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.

```
%-------------------------------------------------------------------------------------------
%-------------------------------------------------------------------------------------------
%-- bench10.erl
%--
%-- AUTHORS:      Tamarit
%-- DATE:         2016
%-- PUBLISHED:    https://github.com/tamarit/hackerrank/tree/master/common-divisors (2016)
%-- 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 with pairs of integers and compute the number of common
%-- divisors of each pair. The program returns a list with the number of common divisors
%-- in the same order that the input pairs.
%-------------------------------------------------------------------------------------------
%-------------------------------------------------------------------------------------------
```

```
-module(bench10).
-export([main/1]).
main(L) ->
  Res = calculate(L).



calculate(List) ->
    calculate(List, []).




calculate([{A,B}|T], Acc) ->

    DA = divs(A),
    DB = divs(B),




    calculate(T,




        [sets:size(
            sets:intersection(
                sets:from_list(DA),
                sets:from_list(DB)))
        | Acc]);
calculate([], Acc) ->
    lists:reverse(Acc).

divs(0) ->



        [];

divs(1) ->



        [1];



divs(N) ->
        [1, N] ++ divisors(2,N,math:sqrt(N)).
```

```
%Given (A), L is necessary w.r.t. calculate(L)
%calculate(L) cannot be deleted because it is the only expression of the clause. Replace it with undef (NOTE2) would prevent to reach the SC
%Given (A), L is necessary w.r.t. calculate(List)
%Given (A), List is necessary w.r.t. calculate(List,[])
%calculate(List,[]) cannot be deleted because it is the only expression of the clause. Replace it with undef (NOTE2) would prevent to reach the SC
%Given (A), List is necessary w.r.t. calculate([{A,B}|T], Acc)
%Given (A), B is necessary w.r.t. DB = divs(B)
%Given (A), T is necessary w.r.t. calculate(T,…)

%DB cannot be deleted because it is the SC
%divs(B) assigns the value of the SC directly. Replace it with undef (NOTE2) would prevent to reach (1),(2),(3)&(4)
%Given (A), B is necessary w.r.t. the clauses of the divs/1 function
%Delete calculate(T,…) would prevent to satisfy (4) because the SC would only be reached once because of the lack of the recursive call
%Given (A), T is necessary w.r.t. calculate([{A,B}|T], Acc). Moreover, replace it with undef (NOTE2) would prevent to reach the SC because of a matching error
```

```
%Delete this clause of the divs/1 function would prevent to satisfy (1)
%Replace 0 with _ (NOTE2) would prevent to satisfy (2),(3)&(4) because this clause would fulfill (F)
%[] cannot be deleted because it is the only expression of the clause and, in consequence, one possible returned value. Replace it with undef (NOTE2) would prevent to satisfy (1)
%Delete this clause of the divs/1 function would prevent to satisfy (2)
%Replace 1 with _ (NOTE2) would prevent to satisfy (3)&(4) because this clause would fulfill (F)
%[1] cannot be deleted because it is the only expression of the clause and, in consequence, one possible returned value. Replace it with undef (NOTE2) would prevent to satisfy (2)
%Replace 1 with undef (NOTE2) would prevent to satisfy (2)
%Given (A), N is necessary w.r.t. [1,N] and math:sqrt(N)
%[1, N] ++ divisors(2,N,math:sqrt(N)) cannot be deleted because it is the only expression of the clause and, in consequence, one possible returned value. Replace it with undef (NOTE2) would prevent to satisfy (3)&(4)
%Replace [1,N] with undef would prevent to reach the SC in (3)&(4) because of a bad argument error in the [1, N] ++ divisors(2,N,math:sqrt(N)) expression
%Replace 1 or N in [1,N] with undef (NOTE2) would prevent to satisfy (3)&(4) because of a modification of the returned value of the divs function
%Replace divisors(2,N,math:sqrt(N)) with undef (NOTE2) would prevent to reach the SC in (3)&(4) because of a matching error
%Given (A), 2, N and math:sqrt(N) are necessary w.r.t. the clauses of the divisors(K,N,Q) function
%Replace N in math:sqrt(N) with undef would prevent to reach the SC in (3)&(4) because of a bad argument error
```

```
divisors(K,_N,Q) when K > Q ->
```

```
%This clause is the base case of the function. Delete this clause would prevent to reach the SC because of the generation of an infinite loop
%Given (A), K and Q are necessary w.r.t. guard when K > Q
%Delete guard when K > Q would prevent to satisfy (3)&(4) because this clause would fulfill (F)
%Replace K with undef (NOTE2) would prevent to satisfy (3)&(4) because this clause would fulfill (F) because of (E)
```

```
        [];
```

%Replace Q with undef (NOTE2) would prevent to reach the SC because this clause would become unreachable because of (E) and it would generate and infinite loop
%[] cannot be deleted because it is the only expression of the clause. Replace it with undef (NOTE2) would prevent to reach the SC in (3)&(4) because of a matching error

```
divisors(K,N,Q) when N rem K =/= 0 ->
```

%Delete this clause would prevent to reach the SC in (3)&(4) because of the generation of an infinite loop
%Given (A), K and N are necessary w.r.t. when N rem K =/= 0
%Given (A), Q is necessary w.r.t. divisors(K+1,N,Q)
%Delete when N rem K =/= 0 would prevent to satisfy (3)&(4) because it would make this clause to fulfill (F)
%Replace N rem K or 0 with undef (NOTE2) would prevent to satisfy (3)&(4) because this clause would fulfill (F). Replace both N rem K and 0 with undef (NOTE2) would prevent to satisfy (3)&(4) because the clause would become unreachable
%Replace N or K in N rem K with undef would prevent to reach the SC in (3)&(4) because of a badarith error

```
    divisors(K+1,N,Q);
```

%divisors(K+1,N,Q) cannot be deleted because it is the only expression of the clause. Replace it with undef (NOTE2) would prevent to reach the SC in (3)&(4) because of a matching error in the [1,N]++divisors(2,N,math:sqrt(N)) expression of the divs/1 function or in the [K, N div K] ++ divisors(K+1,N,Q) expression of the divisors(K,N,Q) function
%Given (A), K+1 and N are necessary w.r.t. the divisors(K,N,Q) when N rem K =/= 0 -> … clause
%Given (A), Q is necessary w.r.t. the clause that represents the base case divisors(K,_N,Q) when K > Q ->… of the divisors/3 function

```
divisors(K,N,Q) when K * K  == N ->
```

%Delete this clause would prevent to satisfy (3) because of (E) in the divisors/3 function
%Given (A), K and N are necessary w.r.t. when K * K == N
%Given (A), Q is necessary w.r.t. divisors(K+1,N,Q)
%Delete when N rem K =/= 0 would prevent to satisfy (4) because it would make this clause to fulfill (F)
%Replace K * K or N with undef (NOTE2) would prevent to satisfy (3) because this clause would become unreachable. Replace both with undef would prevent to satisfy (4) because this clause would fulfill (F)

```
    [K] ++ divisors(K+1,N,Q);
```

%[K] ++ divisors(K+1,N,Q) cannot be deleted because it is the only expression of the clause. Replace it with undef (NOTE2) would prevent to reach the SC in (3) because of a matching error in the [1,N]++divisors(2,N,math:sqrt(N)) expression of the divs/1 function
%Replace [K] with undef (NOTE2) would prevent to reach the SC in (3) because of a bad argument error
%Replace divisors(K+1,N,Q) with undef (NOTE2) would prevent to satisfy (3) because of (E)
%Replace K in [K] with undef (NOTE2) would prevent to satisfy (3)
%According to (D1),K+1 and N can be deleted
%Given (A), Q is necessary w.r.t. the clause that represents the base case divisors(K,_N,Q) when K > Q ->… of the divisors/3 function

```
divisors(K,N,Q) ->
```

%Delete this clause would prevent to reach the SC because of a matching error in the divisors/3 function. This error can be avoided by transforming a previous clause in an absorbent clause, but this would prevent to satisfy (4)
%Given (A), K, N and Q are necessary w.r.t. [K, N div K] ++ divisors(K+1,N,Q)

```
    [K, N div K] ++ divisors(K+1,N,Q).
```

%[K, N div K] ++ divisors(K+1,N,Q) cannot be deleted because it is the only expression of the clause. Replace it with undef (NOTE2) would prevent to satisfy (4) because of (E)
%Replace [K,N div K] with undef (NOTE2) would prevent to reach the SC in executions (3)&(4) because of a bad argument error
%Replace K or N div K with undef (NOTE2) would prevent to satisfy (4) because of (E)
%Replace N or K in N div K would prevent to reach the SC because of a badarith error
%Replace divisors(K+1,N,Q) with undef (NOTE2) would prevent to satisfy (4) because of (E)
%Given (A), K+1 and N are necessary w.r.t. clause divisors(K,N,Q) when N rem K =/= 0 -> … of the divisors/3 function

```
%Given (A), Q is necessary w.r.t. the clause that
represents the base case divisors(K,_N,Q) when K > Q ->…
of the divisors/3 function
```

EXECUTION RESULTS

```
(1) L = [[17,0]]
(2) L = [[92,1]]
(3) L = [[23,121]]
(4) L = [[53,247],[12,54]]
```

SLICING CRITERION

```
SC = []
SC = [1]
SC = [1,121,11]
SC = [1,247,13,19]
     [1,54,2,27,3,18,6,9]
```

DEMONSTRATION 1 (D1)

The Q parameter of the function divisors/3 is not modified in any of its function clauses, so its value will always be $Q = math{:}sqrt(N) \rightarrow Q = \sqrt{N}$.

To match the third clause of the function divisors/3, it is necessary to fulfill $K * K == N$. By some transformations using these two premises, we can guarantee that $K = Q$:

$$\textbf{(I)}\ K * K = N \rightarrow K^2 = N \rightarrow K = \sqrt{N} \rightarrow K = Q$$

In this clause, there is another function call to the function divisors/3. This call will always match the base case of divisors/3 because K will always be greater than because of **(I)**:

$$\textbf{(II)}\ K = Q \rightarrow K + 1 > Q$$

For this reason, in function call `divisors(K+1,N,Q) N` can be replaced with undef (NOTE2) because it is ignored by the base case clause `divisors(K,_N,Q) when K > Q -> []`. In the case of K+1, replace it with undef (NOTE2) would make the guard of the base case to be always true, so it would make no difference because $K + 1$ will always be greater than Q because of **(II)** in this call.

Conclusion: K+1 and N are part of the minimal slice.