Exam Software Testing & Verification 2020/2021

Th. 27th May 2021, 08.45–10.45

Write your solution digitally, and upload it through Blackboard. When a diagram is asked, you can draw it on paper and upload a photo of it. Expected format: pdf/txt/md, and usual standard format for photos.

1 Part I (6pt)

1. Two developers have to test a program P, in particular a path σ with length $k>0$ in P's Control Flow Graph.

   Developer A wrote a test t that covers σ through detour.

   Developer B then proposes to add another test so that σ is also covered with side-trip.

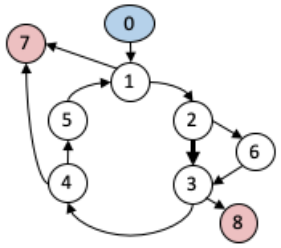
   Question: is developer B's proposal reasonable? Give an argument to support your answer.

2. Consider the program scan shown below —what it exactly does is not so important here. Its CFG is also shown. Numbers in the program's comments refer to the corresponding node-ids in the CFG.

   The programmer wants to test this program, and considers to aim for **full node coverage**. Given the complex control flow of the program this does not seem to be a good idea.

   Describe one critical behavior (of the program) we might **miss** if we go along with the programmer's proposal.

```
1  string scan(int[] dna) {
2    int i=0 ;      /*0*/
3    int sum=0 ;    /*0*/
4    while (i<dna.Length) {   /*1*/
5      sum += dna[i] ;        /*2*/
6      switch(sum % 20) {     /*2*/
7        /*2*/ case 0  : print(sum);   /*6*/
8        /*3*/ case 19 : return "cov";/*8*/
9        /*4*/ case 1  : break
10       /*5*/ default : ; /*5*/
11     }
12     i++ ;            /*5*/
13   }
14   return "clear"   /*7*/
15 }
```



3. Consider again the Control Flow Graph (CFG) of the program Scan() in the question No. 2. The starting node is 0. The exit nodes are marked red.

   How many prime paths pass through the edge 2→3 ? **Explain** how you come up with this answer. Try to use your insight to solve this problem, rather than just brute forcely simulating the Prime Path finding algorithm fram Ammann & Offutt's book.

4. Consider the program below. Given an array a and a value rho which is an element of a, the program returns an index of a pointing to this rho in a.

   int IndexOf(**int**[ ] a, **int** rho)

   Give a formal specification of this program in terms of pre- and post-conditions.

5. Consider the program below. Given an array a whose elements are sorted ascendingly, the program checks whether the array contains duplicate elements. E.g. $[1, 2, 2, 3]$ is such an array. The program should return true on this example.

   bool HasFlatRegion(**int**[ ] a)

   Give a formal specification of this program in terms of pre- and post-conditions.

6. Consider an app to diagnose if a person is ill. The app poses ten questions $Q_1 ... Q_{10}$ to the user, e.g. whether the user has fever. All questions are binary (the answer is a yes or no). **Additionally**, the diagnosis also depends on these factors:

   (a) The user gender.

   (b) The user age.

   (c) The user's proximity with respect to other users of this app in the last two weeks. By the 'proximity' of two users we mean how close they are in terms of the physical distance between them.

   The app is new, so its developer wants to test it. Propose how to apply paritition-based testing to the app:

   (a) What should we take as the characteristics?

   (b) What should be the blocks?

   (c) Which coverage criterion should we use? Mention one strength and one weakness of your proposal.

7. Consider a program tax(x, y, z) that we want to test. The domains of x, y, and z are partitioned as follows, with $k > 2$:

x	is partitioned into k blocks:	$\mathbf{X_1}, \mathbf{X_2}, ..., X_k$
y	is partitioned into two blocks:	$\mathbf{Y_1}, \mathbf{Y_2}$
z	is partitioned into k blocks:	$\mathbf{Z_1}, \mathbf{Z_2}, ..., Z_k$

   The red/embolden blocks can combine to affect the behavior of the program tax(). So the developer wrote the following test set that would exercise all pairs between these embolden blocks:

   tb_0 : (X_1, Y_1, Z_1)
   tb_1 : (X_1, Y_2, Z_2)
   tb_2 : (X_2, Y_1, Z_2)
   tb_3 : (X_2, Y_2, Z_1)

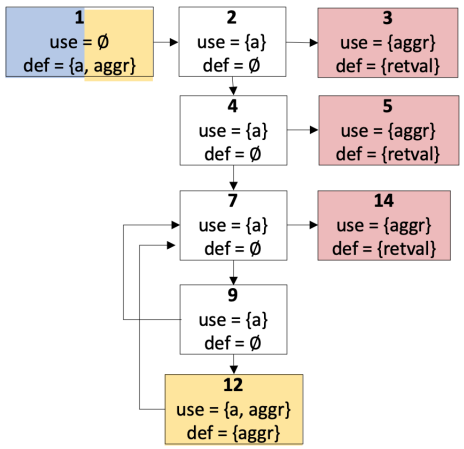
   (a) Give a reasonable proposal on how to cover the remaining blocks.

   (b) What is the minimum number of tests that would be needed in your proposal above.

(c) Mention one strength and one weakness of your proposal.

8. Consider the program below that calculates the sum of all positive elements of an array a. To test this program we will focus on its dataflows with respect to the variables *aggr* and a. The program's Control Flow Graph (CFG) is shown next to it, decorated by the defs and uses of *aggr* and a. The node-numbers correspond to the line numbers of the insrtuctions they represent. Node-1 is the starting node in this CFG. The exit nodes are marked red. Nodes with a def to *aggr* are marked yellow, and one with a def to a is marked blue.

```
1  int sumA(int aggr, int[] a) {
2    if (a == null)
3        return aggr ;
4    if (a.Length==0)
5        return aggr ;
6    int k=0 ;
7    while (k<A.Length) {
8      k++ ;
9      if (a[k-1]<=0)
10         continue ;
11     else
12         aggr = aggr + a[k-1] ;
13   }
14   return aggr ;
15 }
```



We want to apply data-flow based testing to this program. Your task:

(a) List all the du-paths of *aggr* and the du-paths of a. Separate these two groups.

(b) Give a recommendation: which data-flow based coverage should we strive for to test the above program. It should be a reasoanble proposal, and give an argument why your proposal is reasonable.

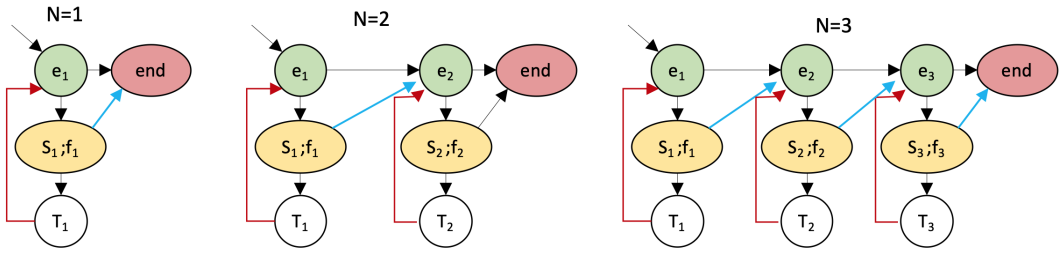
(c) Which of the paths listed in (a) should be included as the your Test Requirement (TR) for your recommendation in (b) ? The TR should be minimum in size.

2 Part II (4 pt)

1. (1.2pt) A programmer wrote a program with a series of N number while-loops. The structure is shown below. All statements S_i and T_i are just simple assignments, and let's assume the program does not throw any exception.

```
1  P(...) {
2    while(e1){
3      S1
4      if(f1) break ;
5      T1
6    }
7    while(e2){
8      S2
9      if(f2) break ;
10     T2
11   }
12   ... // more while-loops with the same structure as above; total there are N such loops
13 }
```

Because the program looks complex. programmer A considers to test the program thoroughly by requiring full prime path coverage, but would like to know first how many prime paths he/she has to cover for different N. The control flow graphs (CFGs) of the program for several N are shown below. The starting node is the node e_1, the exit node is end.



Questions/tasks:

(a) There are 5 prime paths for N=1. How many cyclic and non-cycic prime paths are there for N=2. Explain how you get your numbers.

(b) Programmer B proposes to restrict the testing to cover only all consecutive/connected edges up to 3 consecutive edges. An example of such consecutive edges is:

$$\text{edge } (S_1; f_1) \to T_1 \text{ followed by edge } T_1 \to e_1 \text{ followed by edge } e_1 \to end$$

For N=1 programmer B's proposal is actually just as strong as programmer$A$'s proposal, because all prime paths for N=1 consist of 3 edges.

For higher N, **describe a scenario** that is guaranteed to be covered if we insist on full prime path coverage, but might be **missed** if we follow programmer's B's proposal.

(c) (hard) Give a general formula for the total number of prime paths in $P()$ for a given $N \geq 1$.

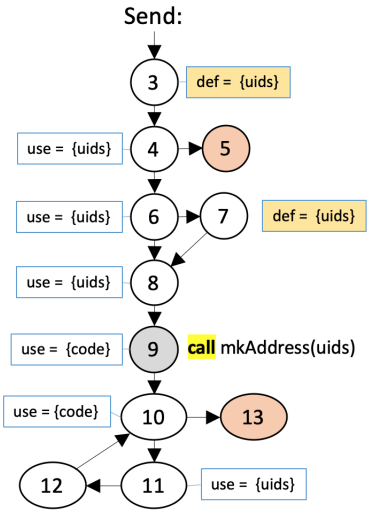
(d) If the **break**s except the last one are actually infeasible (so, in the above picture, all the blue edges are unfeasible), what is the **smallest number** of **tests** needed to cover all remaining prime paths?

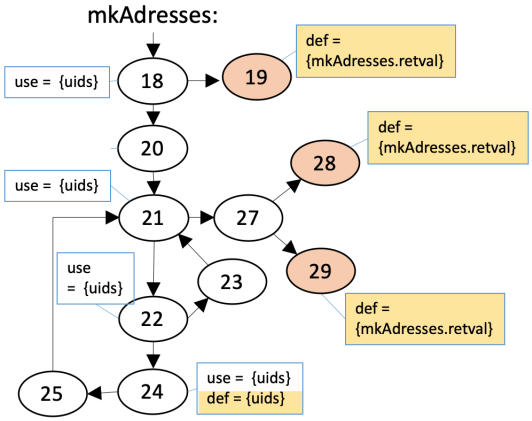
Suppose U is such a minimal test set. Mention one weakness of literally adopting this U as our test set.

2. (1.2pt) The code below shows a procedure/method Send(*uids*) to send an emails to members of the Utrecht University. The procedure takes an array *uids* containing all user-ids of these members. The procedure uses another procedure called mkAddress that replace every user-id in *uids* with its email address. Some special cases such as when the user is the admin, and when the user-id turns out to be empty are also dealth with. The admin should not be mailed, and an empty user-id should be ignored.

   The control flow graphs (CFGs) of each procedure are also shown, along with the defs and uses of the coupling variables between these two procedures. The numbers in the CFGs refer to the instructions in the corresponding line numbers in the code.

```
1  module A {
2   ...
3   void Send(String[] uids) {
4     if(uids==null || uids.Length==0)
5       return ;
6     if(uids[0]=="admin")
7       uids[0] = "" ;
8     int i=0 ; int N=uids.Length ;
9     int code = mkAddresses(uids) ;
10    while(i<N && code>0) {
11      sendEmailTo(uids[i]) ;
12      i++ ;
13    }
14  }}
15
16 module B {
17  int mkAddress(String[] uids) {
18    if (uids.Length<=1) 💣
19      return -1 ;
20    int i=0 ; int count=0 ;
21    while (i<=a.Length) {
22      if (uids[i]=="")
23        { i++ ;continue ; }
24      uids[i] = uids[i]+"@uu.nl" ;
25      i++ ; count++ ;
26    }
27    if(count>1) 💣
28      return count ;
29    return 1 ;
30  }}
```





   We treat the return statements in mkAddress() as assignments to a special variable mkAddress.retval. If a node in the CFGs has both a use and a def of the same variable, the use occurs in the corresponding code fragment before the def.

   Your tasks:

   (a) There are two coupling variables in the above setup: *uids* and mkAddress.retval. List the *coupling du-paths* for each of these variables. Group the paths per variable using a table as shown below:

Coupling var	coupling du-paths
uids	... list here the counpling du-paths of uids
mkAddress.retval/*code*	...list here the counpling du-paths of mkAddress's return value.

(b) After unit-testing Send() in isolation (by mocking mkAddress()), porgrammer A now wants to test it again, but this time using the real mkAddress(). He applies integration testing, aiming at full *All Couping-Defs Coverage* (ACDC).

Programmer B has a different opinion, and wants to have full *All Coupling-Uses Coverage* (ACUC).

- Explain the difference between ACUC and ACDC.
- Then, specify a smallest subset of all the du-paths from (a) that should be covered to get full ACDC.
- And the same for ACUC: give a smallest subset of all the du-paths from (a) that should be covered to get full ACUC.

(c) The program mkAddress() has two errors (marked with bombs).

The condition in line 18 will cause Send() to supress sending an email if *uids* e.g. a singleton ["*harry*"].

The condition in line 27 should use $<$ rather than $>$. The error would cause Send() to still iterate over *uids* even if it consists of only empty user-ids, e.g. ["","",""].

Which of the coupling paths in (a) are *necessary* in order to trigger and observe these errors? We assume the testers observe errors by observing the behavior of Send().

Motivate your answers.

3. (0.8pt) Consider the method SetNewPrice() in the class ShopManager shown below. The method gets the item currently selected by the user, to apply a new price to it, if the new price is lower that its current price.

   The selected item is of type Item, though it may also be a subclass of it, namely SaleItem. The class Item has a getter and a setter for its price, but note that these are overriden by SaleItem.

```
1  class ShopManager {
2    ...
3    void SetNewPrice(int p) {
4      Item item = GetSelectedItem() ;
5      int oldp = item.GetPrice() ;
6      if(oldp > p)
7        item.SetPrice(p) ;
8      Console.Println(oldp) ;
9      int newp = item.GetPrice() ;
10     Console.Println(newp) ;
11   }
12 }
13 class Item {
14   string name ;
15   int price ;
16   ...
17   public virtual void SetPrice(int p) { price = p ; }
18   public virtual int GetPrice() { return price ; }
19 }
20
21 class SaleItem : Item {
22   int discount ;
23   public void SetPrice(int p) { price = p ; discount = 2 ;}
24   public int GetPrice() { return Max(1,price - discount) ; }
25   ...
26 }
```

   (a) The method ShopManager() contains one *coupling sequence* over the variable *item*. Which pair of locations form this coupling sequence? Why is this pair form a coupling sequence?

   (b) List all binding triples of the coupling sequence in (a). Assume that we only have one subclass of Item, namely SaleItem.

   (c) Consider the following binding triples that appear inside some method `foo()` that calls methods of $A1$ (or $A2$). The corresponding coupling paths are also given. In all cases, the context variable is o. All named paths are distinct.

binding triples			
coupling sequence	runtime type of context-var	coupling variables	coupling paths
C_1	A1	{o.name, o.age}	σ_1 for o.name and σ_2 for o.age
C_2	A1	{o.name, o.age}	τ_1 for o.name and τ_2 for o.age
C_2	A2	{o.name}	τ_3

   Explain the difference between requiring All-Poly-Classes (APC) coverage versus All-Coupling-Defs-Uses (ACDU) in the above example. You can explain the difference in terms of which cases would be covered by each, and which ones could be missed.

4. (0.8pt) Consider a program $Q(s)$ that takes a string s as a parameter. The string is an HTML-like fragment describing a logical formula. The syntax is described by the context free grammar shown below. The rightmost column lists the names of the rules.

S	$\rightarrow$	"true"	(RS1)
S	$\rightarrow$	Var	(RS2)
S	$\rightarrow$	$Math$	(RS3)
$Math$	$\rightarrow$	"⟨any⟩" Var ":" S "&&" S "⟨/any⟩"	(RM)
Var	$\rightarrow$	"x"	(RV)

S is the start symbol. Quoted symbols (red) are terminals.

If we ignore the oracle part, a test for Q consists essentially of just an instance of s.

(a) Give a single test for Q (so, an instance of s) that would cover all terminals, all non-terminals, and all production rules of the above grammar. Show the derivation **tree** of this instance of s as a proof of the requested coverage.

(b) Give N tests, $2 \leq N \leq 3$, for Q (so, N instances of s) that would give full Pair-wise Rule Coverage. Explain why your test set fulfills this requirement.

(c) Does your test set from (b) also give full *Each Rule-Rule Coverage* (ERRC)? If you think that is so, explain why. Else, mention two requirements of ERRC which are not met by your tests in (b).