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.

```
%-----------------------------------------------------------------------------------------
%-----------------------------------------------------------------------------------------
%-- bench7.erl
%--
%-- AUTHORS:     Anonymous
%-- DATE:         2016
%-- PUBLISHED:   Software specially developed to test dead code detection.
%-- COPYRIGHT:   Bencher: The Program Slicing Benchmark Suite for Erlang
%--              (Universitat Politècnica de València)
%--              http://www.dsic.upv.es/~jsilva/slicing/bencher/
%-- DESCRIPTION
%-- This benchmark consists in a function receiving two numbers as inputs and calling
%-- another one with a conditional structure that contains a dead code statement.
%-----------------------------------------------------------------------------------------
%-----------------------------------------------------------------------------------------
-module(bench7).
-export([numbers/2]).
```

```
numbers(A,B) ->                 %Given (A), A is necessary w.r.t. C=fn(A,B)
        C=fn(A,B).              %C cannot be deleted because it is the SC
                                %fn(…) 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) & (2)
                                %Given (A), A is necessary w.r.t. ft(X,Y)
fn(X,Y) ->                      %Given (A), X is necessary w.r.t. the if expression
        if                      %The if expression is necessary because it is the only expression of the fn
                                function. If we replace it with undef (NOTE2) it would not be possible to
                                satisfy
                                (1) & (2)
                X>5 ->          %This clause cannot be deleted because it would not be possible to satisfy
                                (1)
                        Z=Y,    %Replace X>5 with true (NOTE2) would produce (F) and thus (2) is not satisfied
                                because of (E)
                                %Replace X with undef (NOTE2) would produce (F) and thus (2) is not satisfied
                                because of (E)
                                %Replace 5 with undef (NOTE2) would make not possible to satisfy (1) because
                                of (E)
                        X;      %Delete X is not possible because it is the only statement of the clause and
                                deleting it would produce (D). If we replace X with undef (NOTE2) it would
                                not be possible to satisfy (1)
                true ->         %This clause cannot be deleted because SC would never be reached due to a
                                matching error in (2)
                        X+2     %Replace X+2 with undef (NOTE2) would make not possible to satisfy (2) because
                                X+2 is one of the possible returned values of the fn function
                                %Replace X or 2 with undef (NOTE2) would make not possible to satisfy (2)
                                due to a badarith error
        end.

EXECUTION RESULT:

INPUTS:                                             SLICING CRITERION

        (1) X>5 -> A=6, B=9                              C = 6
        (2) !(X>5) && (X=<5) -> A=3, B=4                 C = 5 (C=X+2)
```