PROBLEM REDUCTION

Contrast State-Space search

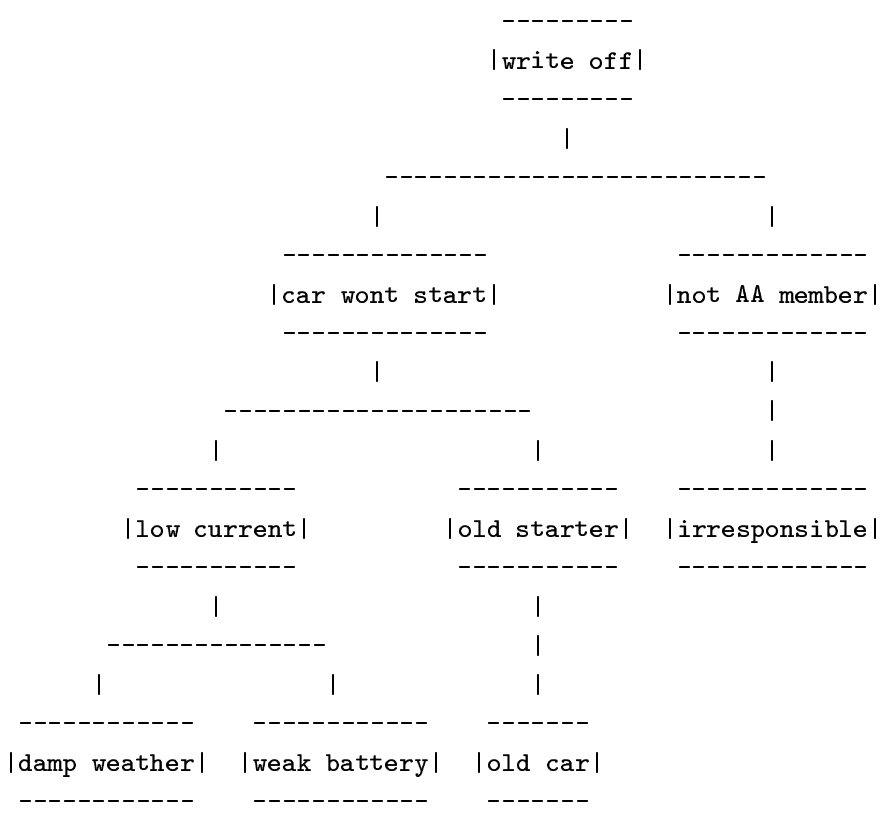
Problem represented as goals to be solved

either immediately solve a goal as a fact

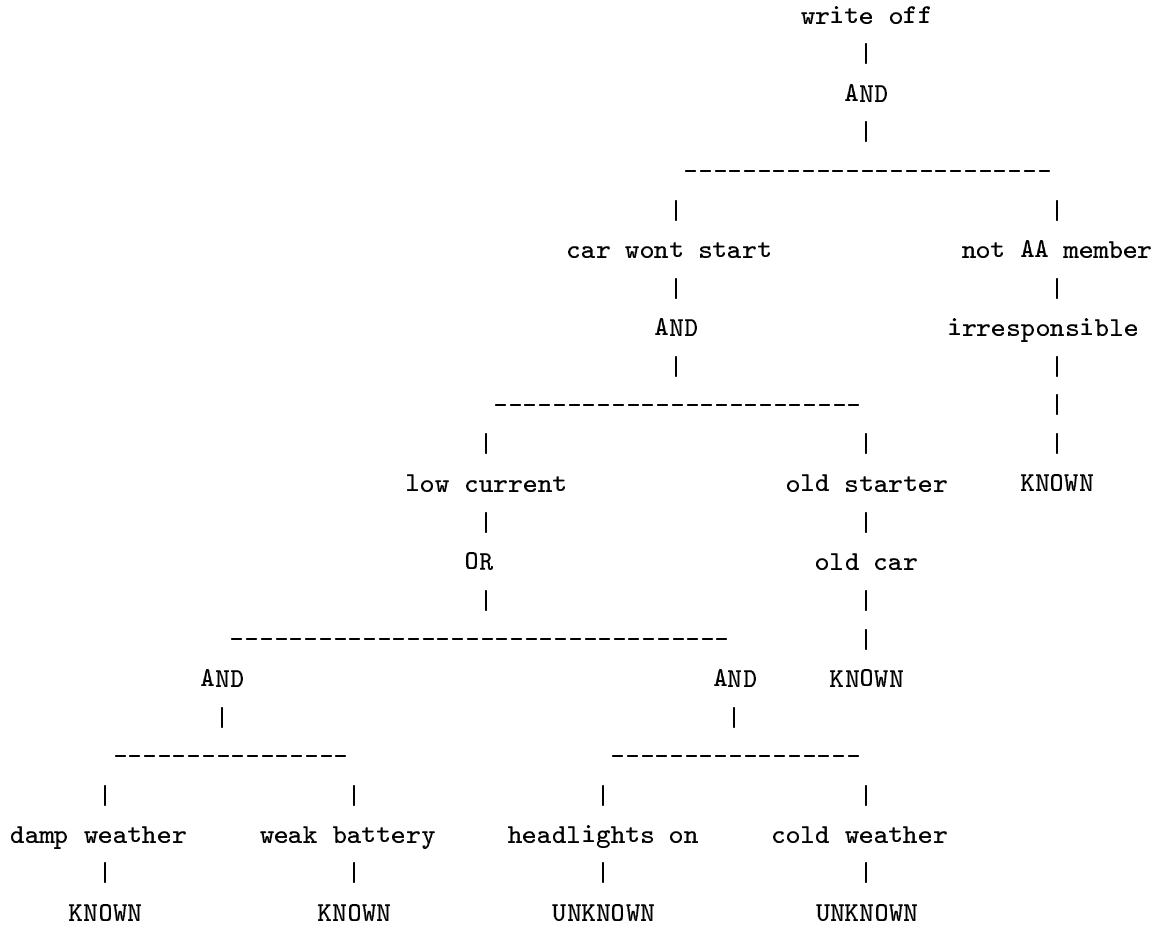
or reduce to subgoals via rules

```
irresponsible.
weak battery.
damp weather.
old car.
low current <- damp weather, weak battery.
low current <- headlights on, cold weather.
old starter <- old car.
car wont start <- low current, old starter.
write off <- car wont start, not AA member.
not AA member <- irresponsible.
```

Proof Tree for "Write Off"



Search Space for "Write Off"



Facts and Rules

```
[[[irresponsible]]
 [[weak battery]]
 [[damp weather]]
 [[old car]]

 [[low current] [damp weather] [weak battery]]
 [[low current] [headlights on] [cold weather]]
 [[old starter] [old car]]
 [[car wont start] [low current] [old starter]]
 [[write off] [car wont start] [not AA member]]
 [[not AA member] [irresponsible]]
] -> rulebase5;
```

Backwards Search

```
define backwards_search1(goals) -> boolean;
  vars goal subgoals other_goals;
  if goals = [] then true -> boolean
  else goals --> [?goal ??other_goals];
       foreach [^goal ??subgoals] in rulebase do
            backwards_search1(subgoals) -> boolean;
            if boolean
            then backwards_search1(other_goals) -> boolean;
                 if boolean then return endif
            endif
       endforeach;
       false -> boolean;
  endif;
enddefine;
```

Backwards Search

```
: backwards_search1([[write off]]) =>
> backwards_search1 [[write off]]
!> backwards_search1 [[car wont start] [not AA member]]
!!> backwards_search1 [[low current] [old starter]]
!!!> backwards_search1 [[damp weather] [weak battery]]
!!!!> backwards_search1 []
!!!!< backwards_search1 <true>
!!!!> backwards_search1 [[weak battery]]
!!!!!> backwards_search1 []
!!!!!< backwards_search1 <true>
!!!!!> backwards_search1 []
!!!!!< backwards_search1 <true>
!!!!< backwards_search1 <true>
!!!< backwards_search1 <true>
!!!> backwards_search1 [[old starter]]
!!!!> backwards_search1 [[old car]]
!!!!!> backwards_search1 []
!!!!!< backwards_search1 <true>
!!!!!> backwards_search1 []
!!!!!< backwards_search1 <true>
!!!!< backwards_search1 <true>
!!!!> backwards_search1 []
!!!!< backwards_search1 <true>
!!!< backwards_search1 <true>
!!< backwards_search1 <true>
!!> backwards_search1 [[not AA member]]
!!!> backwards_search1 [[irresponsible]]
!!!!> backwards_search1 []
!!!!< backwards_search1 <true>
```

```
!!!!> backwards_search1 []
!!!!< backwards_search1 <true>
!!!< backwards_search1 <true>
!!!> backwards_search1 []
!!!< backwards_search1 <true>
!!< backwards_search1 <true>
!< backwards_search1 <true>
!> backwards_search1 []
!< backwards_search1 <true>
< backwards_search1 <true>
** <true>
```

Producing a Proof Tree

```
: proof_tree([write off], rulebase5, 10) ==>
** [[write off]
    [[car wont start]
     [[low current] [[damp weather]] [[weak battery]]]
     [[old starter] [[old car]]]]
    [[not AA member] [[irresponsible]]]]
```

Producing a Proof Tree

```
/* proof_tree takes a single goal as input and returns its
   proof tree or false if there is none. */

define proof_tree(goal, rulebase, depth) -> tree;
  vars database;
  rulebase -> database;
  backwards_search_tree([^goal], depth) -> tree;
  if islist(tree) then hd(tree) -> tree endif
enddefine;
```

Producing a Proof Tree

```
/* backwards_search_tree takes a list of goals as input and returns
  the solution AND tree if it exists, and otherwise false. */

define backwards_search_tree(goals, depth) -> tree;
   vars goal first_tree other_trees goal subgoals other_goals;
   if goals = [] then [] -> tree
   elseif depth <= 0 then false -> tree
   else goals --> [?goal ??other_goals];
       foreach [^goal ??subgoals] do
           backwards_search_tree(subgoals, depth-1) -> first_tree;
           if islist(first_tree)
           then unless present([^goal]) then add([^goal]) endunless;
               backwards_search_tree(other_goals, depth) -> other_trees;
               if islist(other_trees)
               then [[^goal ^^first_tree] ^^other_trees] -> tree; return
               endif
           endif
       endforeach;
       false -> tree;
   endif
enddefine;
```