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 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.

```
%-----------------------------------------------------------------------------------------
%-----------------------------------------------------------------------------------------
%-- bench13.erl
%--
%-- AUTHORS:     Susan Horwitz
%-- DATE:        1990
%-- PUBLISHED:   Interprocedural slicing using dependence graphs
%--              ACM Transactions on Programming Languages and Systems
%--              Volume 12 Issue 1, Jan. 1990 Pages 26-60
%-- COPYRIGHT:   Bencher: The Program Slicing Benchmark Suite for Erlang
%--              (Universitat Politècnica de València)
%--              http://www.dsic.upv.es/~jsilva/slicing/bencher/
%-- DESCRIPTION
%-- The program proposed by Susan Horwitz was designed to illustrate the problem of the
%-- Weiser's slicing algorithm in interprocedural slicing. The program consists in a
%-- recursive call to add the first 11 naturals.
%-----------------------------------------------------------------------------------------
%-----------------------------------------------------------------------------------------
```

```
-module(bench13).
-export([main/0]).

main() ->
  Sum = 0,
  I = 1,                                  %Given (A), I is necessary w.r.t. while(Sum,I,11)
  {Result,_} = while(Sum,I,11),           %Replace while(Sum,I,11) with undef would prevent to satisfy
  Result.                                 (1)
                                          %Replace I with undef would prevent to reach the SC due to a
                                          badarith error in the NewI = increment(I), function call
                                          %Replace 11 with undef would prevent to satisfy (1) because
                                          of a badarith error in the while(NewS,NewI,Top-1) call inside
                                          the while(S,I,Top) clause
while(S,I,Top) ->                         %Given (A), I is necessary w.r.t. NewI = increment(I)
                                          %Given (A), Top is necessary w.r.t. the if expression
  if                                      %The If expression cannot be deleted because it is the only
                                          expression of the while function and it would be not possible
                                          to reach the SC
     Top /= 0 ->                          %Delete this clause would prevent to reach the SC because
     NewS = add(S,I),                     the SC would also be deleted
                                          %Replace Top with undef would convert this clause in absorbent
                                          (F) and prevent to satisfy (1) for (E)
                                          %Replace 0 with undef would prevent to satisfy (1) because
                                          this clause would become absorbent (F) and the execution would
                                          never terminate
                                          %Replace Top and 0 with undef simultaneously would make this
                                          clause unreachable and the SC would never be reached
        NewI = increment(I),              %NewI is necessary because it is the SC
                                          %increment(I) is necessary because it is the only expression
                                          that can assign a value to the SC
                                          %Given (A), I is necessary w.r.t. increment(A)->…
        while(NewS,NewI,Top-1);           %Replace while(NewS,NewI,Top-1) with undef (NOTE2) would
                                          prevent to satisfy (1) because this function call is necessary
                                          to obtain the value of the SC
     Top == 0 ->                          %Given (A), NewI is necessary w.r.t. while(S,I,Top) and used
                                          by the increment(I) call to compute the value of the SC
                                          %Replace Top-1 with undef (NOTE2) would prevent to satisfy (1)
                                          due to a badarith error in the while(NewS,NewI,Top-1) call
        {S,I}                             %Replace Top or 1 with undef (NOTE2) would prevent to satisfy
                                          (1) because of a badarith error

end.
add(A,B) ->                               %Given (A), A and B are necessary w.r.t. A+B
       A+B.                               %A+B cannot be deleted because it is the only expression of
                                          the add function. Its value will be the returned value of the
                                          function and in consequence the value of the SC
                                          %Replace A or B with undef (NOTE2) would prevent to satisfy
                                          (1) because of a badarith error
increment(A) ->                           %Given (A), A is necessary w.r.t. add(A,1)
       add(A,1).                          %add(A,1) is necessary because it is the only expression of
                                          the increment function and its value will be the returned
                                          value of the function and in consequence the value of the SC.
                                          %Given (A), A and 1 are necessary w.r.t. add(A,B)->…
```

```
EXECUTION RESULTS:
ONLY ONE POSSIBLE EXECUTION (PROGRAM WITHOUT INPUTS)

(1) SC VALUES =  2,3,4,5,6,7,8,9,10,11,12
```