**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.

```
%-------------------------------------------------------------------------------------------
%-------------------------------------------------------------------------------------------
%-- bench2.erl
%--
%-- AUTHORS:      Anonymous
%-- DATE:         2016
%-- PUBLISHED:    Software specially developed to test higher order functions and anonymous
%--               functions.
%-- COPYRIGHT:    Bencher: The Program Slicing Benchmark Suite for Erlang
%--               (Universitat Politècnica de València)
%--               http://www.dsic.upv.es/~jsilva/slicing/bencher/
%-- DESCRIPTION
%-- The benchmark receives two input parameters and executes a call to a function using fun
%-- expressions. These expressions are stored in a variable and lately called.
%-------------------------------------------------------------------------------------------
%-------------------------------------------------------------------------------------------
```

```
-module(bench2).
-export([main/2]).
main(A,B)->                          %Given (A), A and B are necessary w.r.t. the if expression
        if                           %The if expression cannot be deleted or because it is the only expression
                                     where the SC can be declared and deleting it would produce (D)
            A>B ->                   %Given (D2), we can state that the clauses of the if expression will
                                     instantiate the C variable with the same value. We can delete one of them
                                     making the other instantiate the C variable in all cases. We delete the
                                     case clause 1 because case clause 2 fulfill (F)
                C=f(A,B);
            true ->                  %Delete this clause would prevent to reach the SC due to an unbound
                                     variable compilation error (D)
                C=f(B,A)             %C=f(B,A) cannot be deleted because it is the only expression of the if
                                     clause. Replacing it with undef (NOTE2) would lead to an unbound variable
                                     error (D) because this is the only expression that can assign a value to
                                     the SC
                                     %Replace C with _ (NOTE2) would prevent to reach the SC due to an unbound
                                     variable error (D)
                                     %Replace  f(B,A)  with  undef  (NOTE2)  would  prevent  to  satisfy
                                     (1),(2),(3)&(4)
        end,
        C,                           %C cannot be deleted because it is the SC
        D = h(fun g/2,A,B),          %The D = h(fun g/2,A,B) expression is executed after the evaluation of the
                                     SC expression. In consequence, given the lack of a recursive call to the
                                     main function, this expression cannot modify the value of the SC and can
                                     be deleted
        D,
        {C,D}.
f(X,Y) ->                            %Given (A), X and Y are necessary w.r.t. the case expression
        Z= X-Y,                      %Z= X-Y can be deleted because its only use (case Z of) is not part of the
                                     minimal slice and in consequence it becomes an unused variable
        W= X+Y,                      %W= X+Y can be deleted because its only use (W -> …) is not part of the
                                     minimal slice and in consequence it becomes an unused variable
        case Z of                    %The case expression cannot be deleted because after deleting the two
                                     previous expressions it is the only expression of the f function and
                                     defines its returned value. Replace it with undef (NOTE2) would prevent
                                     to satisfy (1),(2),(3)&(4)
                                     %Z can be delete because of (D1) there is only one absorbent clause in the
                                     case expression, so it will be executed regardless of the value of this
                                     variable
            W ->                     %Given (D1), we can reduce the case clauses to one absorbent clause by
                                     deleting one of them and transforming its guard in an absorbent condition.
                                     In our case, we have deleted the second clause
                                     _ ->(fun(N) -> N end)(W) and replaced the guard of the first one with _
                (fun(B,E) ->         %This expression cannot be deleted because it is the only expression of
                                     the clause. Replacing it with undef (NOTE2) would prevent to satisfy
                                     (1),(2),(3)&(4)
                   N=B*E,
                   B+E
                end)(X,Y);           %Given (A), X and Y are necessary w.r.t. fun(B,E)
                                     %Given (A), B and E are necessary w.r.t. B+E
                                     %B+E cannot be deleted because it is the only expression in the anonymous
                                     function.  Replace  it  with  undef  (NOTE2)  would  prevent  to  satisfy
                                     (1),(2),(3)&(4)
                                     %B or E cannot be replaced with undef (NOTE2) because it would generate a
                                     badarith error
            _ -> (fun(N) ->
                    N
                 end)(W)
        end.
g(X,Y) ->
        (fun(A,B) ->
                C=A-B,
                A+B
        end)(X,Y).
h(F,A,B) ->                          %The calls of the h function have been deleted from the minimal slice
                                     because they are executed after evaluating the SC, in consequence this
                                     function definition is not part of the slice
        C=B*2,
        D= if
              B>A ->
                B-3;
              B=<A ->
                A+5
           end,
        F(A,B).
```

EXECUTION RESULTS:

| | SLICING CRITERION |
|---|---|
| (1) A > B && Z == W -> A = 1, B = 0 | SC = 1 (B+E => X+Y) |
| (2) A > B && Z /= W -> A = 6, B = 3 | SC = 9 (N => W => X+Y) |
| (3) A =< B && Z == W -> A = 0, B = 5 | SC = 5 (B+E => X+Y) |
| (4) A =< B && Z /= W -> A = 1, B = 3 | SC = 4 (N => W => X+Y) |

**Demonstration 1 (D1)**
---

**Value of Clause 1= (fun(B,E) -> B + E end)(X,Y) = fun(X,Y)-> X+Y end = X+Y**
**Value of Clause 2= W = X+Y, (fun(N) -> N end)(W), = (fun(N) -> N end)(X+Y) = fun(X+Y) -> X+Y end = X+Y**

**Conclusion:** Case clauses 1 and 2 produce the same result for the case expression, in consequence one of them can be deleted

**Demonstration 2 (D2)**
---

**STATE OF f(X,Y) AFTER DELETING EXPRESSIONS WITH D1:**

```
f(X,Y) ->
        case undef of
                _ ->
                    (fun(B,E) ->
                        B+E
                    end)(X,Y);
        end.
```

**f(A,B) = (fun(B,E) -> B+E end)(A,B) = fun(A,B) -> A+B end = A+B**
**f(B,A) = (fun(B,E) -> B+E end)(B,A) = fun(B,A) -> B+A end = B+A**

**Conclusion:** Due to the commutative property of addition we can ensure that the function f(X,Y) will generate the same result for two specific inputs regardless of the order they are given in the function call.