

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.

```
%-----  
%-----  
%-- bench3.erl  
%--  
%-- AUTHORS:      Anonymous  
%-- DATE:         2016  
%-- PUBLISHED:    Software specially developed to test unreachable clauses.  
%-- COPYRIGHT:    Bencher: The Program Slicing Benchmark Suite for Erlang  
%--               (Universitat Politècnica de València)  
%--               http://www.dsic.upv.es/~jsilva/slicing/bencher/  
%-- DESCRIPTION  
%-- This benchmark consists in a function receiving two tuples with two elements as inputs  
%-- and calling another one named ft with two conditional statements. Note that the third  
%-- clause of the second conditional statement is unreachable.  
%-----  
%-----
```

```

-module(bench3).
-export([tuples/2]).
tuples(A,B) ->
    C=ft(A,B).

ft({X1,X2},{Y1,Y2}) ->

    if
        X1 =< Y2 ->
            R = X1;
        true ->
            R = X2
    end,
    if
        X1 > X2 ->
            Z= Y2 + 3,
            S = 5;
        true ->
            S = X2;
        X2 < 7 ->
            S=53
    end,
    S.

```

%Given (A), A is necessary w.r.t. C=ft(A,B)
 %C cannot be deleted because it is the SC
 %ft(...) 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)
 %Given (A), A is necessary w.r.t. ft({X1,X2},{Y1,Y2})
 %We can delete {Y1,Y2} because the original program throws a matching exception when B is not a tuple with two elements and the SC is never evaluated. Given (W), it is not relevant whether the slice computes some values for the SC or not.
 %{X1,X2} cannot be deleted Given (A), w.r.t. the if expression

%Delete the if expression would prevent to satisfy (1) and (2) because of an unbound variable error (variable S). If we also replace S with undef (NOTE2), it would not be possible to satisfy (1) & (2) because S is the last expression of the function and, in consequence, the returned value of it.
 %This clause cannot be deleted because it would not be possible to satisfy (1).
 %Replace X1 > X2 with true (NOTE2), would prevent to satisfy (2) because this clause would become an absorbent clause (F)
 %Replace X1 with undef would prevent to fulfill (2) due to (E)
 %Replace X2 with undef would prevent to fulfill (1) due to (E)
 %S = 5 cannot be replaced with undef because it assigns one of the possible values to the returned expression of the ft function. Neither can it be deleted because it is the only expression of the clause
 %5 cannot be deleted because it would prevent to satisfy (1)
 %This clause cannot be deleted because it would prevent to satisfy (2)
 %S = X2 cannot be replaced with undef because it assigns one of the possible values to the returned expression of the ft function. Neither can it be deleted because it is the only expression of the clause
 %X2 cannot be deleted because it would prevent to satisfy (2)
 %Due to (D1), we can delete this clause
 %The three S variables (S=5,S=X2,S) can be deleted due to property (C). The two possible branches of the if expression define the S variable and this variable is immediately returned in the next expression.

EXECUTION RESULT:

INPUTS:

SLICING CRITERION

(1) X1 > X2 -> A={2,1}, B={1,9}	C = 5
(2) !(X1 > X2) && (X1 =< X2) -> A={1,4},B={8,6}	C = 4 (C = X2)

Demonstration 1 (D1)

In order to execute the third clause of the if expression (X < 7 -> S = 53) the following constrains need to be fulfilled:

$$!(X1 > X2) \ \&\& \ !(X1 =< X2) \ \&\& \ X2 < 7$$

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

$$!(X1 > X2) \ \&\& \ !(X1 =< X2) \ -> \emptyset$$

Conclusion: The third clause of the if expression is not part of the minimal slice