```
% FILE: hps.pro
% TYPE: Prolog source
% LINE: Heuristic problem solver for Crypto problems
% DATE: Fall 2011

%-------------------------------------------------------------------------------
% HEURISTIC CRYPTO PROBLEM SOLVER

%-------------------------------------------------------------------------------
% load some files
:- consult('gv.pro').
:- consult('combosets.pro').

%-------------------------------------------------------------------------------
%  RANDOM CRYPTO PROBLEM GENERATION

establishCryptoProblemParameters :-
  declare(lo,0),
  declare(hi,9).

generateRandomCryptoNumber(R) :-
  valueOf(lo,Lo),
  valueOf(hi,Hi),
  Hip is Hi + 1,
  random(Lo,Hip,R).

generateRandomCryptoProblem :-
  generateRandomCryptoNumber(N1),
  generateRandomCryptoNumber(N2),
  generateRandomCryptoNumber(N3),
  generateRandomCryptoNumber(N4),
  generateRandomCryptoNumber(N5),
  generateRandomCryptoNumber(G),
  addCryptoProblemToKnowledgeBase(N1,N2,N3,N4,N5,G).

addCryptoProblemToKnowledgeBase(N1,N2,N3,N4,N5,G) :-
  retract(problem(_,_)),
  assert(problem(numbers(N1,N2,N3,N4,N5),goal(G))).
addCryptoProblemToKnowledgeBase(N1,N2,N3,N4,N5,G) :-
  assert(problem(numbers(N1,N2,N3,N4,N5),goal(G))).

%-------------------------------------------------------------------------------
% display the problem -- assuming that it has been internalized

displayProblem :-
  displayProblem_nnl, nl.

displayProblem_nnl :-
  problem(numbers(N1,N2,N3,N4,N5),goal(G)),
  write('Problem: numbers = {'),
  write(N1),write(','),
  write(N2),write(','),
  write(N3),write(','),
  write(N4),write(','),
  write(N5),write('} and goal = '),
  write(G),write(' ').

%-------------------------------------------------------------------------------
% internalize the problem

internalizeProblem :-
  problem(numbers(N1,N2,N3,N4,N5),goal(G)),
  eraseProblemBindings,
  eraseProblem,
  eraseSolution,
  assert(problem(numbers(N1,N2,N3,N4,N5),goal(G))),
  assert(binding(n1,N1)),
  assert(binding(n2,N2)),
  assert(binding(n3,N3)),
  assert(binding(n4,N4)),
```

```
  assert(binding(n5,N5)),
  assert(binding(g,G)).

eraseProblem :-
  retract(problem(_,_)),
  fail.
eraseProblem.

eraseSolution :-
  retract(solution(_)),
  fail.
eraseSolution.

eraseProblemBindings :-
  retract(binding(n1,_)),
  retract(binding(n2,_)),
  retract(binding(n3,_)),
  retract(binding(n4,_)),
  retract(binding(n5,_)),
  retract(binding(g,_)),
  fail.
eraseProblemBindings.

%------------------------------------------------------------------------------
% solve the problem heuristically -- assuming internalization

rule(1,situation1,action1).
rule(2,situation2,action2).
rule(3,situation3,action3).
rule(4,situation4,action4).
rule(5,situation5,action5).
rule(6,situation6,action6).
rule(7,situation7,action7).
rule(8,situation8,action8).

solveProblemHeuristically :-
  rule(Number,Situation,Action),
  write('considering rule '),write(Number),write(' ...'),nl,
  Situation,
  write('application of rule '),write(Number),write(' produces '),
  Action.
solveProblemHeuristically.

%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% heuristic 1

situation1 :-
  problem(Numbers,Goal),
  Goal = goal(0),
  Numbers = numbers(N1,N2,N3,N4,N5),
  member(0,[N1,N2,N3,N4,N5]).

action1 :-
  problem(Numbers,_),
  Numbers = numbers(N1,N2,N3,N4,N5),
  assert(solution(ex(N1,*,ex(N2,*,ex(N3,*,ex(N4,*,N5)))))).

%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% heuristic 2

situation2 :-
  problem(numbers(N1,N2,N3,N4,N5),goal(G)),
  member(G,[N1,N2,N3,N4,N5]),
  member(0,[N1,N2,N3,N4,N5]),
  not(G=0).

action2 :-
  problem(_,goal(G)),
  other_numbers(special(G),others(A,B,C,D)),
  assert(solution(ex(G,+,ex(A,*,ex(B,*,ex(C,*,D)))))).
```

```
%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% heuristic 3

situation3 :-
  problem(_,goal(0)),
  doubleton.

action3 :-
  doubleton(doubleton(A,B),rest(C,D,E)),
  assert(solution(ex(ex(A,-,B),*,ex(C,*,ex(D,*,E))))).

%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% heuristic 4

situation4 :-
  problem(_,goal(1)),
  other_numbers(special(0),others(N1,N2,N3,N4)),
  neighbors([N1,N2,N3,N4],neighbors(_,_),others(_,_)).

action4 :-
  problem(_,goal(1)),
  other_numbers(special(0),others(N1,N2,N3,N4)),
  neighbors([N1,N2,N3,N4],neighbors(A,B),others(C,D)),
  A is B+1,
  assert(solution(ex(ex(A,-,B),+,ex(0,*,ex(C,*,D))))).
action4 :-
  problem(_,goal(1)),
  other_numbers(special(0),others(N1,N2,N3,N4)),
  neighbors([N1,N2,N3,N4],neighbors(A,B),others(C,D)),
  B is A+1,
  assert(solution(ex(ex(B,-,A),+,ex(0,*,ex(C,*,D))))).
%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% heuristic 5

situation5 :-
  problem(numbers(N1,N2,N3,N4,N5),goal(G)),
  not(G = 0),
  doubleton(doubleton(A,_),_),
  not(A = G),
  not(member(0,[N1,N2,N3,N4,N5],_)),
  member(G,[N1,N2,N3,N4,N5],_).

action5 :-
  problem(_,goal(G)),
  doubleton(doubleton(A,B),rest(G,C,D)),
  assert(solution(ex(G,+,ex(ex(A,-,B),*,ex(C,*,D))))).
action5 :-
  problem(_,goal(G)),
  doubleton(doubleton(A,B),rest(C,G,D)),
  assert(solution(ex(G,+,ex(ex(A,-,B),*,ex(C,*,D))))).
action5 :-
  problem(_,goal(G)),
  doubleton(doubleton(A,B),rest(C,D,G)),
  assert(solution(ex(G,+,ex(ex(A,-,B),*,ex(C,*,D))))).

%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% heuristic 6

situation6 :-
  problem(numbers(N1,N2,N3,N4,N5),goal(G)),
  G = 1,
  doubleton(doubleton(A,_),_),
  not(A = 0),
  member(0,[N1,N2,N3,N4,N5]).

action6 :-
  doubleton(doubleton(A,B),rest(C,D,E)),
  assert(solution(ex(ex(A,/,B),+,ex(C,*,ex(D,*,E))))).
```

```
%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% heuristic 7

situation7 :-
  problem(numbers(N1,N2,N3,N4,N5),goal(G)),
  G = 2,
  doubleton(doubleton(A,_),_),
  A = 1,
  member(0,[N1,N2,N3,N4,N5]).

action7 :-
  doubleton(doubleton(A,B),rest(C,D,E)),
  A = 1,
  assert(solution(ex(ex(A,+,B),+,ex(C,*,ex(D,*,E))))).

%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% heuristic 8

situation8 :-
  problem(numbers(N1,N2,N3,N4,N5),goal(G)),
  G = 4,
  doubleton(doubleton(A,_),_),
  A = 2,
  member(0,[N1,N2,N3,N4,N5]).

action8 :-
  doubleton(doubleton(A,B),rest(C,D,E)),
  A = 2,
  assert(solution(ex(ex(A,+,B),+,ex(C,*,ex(D,*,E))))).

%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% heuristic support

member(X,[X|R],R).
member(X,[Y|R],Result) :-
  member(X,R,Subresult),
  Result = [Y|Subresult].

neighbor(A,B) :- A is B-1.
neighbor(A,B) :- A is B+1.

neighbor(G,[N|R],N,R) :-
  neighbor(N,G).
neighbor(G,[X|R],Neighbor,Rest) :-
  neighbor(G,R,Neighbor,More),
  Rest = [X|More].

neighbors([N1,N2,N3,N4],neighbors(N1,N2),others(N3,N4)) :- neighbor(N1,N2).
neighbors([N1,N2,N3,N4],neighbors(N1,N3),others(N2,N4)) :- neighbor(N1,N3).
neighbors([N1,N2,N3,N4],neighbors(N1,N4),others(N3,N2)) :- neighbor(N1,N4).
neighbors([N1,N2,N3,N4],neighbors(N2,N3),others(N1,N4)) :- neighbor(N2,N3).
neighbors([N1,N2,N3,N4],neighbors(N2,N4),others(N1,N3)) :- neighbor(N2,N4).
neighbors([N1,N2,N3,N4],neighbors(N3,N4),others(N1,N2)) :- neighbor(N3,N4).

doubleton :-
  problem(numbers(N1,N2,N3,N4,N5),_),
  combos(set(N1,N2,N3,N4,N5),combo(A,B),_),
  A=B.

doubleton(doubleton(A,B),rest(C,D,E)) :-
  problem(numbers(N1,N2,N3,N4,N5),_),
  combos(set(N1,N2,N3,N4,N5),combo(A,B),extras(C,D,E)),
  A=B.

can_make_goal_with_two_numbers(the_two(A,B),the_rest(C,D,E),the_goal(G)) :-
  problem(numbers(N1,N2,N3,N4,N5),goal(G)),
  combos(set(N1,N2,N3,N4,N5),combo(A,B),extras(C,D,E)),
  crypto(A,B,G,_).

other_numbers(special(N),others(N2,N3,N4,N5)) :-
```

```
  problem(numbers(N,N2,N3,N4,N5),goal(_)).
other_numbers(special(N),others(N1,N3,N4,N5)) :-
  problem(numbers(N1,N,N3,N4,N5),goal(_)).
other_numbers(special(N),others(N1,N2,N4,N5)) :-
  problem(numbers(N1,N2,N,N4,N5),goal(_)).
other_numbers(special(N),others(N1,N2,N3,N5)) :-
  problem(numbers(N1,N2,N3,N,N5),goal(_)).
other_numbers(special(N),others(N1,N2,N3,N4)) :-
  problem(numbers(N1,N2,N3,N4,N),goal(_)).

crypto(N1,N2,Goal,ex(N1,+,N2)) :- Goal is (N1+N2).
crypto(N1,N2,Goal,ex(N1,*,N2)) :- Goal is (N1*N2).
crypto(N1,N2,Goal,ex(N1,-,N2)) :- Goal is (N1-N2).
crypto(N1,N2,Goal,ex(N2,-,N1)) :- Goal is (N2-N1).
crypto(N1,N2,Goal,ex(N1,/,N2)) :- N2>0, Goal is (N1/N2).
crypto(N1,N2,Goal,ex(N2,/,N1)) :- N1>0, Goal is (N2/N1).

%-------------------------------------------------------------------------------
% key substitution code

substitute(New,Old,ex(Old,O,Z),ex(New,O,Z)).
substitute(New,Old,ex(X,O,Old),ex(X,O,New)).
substitute(New,Old,ex(X,O,Z),ex(Q,O,Z)) :-
  substitute(New,Old,X,Q).
substitute(New,Old,ex(X,O,Z),ex(X,O,Q)) :-
  substitute(New,Old,Z,Q).

%-------------------------------------------------------------------------------
% display the solution -- assuming that it has been solved

displaySolution :-
  solution(S),
  displayResult(S),
  nl.
displaySolution.

displayResult(ex(A,O,B)) :-
  number(A), number(B),
  write('( '), write(A), write(' '), write(O), write(' '), write(B), write(' )').
displayResult(ex(A,O,B)) :-
  number(A), B = ex(A1,O1,B1),
  write('( '), write(A), write(' '), write(O), write(' '),
  displayResult(ex(A1,O1,B1)), write(' )').
displayResult(ex(A,O,B)) :-
  number(B), A = ex(A1,O1,B1),
  write('( '), displayResult(ex(A1,O1,B1)), write(' '), write(O), write(' '),
  write(A), write(' )').
displayResult(ex(A,O,B)) :-
  A = ex(A1,O1,B1), B = ex(A2,O2,B2),
  write('( '), displayResult(ex(A1,O1,B1)), write(' '), write(O), write(' '),
  displayResult(ex(A2,O2,B2)), write(' )').

%-------------------------------------------------------------------------------
% probabilities of applicability of heuristics

%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% atomic variant

probability1(Situation,NrTrials,Purpose,Probability) :-
  declare(successes,0),
  repeat(NrTrials,trial1(Situation,Purpose)),
  valueOf(successes,Successes),
  Probability is Successes / NrTrials.

trial1(Situation,Purpose) :-
  retract_problem,
  generateRandomCryptoProblem,
  internalizeProblem,
  displayProblem_nnl(Purpose),
  Situation,
```

```
  displayApplicability(Purpose,yes),
  inc(successes).
trial1(_,Purpose) :-
  displayApplicability(Purpose,no).

%- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
% list variant

probability(Situation,NrTrials,Purpose,Probability) :-
  declare(successes,0),
  repeat(NrTrials,trial(Situation,Purpose)),
  valueOf(successes,Successes),
  Probability is Successes / NrTrials.

trial(SL,Purpose) :-
  retract_problem,
  generateRandomCryptoProblem,
  internalizeProblem,
  displayProblem_nnl(Purpose),
  trial_proper(SL,Purpose).

trial_proper([],Purpose) :-
  displayApplicability(Purpose,no).
trial_proper([H|_],Purpose) :-
  H,
  displayApplicability(Purpose,yes),
  inc(successes).
trial_proper([_|T],Purpose) :-
  trial_proper(T,Purpose).

retract_problem :-
  retract(problem(_,_)),fail.
retract_problem.

displayProblem_nnl(testing) :-
  displayProblem_nnl.
displayProblem_nnl(_).

displayApplicability(testing,yes) :-
  write('*'), nl.
displayApplicability(testing,no) :-
  write(' '), nl.
displayApplicability(_,_).

%-----------------------------------------------------------------------------
% demo - generate and solve a random crypto problem

demo :-
  generateRandomCryptoProblem,
  internalizeProblem,
  displayProblem,
  solveProblemHeuristically,
  displaySolution.

demo(0).
demo(N) :-
  demo,
  K is N-1,
  demo(K).

%-----------------------------------------------------------------------------
% establish a particular crypto problem

establishCryptoProblem(numbers(N1,N2,N3,N4,N5),goal(G)) :-
  addCryptoProblemToKnowledgeBase(N1,N2,N3,N4,N5,G).

%-----------------------------------------------------------------------------
% crypto problem solver

solve(numbers(N1,N2,N3,N4,N5),goal(G)) :-
```

```
  establishCryptoProblem(numbers(N1,N2,N3,N4,N5),goal(G)),
  internalizeProblem,
  displayProblem,
  solveProblemHeuristically,
  displaySolution.

%------------------------------------------------------------------------------
% iterator

repeat(0,_).
repeat(N,P) :-
  P,
  K is N-1,
  repeat(K,P).

%------------------------------------------------------------------------------
% initilization

:-establishCryptoProblemParameters.
```