**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.

```
%-----------------------------------------------------------------------------------------
%-----------------------------------------------------------------------------------------
%-- bench6.erl
%--
%-- AUTHORS:      Anonymous
%-- DATE:         2016
%-- PUBLISHED:    Software specially developed to test the detection of unreachable
%--               clauses in case conditional structure and never matching clauses
%--               in 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 consists in a function receiving two tuples as input. It calls other two
%-- functions containing a case statement with unreachable clauses. One of this functions
%-- also makes a call to another function with unreachable and unmatchable clauses.
%-----------------------------------------------------------------------------------------
%-----------------------------------------------------------------------------------------
```

```
-module(bench6).
-export([tuples/2]).
tuples(A,B) ->                  %Given (A), A and B are necessary w.r.t. C=ht(B,A)
        C=ft(A,B),
        D=ht(B,A),              %D is necessary because it is the SC
                                %ht(B,A) is the only expression that can assign a value to the SC. Replace
                                it with undef (NOTE2) would prevent to satisfy (1)
                                %Given (A), B and A are necessary w.r.t. ht({X,Y,Z},{A,B})
        {C,D}.
ft({X,Y},{X,Z,W}) ->
        case Y of
                W ->
                        gt({1,X});
                Y ->
                        gt({0,Y});
                _ ->
                        U=Z+X,
                        gt({U,Z})
        end.
gt({1,2,X}) ->
        X+4;
gt({0,_}) ->
        4;
gt(_) ->
        16;
gt({X,Y}) ->
        Y;
gt({}) ->
        0;
gt(X) ->
        element(1,X).

ht({X,Y,Z},{A,B})->             %Replace {X,Y,Z} or {A,B} with undef (NOTE2) would prevent to reach the SC
                                because of (D)
                                %X,Y are not necessary because their uses inside the ht function (Y -> X*2+A,
                                A -> U=Y+Z, Z+U) are not part of the minimal slice
                                %Given (A), Z, A and B are necessary w.r.t. the case expression
        case Z of               %The case expression cannot be deleted because it is the only expression of
                                the ht function and defines its returned value and in consequence the value
                                of the SC. Replace it with undef (NOTE2) would prevent to satisfy (1)
                                %Z can be deleted because the clause 1 of the case expression fulfills (F)
                                and it would always matches to every possible value of variable Z
                _ ->            %This clause cannot be deleted because it would prevent to satisfy (1)
                        A+Z*B;  %A+Z*B cannot be replaced with undef (NOTE2) because it would prevent to
                                satisfy (1)
                                %A, Z and B cannot be replaced with undef (NOTE2) because it would prevent to
                                reach the SC due to a badarith error
                Y ->            %This clause can be deleted because of (D1)
                        X*2+A;
                A ->            %This clause can be deleted because of (D1)
                        U=Y+Z,
                        Z+U
        end.
```

EXECUTION RESULTS:

| | SLICING CRITERION |
|---|---|
| (1) _ = Z -> A={5,8}, B={1,4,2} | 21 |

**Demonstration 1 (D1)**

---

In order to execute the clause 2 (Y -> X*2+A) or the clause 3 (A -> U=Y+Z, Z+U) of the case expression the following constrains need to be fulfilled:

```
                                !(_ = Z) && Z == Y (case clause 2)
                        !(_ = Z) && !(Z = Y) && Z == A (case clause 3)
```

But these conditions will never succeed because one of these constrains leads to a contradiction:

**!(_ = Z) -> Ø**

The clause 1 of the case expression (_ -> A+Z*B) will always match without taking the value of Z into account, this clause fulfills (F).

**Conclusion:** The clauses 2 and 3 of the case expression are not part of the minimal slice