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 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 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.
NOTE3: The value of the SC is independent of the input values. This conclusion can be reached because the complete slicers used in phase 1 have deleted the two input parameters X and Y.

```
%-------------------------------------------------------------------------------------------
%-------------------------------------------------------------------------------------------
%-- 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]).

%All expressions in red can be deleted for (G) because of the limited number of possible values for the
slicing criterion which is independent of the program inputs (NOTE3).

main(X,Y) ->
        Z = case X of
                terminate -> "the end";
                {A,B} -> {[A+B,B-A],3};
                {3,C} -> g(C);
                _ -> {20*3,8}
        end,
        T = 2,
        V = f(T)+h(2)+h(3),                     %V cannot be deleted because it is the SC
                                                %f(T)+h(2)+h(3) is the only expression that can assign a value
                                                to the SC. If we replace it with undef (NOTE2) it would be
                                                impossible to satisfy (1)
                                                %Replace f(T), h(2) or h(3) with undef (NOTE2) would prevent
                                                to satisfy (1) due to a badarith error
                                                %Replace 2 with undef would prevent to satisfy (1) because of
                                                a change in the behaviour of the h() function. h(undef) and
                                                h(3) would wrongly return the same value preventing to fulfill
                                                (1)
        W = g([X,Y,{X,Y}]),
        Tuple = {Z,W,V},
        Tuple.

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;
f(2) ->
        7;                                      %7 cannot be deleted because it is the only expression of the
                                                f() function and its returned value. Replacing it with undef
                                                (NOTE2) would prevent to reach the SC due to a badarith error.

f(X) -> X.

h(X) ->                                         %Given (A), X is necessary w.r.t. the case expression
        case X of                               %The case expression cannot be deleted because it is the only
                                                expression of the h() function. Replaced it with undef (NOTE2)
                                                would prevent to reach the SC because of a badarith error.
                                                %Replace X with undef (NOTE2) would prevent to satisfy (1)
                                                because it would prevent to enter in more than one clause,
                                                forcing all the h() calls to return the same value
                2->                             %Replace 2 with _ (NOTE2) would prevent to satisfy (1) because
                                                the first clause would fulfill (F) and h() would return always
                                                the same value
```

```
                j({2,4});                %j({2,4}) cannot be deleted because it is the only expression
                                         of the case clause and in consequence a possible returned
                                         value of the function. Replace it with undef would prevent to
                                         satisfy (1) because of a badarith error
            3->                          %This clause cannot be deleted because it would produce a
                                         pattern matching error in the case expression. This error can
                                         be avoided by replacing the previous clause pattern (2) with
                                         _, but as proved before, this 2 cannot be replaced with _
                k([4,8]);                %k([4,8]) cannot be deleted because it is the only expression
                                         of the case clause and in consequence a possible returned
                                         value of the function. Replace it with undef would prevent to
                                         satisfy (1) because of a badarith error
            1->
                l(107)
    end.

j(A) ->                                  %Given (A), A is necessary w.r.t {X,_} = A
    {X,_} = A,                           %Given (A), {X,_} = A is necessary w.r.t. X.
                                         %{X,_} cannot be replaced with _ (NOTE2) because it would
                                         prevent to reach the SC. If we replace it with _ the next
                                         expression ( X ) would produce an unbound variable error. If
                                         we also delete the X expression, the execution would generate
                                         a badarith error
                                         %A cannot be replaced with undef because it would prevent to
                                         reach the SC because of a matching error
    X.                                   %X cannot be deleted because it is the last expression of the
                                         j() function and in consequence its returned value. Deleting
                                         it or replacing it with undef (NOTE2) would prevent to reach
                                         the SC due to a badarith error
k(B) ->                                  %Given (A), B is necessary w.r.t. [H|T] = B
    [H|T] = B,                           %Given (A), [H|T] = B is necessary w.r.t. H.
                                         %[H|T] cannot be replaced with _ (NOTE2) because it would
                                         prevent to reach the SC. If we replace it with _, the next
                                         expression ( H ) would produce an unbound variable error. If
                                         we also delete the H expression, the execution would generate
                                         a badarith error
                                         %B cannot be replaced with undef because it would prevent to
                                         reach the SC because of a matching error
    H.                                   %H cannot be deleted because it is the last expression of the
                                         k() function and in consequence its returned value. Deleting
                                         it or replacing it with undef (NOTE2) would prevent to reach
                                         the SC due to a badarith error

l(C) ->
    C-1.
```

EXECUTION RESULT:

(1) SC = V = f(T)+h(2)+h(3) => SC = 13 (f(T)=7 + h(2)=2 + h(3)=4)