2008 Exam Solutions

December 3, 2010

Problem 1

1.b If we have a perfect failure detector P, we can build consensus (Algorithm from the slides). Also, we know that we can implement *NBAC* using consensus and P. Hence, the answer is *yes*. If we have eventually perfect failure detector $\diamond P$, we *cannot* build *NBAC* because we do not have majority of correct processes guaranteed, and we cannot build consensus from $\diamond P$.

1.d Algorithm works as follows:

- Every process proposes some value, 1 or 0, and keeps it in a variable called $prop$.
- On propose, every process *bebBroadcast*s it's $prop$ to all.
- When process p_i gets a message from process p_j with its proposal, via *bebDeliver*; then p_i sets its $prop_i = prop_i * prop_j$.
- If $prop_i$ becomes 0, process i launches *UniConsPropose* with value 0, if it has not done so yet.
- Also, on every round p_i runs *checkCrash*. If *checkCrash* returns 1, some process has failed. Then p_i sets $prop_i$ to 0, and launches *UniConsPropose* with 0, if it has not done so yet.
- If *checkCrash* did not return 1, and p_i has heard from all processes, and they all proposed 1, then p_i proposes *UniCons* with 1 at the end.
- On *UniConsDecide* all processes *NBACDecide*

Pseudo:

```
init:
    waitingFor = \Pi; cs=true; prop=1; done=false

upon cs=true
    if (checkCrash() == 1) then
       cs = false
       v = 0
    endif

upon nbacPropose(v)
    prop = v
    bebBroadcast(v)

upon bebDeliver(from, v)
    waitingFor = waitingFor \ from;
    prop = prop * v
```

```
upon (done=false) and ((prop == 0) or(waitingFor == empty))
    cs = false
    done = true
    ucPropose(prop)

upon ucDecide(v)
    nbacDecide(v)
```

Commit validity: A process will only launch *UniCons* with 1 if all processes including itself have proposed 1; All other processes will do the same; Therefore, *UniCons* only has 1's in it, and by the validity property of *UniCons*, it must decide 1, and all processes *NBACDecide* this result.

Abort Validity: A process will only launch *UniCons* with 0 if it has 0 as initial value, or it received 0 from another process, or it noted that there was a crash in the system; Then it is possible that *UniCons* will decide 0, and only in this case will processes *NBACDecide* 0.

Agreement: All processes check for a crash in the system at every round, so if there was a crash, they will all detect it and set their *prop* to 0, and propose 0 to *UniCons*. Then by validity property of *UniCons* they must all decide 0. If there were no crashed processes, then by the property of perfect links, all processes eventually *bebDeliver* all proposals from all processes. If any one process (or more) proposed 0, then at the end (due to multiplication by zero) all processes will have $prop = 0$, and will all *UniConsPropose* 0. Then, *UniCons* will decide 0, and all processes will *NBACDecide* 0. If all processes proposed 1, then $prop = 1$, and similarly all processes decide 1.

Termination: Follows from the termination properties of *beb* and *UniCons*.

Problem 2

2.a All processes propose their values and broadcast it to other processes. At the end of round 1, each process adopts the the value proposed by process with the lowest id among the messages received. In round 2, if there were no crashes in round 1, processes decide those values that they adopted in round 1. If there was a crash, processes exchange messages again, re-sending their adopted values. Again they adopt the value received from the process with the lowest id and decide upon this value.

Proof: If there are no crashes, p_2 and p_3 adopt the proposed value of p_1 in round 1. Then in round 2 they all decide it, so consensus properties are satisfied.

If there is one crash, and either p_2 or p_3 crashes, then it has no effect on algorithm, and processes decide as in the previous sentence.

If p_1 crashes in round 1, it is possible that not all processes got p_1's value. Thus, processes exchange values again, and both decide in round 2.

2.b Every process keeps a proposal value. In each round, every process first sends the proposal to all. Then, in the set of received values, the process adopts as its proposal value the value of the process with the minimal id. Each process decides in the first round after the *all-synchronous* round, i.e. the round in which the process did not detect any crash.

2.c Sketch of Proof: Regardless on the number of crashes, there will be some round i at which no process crashes. The highest value for i in some execution E is $f(E) + 1$. Thus, all processes will decide by round $f(E) + 2$. At round i, the correct remaining processes may have different values stored in their proposed variable, depending on which processes crashed earlier, and whether some of their messages were delivered before they crashed. The first all-synchronous round will update all these values such that all processes will have the same value – the one of the lowest-id remaining process. They will all decide in round $f(E) + 2$.

2.d No, this is not possible. The proof is by indistinguishable executions. Execution A: no process crashes, all receive values from all other processes (in further text, $v1$, $v2$, $v3$).

Execution B: p_1 crashes, p_2 receives $v1$, $v2$, $v3$, and p_3 receives $v2$, $v3$.

For p_2 these two executions are indistinguishable, hence p_2 may decide on the value v_1 from p_1. If p_2 crashes at the beginning of round 2, p_3 may fail to receive value $v1$, and thus violate uniform agreement.

NB. 2.d does not contradict 2.a, because this proof relies on 2 processes failing, while in 2.a only a single process fails.

Problem 3

3.b Algorithm

- every process keeps a list of correct processes.
- there is a periodic *leader* event, at which each process *bebBroadcast*s $\langle leader, p_i \rangle$, where p_i is the process with the smallest id from the list of correct processes.
- at crash event, if p_j id is returned, current process removes p_j from the set of correct processes. Similarly, if $\diamond P$ removes suspicion on p_j, process puts p_j back in the list of correct processes.

Due to the properties of $\diamond P$, eventually the list of correct processes will be the same at all correct nodes, and will contain all correct processes. Thus, termination will hold, as all processes will trust the same correct process. Similarly, agreement holds as all processes will stop outputting different *leader* events, once all processes have the same list of correct processes (as they choose the leader deterministically from this list).

3.c The Uniform Consensus in this model is a multi-round algorithm – it runs until the failure detector becomes perfect (eventually), and a correct process is not suspected by any correct processes, and all the crashed/omission-faulty processes have been identified.

A process becomes leader based on the Eventual Leader Election: eventually all processes will agree on a single leader. At every round, the leader will ask the majority of processes if it is suspected by them. Finally, when Eventual Leader Election succeeds, no correct process will suspect the leader, and once the majority states it, the leader can decide the previously imposed value, and then impose it on the other processes, which will decide in the next round.

The following algorithm assumes implementation of $pp2p$ links, using *bebBroadcast*.

```
init:
    currentleader = nil
    sn = 0
    qack = empty
    iack = empty
    v = nil
    ucdecided = false

upon event ucPropose(v')
    v = v'

upon event leader(p) and v not nil and not ucdecided
    if p not current leader
        currentleader = p
        if leader = self
            sn++;
            qack = empty
            iack = emtpy
        endif
    else
```

```
        pp2p(QUERY, leader, v, sn)
    endif

upon event pp2pDeliver(QUERY, l, v', sn')
    if (l=self and leader =self)
        qack = qack \union (v',sn')
        if (|qack| >N/2)
            (v,sn) = highest(qack) #sorts by sn, then by v
            bebBroadcast(IMPOSE, v, sn)
        endif
    endif

upon event bebDeliver(IMPOSE, v', sn')
    if ( (sn',v') >= (sn, v) )
        sn = sn'
        v = v'
        pp2p(IMPOSE, leader, v, sn)
    endif

upon event p2pDeliver(IMPOSE, l, v', sn')
    if (leader = l = self and sn' = sn and v' = v)
        iack = iack \union (v',sn')
        if (|iack| > N/2)
            ucDecide(v)
            ucdecided = true
            bebBroadcast(DECIDE, v)
        endif
    endif

upon event bebDeliver(DECIDE, v') and ucdecided = false
    v = v'
    sn = \inf
    ucDecide(v)
    ucdecided = true
    bebBroadcast(DECIDE, v)
```

3.d We can implement *TOB* using *UniCons*. Each process *toBroadcast*s messages using *bebBroadcast*. Each process keeps an internal queue of unordered, non-delivered messages.

When a process p_i *bebDeliver*s a message, it puts the message into the internal list. When the internal list is not empty, the process p_i launches *ucPropose* with the list. When a process *ucDecide*s, it receives a decided list, $decided_u$. Then p_i removes all messages from its unordered list, which are in $decided_u$. Finally, p_i deterministically sorts $decided_u$ and p_i delivers messages from that list. While delivering messages, p_i skips delivery if a message exists in its *delivered* list. Every *toDeliver*-ed message is added to the *delivered* list.