

PROOF FOUNDATIONS

(W) WEISER DEFINITION OF SLICING:

Given a program P, a slicing criterion $C=\langle v,s \rangle$ 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 ERROR:

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.

```
%-----  
%-----  
%-- bench14.erl  
%--  
%-- AUTHORS:      Anonymous  
%-- DATE:         2016  
%-- PUBLISHED:    Software specially developed to test challenging slicing problems  
%-- COPYRIGHT:    Bencher: The Program Slicing Benchmark Suite for Erlang  
%--                (Universitat Politècnica de València)  
%--                http://www.dsic.upv.es/~jsilva/slicing/bencher/  
%-- DESCRIPTION  
%-- The program presents a set of difficult slicing problems like unreachable clauses in case  
%-- statements or never called function clauses. It receives two inputs of any nature and  
%-- processes them with a suit of case statements and function calls to obtain a final  
%-- result.  
%-----  
%-----
```

```

-module(bench14).
-export([main/2]).

main(X,Y) ->
    Z = case X of
        terminate -> "the end";
        {A,B} -> {[A+B,B-A],3};
        {3,C} -> g(C);
        _ -> {20*3,8}
    end,
    T = 2,
    V = f(T)+h(2)+h(3),
    W = g([X,Y,{X,Y}]),

    Tuple = {Z,W,V},
    Tuple.

g(X) ->
    [_,_,{R,S}] = X,

    case R of

        [1,3] ->
            21;

        [A,B] ->
            (A*B)/9;

        T ->
            T;

        - ->
            f(4)
    end.

f(7) ->
    L = 2+9,
    F = L*3,
    F+L;

```

%Given (A), X is necessary w.r.t. $W=g([_,_,{X,_}])$
 %W cannot be deleted because it is the SC
 %g([_,_,{X,_}) with undef (NOTE3) would prevent to satisfy (1),(2),(3)&(4) because it is the only expression that can assign a value of the SC
 %Replace [X,Y,{X,Y}] with undef (NOTE3) would prevent to reach the SC due to a matching error in the g() function
 %Replace {X,Y} with undef (NOTE3) would prevent to satisfy (1),(2),(3)&(4)
 %Replace X with undef (NOTE3) would prevent to satisfy (1),(2)&(3) because of (E) in the g() function

%Given (A), X is necessary w.r.t. $[_,_,{R,_}] = X$
 %[_,_,{R,S}] = X is necessary because of (A) w.r.t. the case expression
 %Replace [_,_,{R,S}] with _ (NOTE3) would prevent to satisfy (1),(2),(3)&(4) due to an unbound variable error. This error could be avoided replacing the R expression in the case with undef, but it would not be possible to satisfy (1),(2),(3)&(4) by any modifications
 %Replace X with undef (NOTE3) would make not possible to reach the SC because of a matching error
 %Replace R with undef (NOTE3) would prevent to reach the SC because of an unbound variable error. This error could be avoided replacing the R expression in the case with undef, but then would not be possible to satisfy (1),(2),(3)&(4)
 %The case expression cannot be deleted because it is the last expression and in consequence the returned value of the g() function. Delete this expression would prevent to satisfy (1),(2),(3)&(4)
 %R cannot be replaced with undef (NOTE3) because it would prevent to satisfy (1),(2),(3)&(4). This would make the case expression enter always in the third clause and g(X) would always return undef

%This clause cannot be deleted because it would not be possible to satisfy (1)
 %Replace [1,3] with _ (NOTE3) would prevent to satisfy (2),(3)&(4)
 %Replace 1 with _ (NOTE3) would prevent to satisfy (3)
 %Replace 3 with _ (NOTE3) would prevent to satisfy (2)
 %21 cannot be deleted because it is the only expression in the case clause and also one of the possible values of the g() function. Replace it with undef (NOTE3) would prevent to satisfy (1)
 %This clause cannot be deleted because it would not be possible to satisfy (2)&(3)
 %Given (A), A and B are necessary w.r.t. $(A*B)/9$, in consequence [A,B] cannot be deleted
 % $(A*B)/9$ cannot be deleted because it is the only expression in the case clause and also one of the possible values of the g() function. Replace it with undef (NOTE3) would prevent to satisfy (2)&(3)
 %Replace A,B,A*B or 9 with undef would prevent to reach the SC due to a badarith error
 %This clause cannot be deleted because it would not be possible to reach the SC in (4) due to a pattern matching error
 %Given (A), T is necessary w.r.t. T
 %T cannot be deleted because it is the only expression in the case clause and also one of the possible values of the g() function. Replace it with undef (NOTE3) would prevent to satisfy (4)
 %Due to (D1), this clause can be deleted

```
f(4) -> 9;                                     %Given (D1), we have deleted from the program all the calls of the
                                                f() function. In consequence, we can delete also from the program the
                                                definitions of this function: f(4)-> 9 and f(X) -> X
```

```
f(2) -> 7;
f(X) -> X.
```

```
h(X) ->
  case X of
    2-> j({2,4});
    3-> k([4,8]);
    1-> l(107)
  end.
```

```
j(A) -> {X,_} = A, X.
k(B) -> [H|T] = B, H.
l(C) -> C-1.
```

EXECUTION RESULTS:

	SLICING CRITERION
(1) R == [1,3] -> X=[1,3]	SC = 21
(2) R != [1,3] && R == [A,B] -> X = [1,18]	SC = 2
(3) R != [1,3] && R == [A,B] -> X = [27,3]	SC = 9
(4) R != [1,3] && R != [A,B] && T = R -> X = 972	SC = 972

Demonstration 1 (D1)

 In order to execute the clause 4 of the case expression (`_ -> f(4)`) the following constrains need to be fulfilled:

$$R \neq [1,3] \ \&\& \ R \neq [A,B] \ \&\& \ !(T=R) \ \&\& \ _ = R$$

Being A, B and T undefined variables

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

$$!(T=R) \ \text{when } T \ \text{undefined} \ -> \emptyset$$

When a variable is unbound, it can always be assigned in a case clause pattern, in consequence, the clause 3 of the case expression fullfils (F).

Conclusion: The clause 4 of the case expression is not part of the minimal slice