PROOF FOUNDATIONS

(W) WEISER DEFINITION OF SLICING:
Given a program P, a slicing criterion C=<v,s> where v is a variable at statement s, and a 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 consequence.

(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 whether 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.

```
%------------------------------------------------------------------------------------------
%------------------------------------------------------------------------------------------
%-- bench14.erl
%--
%-- AUTHORS:     Anonymous
%-- DATE:        2016
%-- PUBLISHED:   Software specially developed to test challenging slicing problems
%-- COPYRIGHT:   Bencher: The Program Slicing Benchmark Suite for Erlang
%--              (Universitat Politècnica de València)
%--              http://www.dsic.upv.es/~jsilva/slicing/bencher/
%-- DESCRIPTION
%-- The program presents a set of difficult slicing problems like unreachable clauses in case
%-- statements or never called function clauses. It receives two inputs of any nature and
%-- processes them with a suit of case statements and function calls to obtain a final
%-- result.
%------------------------------------------------------------------------------------------
%------------------------------------------------------------------------------------------
```

```
-module(bench14).
-export([main/2]).

main(X,Y) ->
        Z = case X of
                                  %Given (A), X is necessary w.r.t. the case X of expression
                                  %Given (A), Z is necessary w.r.t. the SC Z. In consequence,
                                  the whole expression Z = case … canot be deleted
                                  %The case expression is the only expression that can assign a
                                  value to the SC. Replace it with undef (NOTE2) would prevent
                                  to satisfy (1),(2)&(3)
                                  %Replace X with undef (NOTE2) would prevent to satisfy (1)&(2)
                                  if we modify clauses 1 or 2 of the case expression it would
                                  be possible to satisfy one of this executions, but this would
                                  prevent to fulfill (3)
                terminate ->      %This clause cannot be deleted because it would prevent to
                                  satisfy (1)
                                  %Replace terminate with _ (NOTE2) would prevent to satisfy
                                  (2)&(3)
                    "the end";    %This expression cannot be deleted because it is the only
                                  expression of the clause and also one of the possible values
                                  of the SC variable Z
                {A,B} ->          %This clause cannot be deleted because it would prevent to
                                  satisfy (2)
                                  %Given (A), A and B are necessary w.r.t. {[A+B,B-A],3}. In
                                  consequence {A,B} cannot be deleted
                    {[A+B,B-A],3};  %{[A+B,B-A],3} cannot be deleted because it is the only
                                  expression of the clause and also one of the possible values
                                  of the SC variable Z
                                  %[A+B,B-A],A+B,B-A or 3 cannot be replaced with undef (NOTE2)
                                  because it would prevent to satisfy (2)
                                  %A or B cannot be replaced with undef (NOTE2) because it would
                                  prevent to reach the SC due to a badarith error in execution
                                  (2)
                {3,C} -> g(C);    %Due to D1, this clause can be deleted
                _ ->              %This clause cannot be deleted because it would prevent to
                                  satisfy (3)
                    {20*3,8}      %{20*3,8} cannot be deleted because it is the only expression
                                  of the clause and also one of the possible values of the SC
                                  variable Z
                                  %20*3 or 8 cannot be replaced with undef (NOTE2) because it
                                  would prevent to satisfy (3)
                                  %20 or 3 cannot be replaced with undef (NOTE2) because it
                                  would prevent to reach the SC due to a badarith error in
                                  execution (3)
        end,
        T = 2,
        V = f(T)+h(2)+h(3),
        W = g([X,Y,{X,Y}]),
        Tuple = {Z,W,V},          %Z cannot be deleted because it is the SC
        Tuple.

g(X) ->                           %Given (D1), we have deleted from the program all existent
                                  calls to the g() function. In consequence, we can delete from
                                  the program the definition of this function: g(X)-> …
        [_,_,{R,S}] = X,
        case R of
                [1,3] -> 21;
                [A,B] -> (A*B)/9;

                T ->  T;
                _ -> f(4)
        end.

f(7) ->
        L = 2+9,
        F = L*3,
        F+L;
f(4) -> 9;                        %Given (D1), we have deleted from the program all the calls
                                  to the g() function. In consequence, by deleting the g()
                                  function from the minimal slice, we have also deleted the
                                  calls to the function f(), so we can delete from the program
                                  all the definitions of this function: f(4)-> 9 and f(X) -> X

f(2) -> 7;
f(X) -> X.

h(X) ->
        case X of
```

```
              2-> j({2,4});
              3->    k([4,8]);
              1-> l(107)
       end.

j(A) -> {X,_} = A, X.
k(B) -> [H|T] = B, H.
l(C) -> C-1.
```

EXECUTION RESULTS:

A,B,C -> Undefined variables

	SLICING CRITERION
(1) X == terminate -> X = terminate	SC = “the end”
(2) X != terminate && X == {A,B} -> X = {1,6}	SC = {[7,5],3}
(3) X != terminate && X != {A,B} && X != {3,C} && _=X -> X = 21	SC = {60,8}

Demonstration 1 (D1)

In order to execute the clause 3 of the case expression({3,C} -> g(C)) the following constrains need to be fullfiled:

!({A,B}=X) && {3,C}=X

Being A, B and C undefined variables

But this will never succeed because one of these constrains lead to a contradiction:

!({A,B}=X) when A,B undefined ∧ X ∈ two-elements tuple -> ∅

When a variable is unbound, it can always be assigned in a case clause pattern, in consequence, clause 2 of the case expression fullfil (F) when X is a tuple of two elements. The clause 3 is a particular case of the clause 2 and will never match.

Conclusion: The clause 3 of the case expression is not part of the minimal slice