Distributed Algorithms 2010/2011
The Final Exam (Solutions)

Prof. Rachid Guerraoui

January 22, 2011

Problem 1: Consensus

Part 1.a. As a syntactical rule, the word "eventually" in the property usually indicates a liveness property. As an intuitive rule, if a system doing nothing satisfies the property, then it is probably a safety property. As a more formal guideline: If you can describe a finite point in time at which it is possible to check whether the property is satisfied or violated, then it's probably a safety property.

Properties 1 and 2 are safety properties: One could instruct an observer to wait until one general attacks and at that point in time check whether the other general also attacks. If not, then there is no chance to satisfy the property again. This is characteristic of a safety property. Property 3 is unclear. On the one hand, a system doing nothing implements the property. This indicates a safety property. But on the other hand, any finite behavior can be âĂIJmade goodâĂİ again. For example, if general B attacks, then the property is only violated as long as general A never attacks. So if A eventually attacks, then the property is satisfied. This indicates a liveness property. Property 4 is a classical liveness property. There is no finite point in time where this property can be violated. Properties 5 and 6 are very similar, but they don't belong to the same class. Property 5 is a liveness property because it does not state that both generals must attack at the same time. Any finite behavior can be made good again: If some general attacks then the other general should also attack. If no general has attacked yet, this can be made good again by letting both generals attack. Property 6 is a safety property like properties 1 and 2. Wait for the time when one general attacks and then check if the other general attacks at the same instant. If not, then you have a finite point in time where the property is violated.

Part 1.b. One needs 10 generals.

Part 1.d. See Exercise 5.5 (page 265) in the book.

Problem 2: Broadcasts

Part 2.b. Whenever processes propose, they broadcast their proposal value to all other processes using total order broadcast. Deliveries of that broadcast happen at all processes in the same (total) order. This means that all processes will deliver the same first message. The idea is to simply decide the first message.

```
Implements:
    Consensus, instance cons;
Uses:
    TotalOrderBroadcast, instance tob;
upon event <cons, Init> do
    done = false
upon event <cons, Propose| v> do
    trigger <tob, Broadcast| v>
upon event <tob, Deliver| x> do
    if (done = false) then
        trigger <cons, Decide| x>
        done := true
```

Consensus integrity is guaranteed by the use of the variable *done*. Consensus validity follows from the algorithm and the *No creation* property of total order broadcast. Consensus termination follows from the algorithm and the *Validity* and *Agreement* properties of total order broadcast. Finally, consensus Agreement follows from the algorithm and the total order property.

Part 2.c. First, implement Consensus in this model. Then, using Consensus, and Reliable Broadcast, implement Total Order Broadcast.

```
Implements:
    Consensus, instance cons;
Uses:
    ReliableBroadcast, instance rb;
    Strong Failure Detector, instance S;

upon event <cons, Init>
    correct := \Pi;
    V := <\bot>^N; // p's estimate of proposed value
    round := 1;
    decision := \bot;
    phase1 := \bot;
    receiveddelta := < <\bot>^N >^N;
    receivedfrom := \emptyset;
    receivedVfrom := \emptyset;

upon event <S,Suspect |p> do
    correct := correct \ {p};

upon event <S,Restore |p> do
    correct := correct \union {p};

upon event <cons, Propose | v> do
    V[rank(self)] := v;
    \Delta := V;
    trigger < rb, Broadcast | [PROPOSAL, round, \Delta]>;

upon event < rb, Deliver | p, [PROPOSAL, r, \Delta_p] > such that r = round do
    receivedfrom := receivedfrom \bot {p};
    receiveddelta[rank(p)] := \Delta_p;

upon correct \subset receivedfrom \cap phase1 = \bot do
    if round = N then
        phase1 := \top
        trigger <rb, Broadcast | [DECIDE, V]>
    else
        \Delta := <\bot>^N;
        for k:=1 to N
            if V[k] = \bot
                \cap (\exists p_i \in receivedfrom:  receiveddelta[i][k] /= \bot) then
                V[k] := receiveddelta[i][k]
                \Delta[k] := receiveddelta[i][k]

        round := round + 1;
        receivedfrom := \emptyset;
        trigger <rb, Broadcast | [PROPOSAL, round, \Delta]>;

upon event < rb, Deliver | p, [DECIDE, V_p] > do
```

```
    receivedVfrom := receivedVfrom \bot {p};
    receivedV[rank(p)] := V_p;

upon correct \subset receivedVfrom \cap decision = \bot do
    for k:=1 to N
        if (\exists V_p \in receivedV: V_p[k] = \bot) then
            V[k] := \bot
    decision := first non \bot component of V
    trigger <cons, Decide | decision>;
```

For the explanation of the above algorithm, consult the paper "Unreliable failure detectors for reliable distributed systems", by Chandra and Toueg. The algorithm is described in Section 6.1. Essentially, this algorithm goes in asynchronous rounds. In each round, a process waits for a message from all other processes it sees as correct. In each round, processes exchange a vector containing estimates of all other processes. The final decision is made in the N-th round, where processes decide on the value from the lowest ranked process. Proof of correctness is in Chandra and Toueg's paper.

For the implementation of Total Order Broadcast, using Reliable Broadcast and Consensus, consult the book, Algorithm 6.1 (page 285).

Part 2.d. The answer is *no*, you can't implement *tob* using $\diamond S$, because one needs a majority of correct processes when working with Eventual failure detectors.

Problem 3: Lucky Registers

Part 3.a. The answer depends on the stated assumptions. If one assumes initial values in the registers were 2, then the execution is atomic. If initial values are not 2, and if p_3's last read overlaps with p_1's last write, then the execution is also atomic. Else, if p_3's last read doesn't overlap with p_1's last write, the execution is not atomic. We accept any of these answers, as long as the assumptions are clearly stated.

Part 3.c. Consult the "Lucky Registers" pdf on the Course website.