Prolog (2)

Carlos Hurtado L.

Depto de Ciencias de la Computación, Universidad de Chile

Clase Pasada: Prolog

- ***Pro****gramation et* ***Log****ique*
- Desarrollado por Colmerauer y Roussel de la Universidad de Aix-Marselle a principios de los 70s.
- Define el paradigma de programación lógica (versus programación procedural).
- Resolución sobre cláusulas de Horn

Clase Pasada: Compiladores e IDEs para Prolog

- GNU Prolog (GPL)
- **SWI Prolog (LGPL)**
- Kernel Prolog (GPL)
- B-Prolog (comercial)
- Strawberry Prolog (comercial)
- ... etc
- www.prolog.info lista 34 compiladores e IDEs

Clase Pasada: Notación de Listas en Prolog

- [] is a constant symbol (empty list)
- [|] is a binary function symbol: infix notation (cons)
  - [X|Y] ≡ cons(X,Y)
- [a,b,c] shorthand for [a | [b | [c | []]]] ≡ cons(a,cons(b,cons(c,el))).
  - [c | []] ≡ [c] the list ‘c’—different from ‘c’ by itself.
  - [b | [c]] = [b,c] the list ‘b,c’.
  - [a | [b,c]] = [a,b,c].

Concatenación (Append)

- El predicado Append(X,Y,Z) se define usando dos reglas:

```
(A1)  append([], Z, Z).

(A2)  append([E1 | R1], Y, [E1 | Rest]) <-
                append(R1, Y, Rest).
```

Ejemplo Concatenación (1)

Query: ? append([a,b], [c,d], [a,b,c,d]).

(A1) append([], Z, Z).

(A2) append([E1 | R1], Y, [E1 | Rest]) <-
append(R1, Y, Rest).

Derivation:

yes <- append([a,b], [c,d], [a,b,c,d]).

yes <- append([b], [c,d], [b,c,d]).

Resolve with (A2) using { E1/a, R1/[b], Y/[c,d], Rest/[b,c,d] }

yes <- append([], [c,d], [c,d]).

Resolve with (A2) using { E1/b, R1/[], Y/[c,d], Rest/[c,d] }

yes <- .

Resolve with (A1) using { Z/[c,d] }

Answer: yes

Clase Pasada: Ejercicio: Concatenación

- Exercise: Give derivations for at least two other answers for the previous query:
- Query: ? append(L, M, [a,b,c,d]).
  - L = [], M = [a,b,c,d]
  - L = [a], M = [b,c,d]
  - L = [a,b], M = [c,d]
  - L = [a,b,c], M = [d]
  - L = [a,b,c,d], M = []
- Note that specifying append "declaratively" (as as set of clauses) means that we can use these clauses in very flexible ways. This would not be possible with a "procedural" representation!

Ejercicios de Prolog

- Werner Hett, Bern U. of Applied Science.
- P-99: Ninety-Nine Prolog Problems
- https://prof.ti.bfh.ch/hew1/informatik3/prolog/p-99/

P01

- Entregar el u'ltimo elemento de una lista
- Ejemplo:

  ?- my_last(X,[a,b,c,d]).

  X = d

P01: Resp

- Entregar el u'ltimo elemento de una lista
- Ejemplo:

  ?- my_last(X,[a,b,c,d]).

  X = d
- Resp:

```
my_last(X,[X]).
my_last(X,[_|L]) :- my_last(X,L).
```

P03

- Entregar el elemento en la posición K de una lista
- Ejemplo:

?- element_at(X,[a,b,c,d,e],3).

X = c

P03

- Entregar el elemento en la posición K de una lista
- Ejemplo:

?- element_at(X,[a,b,c,d,e],3).

X = c

- Resp:

```
element_at(X,[X|_],1).
element_at(X,[_|L],K) :- K > 1, K1 is K - 1,
    element_at(X,L,K1).
```

P14

- Duplicar los elementos de una lista
- Ejemplo:

?- dupli([a,b,c,c,d],X).

X = [a,a,b,b,c,c,c,c,d,d]

P14

- Duplicar los elementos de una lista
- Ejemplo:

?- dupli([a,b,c,c,d],X).

X = [a,a,b,b,c,c,c,c,d,d]

- Resp:

```
dupli([],[]).
dupli([X|Xs],[X,X|Ys]) :- dupli(Xs,Ys).
```

P04

- Num. de elementos de una lista
- Ejemplo:

?-my_length([a,b,c],X).

X=3

P04

- Num. de elementos de una lista
- Ejemplo:

?-my_length([a,b,c],X).

X=3

- Resp:

```
my_length([],0).
my_length([_|L],N) :- my_length(L,N1), N is N1 +
   1.
```

P05

- Invertir una lista:
- Ejemplo:

?-my_reverse([a,b,c],X).

X=[c,b,a].

Ojo: en [Y|L] Y es un elemento y L es una lista.

P05

- Invertir una lista:
- Ejemplo:

?-my_reverse([a,b,c],X).

X=[c,b,a].

- Resp:

```
my_reverse(L1,L2) :- my_rev(L1,L2,[]).
my_rev([],L2,L2).
my_rev([X|Xs],L2,Acc):-my_rev(Xs,L2,[X|Acc]).
```

P05

- Solución alternativa:

```
my_reverse([],[]).
my_reverse([X|L1],[Y|L2]):- my_reverse([X|
   L3],L2), my_reverse([Y|
   L4],L1),my_reverse(L3,L4).
?-my_reverse([a,b,c],L).
```

- Produce recursión infinita.

  “Warning: Out of local stack -- abort”.

Ejemplo: Recursión Infinita

```
a :- b, c, d.
b.
c :- b, e.
c :- b, a.
d.
f.
?-a.
```

Ejemplo: Recursión Infinita

```
owes(andy,bill).
owes(bill,carl).
owes(carl,bill).
avoids(X,Y) :- owes(X,Y).
avoids(X,Y) :- owes(X,Z), avoids(Z,Y).

?-avoids(andy,x).
X=bill;
X=andy;
X=bill;
.... etc.
```

Ejemplo: Recursión infinita

```
isa(garfield,cat).
isa(tom,cat).
isa(sylvester,cat).
isa(tweety,canary).
isa(heckle,magpie).
isa(jeckle,magpie).
isa(woody,woodpecker).
isa(canary,bird).
isa(magpie,bird).
isa(woodpecker,bird).
isa(cat,mammal).
isa(cat,pest).
isa(bird,vertebrate).
isa(mammal,vertebrate).
isa(X,Y) :- isa(X,Z),isa(Z,Y).
```

Ejemplo: Recursión Infinita

```
isa(garfield,cat).
isa(tom,cat).
isa(sylvester,cat).
isa(tweety,canary).
isa(heckle,magpie).
isa(jeckle,magpie).
isa(woody,woodpecker).
isa(canary,bird).
isa(magpie,bird).
isa(woodpecker,bird).
isa(cat,mammal).
isa(cat,pest).
isa(bird,vertebrate).
isa(mammal,vertebrate).
isa(X,Y) :- isa(X,Z),isa(Z,Y).
```

```
?- isa(tom,What).

What = cat ;

What = mammal ;

What = pest ;

What = vertebrate ;
[WARNING: Out of local stack]
   Exception: (42321)
     isa(vertebrate, _L507898) ?
     abort
```

Solución

```
isa(garfield,cat).
isa(tom,cat).
isa(sylvester,cat).
isa(tweety,canary).
isa(heckle,magpie).
isa(jeckle,magpie).
isa(woody,woodpecker).
isa(canary,bird).
isa(magpie,bird).
isa(woodpecker,bird).
isa(cat,mammal).
isa(cat,pest).
isa(bird,vertebrate).
isa(mammal,vertebrate).
isin(X,Y) :- isa(X,Y).
isin(X,Y) :- isa(X,Z),isin(Z,Y).
```

```
?- isin(tom,What).
What=cat;
What=mammal;
What=pest;
What=vertebrate;
no
```

P06

- Decir si una lista es palíndrome
- Resp:

```
is_palindrome(X):-reverse(X,X).
```

P08

- Eliminar duplicados consecutivos
- Ejemplo:

?-compress([a,a,a,b,c,c]).

X=[a,b,c]

P08

- Eliminar duplicados consecutivos
- Ejemplo:

?-compress([a,a,a,b,c,c],X).

X=[a,b,c]

- Resp:

```
compress([],[]).
compress([X],[X]).
compress([X,X|Xs],Zs) :- compress([X|Xs],Zs).
compress([X,Y|Ys],[X|Zs]) :- X \= Y,
   compress([Y|Ys],Zs).
```

P18

- Extraer una sublista de una lista
- Ejemplo:

?- slice([a,b,c,d,e,f,g,h,i,k],3,7,L).

L= [c,d,e,f,g]

Nota: usar predicados “>” y asignación “is”.

P18

- Extraer una sublista de una lista
- Ejemplo:

?- slice([a,b,c,d,e,f,g,h,i,k],3,7,L).

L= [c,d,e,f,g]

- Resp.

```
slice([X|_],1,1,[X]).
slice([X|Xs],1,K,[X|Ys]) :- K > 1, K1 is K - 1,
   slice(Xs,1,K1,Ys).
slice([_|Xs],I,K,Ys) :- I > 1, I1 is I - 1, K1 is K - 1,
   slice(Xs,I1,K1,Ys).
```

P22

- Crear una lista que contenga todos los enteros en un rango dado
- Ejemplo:

?- range(4,9,L).

L = [4,5,6,7,8,9]

P22

- Crear una lista que contenga todos los enteros en un rango dado
- Ejemplo:

?- range(4,9,L).

L = [4,5,6,7,8,9]

- Resp:

```
range(I,I,[I]).
range(I,K,[I|L]) :- I < K, I1 is I + 1,
  range(I1,K,L).
```

P31

- Determinar si un número es primo
- Ejemplo:

?- is_prime(7).

Yes

Hint: Usar negación \+

P31

- Determinar si un número es primo
- Ejemplo:

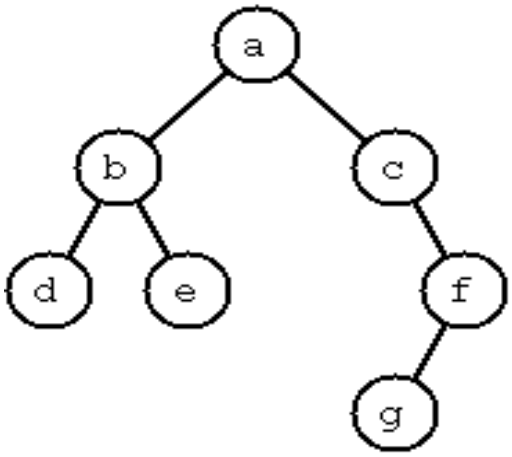
?- is_prime(7).

Yes

- Resp:

```
is_prime(2).
is_prime(3).
is_prime(P) :- integer(P), P > 3, P mod 2 =\= 0, \+
    has_factor(P,3).
has_factor(N,L) :- N mod L =:= 0.
has_factor(N,L) :- L < N, L2 is L + 2, has_factor(N,L2).
```

Arboles



t(a,t(b,t(d,nil,nil),t(e,nil,nil)),t(c,nil,t(f,t(g,nil,nil),nil)))

P54

- Verificar si un término es un árbol
- Ejemplo:

?- istree(t(a,t(b,nil,nil),nil)).

Yes

?- istree(t(a,t(b,nil,nil))).

No

P54

- Verificar si un t'ermino es un árbol
- Ejemplo:

?- istree(t(a,t(b,nil,nil),nil)).

Yes

?- istree(t(a,t(b,nil,nil))).

No

- Resp:

```
istree(nil).
istree(t(_,L,R)) :- istree(L), istree(R).
```

P61

- Contar los nodos internos de un árbol binario

P61

- Contar los nodos internos de un árbol binario
- Resp:

```
internal_node(t(_,L,R)) :- \+ (L = nil, R = nil).

count_internal_nodes(t(_,L,R),N) :-
  internal_node(t(_,L,R)),
  count_internal_nodes(L,NL),
  count_internal_nodes(R,NR),
  N is NL + NR + 1.
count_internal_nodes(_,0).
```

P61A

- Recolectar las hojas de un árbol binario en una lista
- Ejemplo:

?-leaves(t(a,t(b,nil,nil),t(c,nil,nil),S)

S=[b,c]

Se puede usar el predicado append.

P61A: Resp

```
leaves(nil,[]).
leaves(t(X,nil,nil),[X]).
leaves(t(_,L,nil),S) :- L = t(_,_,_), leaves(L,S).
leaves(t(_,nil,R),S) :- R = t(_,_,_), leaves(R,S).
leaves(t(_,L,R),S) :- L = t(_,_,_), R = t(_,_,_),
    leaves(L,SL), leaves(R,SR), append(SL,SR,S).
```

Cut

- La idea del cut en prolog es agregar información explícita sobre control del flujo del programa.
- Podar el árbol de ejecución.
- Caso 1:

p(X) :- q(X), !, r(X).

Si q tiene éxito con un valor x de X, entonces si r(x) falla, no podemos hacer backtracking.

- Caso 2:

p(X) :- r(X), !.

p(X) :- s(X).

Si usamos la primera regla y esta tiene éxito con un valor x de X, no usamos la segunda regla.

Cut: Ejemplo 0

- Sin Cut

```
q(a).
q(b).
r(b).
p(X) :- q(X), r(X).
?-p(X).
X=b
```

- Con Cut:

```
q(a).
q(b).
r(b).
p(X) :- q(X), !, r(X).
?-p(X).
No
```

Cut: Ejemplo 1

```
q(a).
q(b).
r(1).
r(2).
r(3).
p(X,Y) :- q(X), !, r(Y).
?-p(X,Y).
```

Cut: Ejemplo 1

```
q(a).
q(b).
r(1).
r(2).
r(3).
p(X,Y) :- q(X), !, r(Y).
?-p(X,Y).
X=a,Y=1;
X=a,Y=2;
X=a,Y=3;
No
```

Cut: Ejemplo 2

```
grade(Mark, first) :- Mark>=70.
grade(Mark, two_1) :- Mark<70, Mark>=63.
grade(Mark, two_2) :- Mark<63, Mark>=55.
grade(Mark, third) :- Mark<55, Mark>=50.
grade(Mark, pass)  :- Mark<50, Mark>=40.
grade(Mark, fail)  :- Mark<40.
?-grade(60,X)
X=two_1;
No
```

Cut: Ejemplo 2

```
grade(N,first) :- N>=70, ! .
grade(N,two_1) :- N>=63, ! .
grade(N,two_2) :- N>=55, ! .
grade(N,third) :- N>=50, ! .
grade(N,pass) :- N>=40, ! .
grade(N,fail) :- N<40.
?-grade(60,X)
X=two_1;
No
```

P05

- Invertir una lista:
- Ejemplo:

?-my_rev([a,b,c],X).

X=[c,b,a].

- Resp:

```
my_reverse(L1,L2) :- my_rev(L1,L2,[]).
my_rev([],L2,L2):- !.
my_rev([X|Xs],L2,Acc):-my_rev(Xs,L2,[X|Acc]).
```

Negación en Prolog

- La negación se define como:

```
not(P) :- call(P), !, fail.
not(P).
```

- En SWI-Prolog not(p) se denota \+

Ejemplo

```
bachelor(P) :- male(P), not(married(P)).
male(henry).
male(tom).
married(tom).
```

```
?- bachelor(henry).
```

Ejemplo

```
bachelor(P) :- male(P), not(married(P)).
male(henry).
male(tom).
married(tom).
```

```
?- bachelor(henry).
yes
?- bachelor(tom).
```

Ejemplo

```
bachelor(P) :- male(P), not(married(P)).
male(henry).
male(tom).
married(tom).
```

```
?- bachelor(henry).
yes
?- bachelor(tom).
no
?- bachelor(Who).
```

Ejemplo

bachelor(P) :- male(P), not(married(P)).

male(henry).

male(tom).

married(tom).

?- bachelor(henry).

yes

?- bachelor(tom).

no

?- bachelor(Who).

Who= henry ;

no

?- not(married(Who)).

Ejemplo

bachelor(P) :- male(P), not(married(P)).

male(henry).

male(tom).

married(tom).

?- bachelor(henry).

yes

?- bachelor(tom).

no

?- bachelor(Who).

Who= henry ;

no

?- not(married(Who)).

no.