```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%
% Computing the Maximal Boolean Complexity of Families of Aristotelian Diagrams
% Lorenz Demey
%
% Journal of Logic and Computation
%
% online appendix (Prolog code)
%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% maxpartition(+Fragment, +Contrarieties, -Partition):
% Partition is the largest possible partition that can be induced
% by the Aristotelian family represented by Fragment and Contrarieties

% Partition is a list of lists, with the inner lists representing conjunctions

maxpartition([X], _, [[X],[not(X)]]).
maxpartition([First|Rest], Contrarieties, Partition) :-
   maxpartition(Rest, Contrarieties, PartitionInducedByRest),
   safelyaddconjunct(First, PartitionInducedByRest, Contrarieties, Part1),
   safelyaddconjunct(not(First), PartitionInducedByRest, Contrarieties, Part2),
   append(Part1, Part2, Partition).


% safelyaddconjunct(+Formula, +List, +Contrarieties, -Result):
% Result is the list of conjunctions (i.e. inner lists) that results from
% adding Formula as a conjunct to each of the conjunctions (i.e. inner lists) of List

% this 'adding of a conjunct' is done safely,
% i.e. so as to preserve consistency:
% if adding X as a conjunct to one of the conjunctions in List results in an
inconsistency,
% then the new, inconsistent conjunction is discarded

% example:
%    ?- safelyaddconjunct(p, [[x,y,z], [_,q,_], [_,_,not(p)], [a,b,c]], [c(p,q)], Result)
%    will yield Result = [[p,x,y,z], [p,a,b,c]]

safelyaddconjunct(X, [Y], Contrarieties, []) :- inconsistency(X, Y, Contrarieties), !.
safelyaddconjunct(X, [Y], _ , [[X|Y]]).
safelyaddconjunct(X, [First|Rest], Contrarieties, Result) :-
       inconsistency(X, First, Contrarieties), !,
       safelyaddconjunct(X, Rest, Contrarieties, Result).
safelyaddconjunct(X, [First|Rest], Contrarieties, [[X|First]|IntermediateResult]) :-
       safelyaddconjunct(X, Rest, Contrarieties, IntermediateResult).
```

```
% inconsistency(+X, +Conjunction, +Contrarieties):
% the formula X is inconsistent with some conjunct in Conjunction

% this can be due to two reasons (two Aristotelian relations):
% (i) the contradictory of X is present as a conjunct in Conjunction
% (ii) some contrary of X (as specified in Contrarieties) is present as a conjunct in
Conjunction

inconsistency(X, [not(X)|_], _).
inconsistency(not(X), [X|_], _).
inconsistency(X, [First|_], Contrarieties) :-
        member(c(X,First), Contrarieties) ;
        member(c(First,X), Contrarieties).
inconsistency(X, [_|Last], Contrarieties) :- inconsistency(X, Last, Contrarieties).


%%%%%
%%%%%
%%%%%


% maxbooleancomplexity(+Fragment, +Contrarieties, -MBC):
% MBC is the maximal Boolean complexity of
% the Aristotelian family represented by Fragment and Contrarieties,

maxbooleancomplexity(Fragment, Contrarieties, MBC) :-
    maxpartition(Fragment, Contrarieties, Partition),
    length(Partition, MBC).


%%%%%
%%%%%
%%%%%


% maxpartitionwithsimple(+Fragment, +Contrarieties, -SimplifiedPartition):
% the same as maxpartition/3, but with the conjunctions in the partition simplified

maxpartitionwithsimple(Fragment, Contrarieties, SimplifiedPartition) :-
    maxpartition(Fragment, Contrarieties, Partition),
    reduceall(Partition, Contrarieties, SimplifiedPartition).


% reduceall(+Partition, +Contra, -SimplifiedPartition):
% SimplifiedPartition is the result of applying
% reduce/3 to every conjunction (i.e. inner list) in Partition

reduceall([],_,[]).
```

```
reduceall([First|Rest],Contra,[ReducedFirst|ReducedRest]) :-
    reduce(First,Contra,ReducedFirst),
    reduceall(Rest,Contra,ReducedRest).


% reduce(+Conjunction, +Contrarieties, -SimplifiedConjunction):
% SimplifiedConjunction is the result of simplifying Conjunction,
% based on Contrarieties

% base case: Contrarieties is empty

reduce(F,[],F).

% recursive cases: Contrarieties is non-empty

%  case distinction on the form of the first contrariety in Contrarieties
%  1: c(p,q)
%  2: c(p,not(q))
%  3: c(not(p),q)
%  4: c(not(p),not(q))

% case 1: c(p,q)
%
% hence:
%
% p entails not(q)
% q entails not(p)
%
% hence:
%
% p /\ not(q)   ==simplifies to==>  p
% q /\ not(p)   ==simplifies to==>  q

reduce(F, [c(X,Y)| Rest], Res) :-
     atom(X), atom(Y),
     member(X,F), member(not(Y),F), !, subtract(F, [not(Y)], Inter),
     reduce(Inter, Rest, Res).

reduce(F, [c(X,Y)| Rest], Res) :-
    atom(X), atom(Y),
     member(not(X),F), member(Y,F), !, subtract(F, [not(X)], Inter),
     reduce(Inter, Rest, Res).

reduce(F, [c(X,Y)| Rest], Res) :-
    atom(X), atom(Y),
    reduce(F, Rest, Res).

% case 2: c(p,not(q))
%
% hence:
```

```
%
% p entails not(not(q)) = q
% not(q) entails not(p)
%
% hence:
%
% p /\ q                    ==simplifies to==>  p
% not(p) /\ not(q)          ==simplifies to==>  not(q)

reduce(F, [c(X,not(Y))| Rest], Res) :-
     atom(X), atom(Y),
     member(X,F), member(Y,F), !, subtract(F, [Y], Inter),
     reduce(Inter, Rest, Res).

reduce(F, [c(X,not(Y))| Rest], Res) :-
    atom(X), atom(Y),
    member(not(X),F), member(not(Y),F), !, subtract(F, [not(X)], Inter),
     reduce(Inter, Rest, Res).

reduce(F, [c(X,not(Y))| Rest], Res) :-
    atom(X), atom(Y),
    reduce(F, Rest, Res).

% case 3: c(not(p),q)
%
% hence:
%
% not(p) entails not(q)
% q entails not(not(p)) = p
%
% hence:
%
% not(p) /\ not(q)     ==simplifies to==> not(p)
% p /\ q               ==simplifies to==> q

reduce(F, [c(not(X),Y)| Rest], Res) :-
    atom(X), atom(Y),
     member(X,F), member(Y,F), !, subtract(F, [X], Inter),
     reduce(Inter, Rest, Res).

reduce(F, [c(not(X),Y)| Rest], Res) :-
    atom(X), atom(Y),
     member(not(X),F), member(not(Y),F), !, subtract(F, [not(Y)], Inter),
     reduce(Inter, Rest, Res).

reduce(F, [c(not(X),Y)| Rest], Res) :-
    atom(X), atom(Y),
     reduce(F, Rest, Res).

% case 4: c(not(p),not(q))
```

```
%
% hence:
%
% not(p) entails not(not(q)) = q
% not(q) entails not(not(p)) = p
%
% hence:
%
% not(p) /\ q   ==simplifies to==> not(p)
% p /\ not(q)    ==simplifies to==> not(q)

reduce(F, [c(not(X),not(Y))| Rest], Res) :-
    atom(X), atom(Y),
     member(X,F), member(not(Y),F), !, subtract(F, [X], Inter),
     reduce(Inter, Rest, Res).

reduce(F, [c(not(X),not(Y))| Rest], Res) :-
    atom(X), atom(Y),
     member(not(X),F), member(Y,F), !, subtract(F, [Y], Inter),
     reduce(Inter, Rest, Res).

reduce(F, [c(not(X),not(Y))| Rest], Res) :-
    atom(X), atom(Y),
     reduce(F, Rest, Res).


%%%%%
%%%%%
%%%%%


% maxbitstrings(+Fragment, +Contrarieties, -Bitstrings):
% computes Bitstrings for all the formulas in Fragment,
% based on the maximal partition that is induced by the Aristotelian family
% represented by Fragment and Contrarieties


maxbitstrings(Fragment, Contrarieties, Bitstrings) :-
    maxpartition(Fragment, Contrarieties, Partition),
    assignallbitstrings(Fragment, Partition, Bitstrings).

assignallbitstrings([], _ ,[]).
assignallbitstrings([First|Rest], Partition, [FirstB|RestBs]) :-
    assignbitstring(First, Partition, FirstB),
    assignallbitstrings(Rest, Partition, RestBs).

assignbitstring(X, [], [X]).
assignbitstring(X, [First|Rest], [FirstBit|RestBitstring]) :-
    assignbit(X, First, FirstBit),
    assignbitstring(X, Rest, RestBitstring).
```

```
assignbit(X,List,1) :- member(X, List), !.
assignbit(_, _, 0).
```