PROOF FOUNDATIONS

(W) WEISER DEFINITION OF SLICING:
Given a program P, an slicing criterion C=<v,s> where v is a variable at statement s, and an slice S:
If P halts on input I, then the value of v at statement s each time s is executed in P is the same in P and S. If P fails to terminate normally, s may be executed more times in S than in P, but P and S compute the same values for v each time s is executed by P.

(A) DATA DEPENDENCE:
We say there exists a data dependence between two expressions when the first expression defines the value of a variable and the second one uses this value in at least one of the possible program executions without being any other expression modifying it.
NOTE: We consider that the arguments passed in a function call and the parameters of that function are a specific case of data dependence where the expression changes its name.

(B) CONTROL DEPENDENCE:
There exists a control dependence between two expressions when the second expression cannot be evaluated without evaluating the first expression.

(C) SEQUENTIAL REDUNDANCE:
When the return expression of a block or a function (the last expression of the block in Erlang) is a variable defined in the previous expression, this can be deleted avoiding the definition of this variable and returning the result of the previous expression, taking this expression the last position of the block and being returned in consecuense.

(D) SYNTAX ERRROR:
We say there exists a syntax error in a program when the removal or modification of a chosen expression transforms the program into a non-executable state.

(E) SEMANTIC MODIFICATION:
There exists a semantic modification in an expression when the modification of one of its subexpressions modifies the behaviour of the whole expression.

(F) ABSORBING PROPERTY:
A clause of a conditional or a function statement is absorbing when its guard is always evaluated to true or its pattern always matches.

(G) FULL TEST VALIDATION:
There exists full test validation when an original program and a slice extracted from it can be executed with all possible input values of the original program and the values of the slicing criterion are the same in both executions.
NOTE: We consider in this definition also programs with slicing criteria that are independent of program inputs, where there is only one possible execution.

COLOUR LEGEND

Black: Expressions deleted by executing phase 1 (iterative slicing with the selected slicers)
Red: Expressions deleted by executing phase 2 (modified ORBS algorithm)
Green: Expressions remaining in the quasi-minimal slices
Orange: Slicing Criterion

NOTE1: We will not prove wether black expressions of the program code can be deleted or not because they have been deleted by phase 1. Phase 1 produces a complete slice of the original code, so we can guarantee that these expressions are not part of the slice.
NOTE2: Our slices keep the syntax of the original program (we are not interested in amorphous slices). However, in order to make the final slice executable, some modifications of the source code are compulsory (e.g., replacing calls to deleted functions with a constant called "undef"). Therefore, we allow for some modifications of the source code to produce executable slices. The modifications made never affect the behaviour of the source code, they just ensure that the final code is a valid Erlang program.

```
%-----------------------------------------------------------------------------------------
%-----------------------------------------------------------------------------------------
%-- bench15.erl
%--
%-- AUTHORS:      Tamarit
%-- DATE:         2015
%-- PUBLISHED:    http://rosettacode.org/wiki/Combinations_and_permutations#Erlang (2016)
%-- COPYRIGHT:    Bencher: The Program Slicing Benchmark Suite for Erlang
%--               (Universitat Politècnica de València)
%--               http://www.dsic.upv.es/~jsilva/slicing/bencher/
%-- DESCRIPTION
%-- The program implements the procedure for making permutations and combinations in an
%-- interval of numbers given. The inputs are the first and last numbers of the
%-- permutation interval and the first and last number of the combination interval. The
%-- output is the result of the permutations and combinations calculated.
%-----------------------------------------------------------------------------------------
%-----------------------------------------------------------------------------------------
```

```
-module(bench15).
-export([main/4]).


perm(N, K) ->
                                                  %Given (A), N and K are necessary w.r.t.
                                                  product(lists:seq(N - K + 1, N)).
    product(lists:seq(N - K + 1, N)).             %product(lists:seq(N - K + 1, N)) cannot be deleted because
                                                  it is the only expression of the clause. Replacing it with
                                                  undef (NOTE3) would prevent to reach the SC because of a
                                                  bad argument error in the integer_to_list(N) function call
                                                  in the show_big function
                                                  %Replace lists:seq(N-K+1,N)) with undef (NOTE3) would
                                                  prevent to reach the SC because of a matching error in the
                                                  product function
                                                  %Replace N-K+1 or N with undef (NOTE3) would prevent to
                                                  reach the SC because of a matching error in the lists:seq
                                                  function call
                                                  %Replace N, K or 1 with undef (NOTE3) would prevent to
                                                  reach the SC due to a badarith error
comb(N, K) ->                                     %Given (A), N and K are necessary w.r.t. perm(N,K)
    perm(N, K) div product(lists:seq(1, K)).      %perm(N, K) div product(lists:seq(1, K)) cannot be deleted
                                                  because it is the only expression of the clause. Replace
                                                  it with undef (NOTE3) would prevent to reach the SC because
                                                  of a bad argument error in the integer_to_list(N) function
                                                  call of the show_big function
                                                  %perm(N,K) or product(lists:seq(1,K)) cannot be replaced
                                                  with undef (NOTE3) because it would prevent to reach the
                                                  SC due to a badarith error
                                                  %Given (A), N and K in perm(N,K) function call are
                                                  necessary w.r.t. the clause of the perm(N,K) function
                                                  %Replace lists:seq(1,K) with undef (NOTE3) would prevent
                                                  to reach the SC because of a matching error in the product
                                                  function
                                                  %Replace 1 or K in lists:seq(1,K) with undef (NOTE3) would
                                                  prevent to reach the SC because of a bad argument error
product(List) ->                                  %Given (A), List is necessary w.r.t.
                                                  lists:foldl(fun(N,Acc) -> N * Acc end, 1, List)
    lists:foldl(fun(N, Acc) -> N * Acc end, 1, List).  %lists:foldl(fun(N, Acc) -> N * Acc end, 1, List) cannot
                                                  be deleted because it is the only expression of the
                                                  function. Replace it with undef (NOTE3) would prevent to
                                                  reach the SC because of a badarith error in the comb(N,K)
                                                  function
                                                  %Replace fun(N,Acc)->N*Acc end, 1 or List with undef would
                                                  prevent to reach the SC because of an execution error in
                                                  the lists:foldl function
                                                  %Given (A), N and Acc in fun(N,Acc) are necessary w.r.t.
                                                  N*Acc
                                                  %Replace N * Acc with undef (NOTE3) would prevent to reach
                                                  the SC because of a badarith error in the comb(N,K)
                                                  function
                                                  %Replace N or Acc in N * Acc with undef (NOTE3) would
                                                  prevent to reach the SC because of a badarith error
main(PFrom,PTo,CFrom,CTo) ->                      %Given (A), PFrom and PTo are necessary w.r.t. the if
                                                  expression if (PTo - PFrom >= 10) -> …
                                                  %Given (A), CFrom and CTo are necessary w.r.t. the if
                                                  expression if (CTo - CFrom >= 10) -> …
        IncremP = if (PTo - PFrom >= 10) ->       %Given (A), IncremP is necessary w.r.t.
                                                  lists:seq(PFrom, PTo, IncremP)
                                                  %Replace the if expression with undef (NOTE3) would
                                                  prevent to reach the SC because of a bad argument error
                                                  in the lists:seq(PFrom, PTo, IncremP) function call
                                                  %This clause cannot be deleted because it would not be
                                                  possible to satisfy (2),(4)&(5)
                                                  %Replace (PTo - PFrom >= 10) with true would force this
                                                  clause to fulfill (F) and it would prevent to satisfy
                                                  (1)&(5) because of a matching error
                                                  %Replace PTo - PFrom with undef (NOTE3) would prevent to
                                                  reach the SC in (1)&(5) because of a matching error in the
                                                  lists:seq function
                                                  %Replace PTo or PFrom with undef (NOTE3) would prevent to
                                                  reach the SC because of a badarith error
                                                  %Replace 10 with undef (NOTE3) would prevent to satisfy
                                                  (2),(4)&(5) because this clause would become unreachable
                    (PTo-PFrom) div 10;           %(PTo-PFrom) div 10 cannot be deleted because it is the
                                                  only expression of the clause. Replace it with undef
                                                  (NOTE3) would prevent to reach the SC because of a bad
                                                  argument error in the lists:seq function call
                                                  %Replace PTo-PFrom or 10 with undef (NOTE3) would prevent
                                                  to reach the SC because of a badarith error
                                                  %Replace PTo or PFrom with undef (NOTE3) would prevent to
                                                  reach the SC because of a badarith error
```

```
            true ->                                 %This clause cannot be deleted because it would prevent to
                                                    satisfy (1)&(5) because of a matching error

                                                    %1 cannot be deleted because it is the only expression of
                                                    the clause. Replace it with undef (NOTE3) would prevent to
                                                    satisfy (1)&(5) because of a bad argument error in the
                                                    lists:seq function call
        end,
    IncremC = if (CTo - CFrom >= 10) ->             %Given (A), IncremC is necessary w.r.t.
                                                    lists:seq(CFrom, CTo, IncremC)
                                                    %Replace the if expression with undef (NOTE3) would
                                                    prevent to reach the SC because of a bad argument error in
                                                    the lists:seq(CFrom, CTo, IncremC) function call
                                                    %This clause cannot be deleted because it would not be
                                                    possible to satisfy (3),(4)&(5)
                                                    %Replace (CTo - CFrom >= 10) with true would force this
                                                    clause to fulfill (F) and it would prevent to satisfy
                                                    (1)&(5) because of a matching error
                                                    %Replace CTo - CFrom with undef (NOTE3) would prevent to
                                                    reach the SC in (1)&(5) because of a matching error in the
                                                    lists:seq function
                                                    %Replace CTo or CFrom with undef (NOTE3) would prevent to
                                                    reach the SC because of a badarith error
                                                    %Replace 10 with undef (NOTE3) would prevent to satisfy
                                                    (3),(4)&(5) because this clause would become unreachable
                (CTo-CFrom) div 10;                 %(CTo-CFrom) div 10 cannot be deleted because it is the
                                                    only expression of the clause. Replace it with undef
                                                    (NOTE3) would prevent to reach the SC because of a bad
                                                    argument error in the lists:seq function call
                                                    %Replace CTo-CFrom or 10 with undef (NOTE3) would prevent
                                                    to reach the SC because of a badarith error
                                                    %Replace CTo or CFrom with undef (NOTE3) would prevent to
                                                    reach the SC because of a badarith error
            true ->                                 %This clause cannot be deleted because it would prevent to
                                                    satisfy (1)&(5) because of a matching error
                1                                   %1 cannot be deleted because it is the only expression of
                                                    the clause. Replace it with undef (NOTE3) would prevent to
                                                    satisfy (1)&(5) because of a bad argument error in the
                                                    lists:seq function call

        end,
io:format("\nPermutations from ~p to ~p:\n",[PFrom,PTo]),
L1=[show_perm({N, N div 3}) || N <- lists:seq(PFrom, PTo, IncremP)],
                                                    %Delete  L1=[show_perm({N,  N  div  3})  ||  N  <-
                                                    lists:seq(PFrom, PTo, IncremP)] would prevent to satisfy
                                                    (2),(4)&(5)
                                                    %The expression show_perm({N, N div 3}) cannot be replaced
                                                    with undef (NOTE3) because it would prevent to satisfy
                                                    (2),(4)&(5)
                                                    %Replace {N, N div 3} with undef (NOTE3) would prevent to
                                                    reach the SC because of a matching error in the show_perm
                                                    function
                                                    %Given  (A),  N  and  N  div  3  are  necessary  w.r.t.
                                                    show_perm({N,K})
                                                    %Replace N or 3 in N div 3 with undef (NOTE3) would prevent
                                                    to reach the SC because of a badarith error
                                                    %Replace N <- lists:seq(PFrom, PTo, IncremP) with undef
                                                    (NOTE3) would prevent to reach the SC because of an unbound
                                                    variable  error.  This  could  be  avoided  by  replacing
                                                    show_perm({N, N div 3}) with undef but this would prevent
                                                    to satisfy (2),(4)&(5)
                                                    %Given (A), N is necessary w.r.t. show_perm({N, N div 3})
                                                    %Replace lists:seq(PFrom, PTo, IncremP) with undef (NOTE3)
                                                    would prevent to reach the SC because of a bad generator
                                                    error
                                                    %Replace PFrom, PTo or IncremP with undef (NOTE3) would
                                                    prevent to reach the SC because of a bad argument error
io:format("\nCombinations from ~p to ~p:\n",[CFrom,CTo]),
L2=[show_comb({N, N div 3}) || N <- lists:seq(CFrom, CTo, IncremC)],
                                                    %Delete  L2=[show_comb({N,  N  div  3})  ||  N  <-
                                                    lists:seq(CFrom, CTo, IncremC)] would prevent to satisfy
                                                    (3),(4)&(5)
                                                    %The expression show_comb({N, N div 3}) cannot be replaced
                                                    with undef (NOTE3) because it would prevent to satisfy
                                                    (3),(4)&(5)
                                                    %Replace {N, N div 3} with undef (NOTE3) would prevent to
                                                    reach the SC because of a matching error in the show_comb
                                                    function
                                                    %Given  (A),  N  and  N  div  3  are  necessary  w.r.t.
                                                    show_comb({N,K})
                                                    %Replace N or 3 in N div 3 with undef (NOTE3) would prevent
                                                    to reach the SC because of a badarith error
```

```
                                        %Replace N <- lists:seq(CFrom, CTo, IncremC) with undef
                                        (NOTE3) would prevent to reach the SC because of an unbound
                                        variable error. This could be avoided by replacing
                                        show_comb({N, N div 3}) with undef but this would prevent
                                        to satisfy (3),(4)&(5)
                                        %Given (A), N is necessary w.r.t. show_comb({N, N div 3})
                                        %Replace lists:seq(CFrom, CTo, IncremC) with undef (NOTE3)
                                        would prevent to reach the SC because of a bad generator
                                        error
                                        %Replace CFrom, CTo or IncremC with undef (NOTE3) would
                                        prevent to reach the SC because of a bad argument error
    {L1,L2}.

show_perm({N, K}) ->                    %Given (A), N and K are necessary w.r.t.
                                        show_gen(N, K, "perm", fun perm/2)
    show_gen(N, K, "perm", fun perm/2).  %show_gen(N, K, "perm", fun perm/2) cannot be deleted
                                        because it is the only function of the clause. Replace it
                                        with undef (NOTE3) would prevent to satisfy (2),(4)&(5)
                                        %N, K and fun perm/2 are necessary w.r.t.
                                        show_gen(N, K, StrFun, Fun)
show_comb({N, K}) ->                    %Given (A), N and K are necessary w.r.t.
                                        show_gen(N, K, "comb", fun perm/2)
        show_gen(N, K, "comb", fun comb/2). %show_gen(N, K, "comb", fun perm/2) cannot be deleted
                                        because it is the only expression of the clause. Replace
                                        it with undef (NOTE3) would prevent to satisfy (3),(4)&(5)
                                        %N, K and fun perm/2 are necessary w.r.t.
                                        show_gen(N, K, StrFun, Fun)

show_gen(N, K, StrFun, Fun) ->          %Given (A), N, K and Fun are necessary w.r.t.
                                        show_big(Fun(N, K), 40)

    io:format("~s(~p, ~p) = ~s\n",[StrFun, N, K, show_big(Fun(N, K), 40)]).
                                        %This function call cannot be deleted because it is the
                                        only expression of the clause. Replace it with undef
                                        (NOTE3) would prevent to reach the SC because the only
                                        call to the function containing the SC would be deleted
                                        %Replace [StrFun, N, K, show_big(Fun(N, K), 40)] with
                                        undef (NOTE3) would prevent to reach the SC because the
                                        only call to the function containing the SC would be
                                        deleted
                                        %Replace show_big(Fun(N, K), 40) with undef (NOTE3) would
                                        prevent to reach the SC because the only call to the
                                        function containing the SC would be deleted
                                        %Given (A), Fun(N,K) and 40 are necessary w.r.t.
                                        show_big(N,Limit)
                                        %Given (A),Fun, N and K are necessary w.r.t. perm(N,K) and
                                        comb(N,K). From the function calls we can ensure Fun would
                                        only be evaluated as perm (fun perm/2) or comb (fun
                                        comb/2). In consequence, these two functions are necessary

show_big(N, Limit) ->                   %Given (A), N is necessary w.r.t.
                                        StrN = integer_to_list(N)
                                        %Given (A), Limit is necessary w.r.t.
                                        lists:split(Limit,StrN)
    StrN = integer_to_list(N),          %Given (A), StrN is necessary w.r.t.
                                        lists:split(Limit,StrN)
                                        %Replace integer_to_list(N) with undef (NOTE3) would
                                        prevent to reach the SC because of a bad argument error in
                                        the lists:split(Limit, StrN) function call
                                        %Replace N with undef (NOTE3) would prevent to reach the
                                        SC because of a bad argument error
    case length(StrN) < Limit of        %The case expression cannot be deleted because the SC would
                                        be deleted
        true ->                         %Delete this clause would prevent to reach the SC in
                                        (1),(2),(3),(4)&(5) because of a matching error in the
                                        case expression. This could be avoided by replacing the
                                        atom false in the second clause with _ (NOTE3) but this
                                        would produce a bad argument error in the
                                        lists:split(Limit,StrN) function call
            StrN;                       %StrN cannot be deleted because it is the only expression
                                        of the clause and, in consequence, one possible returned
                                        value of the function. The execution of this clause
                                        prevents the execution of the SC because it is located in
                                        another case clause. Besides, the returned value of the
                                        show_big function is not used so we can replace StrN with
                                        undef (NOTE3)

        false ->                        %This clause cannot be deleted because the SC would be
                                        deleted
                                        %The case expression is a logic operator with two possible
                                        result values. Every time the expression value is not
                                        evaluated to true it would be evaluated to false. For this
```

```
                                                reason, this clause would always match and it could be
                                                replaced with _ (NOTE3)
            {Shown, Hidden} = lists:split(Limit, StrN),
                                                %Delete {Shown, Hidden} = lists:split(Limit, StrN) would
                                                prevent to reach the SC because it would be deleted
                                                %{Shown, Hidden} cannot be deleted because the SC would be
                                                deleted
                                                %Shown cannot be deleted because it is the SC
                                                %Hidden can be deleted because it is a definition never
                                                used in the case clause. There are no dependences between
                                                this expression and any other expression of the minimal
                                                slice
                                                %Replace lists:split(Limit,StrN) with undef (NOTE3) would
                                                prevent to reach the SC because of a matching error
                                                %Limit and StrN are necessary because the split function
                                                is a remote function from the erlang library with
                                                unavailable code and replace any of them with undef (NOTE3)
                                                generates a bad argument error
            io_lib:format("~s... (~p more digits)", [Shown, length(Hidden)])
                                                %This function call is executed after executing the SC.
                                                Its returned value would be the returned value of the
                                                function, but this value is ignored by the call. In
                                                consequence, this function call is not part of the minimal
                                                slice
    end.
```

EXECUTION RESULTS:

(1) length(StrN) < Limit (40) ->
PFrom=10,PTo=11,CFrom=10,CTo=11 SC = ∅

(2) length(StrN) >= Limit (40) (only permutations reach the SC) ->
PFrom=10,PTo=100,CFrom=10,CTo=100 SC = "73487567619817453101226654004682424320000"
"37440865768045653413776110658813362809333"
"26636214153245729506242862958589555647415"
"25589079436952654330094306420626434538711"

(3) length(StrN) >= Limit (40) (only combinations reach the SC) ->
PFrom=10,PTo=20,CFrom=100,CTo=200 SC = "20128660909731932294240234380929315748144"
"89920201741625221827800116975578196735960"
"40130199204054316467738730741082931362610"
"36091316841647245952229588716777245148000"
"16207005768165267765533891202388950077420"
"72697525451692783415270666511927389767550"

(4) length(StrN) >= Limit (40) (combinations and permutations reach the SC) ->
PFrom=40,PTo=80,CFrom=120,CTo=170 SC = "49327271416041852081645288304512860160000"
"12155708243936863418861726533009822187520"
"31003426938309408879218399371194443759610"
"20128660909731932294240234380929315748144"
"29978989025985786628387799688747410537670"
"89920201741625221827800116975578196735960"
"26897639707123797206979509942005572225360"
"40130199204054316467738730741082931362610"

(5) length(StrN) >= Limit (40) (combinations and permutations reach the SC) ->
PFrom=199,PTo=200,CFrom=199,CTo=200 SC = "26513595142265325083433914543093366876210"
"39572530063082574751393902303124428173450"
"48707342052634164888231346562991351144250"
"72697525451692783415270666511927389767550"