DD2459 Software Reliability
Take Home Examination 2020

Name	Bryan Leow Xuan Zhen
Number of Pages (excluding cover page)	9
T-number	960107T377
Personal Mobile Number	0728707109
E-mail addresses	bryanlxz@kth.se blxz96@gmail.com xleow002@e.ntu.edu.sg

Bryan Leow Xuan Zhen (bryanlxz@kth.se)

Question 1

(i)

```
/*@ requires substr!= null && superstr!= null

  @ ensures ((\exists int k; 0<=k<=superstr.length-substr.length;
            (\forall int i,j ; 0<=i<=substr.length-1 && k<=j<=k+substr.length-1; substr[i] == superstr[j])) &&
            (substr.length <= superstr.length)) ==>
            \result == true
            &&
            ((\forall int k; 0<=k<=superstr.length-substr.length;
            (\exists int i,j ; 0<=i<=substr.length-1 && k<=j<=k+substr.length-1; substr[i] != superstr[j])) ||
            (substr.length > superstr.length)) ==>
            \result == false
  @*/
```

(ii)

```
/*@ requires mainstr!= null && suffixstr!=null

  @ ensures ((\forall int i,j; 0 <= i <= suffixstr.length-1 && mainstr.length-suffixstr.length <= j <= mainstr.length-1; suffixstr[i] == mainstr[j]) &&
            (suffixstr.length <= mainstr.length)) ==>
            \result == true
            &&
            ((\exists int i,j; 0 <= i <= suffixstr.length-1 && mainstr.length-suffixstr.length <= j <= mainstr.length-1; suffixstr[i] != mainstr[j]) ||
            (suffixstr.length > mainstr.length)) ==>
            \result == false
  @*/
```

(iii)

```
/*@ requires myCharSet!= null , myChar
  @ requires (\forall int i; 0 <= i <= myCharSet.length-2; myCharSet[i]!=myCharSet[i+1])

  @ ensures (\exists int i; 0 <= i <= \old (myCharSet.length)-1; \old (myCharSet[i]) == myChar) ==>
            \result == myCharSet == \old (myCharSet)
            &&
            (\forall int i; 0 <= i <= \old (myCharSet.length)-1; \old (myCharSet[i]) != myChar) ==>
            (myCharSet != \old(myCharSet)) &&
            (myCharSet.length = \old (myCharSet.length) + 1) &&
            (\exists int i; 0 <= i <= myCharSet.length-1; myCharSet[myCharSet.length-1] == myChar) &&
            \result == myCharSet
   @*/
```

(iv) From the precondition, we can see that:
myCharSet can take any non-null array with unique characters as inputs for test case formulation.
myChar can take any character as inputs for test case formulation.

For pairwise test case, we first need to set the defaults for *myCharSet* and *myChar.*
Since pairwise testing includes 0-wise testing, 1-wise testing and 2-wise testing, our test suite must contain the following:
a default test case, a test case whereby only *myCharSet* is taking non-default value, a test case whereby only *myChar* is taking non-default value and a test case whereby both *myCharSet* and *myChar* is taking non-default values.

Then, we evaluate the postcondition after execution of each test case to ensure that should there exist an index *i* in *\old (myCharSet)* where *\old (myCharSet[i]) == myChar,* then *myCharSet* is unmodified and *\result == myCharSet == \old*

	(myCharSet). Else, *myCharSet* will be modified to take in *myChar* and the length of *myCharSet* will be incremented by 1. Subsequently, *\result == myCharSet && myCharSet! = \old (myCharSet).*
(v)	Best way to look at the dimensionality of system under test is to see *myCharSet* as one dimension and *myChar* as another dimension. As mentioned earlier, *myCharSet* can take any non-null array with unique character. Should we be considering the possible combination for *myCharSet,* there are infinite possibilities. Hence, the best way is to just consider *myCharSet* as a variable and myChar as another. Hence, n = 2.

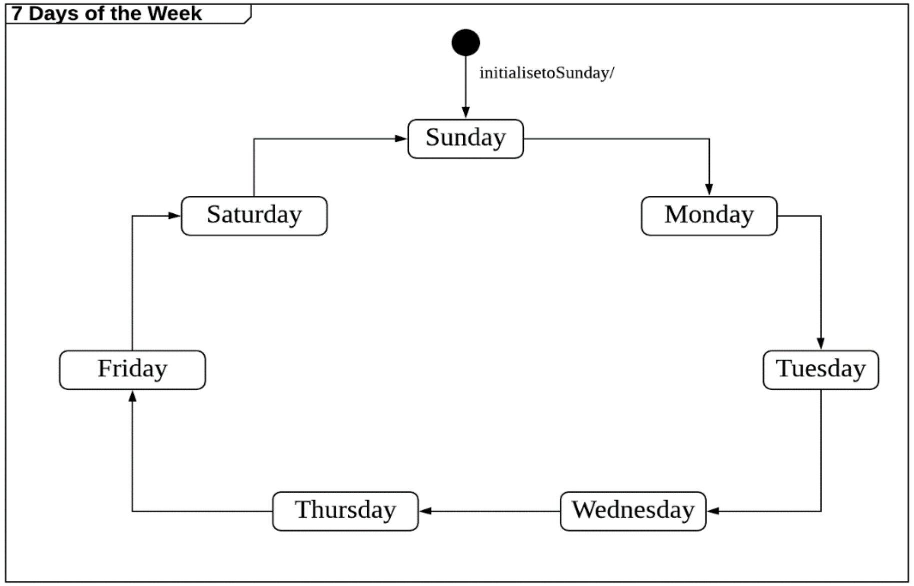
Question 2

(a)	**G (saveMoney -> F(rich))**
(b)	**G (brake -> X^2(stop) | X^3(stop) -> (stop ∪ !brake))**
(c)	**G (green | red) & G (! (green & red)) & red & G (red & press -> X(press) -> X^2(green) & green & press -> X(press) -> X^2(red))**
(d)	

Test Case	**Test Case Values**	**Expected Values**
TC1	**red & press -> X(press)**	**X^2(green)**
TC2	**green & press -> X(press)**	**X^2(red)**

The aspect of (c) that we cannot test is the time taken between the current state in which the button is pressed for the first time and the next state in which the button is pressed for the second time.

Bryan Leow Xuan Zhen (bryanlxz@kth.se)

(e)	**State Machine Diagram**	
	"If tomorrow is Sunday then the day before yesterday was Wednesday"	**G (X(Sunday) -> X^5 (Wednesday))**

According to the false PTPL temporal formula, **G (X(Sunday) -> X^5 (Wednesday)).**

<u>Counterexample:</u>

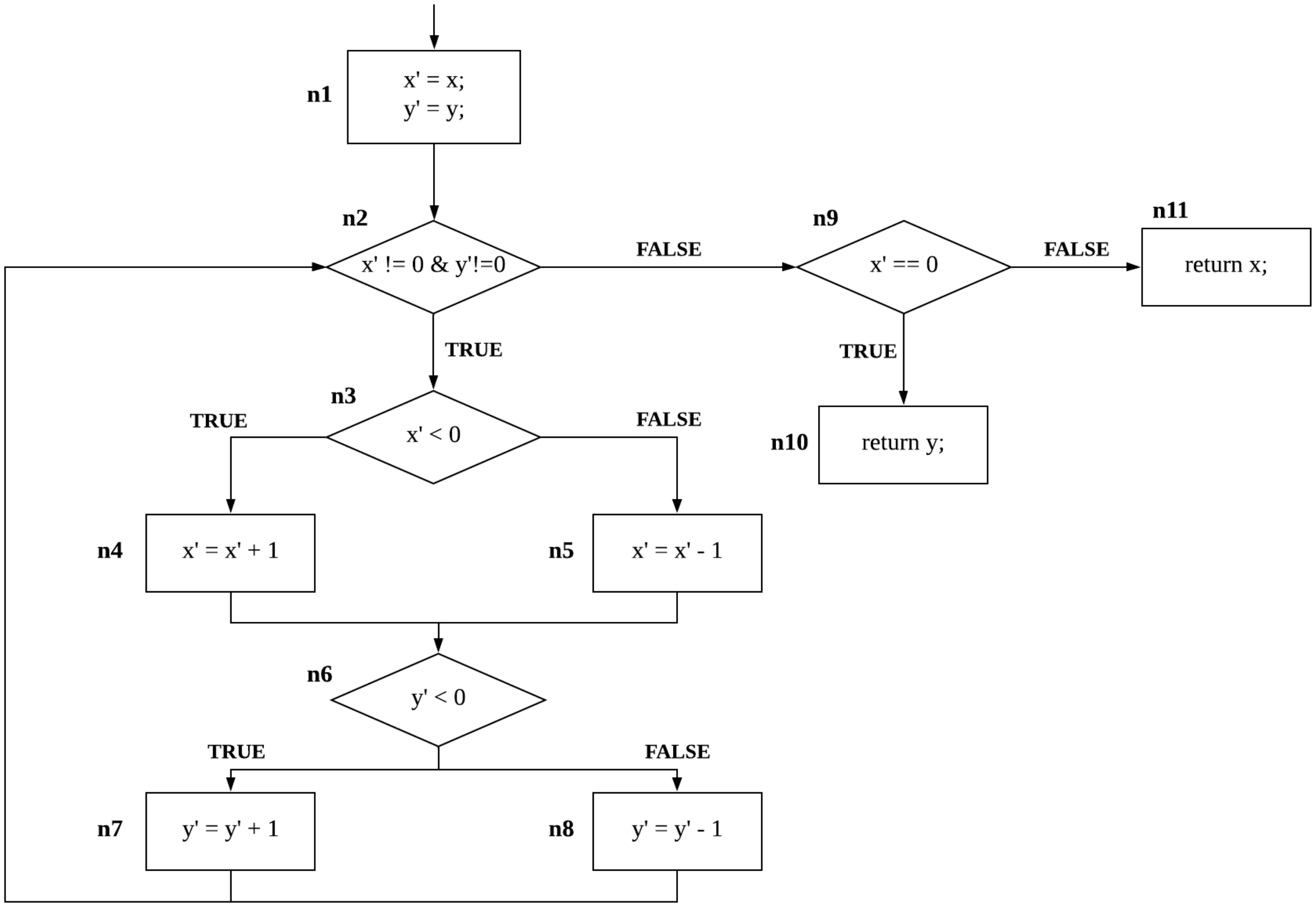
From the state machine diagram, assume X(Sunday).

Then, **X(Sunday) -> X^2 (Monday) -> X^3 (Tuesday) -> X^4 (Wednesday) -> X^5 (Thursday).**

Therefore, **!G (X(Sunday) -> X^5 (Wednesday)).**

Question 3

(a)


n1
x' = x;
y' = y;
n2
x' != 0 & y'!=0
FALSE
n9
x' == 0
FALSE
n11
return x;
TRUE
TRUE
n10
return y;
n3
x' < 0
TRUE
FALSE
n4
x' = x' + 1
n5
x' = x' - 1
n6
y' < 0
TRUE
FALSE
n7
y' = y' + 1
n8
y' = y' - 1

(b)	There are 13 edge requirements in ER. **ER_TR1: n1, n2** **ER_TR2: n2, n3** **ER_TR3: n3, n4** **ER_TR4: n3, n5** **ER_TR5: n4, n6** **ER_TR6: n5, n6** **ER_TR7: n6, n7** **ER_TR8: n6, n8** **ER_TR9: n7, n2** **ER_TR10: n8, n2** **ER_TR11: n2, n9** **ER_TR12: n9, n10** **ER_TR13: n9, n11**
(c)	There are 2 requirements in RER. **RER_TR1: n1, n2, n3, n4, n6, n7, n2, n9, n10** **RER_TR2: n1, n2, n3, n5, n6, n8, n2, n9, n11** Covered by both **RER TR1** and **TR2 : ER TR1, TR2** and **TR11** *Covered by only **RER_TR1 : ER TR3, TR5, TR7, TR9** and **TR12*** *Covered by only **RER_TR2 : ER TR4, TR6, TR8, TR10** and **TR13*** Since there are TR in ER that is covered only by RER_TR1 or RER_TR2, the smallest subset of RER has a size of 2.

(d) The goal of structural testing is to exercise a minimum collection of locations. “Enough” testing is defined in terms of coverage rather than test suite size in structural testing. Hence, RER is a better requirement set for testing than ER because it yields the same edge coverage with a smaller test suite size (2 < 13) and this will help to save a lot of time, allowing the coverage to be more easily and accurately defined.

(e)

Test Case	Test Case Values	Setup Values	Teardown Values	Expected values
TC1	**x = -1 , y = -2**	**x’= x, y’= y**	**x’= x, y’= y**	**-2 (since return y ⇔ return -2)**
TC2	**x = 2 , y = 1**	**x’= x, y’= y**	**x’= x, y’= y**	**2 (since return x ⇔ return 2)**

(f) There are 16 test requirements for E^2R.

E^2R_TR1: n1, n2, n3
E^2R_TR2: n1, n2, n9
E^2R_TR3: n2, n3, n4
E^2R_TR4: n2, n3, n5
E^2R_TR5: n3, n4, n6
E^2R_TR6: n3, n5, n6
E^2R_TR7: n4, n6, n7
E^2R_TR8: n4, n6, n8
E^2R_TR9: n5, n6, n7
E^2R_TR10: n5, n6, n8
E^2R_TR11: n6, n7, n2
E^2R_TR12: n6, n8, n2
E^2R_TR13: n7, n2, n9
E^2R_TR14: n8, n2, n9
E^2R_TR15: n2, n9, n10
E^2R_TR16: n2, n9, n11

Bryan Leow Xuan Zhen (bryanlxz@kth.se)

(g)	There are 5 test requirements for RE^2R. **RE^2R_TR1: n1, n2, n3, n4, n6, n7, n2, n9, n10** **RE^2R_TR2: n1, n2, n3, n4, n6, n8, n2, n9, n11** **RE^2R_TR3: n1, n2, n3, n5, n6, n7, n2, n9, n10** **RE^2R_TR4: n1, n2, n3, n5, n6, n8, n2, n9, n11** **RE^2R_TR5: n1, n2, n9, n10** Covered by **RE^2R TR 1, 2, 3, 4 : E^2R TR1** *Covered by only **RE^2R TR 5 : E^2R TR2*** Covered by both **RE^2R TR 1, 2 : E^2R TR3, TR5** Covered by both **RE^2R TR 3, 4 : E^2R TR4, TR6** *Covered by only **RE^2R TR 1 : E^2R TR7*** *Covered by only **RE^2R TR 2 : E^2R TR8*** *Covered by only **RE^2R TR 3 : E^2R TR9*** *Covered by only **RE^2R TR 4 : E^2R TR10*** Covered by both **RE^2R TR 1, 3 : E^2R TR11, TR13** Covered by both **RE^2R TR 2, 4 : E^2R TR12, TR14, TR16** Covered by **RE^2R TR 1, 3, 5 : E^2R TR15** Since there are TR in E^2R that is covered only by RE^2R TR1, TR2, TR3, TR4 or TR5 alone, the smallest subset of RE^2R has a size of 5.

Bryan Leow Xuan Zhen (bryanlxz@kth.se)

(h) RE^2R is more effective than RER because it allows more coverage in nature. From the requirement, we can see that RER only allows us to check the case where x' and y' are both negative or are both greater than or equal to zero. However, RE^2R allows us to check both or them are positive or negative or only one of them is positive and another negative. By virtue of this, RE^2R is more effective than RER.

(i) The mathematical function is defined as

$$f(x,y) = \begin{cases} y, & if\ |y| \geq |x| \\ x, & if\ |y| < |x| \end{cases}$$

(j)

Test Case	Test Case Values	Setup Values	Teardown Values	Expected values
TC1	**x = -1 , y = -2**	**x'= x, y'= y**	**x'= x, y'= y**	**-2 (since $|-2| \geq |-1|$)**
TC2	**x = 2 , y = 1**	**x'= x, y'= y**	**x'= x, y'= y**	**2 (since $|1| < |2|$)**