y)).
-p(union(X,Y)) | $ans(union,X,Y,union(X,Y)).
end_of_list.

list(sos).
p(subconjunto([],[a,b])).
p(subconjunto([a],[a,b])).
p(subconjunto([a,b],[a,b])).
p(subconjunto([c],[a,b])).
p(subconjunto([a,c],[a,b])).
p(interseccion([b,d],[a,b,c,d])).
p(union([b,e],[a,b,c,d])).
end_of_list.
```

Operaciones conjuntistas

```
                PROOF
3 [] subconjunto([],L)=$T.
5 [] -p(subconjunto(X,Y))| -subconjunto(X,Y)|$ans(subc,X,Y).
7 [] p(subconjunto([],[a,b])).
12 [binary,7.1,5.1,demod,3] $ans(subc,[],[a,b]).
---------------- PROOF ----------------
2 [] pertenece(X,[Y|L])=$IF($ID(X,Y),$T,pertenece(X,L)).
3 [] subconjunto([],L)=$T.
4 [] subconjunto([X|L1],L2)= (pertenece(X,L2)&subconjunto(L1,L2)).
5 [] -p(subconjunto(X,Y))| -subconjunto(X,Y)|$ans(subc,X,Y).
8 [] p(subconjunto([a],[a,b])).
13 [binary,8.1,5.1,demod,4,2,3] $ans(subc,[a],[a,b]).
---------------- PROOF ----------------
1 [] pertenece(X,[])=$F.
2 [] pertenece(X,[Y|L])=$IF($ID(X,Y),$T,pertenece(X,L)).
3 [] subconjunto([],L)=$T.
4 [] subconjunto([X|L1],L2)= (pertenece(X,L2)&subconjunto(L1,L2)).
6 [] -p(subconjunto(X,Y))|subconjunto(X,Y)|$ans(no_subc,X,Y).
10 [] p(subconjunto([c],[a,b])).
14 [binary,10.1,6.1,demod,4,2,2,1,3] $ans(no_subc,[c],[a,b]).
```

Operaciones conjuntistas

```
---------------- PROOF ----------------
2 [] pertenece(X,[Y|L])=$IF($ID(X,Y),$T,pertenece(X,L)).
3 [] subconjunto([],L)=$T.
4 [] subconjunto([X|L1],L2)= (pertenece(X,L2)&subconjunto(L1,L2)).
5 [] -p(subconjunto(X,Y))| -subconjunto(X,Y)|$ans(subc,X,Y).
9 [] p(subconjunto([a,b],[a,b])).
15 [binary,9.1,5.1,demod,4,2,4,2,2,3] $ans(subc,[a,b],[a,b]).
---------------- PROOF ----------------
1 [] pertenece(X,[])=$F.
2 [] pertenece(X,[Y|L])=$IF($ID(X,Y),$T,pertenece(X,L)).
3 [] subconjunto([],L)=$T.
4 [] subconjunto([X|L1],L2)= (pertenece(X,L2)&subconjunto(L1,L2)).
6 [] -p(subconjunto(X,Y))|subconjunto(X,Y)|$ans(no_subc,X,Y).
11 [] p(subconjunto([a,c],[a,b])).
16 [binary,11.1,6.1,demod,4,2,4,2,2,1,3] $ans(no_subc,[a,c],[a,b]).
```

Operaciones conjuntistas

```
---------------- PROOF ----------------
2 [] pertenece(X,[Y|L])=$IF($ID(X,Y),$T,pertenece(X,L)).
5 [] interseccion([],L)=[].
6 [] interseccion([X|L1],L2)=
     $IF(pertenece(X,L2),[X|interseccion(L1,L2)],interseccion(L1,L2)).
11 [] -p(interseccion(X,Y))|$ans(interseccion,X,Y,interseccion(X,Y)).
18 [] p(interseccion([b,d],[a,b,c,d])).
25 [binary,18.1,11.1,demod,6,2,2,6,2,2,2,2,5]
   $ans(interseccion,[b,d],[a,b,c,d],[b,d]).
---------------- PROOF ----------------
1 [] pertenece(X,[])=$F.
2 [] pertenece(X,[Y|L])=$IF($ID(X,Y),$T,pertenece(X,L)).
7 [] union([],L)=L.
8 [] union([X|L1],L2)=$IF(pertenece(X,L2),union(L1,L2),[X|union(L1,L2)]).
12 [] -p(union(X,Y))|$ans(union,X,Y,union(X,Y)).
19 [] p(union([b,e],[a,b,c,d])).
26 [binary,19.1,12.1,demod,8,2,2,8,2,2,2,2,1,7]
   $ans(union,[b,e],[a,b,c,d],[e,a,b,c,d]).
```

Ordenación

```
set(prolog_style_variables).
make_evaluable(_@<=_, $LE(_,_)).  make_evaluable(_@>_,  $GT(_,_)).
set(binary_res).                  assign(max_proofs,-1).

list(demodulators).
concatenacion([],L)      = L.
concatenacion([X|L1],L2) = [X|concatenacion(L1,L2)].

ordenacion([])    = [].
ordenacion([X|L]) = concatenacion(ordenacion(menores(X,L)),
                                  [X|ordenacion(mayores(X,L))]).

menores(X,[])    = [].
menores(X,[Y|L]) = $IF(Y @<= X, [Y|menores(X,L)],
                                menores(X,L)).

mayores(X,[])    = [].
mayores(X,[Y|L]) = $IF(Y @> X, [Y|mayores(X,L)],
                               mayores(X,L)).
end_of_list.
```

Ordenación

```
list(usable).
-p(ordenacion(X)) | $ans(ordenacion,X,ordenacion(X)).
end_of_list.

list(sos).
p(ordenacion([3,1,2])).
end_of_list.
```

Ordenación

```
---------------- PROOF ----------------
1 [] concatenacion([],L)=L.
2 [] concatenacion([X|L1],L2)=[X|concatenacion(L1,L2)].
3 [] ordenacion([])=[].
4 [] ordenacion([X|L])=
     concatenacion(ordenacion(menores(X,L)),[X|ordenacion(mayores(X,L))]).
5 [] menores(X,[])=[].
6 [] menores(X,[Y|L])=$IF(Y@<=X,[Y|menores(X,L)],menores(X,L)).
7 [] mayores(X,[])=[].
8 [] mayores(X,[Y|L])=$IF(Y@>X,[Y|mayores(X,L)],mayores(X,L)).
9 [] -p(ordenacion(X))|$ans(ordenacion,X,ordenacion(X)).
10 [] p(ordenacion([3,1,2])).
11 [binary,10.1,9.1,demod,4,6,6,5,4,6,5,3,8,7,4,5,3,7,3,1,1,8,8,7,3,2,2,1]
   $ans(ordenacion,[3,1,2],[1,2,3]).
------------ end of proof -------------
```

Rompecabeza: Problema del baile

- **Problema: En un baile hay 25 personas. La primera mujer ha bailado con los 10 primeros hombres, la segunda con los 11 primeros, la tercera con los 12 primeros y así sucesivamente. Los 10 primeros hombres han bailado con todas las mujeres, el undécimo con todas menos con la primera y así sucesivamente. ¿Cuántas mujeres y hombres hay en el baile?.**

- **Entrada**

```
make_evaluable(_+_, $SUM(_,_)).
make_evaluable(_<_, $LT(_,_)).
make_evaluable(_==_, $EQ(_,_)).
set(binary_res).
set(hyper_res).

list(sos).
total(25).
bailado(mujer(1),10).
-bailado(mujer(x),y) | -total(z) | - (x + y < z) | bailado(mujer(x+1),y+1).
-bailado(mujer(x),y) | -total(x+y) | $ans(x,y).
end_of_list.
```

Rompecabeza: Problema del baile

- **Solución**

```
---------------- PROOF ----------------
1 [] total(25).
2 [] bailado(mujer(1),10).
3 [] -bailado(mujer(x),y)| -total(z)| -(x+y<z)|bailado(mujer(x+1),y+1).
4 [] -bailado(mujer(x),y)| -total(x+y)|$ans(x,y).
12 [hyper,3,2,1,eval,demod] bailado(mujer(2),11).
15 [hyper,12,3,1,eval,demod] bailado(mujer(3),12).
18 [hyper,15,3,1,eval,demod] bailado(mujer(4),13).
21 [hyper,18,3,1,eval,demod] bailado(mujer(5),14).
24 [hyper,21,3,1,eval,demod] bailado(mujer(6),15).
27 [hyper,24,3,1,eval,demod] bailado(mujer(7),16).
30 [hyper,27,3,1,eval,demod] bailado(mujer(8),17).
31 [binary,30.1,4.1,demod] -total(25)|$ans(8,17).
32 [binary,31.1,1.1] $ans(8,17).
------------ end of proof -------------
```

Planificación: Problema del mono

- **Problema**

```
+-----------------+-----------------+
|                 |                 |
|                 |                 |
|              Plátano              |
|                                   |
|     Mono                Silla     |
+-----------------------------------+
         A         B         C
```

- **Representación:**

  `vale(pos_mono(X),pos_platano(Y),pos_silla(Z),Plan)`

  significa que en el estado obtenido aplicando el `Plan` **(inverso) al estado inicial se verifica que la posición del mono es** `X`**, la del plátano es** `Y` **y la de la silla es** `Z`

Planificación: Problema del mono

```
set(prolog_style_variables).
set(input_sequent).
set(output_sequent).
set(ur_res).

list(usable).
posicion(X), posicion(Y),
vale(pos_mono(X),pos_platano(Pp),pos_silla(Ps),Plan)
->
vale(pos_mono(Y),pos_platano(Pp),pos_silla(Ps),[andar(X,Y)|Plan]).

posicion(Y),
vale(pos_mono(X),pos_platano(Pp),pos_silla(X),Plan)
->
vale(pos_mono(Y),pos_platano(Pp),pos_silla(Y),[empujar(X,Y)|Plan]).

vale(pos_mono(P),pos_platano(P),pos_silla(P),Plan)
->
coge_platano([subir|Plan]).
end_of_list.
```

Planificación: Problema del mono

```
list(sos).
-> posicion(a).
-> posicion(b).
-> posicion(c).

-> vale(pos_mono(a),pos_platano(b),pos_silla(c),[]).

coge_platano(Plan) -> resp(inversa(Plan,[])).
end_of_list.

list(passive).
resp(Plan) -> $ans(Plan).
end_of_list.

list(demodulators).
-> inversa([X|L1],L2) = inversa(L1,[X|L2]).
-> inversa([],L) = L.
end_of_list.
```

Planificación: Problema del mono

```
1 [] posicion(X), posicion(Y),
     vale(pos_mono(X),pos_platano(Pp),pos_silla(Ps),Plan)
     -> vale(pos_mono(Y),pos_platano(Pp),pos_silla(Ps),[andar(X,Y)|Plan]).
2 [] posicion(Y), vale(pos_mono(X),pos_platano(Pp),pos_silla(X),Plan)
     -> vale(pos_mono(Y),pos_platano(Pp),pos_silla(Y),[empujar(X,Y)|Plan]).
3 [] vale(pos_mono(P),pos_platano(P),pos_silla(P),Plan)
     -> coge_platano([subir|Plan]).
4 [] -> posicion(a).
5 [] -> posicion(b).
6 [] -> posicion(c).
7 [] -> vale(pos_mono(a),pos_platano(b),pos_silla(c),[]).
8 [] coge_platano(Plan) -> resp(inversa(Plan,[])).
9 [] resp(Plan) -> $ans(Plan).
10 []  -> inversa([X|L1],L2)=inversa(L1,[X|L2]).
11 []  -> inversa([],L)=L.
12 [hyper,7,1,4,6]  -> vale(pos_mono(c),pos_platano(b),pos_silla(c), [andar(a,c)]).
16 [hyper,12,2,5]   -> vale(pos_mono(b),pos_platano(b),pos_silla(b),
                            [empujar(c,b),andar(a,c)]).
33 [hyper,16,3]     -> coge_platano([subir,empujar(c,b),andar(a,c)]).
40 [hyper,33,8,demod,10,10,10,11]  -> resp([andar(a,c),empujar(c,b),subir]).
41 [binary,40.1,9.1]  -> $ans([andar(a,c),empujar(c,b),subir]).
```

Problema de las jarras

- **Enunciado:**
  - **Se tienen dos jarras, una de 4 litros de capacidad y otra de 3.**
  - **Ninguna de ellas tiene marcas de medición.**
  - **Se tiene una bomba que permite llenar las jarras de agua.**
  - **Averiguar cómo se puede lograr tener exactamente 2 litros de agua en la jarra de 4 litros de capacidad.**
- **Entrada**

```
set(prolog_style_variables).
set(input_sequent).
set(output_sequent).
make_evaluable(_+_, $SUM(_,_)).
make_evaluable(_-_, $DIFF(_,_)).
make_evaluable(_<=_, $LE(_,_)).
make_evaluable(_>_, $GT(_,_)).
set(hyper_res).
```

Problema de las jarras

```
list(usable).
e(X,Y)            -> e(3,Y).
e(X,Y)            -> e(0,Y).
e(X,Y)            -> e(X,4).
e(X,Y)            -> e(X,0).
e(X,Y), X+Y <= 4 -> e(0,Y+X).
e(X,Y), X+Y >  4 -> e(X - (4-Y), 4).
e(X,Y), X+Y <= 3 -> e(X+Y, 0).
e(X,Y), X+Y >  3 -> e(3, Y - (3-X)).
end_of_list.

list(sos).
-> e(0,0).         % Estado inicial
e(X,2) ->.  % Estado final
end_of_list.
```

Problema de las jarras

- **Prueba**

```
2 [] e(X,Y) -> e(0,Y).
3 [] e(X,Y) -> e(X,4).
7 [] e(X,Y), X+Y<=3 -> e(X+Y,0).
8 [] e(X,Y), X+Y>3 -> e(3,Y- (3-X)).
9 []  -> e(0,0).
10 [] e(X,2) -> .
11 [hyper,9,3]  -> e(0,4).
13 [hyper,11,8,eval,demod]  -> e(3,1).
16 [hyper,13,2]  -> e(0,1).
18 [hyper,16,7,eval,demod]  -> e(1,0).
20 [hyper,18,3]  -> e(1,4).
22 [hyper,20,8,eval,demod]  -> e(3,2).
23 [binary,22.1,10.1]  -> .
```

Bibliografía

- **Alonso, J.A.; Fernández, A. y Pérez, M.J.** ***Razonamiento automático*** **(**`http://www-cs.us.es/~jalonso/ra/documentacion/alonso-94.ps`**)**
- **Baj, F.** ***Logic with PAIL*** **(**`http://cbl.leeds.ac.uk/nikos/pail/logic/logic.html`**)**
- **Chang, C.L.; Lee, R.C.T.** ***Symbolic logic and mechanical theorem proving.*** **(Academic Press, 1973)**
- **McCune, W.** OTTER ***3.0 Reference Manual and Guide.*** **(Technical Report ANL–94/6, Argone National Laboratory, 1994)**
- **Wos, L.; Overbeek, R.; Lusk, E. y Boyle, J.** ***Automated Reasoning: Introduction and Applications, (2nd ed.)*** **(McGraw–Hill, 1992)**