CSE 4/586: Project 1

Name:

Due Date *[2017-10-22 Sun]*

1 Synchronized Round Consensus Algorithm

Every process broadcasts (to all other processes, including itself) its initial value v_i. In a synchronous network, this can be done in a single "round" of messages. After this round, each process decides on the minimum value it received.

If no faults occur, this algorithm is correct. In the presence of a crash fault, however, a problem can arise. In particular, if a process crashes *during* a round, some processes may have received its (low) initial value, but others may not have. (Note that the channels are always assumed to be fault-free; they deliver messages reliably once a message is put to the channel.)

1.1 Write a PlusCal program to represent this naive algorithm. (20 points)

Use the template in the last page as your starting point, and fill in the redacted parts. Use the toolkit to translate your code to TLA+ and model-check for correctness.

1.2 Model-check safety properties with TLA+ (30 points)

- Write and test an *invariant* property to capture the Agreement property of the consensus protocol.
- Agreement property should be satisfied when *FAILNUM=0*, i.e., when no node is allowed to fail. The property will fail to be satisfied when *FAILNUM>0*.

- Write in the comments section, after the "==========" line, your findings/observations about how the Agreement property is violated.

1.3 Write a PlusCal program to achieve consensus with crash faults (50 points)

To address crash faults, consider this simplifying assumption: say that at most 1 process can crash. How can we modify the algorithm to handle such a failure? (Note again that the channels are always fault-free; they deliver messages reliably once a message is put to the channel.)

Answer: by using 2 rounds. In the 1st round, processes broadcast their own initial value. In the 2nd round, processes broadcast the minimum value they heard. Each process then decides on the min value among all the sets of values it received in the 2nd round.

If the one crash occurs during the first round, the second round ensures that all processes have the same set of values from which to decide. Else, if the one crash occurs during the second round, the first round must have completed without a crash and hence all processes have the same set of values from which to decide.

Without knowing/referring-to FAILNUM, modify your first PlusCal algorithm to achieve consensus in the presence of crash faults. The key observation is that if no crash occurs during a round, all processes have the same set of values from which to decide and they correctly decide on the same minimum value.

2 Submission

Your TLA+ files should be named *syncCon1.tla* and *syncCon2.tla* . Your model's name should be the default name *Model_1* (do not name your model file differently). Generate a pdf print of your TLA+ program using the "Produce Pdf version" from the TLA+ menu. (This will get included in your submission as it is created under the ".toolbox" directory.)

Now create a zip file from the ".tla" file and the corresponding ".toolbox" directory. **Name the zipfile as: proj1.zip**

You will use the submit command (*submit_cse*486 or *submit_cse*586 respectively) to submit your work. The submit command instructions are here: `http://wiki.cse.buffalo.edu/services/content/submit-script`

```
1  ---------- MODULE syncCon1 ----------
   Synchronized consensus
5  EXTENDS Integers, Sequences, FiniteSets, TLC
6  CONSTANTS N, FAILNUM
7  ASSUME N ≤ 5 ∧ 0 ≤ FAILNUM ∧ FAILNUM ≤ 2
8  Nodes ≜ 1 .. N

11 --algorithm syncCon1
12 { variable FailNum = FAILNUM,   Initialization block
13           up = [n ∈ Nodes ↦ TRUE] ;   nodes are up
14           pt = [n ∈ Nodes ↦ 0] ;   nodes are at round 0
15           t = [n ∈ Nodes ↦ FALSE] ;   nodes are not terminated
16           d = [n ∈ Nodes ↦ −1] ;   nodes are not decided
17           mb = [n ∈ Nodes ↦ {}] ;   nodes have mailbox as emptyset

19    define {
20    SetMin(S) ≜ CHOOSE i ∈ S : ∀ j ∈ S : i ≤ j
21     }

23    macro MaybeFail( ) {
24       if ( FailNum > 0 ∧ up[self] )
25         { either
26            { up[self] := FALSE ; FailNum := FailNum − 1 ; }   Node may fail
27           or skip ; } ;   or not
28     }

30    fair process ( n ∈ Nodes )
31    variable v = 0, pv = 0, Q = {} ;
32    {
33 P: if ( up[self] ) {
34       v := self ;   value is set to your id
35       Q := Nodes ;
36 PS:  while ( up[self] ∧ Q ≠ {} ) {   send vote to mb[p] one by one; this node can fail in between
37          with ( p ∈ Q ) {
38
39
40
41          } ;
42       } ;   end_while
43       if ( up[self] ) pt[self] := pt[self] + 1 ;   move to next round
44 PR:    await (up[self]                                   wait for others to move
45       d[self] :=
46       t[self] :=
47    } ;   end_if
48  }   process
49 }
```