|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  | %----------2016 Q1)---------- |
|  |  |
|  | %1b) |
|  | isSet([]). |
|  | isSet([H|T]):- \+member(H,T), isSet(T). |
|  |  |
|  | %1c) |
|  | moreThanOne([A,B|\_]):- A\=B. |
|  | moreThanOne([\_,B|C]):- moreThanOne([B|C]). |
|  |  |
|  | %1d) |
|  | moreThan(List,N):- countDistincts([],List,Y), N>=0, Y > N. |
|  |  |
|  | countDistincts(\_,[],0). |
|  | countDistincts(Distinct,[H|T],N+1):- \+member(H,Distinct), countDistincts([H|Distinct],T,N). |
|  | countDistincts(Distinct,[\_|T],N):- countDistincts(Distinct,T,N).  % David: The above countDistincts produces lots of values for N.  countDistincts(\_,[],0).  countDistincts(Distinct,[H|T],M):- \+member(H,Distinct), countDistincts([H|Distinct],T,N), M is N + 1.  countDistincts(Distinct,[H|T],N):- member(H,Distinct), countDistincts(Distinct,T,N). |
|  |  |
|  |  |
|  |  |
|  | %----------2016 Q2)---------- |
|  |  |
|  | %2ai) |
|  | |  | | --- | | fac(0,1). | | fac(N,X):- N>0, N2 is N-1, fac(N2,Y), X is N\*Y. | |
|  | |  | | --- | | fac(0,1). | | fac(N,X):- N>0, N2 is N-1, fac(N2,Y), X is N\*Y. | |
|  |  |
|  | %2aii) |
|  | facTail(0,1). |
|  | facTail(N,Fac):- integer(N), N>0, facTail(N,1,Fac). |
|  |  |
|  | facTail(0,N,N). |
|  | facTail(N,X,Fac):- N>0, N2 is N-1, X2 is N\*X, facTail(N2,X2,Fac). |
|  |  |
|  | %2bi) |
|  | fib(0,0). |
|  | fib(1,1). |
|  | fib(N,Fib):- N>1, X is N-1, Y is N-2, fib(X,F1), fib(Y,F2), Fib is F1 + F2. |
|  |  |
|  | %2bii) |
|  | fibTail(0,0). |
|  | fibTail(N,Fib):- N>0, fibTail(N,0,1,Fib). |
|  |  |
|  | fibTail(0,X,\_,X). |
|  | fibTail(N,X,Y,Fib):- N>0, N2 is N-1, X2 is Y, Y2 is X+Y, fibTail(N2,X2,Y2,Fib). |
|  |  |
|  |  |
|  |  |
|  | %----------2016 Q3)---------- |
|  |  |
|  | %3a) |
|  | s --> a(N), b(M), c(K), {N + M =< K}. |
|  |  |
|  | a(0) --> []. |
|  | a(NN) --> [a], a(Y), {NN is Y+1}. |
|  |  |
|  | b(0) --> []. |
|  | b(NN) --> [b], b(Y), {NN is Y+1}. |
|  |  |
|  | c(0) --> []. |
|  | c(NN) --> [c], c(Y), {NN is Y+1}. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  | %---------------------------------------2015------------------------------------------ |
|  |  |
|  | %----------2015 Q1)---------- |
|  |  |
|  | %1a) |
|  | % fact, fact, rule. |
|  | % The third sentence is the best translation as it separates the encoding of the verbs, nouns and adjectives in the sentence, so more than just that one particular sentence in the form 'Mary owns every lamb that is white' is true.  % David: I would say the third sentence is best because it structures the information, allowing you to query it, as well as making it extensible for when there is more information to add. |
|  |  |
|  | %1bi) false. |
|  | %1bii) X = 3+2. |
|  | %1biii) false. |
|  | %1biv) Error: is/2 Arguments are not sufficiently instantiated. |
|  | %1bv) X = 5. |
|  | %1bvi) false. |
|  | %1bvii) true. |
|  | %1bviii) true. |
|  |  |
|  | %1c) |
|  | split(\_,[],[],[]). |
|  | split(N,[H|T],[H|Small],Big):- H<N, split(N,T,Small,Big). |
|  | split(N,[H|T],Small,[H|Big]):- H>=N, split(N,T,Small,Big). |
|  |  |
|  | %1d) |
|  | sumOfPowers(N,Sop):- N>=0, sumP(N,0,Sop). |
|  |  |
|  | sumP(0,P,P). |
|  | sumP(N,P,SoP):- N>0, N2 is N-1, P2 is (N\*\*N)+P, sumP(N2,P2,SoP). |
|  |  |
|  |  |
|  |  |
|  | %----------2015 Q2)---------- |
|  |  |
|  | %2a) |
|  | member(X,[X|\_]). |
|  | member(X,[\_|T]):- member(X,T). |
|  |  |
|  | %2b) |
|  | % Red, as the outcome of the program is changed by the cut. |
|  | %correction?  GREEN CUT - without cut it’s just member (as defined above) - only improves efficiency |
|  | %2c) |
|  | last(X,[X]). |
|  | last(X,[\_|T]):- last(X,T). |
|  |  |
|  | %2d) |
|  | multiple(X,List):- countOccurrences(X,0,List,Y), Y > 1. |
|  |  |
|  | countOccurrences(\_,N,[],N). |
|  | countOccurrences(X,N,[X|T],Y):- countOccurrences(X,N+1,T,Y). |
|  | countOccurrences(X,N,[\_|T],Y):- countOccurrences(X,N,T,Y). |
|  |  |
|  | %2e) |
|  | next(A,B,[A,B|\_]). |
|  | next(A,B,[\_|T]):- next(A,B,T). |
|  |  |
|  | %2f) |
|  | first(X,[X|\_]). |
|  | memb(X1,X2,X3,L1,L2,L3):- member(X1,L1), member(X2,L2), member(X3,L3). |
|  |  |
|  | next3(A1,A2,A3,A1,A2,B3,L1,L2,L3):- memb(A1,A2,A3,L1,L2,L3), next(A3,B3,L3). |
|  | next3(A1,A2,A3,A1,B2,B3,L1,L2,L3):- memb(A1,A2,A3,L1,L2,L3), last(A3,L3), next(A2,B2,L2), first(B3,L3). |
|  | next3(A1,A2,A3,B1,B2,B3,L1,L2,L3):- memb(A1,A2,A3,L1,L2,L3), last(A2,L2), last(A3,L3), next(A1,B1,L1), first(B2,L2), first(B3,L3). |
|  |  |
|  | % David’s cheat attempt  % Use findall to produce [[1,a,x],[1,a,y], ... ],  % then set the As and Bs to any 2 sequential members of that. mem3(X1,X2,X3,L1,L2,L3):-  member(X1,L1),  member(X2,L2),  member(X3,L3).  combos(L1,L2,L3,List) :-  findall([X1,X2,X3], mem3(X1,X2,X3,L1,L2,L3), List).  getcombos(A1,A2,A3,B1,B2,B3,[[A1,A2,A3],[B1,B2,B3]|\_]).  getcombos(A1,A2,A3,B1,B2,B3,[\_|T]) :- getcombos(A1,A2,A3,B1,B2,B3,T).  next3(A1,A2,A3,B1,B2,B3,L1,L2,L3) :-  combos(L1,L2,L3,List),  getcombos(A1,A2,A3,B1,B2,B3,List).  % David's good attempt - "Values" is like a stack of possible outcomes.  % getnext(+DefaultValues, +Values, ?NextValues)  % Get the next set of possible outcomes.  % e.g. [[x,y],[a],[1,2]] -> [[y],[a],[1,2]] or [[y],[a],[1,2]] -> [[x,y],[a],[2]]  getnext(\_, [[\_,H2|T]|Values], [[H2|T]|Values]).  getnext([DH|DT], [[\_]|Values], [DH|NextValues]) :-  getnext(DT, Values, NextValues).  next3(A1,A2,A3,B1,B2,B3,L1,L2,L3) :-  helper(A1,A2,A3,B1,B2,B3,[L3,L2,L1],[L3,L2,L1]).  helper(A1,A2,A3,B1,B2,B3,Values,DefaultValues) :-  Values = [[A3|\_],[A2|\_],[A1|\_]],  getnext(DefaultValues, Values, [[B3|\_],[B2|\_],[B1|\_]]).  helper(A1,A2,A3,B1,B2,B3,Values,DefaultValues) :-  getnext(DefaultValues, Values, NextValues),  helper(A1,A2,A3,B1,B2,B3,NextValues,DefaultValues). |
|  | % simple next3  next3(A,B,C,D,E,F, L1,L2,L3) :- next3(Answers, L1, L2, L3, L2, L3),  next([A,B,C],[D,E,F],Answers).  next3([], [], \_, \_, \_, \_).  next3([[A,B,C]|Tail], [A|As],[B|Bs],[C|Cs], L2, L3) :- next3(Tail, [A|As], [B|Bs], Cs, L2, L3).  next3([[A,B,C]|Tail], [A|As],[B|Bs],[C], L2, L3) :- next3(Tail, [A|As], Bs, L3, L2, L3).  next3([[A,B,C]|Tail], [A|As],[B], [C], L2, L3) :- next3(Tail, As, L2, L3, L2, L3). |
|  |  |
|  |  |
|  |  |
|  | %---------------------------------------2014------------------------------------------ |
|  |  |
|  | %----------2014 Q1)---------- |
|  |  |
|  | %1a) |
|  | % list sum is not tail recursive because it does not return what its recursive call to itself returns. |
|  |  |
|  | %1b) |
|  | listSum(List,Sum):- listSum(List,0,Sum). |
|  |  |
|  | listSum([],Sum,Sum). |
|  | listSum([H|T],X,Sum):- X2 is H+X, listSum(T,X2,Sum). |
|  |  |
|  | %1c) |
|  | listProd(List,P):- listProd(List,1,P). |
|  |  |
|  | listProd([],P,P). |
|  | listProd([H|T],X,P):- X2 is H\*X, listProd(T,X2,P). |
|  |  |
|  | %1d) |
|  | list2N(0,[]). |
|  | list2N(N,[N|L]):- N>0, N2 is N-1, list2N(N2,L). |
|  |  |
|  | %1e) |
|  | nonInc([]). |
|  | nonInc([\_]). |
|  | nonInc([H,M|T]):- H>=M, nonInc([M|T]). |
|  |  |
|  | %1f) |
|  | sumList(0,[]). |
|  | sumList(N,[H|T]):- N>0, list2N(N, L2), member(H, L2), N2 is N-H, sumList(N2, T), nonInc([H|T]). |
|  |  |
|  |  |
|  |  |
|  | %----------2014 Q2)---------- |
|  |  |
|  | %2a) |
|  | subset([],\_). |
|  | subset([H|T],L):- member(H,L), subset(T,L). |
|  |  |
|  | %2b) |
|  | setEq(L1,L2):- subset(L1,L2), subset(L2,L1). |
|  |  |
|  | %2c) |
|  | setEq2(L1,L2):- setof(X,member(X,L1),Z), setof(Y,member(Y,L2),Z). |
|  |  |
|  | %2d) |
|  | nonmember(X,L):- \+ member(X,L). |
|  |  |
|  | %2e) |
|  | setInt(L1,L2,L3):- append(L1,L2,LU), setEq(LU,L3). |
|  |  |
|  | %2f) |
|  | setInt2(L1,L2,L3):- setof(X,member(X,L1),A1), setof(Y,member(Y,L2),A2), append(A1,A2,L3). |
|  |  |
|  | memberOfBoth(X,L1,L2):- member(X,L1), member(X,L2). |
|  |  |
|  | %2g) |
|  | %(Tim Fernando verified !) |
|  | if(A,B,C):- A, !, B; C. |
|  |  |
|  | %2h) |
|  | maxHead(L1,L2):- maxHead([],L1,L2). |
|  |  |
|  | maxHead(L1,[H|T],L3):- isMax(H,T), append([H|L1],T,L3). |
|  | maxHead(L1,[H|T],L3):- append(L1,[H],X), maxHead(X,T,L3). |
|  |  |
|  | isMax(\_,[]). |
|  | isMax(X,[H|T]):- X>=H, isMax(X,T). |
|  |  |
|  |  |