Introduction to Artificial Intelligence

Department of Computer Science

Prolog
prefix([],_).
prefix([H|T],[H|L]) :- prefix(T,L).

| ?- prefix(X,[a,b,c]).
X = [] ;
X = [a] ;
X = [a,b] ;
X = [a,b,c] ;
no

yes
| ?- prefix([a,b],[a,b,c]).
yes
| ?- prefix([a | X],[a, b ,c]).

X = [] ;
X = [b] ;
X = [b,c] ;
no
| ?-
suffix(X,X).
suffix(L,[_|T]) :- suffix(L,T).

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

X = [a,b,c] ;

X = [b,c] ;

X = [c] ;

X = [] ;

no
Sublist

sublist(L1,L2) :- suffix(L1,M),prefix(M,L2).

?- sublist(X,[a,b,c]).
X = [] ;
X = [a] ;
X = [a,b] ;
X = [a,b,c] ;
X = [] ;
X = [b] ;
X = [b,c] ;
X = [] ;
X = [c] ;
X = [] ;

...Infinite Loop...
Negation by Failure

The goal \texttt{\(+\)(G)} succeeds whenever the goal \texttt{G} fails.

\begin{verbatim}
| ?- member(b,\[a,b,c\]).
  yes
| ?- \+(member(b,\[a,b,c\))).
  no
\end{verbatim}
overlap(S1, S2) :- member(X, S1), member(X, S2).

disjoint(S1, S2) :- \+(overlap(S1, S2)).

| ?- overlap([a, b, c], [c, d, e]).  
  yes  
| ?- overlap([a, b, c], [d, e, f]).  
  no  
| ?- disjoint([a, b, c], [c, d, e]).  
  no  
| ?- disjoint([a, b, c], [d, e, f]).  
  yes  
| ?- disjoint([a, b, c], X).  
  no  %<---------Not what we wanted???

Proper use of Negation by Failure

Negation by failure \(\neg+(G)\) works properly only in the following cases:

- When \(G\) is fully instantiated at the time PROLOG processes the goal \(\neg+(G)\). (In this case, we interpret \(\neg+(G)\) to mean “goal \(G\) does not succeed”.)

- When all variables in \(G\) are unique to \(G\), i.e., they don’t appear elsewhere in the same clause. (In this case, we interpret \(\neg+(G(X))\) to mean “There is no value of \(X\) that will make \(G(X)\) succeed”.)
Arithmetic in Prolog

| ?- AGE is 1998 - 1967.
AGE = 31 ;
| ?- DATE is 1967 + 31.
DATE = 1998 ;
| ?- 31 is DATE - 1967.
! Instantiation error in argument 2 of is/2
! goal: 31 is _6668-1967
| ?- 1998 is 1967 + AGE.
! Instantiation error in argument 2 of is/2
! goal: 1998 is 1967+_6728
At the time PROLOG begins processing a goal of the form: \( X \text{ is } <\text{Exp}> \), \(<\text{Exp}>\) must be a fully instantiated arithmetic expression, i.e., it cannot have any unbound variables.
Adding up the Numbers on a List

sumlist([],0).

sumlist([X|Y],S) :- sumlist(Y,T),
    S is X + T.

| ?- sumlist([1,2,3],X).
X = 6

| ?- sumlist([],X).
X = 0

| ?- sumlist(X,3).
! Instantiation error in argument 2 of is/2
! goal:  3 is _6786+0
Computing the Length of a List

length([],0).

length([_|Y],L) :- length(Y,M),
               L is M + 1.

| ?- length([a,b,c],L).
  L = 3

| ?- length([],L).
  L = 0

| ?- length(X,3).
  X = [_6755,_6757,_6759]

| ?- length(X,0).
  X = []
bounded-list(LIST,BOUND)
    :- BOUND > 0,
       LOWER is BOUND - 1,
       bounded-list(LIST,LOWER).

bounded-list(LIST,BOUND)
    :- length(LIST,BOUND).

| ?- bounded-list(L,3).

L = [] ;
L = [_6778] ;
L = [_6775,_6777] ;
L = [_6772,_6774,_6776] ;

no
Bounded Depth First Search

flight(ny,chicago). flight(chicago,ny).
flight(ny,miami). flight(miami,ny).
flight(miami,austin). flight(austin,miami).
flight(chicago,la). flight(la,chicago).
flight(la,austin). flight(austin,la).

journey(X,Y,Path,Bound)
   :- Bound > 0,  
       Lower is Bound - 1,  
       journey(X,Y,Path,Lower).

journey(X,Y,Path,Bound)
   :- length(Path,Bound),trip(X,Y,Path).

trip(X,Z,[X,Z]) :- flight(X,Z).
trip(X,Z,[X|Path])
   :- flight(X,Y),trip(Y,Z,Path),  
      +(member(X,Path)).
Bounded Depth First Search

?- journey(ny,austin,P,1).
no
?- journey(ny,austin,P,2).
no
?- journey(ny,austin,P,3).
P = [ny,miami,austin] ;
no
?- journey(ny,austin,P,4).
P = [ny,miami,austin] ;
P = [ny,chicago,la,austin] ;
no
Problems with Multiple Solutions

isa-mother(X) :- female(X),parent(X,_).
isa-father(X) :- male(X),parent(X,_).

parent(fred,sue).
parent(janet,sue).
parent(fred,tim).
parent(janet,tim).

male(fred).
female(janet).

| ?- isa-mother(X).

X = janet ;
X = janet ;
no
• The cut symbol ! indicates a goal that always succeeds.

• Cut has the side effect of suppressing backtracking.

• PROLOG will not attempt to find additional solutions to any goals to the left of the cut.
Surpressing Multiple Solutions with Cut

\[
\text{isa-mother}(X) :- \text{female}(X),\text{parent}(X,\_),!.
\text{isa-father}(X) :- \text{male}(X),\text{parent}(X,\_),!.
\]

\[
\begin{align*}
\text{parent}(\text{fred},\text{sue}). \\
\text{parent}(\text{janet},\text{sue}). \\
\text{parent}(\text{fred},\text{tim}). \\
\text{parent}(\text{janet},\text{tim}). \\
\text{male}(\text{fred}). \\
\text{female}(\text{janet}).
\end{align*}
\]

\[
\text{?- isa-mother}(X).
\]

\[
X = \text{janet} ;
\]

\[
\text{no}
\]