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.

```
%-------------------------------------------------------------------------------------------
%-------------------------------------------------------------------------------------------
%-- bench4.erl
%--
%-- AUTHORS:      Anonymous
%-- DATE:         2016
%-- PUBLISHED:    Software specially developed to test list comprehensions and blocks.
%-- COPYRIGHT:    Bencher: The Program Slicing Benchmark Suite for Erlang
%--               (Universitat Politècnica de València)
%--               http://www.dsic.upv.es/~jsilva/slicing/bencher/
%-- DESCRIPTION
%-- The program receives a list of chemical elements with their abbreviation and their
%-- melt temperature in the format {Element,Abbreviation,Melting_Temp} and a temperature.
%-- It returns a list with the elements that have a melt temperature greater than a
%-- temperature given. The format of each element is {Element,Melting_Temp}.
%-- It also returns a list with the abbreviations of the selected elements when this
%-- abbreviation is one character long.
%-------------------------------------------------------------------------------------------
%-------------------------------------------------------------------------------------------
```

```
-module(bench4).
-export([main/2]).

main(Elements,Temp) ->
        Res = templessthan(Temp,Elements),
        Abb = abbreviation(Elements,Res),
        {Res,Abb}.

templessthan(Temp,Elem) ->
        [begin
             T1=T*7+1,
             T2=T*2,
             T3=T2/2,
             {Name,T3}
         end||{Name,Abb,T} <- Elem, T > Temp].

abbreviation(Elements,List) ->
        [Abb||
           {RN,RT} <- List,
```

%Given (A), Elements and Temp are necessary w.r.t. templessthan(Temp,Elements)
%Given (A), Res is necessary w.r.t. abbreviation(Elements,Res)
%Replace templessthan(Temp,Elements) with undef (NOTE2) would prevent to reach the SC due to a badgenerator error
%Given (A), Temp and Elements are necessary w.r.t. templessthan(Temp,Elem)
%Abb cannot be deleted because it is the SC
%Replace abbreviation(Elements,Res) with undef (NOTE2) would prevent to satisfy (1) & (2)
%Given (A), Elements and Res are necessary w.r.t. abbreviation(Elements,List)

%Given (A), Temp and Elem are necessary w.r.t filter T > Temp and generator {Name,Abb,T} <- Elem in the list comprehension expression
%List comprehension expression cannot be deleted because it is the only expression of the templessthan function. Replacing it with undef would prevent to reach the SC due to a bad generator error
%Replace begin … end with undef would prevent to reach the SC because of a matching error in the {RN,RT} <- List expression in the abbreviation function

%T2 = T*2 and T3 = T2/2 expressions can be deleted because the value defined by these expressions is never used because T3 is not part of the minimal slice

%Replace {Name,T3} with undef (NOTE2) would prevent to satisfy (1) because undef elements of the list comprehension result would be ignored in the list comprehension of the abbreviation function [Abb||{RN,RT} <- List, …] generating always an empty list as result
%Given (A), Name is necessary w.r.t. the {RN,RT}<- List expression in the abbreviation function
%T3 can be deleted because its value is never used in the abbreviation function. We know that because phase 1 deleted the expression getting the value of this one in the abbreviation function call {RN,RT} <- List
%Replace {Name,Abb,T} with _ would produce (D). This could be solved by deleting the filter T > Temp and replacing {Name,T3} with {undef,T3} but this would prevent to satisfy (1) even performing more transformations in some expressions of the abbreviation() function
%Given (A), Name is necessary w.r.t. {Name,T3}
%Given (A), T is necessary w.r.t. T > Temp
%Replace Elem with undef (NOTE2) would prevent to reach the SC because of a bad generator error
%Delete T > Temp would prevent to satisfy (1)&(2) because the expression would select all the elements of the list
%Replace T with undef (NOTE2) would prevent to satisfy (1)&(2) because of (E)
%Replace Temp with undef (NOTE2) would prevent to satisfy (1) because of (E)

%Given (A), Elements and List are necessary w.r.t. {RN,RT} <- List, {Name,Abb,T} <- Elements in the list comprehension expression
%The list comprehension expression cannot be deleted because it is the only expression of the abbreviation function. Replacing it with undef (NOTE2) would prevent to satisfy (1)&(2)
%Replace Abb with undef (NOTE2) would prevent to satisfy (1) because it defines the values of the list returned in the list comprehension expression and in consequence the list of values that will be assigned to the SC
%Replace List with undef (NOTE2) would prevent to reach the SC because of a bad generator error
%Replace {RN,RT} with _ (NOTE2) would prevent to reach the SC because of (D). This could be solved by replacing RN == Name with undef == Name, but this would prevent to satisfy (1). Even if we also replace {Name,Abb,T} with {undef,Abb,T} the condition would become true but (1) would neither be satisfied

```
        {Name,Abb,T} <- Elements,

                RN == Name andalso
        erlang:length(atom_to_list(Abb)) == 1].
```

```
%Replace RN with _ (NOTE2) would prevent to reach the SC
because of (D). This syntax error could be solved by
replacing RN == Name with undef == Name (NOTE2), but it
would prevent to satisfy (1), even if we perform any more
transformations (1) would never be satisfied
%Replace Elements with undef (NOTE2) would prevent to
reach the SC due to a bad generator error
%Replace {Name,Abb,T} with _ (NOTE2) would prevent to
reach the SC because of (D). This could be solved by
replacing RN == Name with RN == undef and [Abb ||…] with
[undef || …] but this would prevent to satisfy (1) because
Abb has been proven necessary in the minimal slice
%Replace Name with _ would prevent to satisfy (1)&(2)
because of (D). This could be solved by replacing
RN == Name with RN == undef but this would prevent to
satisfy (1)
%Replace Abb with _ would prevent to satisfy (1) because
Abb values are assigned directly to the SC
%Delete RN == Name andalso
erlang:length(atom_to_list(Abb)) == 1 would prevent to
satisfy (1) because of (E)
%Replace RN == Name with true (NOTE2) would prevent to
satisfy (1) because of (E)
%Replace erlang:length(atom_to_list(Abb)) == 1 with true
(NOTE2) would prevent to satisfy (1) because of (E)
%Replace RN, Name, erlang:length(atom_to_list(Abb)) or 1
with undef would prevent to satisfy (1) because of (E)
%Replace Abb in atom_to_list(Abb) with undef would prevent
to satisfy (1) because of (E)
```

EXECUTION RESULTS:

```
(1) Elements = [{magnesium,mg,639},
                {uranium,u,1132},
                {sulfur,s,113}], Temp = 444

(2) Elements = [{bismuth,bi,271}], Temp = 444
```

SLICING CRITERION

```
SC = [u]

SC = []
```