

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.

```
%-----  
%-----  
%-- bench5.erl  
%--  
%-- AUTHORS:      Anonymous  
%-- DATE:         2016  
%-- PUBLISHED:    Software specially developed to test the detection of unreachable  
%--               clauses in functions due to a previous clause always matches or another  
%--               never matches. In this case, the slicer need to notice the structure of  
%--               the input (a list) to avoid clauses.  
%-- COPYRIGHT:    Bencher: The Program Slicing Benchmark Suite for Erlang  
%--               (Universitat Politècnica de València)  
%--               http://www.dsic.upv.es/~jsilva/slicing/bencher/  
%-- DESCRIPTION  
%-- The benchmark has a function receiving two lists as inputs. These lists are processed  
%-- in a call using their elements in a case statement. Another function can be called  
%-- in the case using as input the tail of one of the lists.  
%-----  
%-----
```

```

-module(bench5).
-export([lists/2]).
lists(A,B) ->

    C=f1(A,B).

f1([H1|T1],[H2|T2]) ->
    if

        H1 >= 3 ->

            H2;

            true ->

                H1 + gl(T2)

    end.
gl(0) ->

    3;
gl([]) ->

    5;
gl([1|_]) ->

    0;
gl([H|T]) ->

    1;
gl(_) ->

    7.

```

%Given (A), A and B are necessary w.r.t. C = fn(A,B)
 %C is necessary because it is the SC
 %f1(...) is the only expression that can assign a value to the SC. If we replace it with undef (NOTE2) it would be impossible to satisfy (1),(2),(3),(4),(5)&(6)
 %Given (A), A and B are necessary w.r.t. f1([H1|T1],[H2|T2])
 %Given (A), H1,H2 and T2 are necessary w.r.t. the if expression
 %The if expression is necessary because it is the only expression of the function and defines the returned value of the f1 function. If we replace it with undef (NOTE2) it would be impossible to satisfy (1),(2),(3),(4),(5)&(6)
 %This clause cannot be deleted because it would not be possible to satisfy (1)
 %Replace H1 >= 3 with true (NOTE2) would produce (F) and thus (2),(3),(4),(5)&(6) are not satisfied because of (E)
 %Replace H1 with undef (NOTE2) would produce (F) and thus (2),(3),(4),(5)&(6) are not satisfied because of (E)
 %Replace 3 with undef (NOTE2) would make not possible to satisfy (1) because of (E)
 %Replace H2 with undef (NOTE2) would make not possible to satisfy (1) because H2 is one of the possible return values of the f1() function and consequently one of the possible values of the SC. Neither can this expression be deleted because it is the only expression of the if clause
 %This clause cannot be deleted because SC would never be reached due to a matching error in (2),(3),(4),(5)&(6)
 %Replace H1 + gl(T2) with undef (NOTE2) would make not possible to satisfy (2),(3),(4),(5)&(6) because H1 + gl(T2) is one of the possible return values of the f1() function
 %Replace H1 or gl(T2) with undef (NOTE2) would make not possible to satisfy (2),(3),(4),(5)&(6) due to a badarith error
 %Replace T2 with undef (NOTE2) would make not possible to satisfy (2),(3),(4)&(5) because of (E)
 %This clause cannot be deleted because it would not be possible to satisfy (2)
 %Replace 0 with _ (NOTE2) would make not possible to satisfy (3),(4),(5)&(6) because this clause would fullfils (F)
 %3 is one of the possible return values of gl() necessary to compute one of the returned values of f1()
 %This clause cannot be deleted because it would not be possible to satisfy (3)
 %Replace [] with _ (NOTE2) would make not possible to satisfy (4),(5)&(6) because this clause would fullfils (F)
 %5 is one of the possible return values of gl() necessary to compute one of the returned values of f1()
 %This clause cannot be deleted because it would not be possible to satisfy (4)
 %Replace [1|_] with _ (NOTE2) would make not possible to satisfy (5)&(6) because this clause would fullfils (F)
 %Replace 1 with _ (NOTE2) would make not possible to satisfy (5)&(6) because of (E). This clause would become more general than the next one and would absorb cases that should not.
 %0 is one of the possible return values of gl() necessary to compute one of the returned values of f1()
 %This clause cannot be deleted because it would not be possible to satisfy (5)
 %Replace [H|T] with _ (NOTE2) would make not possible to satisfy (6) because this clause would fullfils (F)
 %1 is one of the possible return values of gl() necessary to compute one of the returned values of f1()
 %This clause cannot be deleted because it would not be possible to satisfy (6)
 %7 is one of the possible return values of gl() necessary to compute one of the returned values of f1()

EXECUTION RESULT:

	SLICING CRITERION
(1) H1 >= 3 -> A=[5,2],B=[18,9]	SC = 18 (H2)
!(H1 >= 3) && H1 < 3 :	
(2) 0 = T2 -> A=[1,2],B=[12 0]	SC = 4 (H1 + 3)
(3) [] = T2 -> A=[1,2],B=[8]	SC = 6 (H1 + 5)
(4) [1 _] = T2 -> A=[1,2],B=[7,1]	SC = 1 (H1 + 0)
(5) [H T] = T2 -> A=[1,2],B=[7,9,2]	SC = 2 (H1 + 1)
(6) _ = T2 -> A=[1,2],B=[6 15]	SC = 8 (H1 + 7)