```prolog
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% macro magic (not actually used)
:- op(1001,xfy,ors).

ors2L(X ors Y,[X|T]) :- !, ors2L(Y,T).
ors2L(X,[X]).

goal_expansion(X ors Y,or(Out)) :- ors2L(X  ors Y,Out).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% knowledge base
happy/t        :- rich/t,healthy/t.
happy/t        :- tranquility /hi.
tranquility/hi :- conscience/clear.
tranquility/hi :- satiated/t.
satiated/t     :- diet/fatty.
healthy/t      :- diet/light.
rich/t.

todo1/t        :- happy/t.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% misc support stuff
b/X :- X-Y, c/Y,d/X.
b/X :- X+Y, c/Y,a/Y, e/Y.

 1 +  1.
-1 + -1.
 1 - -1.
-1 -  1.

calls( _ + _ ).
calls( _ - _ ).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% main control
% working memory
go    :- reset, run,    report.
go(X) :- reset, run(X), report.

% zap past assumptions
reset  :- retractall(assumption(_,_,_)).

% call the maybe interpreter
run    :- maybe(todo/t).
run(X) :- maybe(X).

% show assumptions
report :- write('% assumptions\n'), report1, fail.
report :- nl.

report1 :-
        setof(Z*(X/Y),assumption(X,Y,Z),All),
        member(One,All),
        write(One),
        nl.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% the maybe interpreter
% allow any conclusion as long as there is no
% evidence against it.
maybe(X) :- once(maybe0(X,Y)), maybe1(Y).

maybe0(true,       true).
maybe0((X,Y),      (X,Y)).
maybe0((X;Y),      (X;Y)).
maybe0(X,          calls(X))       :- calls(X).
maybe0(X/Y,        fact(X/Y))      :- clause(X /_,true).
maybe0(X/Y,        rule(X/Y))      :- clause(X/_,_).
maybe0(X/Y,        abducible(X/Y)).
maybe0(or([H|T]),  or([H|T])).
maybe0(or([]),     true).

maybe1(true).
maybe1((X,Y))        :- maybe(X),maybe(Y).
maybe1((X;Y))        :- maybe(X);maybe(Y).
maybe1(abducible(X)) :- assume(X,0).
maybe1(fact(X))      :- assume(X,1), one(X).
maybe1(rule(X))      :- assume(X,2), one(clause(X,Y)), maybe(Y).
maybe1(calls(X))     :- one(clause(X,Y)), maybe(Y).
maybe1(or([H|T]))    :- any(One,Rest,[H|T]), ignore(maybe(One)), maybe(or(Rest)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% assumption management

% if seen before, then new value must agree with old
assume(X/Y,_)   :- assumption(X,Z,_),!,Y=Z.
% otherwise, we can tentatively assert it
assume(X/Y,How) :- bassert(assumption(X,Y,How)).

% evil backtrackable assert
bassert(X) :- assert(X).
bassert(X) :- retract(X),fail.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% lower-level utilities.
one(X) :- bagof(X,X,All), any(X,_,All).

any(X,L) :- any(X,_,L).

any(X,Rest,L) :-
        length(L,N),
        any(L,N,X,Rest).

any([H],_,H,[])   :- !.
any([H|T],N,X,Rest) :-
        Pos is random(N) + 1,
        lessN(Pos,Y,[H|T],L),
        ( X=Y,
          Rest=L
        ; Rest=[Y|Rest1],
          N1 is N - 1,
          any(L,N1,X,Rest1)).

lessN(1,H,[H|T],T) :- !.
lessN(N0,X,[H|T],[H|L]) :-  N is N0 - 1, lessN(N,X,T,L).

/************************************
?- go(happy/t).
% assumptions
0* (diet/fatty)
2* (happy/t)
2* (satiated/t)
2* (tranquility/hi)

true ;
% assumptions
0* (conscience/clear)
2* (happy/t)
2* (tranquility/hi)

true ;
% assumptions
0* (diet/light)
1* (rich/t)
2* (happy/t)
2* (healthy/t)
************************************/
```