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.

```
%-----------------------------------------------------------------------------------------
%-----------------------------------------------------------------------------------------
%-- 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. ft(A,B)
       C=ft(A,B),                       %C cannot be deleted because it is the SC
                                        %ft(A,B) is the only expression that can assign a value to the SC.
                                        Replace it with undef (NOTE2) would prevent to satisfy (1)&(2)
                                        %Replace A or B with undef (NOTE2) would prevent to reach the SC due
                                        to a matching error in the clause of the ft function
       D=ht(B,A),
       {C,D}.
ft({X,Y},{X,Z,W}) ->                    %Given (A), Y and W are necessary w.r.t. the case expression
                                        %After deleting the third case clause (D1) there are dependencies
                                        between the defined variable Z and any variable of the minimal slice,
                                        in consequence Z can be deleted
                                        %Given (W), the two X expressions of the clause ft({X,Y},{X,Z,W}) can
                                        be replaced with _ (NOTE2). These expressions are defining an explicit
                                        constraint and deleting them would make the slice accept some possible
                                        inputs that would prevent the original program to terminate normally,
                                        but in the cases where the SC is reached by the original program, the
                                        slice and the original program would generate the same value/s for
                                        the SC. It does not exist any data dependence between the X received
                                        as a parameter and any other expression contained in the minimal slice
                                        (because {1,X} is not part of the minimal slice), in consequence, the
                                        two X parameters can be deleted
       case Y of                        %The case expression cannot be deleted because it is the only
                                        expression of the ft function and in consequence its returned value
                                        is assigned to the SC. Replace it with undef (NOTE2) would prevent to
                                        satisfy (1)&(2)
                                        %Replace Y with undef (NOTE2) would prevent to satisfy (1)&(2)
                                        simultaneously. It would be possible to satisfy (1) by replacing W
                                        with _ but this would prevent to satisfy (2). It would also be possible
                                        to satisfy (2) replacing Y with _ but this would prevent to satisfy
                                        (1)
              W ->                      %This clause cannot be deleted because it defines one of the returned
                                        values of the ft function. Deleting it would prevent to reach (1)
                                        %Replace W with _ would prevent to satisfy (2) because it would make
                                        this clause to fulfill (F)
                     gt({1,X});         %gt({1,X}) cannot be deleted because it is the only expression of the
                                        clause. Replace it with undef (NOTE2) would prevent to satisfy (1)
                                        %{1,X} can be deleted because the function call would never match to
                                        the second clause of the gt function. The first element in the tuple
                                        of the call "1" would never match to the first element in the tuple
                                        of the clause "0". Then, this function call will always match to the
                                        third clause gt(_) and we can delete {1,X} because there is no
                                        dependence between the parameters of the call and the returned result
                                        of the function clause
              Y ->                      %This clause cannot be deleted because it would produce a matching
                                        error in execution (2). This may be solved by replacing the previous
                                        clause W with _ (NOTE2) but this would prevent to satisfy (2)
                                        %Y can be deleted because this clause fulfills (F) and the value of
                                        Y has already been defined
                     gt({0,Y});         %gt({0,Y}) cannot be deleted because it is the only expression of the
                                        clause. Replace it with undef (NOTE2) would prevent to satisfy (2)
                                        %Tuple {0,Y} cannot be deleted because it would prevent to satisfy
                                        (2). It would be possible to satisfy (2) by replacing gt({0,_}) -> …
                                        with gt(_) -> … but this would prevent to satisfy (1)
                                        %Y can be deleted because in clauses 2 and 3 of the gt function (the
                                        remaining clauses after the execution of the phase 1) the returned
                                        value is independent of the input parameters received
                                        %0 can be deleted because of (L1)
              _ ->                      %This clause can be deleted because of (D1)
                     U=Z+X,
                     gt({U,Z})

       end.
gt({1,2,X}) ->
       X+4;
gt({0,_}) ->                            %This clause cannot be deleted because it would prevent to satisfy
                                        (2)
                                        %Replace {0,_} with _ (NOTE2) would prevent to satisfy (1) because
                                        this clause would fulfill (F)
                                        %0 can be deleted because of (L1)
       4;                               %4 cannot be deleted because it is the only expression of the clause,
                                        its returned value is also one of the possible returned values of the
                                        ft function and, in consequence, one of the possible values of the
                                        SC. Replace it with undef (NOTE2) would prevent to satisfy (2)
```

```
gt(_) ->                                %This clause cannot be deleted because it would prevent to reach the
                                        SC in execution (1) due to a matching error in the gt({1,X}) function
                                        call
       16;                              %16 cannot be deleted because it is the only expression of the clause,
                                        its returned value is also one of the possible returned values of the
                                        ft function and, in consequence, one of the possible values of the
                                        SC. Replace it with undef (NOTE2) would prevent to satisfy (1)

gt({X,Y}) ->
       Y;
gt({}) ->
       0;
gt(X) ->
       element(1,X).
ht({X,Y,Z},{A,B})->
       case Z of
              _ ->
                     A+Z*B;
              Y ->
                     X*2+A;
              A ->
                     U=Y+Z,
                     Z+U
       end.
```

%(L1): 0 in the gt({0,Y}) function call would always match to the clause of the gt function (gt({0,_})) due to a static constraint. Replacing 0 in gt({0,Y}) call with undef would prevent to satisfy (2) because it would prevent to fulfill the defined constraint but replacing also the 0 in the clause gt({0,_}) of the gt function with _ would make the constraint to be fulfilled again. Because of this, both 0 expressions (call and clause) can be replaced without modifying the behaviour of the program. In the case expression of the gt({1,X}) function call, delete 0 in gt({0,_}) clause would make no difference because the whole expression {1,X} has been proven not necessary and the gt(undef) call (NOTE2) would never match the clause

EXECUTION RESULTS:

	SLICING CRITERION
(1) Y==W -> A={1,2},B={1,3,2} -> Y=2, W=2	16
(2) !(Y==W) && Y==Y -> A={2,3},B={2,7,8}-> Y=3, W=8	4

Demonstration 1 (D1)

In order to execute the clause 3 of the case expression (_ -> U=Z+X,gt({U,Z})) the following constraints need to be fulfilled:

!(Y==W) && !(Y==Y) && _=Y

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

!(Y==Y) -> ∅

A variable is always equals to itself, in consequence, the clause 2 of the case expression fulfills (F).

Conclusion: The clause 3 of the case expression is not part of the minimal slice