Name: _____
Personal Number: _____

KTH Microelectronics
and Information Technology

Exam in 2G1513 Distributed Systems FK, 2007-03-08, 15:00

NAME:
PERSONAL NUMBER:

Rules

This exam is "closed book" and you are not allowed to bring any material or equipment (such as laptops, PDA's, or mobile phones) with you. The only exception is English-to-"your favorite language" dictionary and **pencils.**

Instructions

- Write your name and personal number **clearly above on every page (including this one)**
- **Write only on the exam sheets**!
- Some of the questions are multiple-choice, **mark** the right answer by circling it! Marking the correct answer gives you 1 point
- For other questions, you have to write your answer in the allocated solution space **solution** … **end solution.** There may be more space allocated than it is needed.
- Check that you really have all pages of this exam, it should be 16 pages.

Time

You have 300 minutes to complete the exam.

Grading

Total 100 points, 4 point for each multiple-choice question, 6 points for the rest.

Name: _____
Personal Number: _____

1. Which of the following statements is true about synchronous networks vs. synchronous message passing?

a) In synchronous message passing, there is an upper bound on the time it takes to send a message from one process to another

b) In synchronous message passing, each send event immediately applies the corresponding receive event

c) Synchronous message passing simulates synchronous networks, by using message passing

d) In synchronous networks, each send event immediately applies the corresponding receive event.

2. Which of the following statements is true about all wave algorithms?

a) all processes have to decide and they all influence each other's decisions

b) at least one process has to decide and this decision is influenced by all processes

c) at least one process decides and this decision is influenced by at least one process

d) all processes have to decide and at least one process influences each decision

3. Which of the following statements is true?

a) a perfect failure detector can be implemented in a network where there is no bound on the communication delay

b) an eventually perfect failure detector can be implemented in a network where there is an unknown bound on the communication delay

c) a perfect failure detector can be implemented in a network where there is a known bound on the communication delay

d) a perfect failure detector can be implemented in a network where there is an unknown bound on the communication delay

4. Which of the following statements is true?

a) consensus can be implemented in a fail-noisy model by assuming an eventual leader detector

b) consensus can be implemented in a fail-silent model by assuming a majority of correct processes

c) consensus can be implemented in a fail-noisy model by assuming a majority of correct processes

d) consensus can be implemented in a fail-silent model by assuming an eventual leader detector.

5. Which of the following statements is true about a fail-stop consensus algorithm?

a) if the underlying failure detector would not satisfy completeness, the termination requirement of consensus would not be satisfied.

b) if the underlying failure detector would not satisfy accuracy, the termination requirement of consensus would not be satisfied.

c) if the underlying failure detector would not satisfy safety, the termination requirement of consensus would not be satisfied.

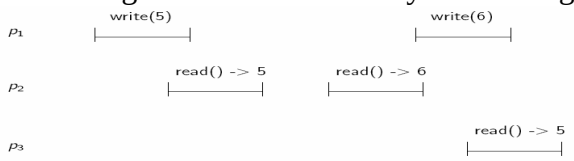
d) if the underlying failure detector would not satisfy completeness, the agreement requirement of consensus would not be satisfied.

Name: _____
Personal Number: _____

6. The Uniform Agreement property of Uniform Reliable Broadcast (Uniform Consensus) states that:

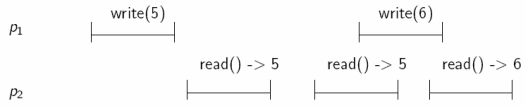
a) if any process delivers (decides) then all processes should deliver (decide)
b) if any correct process delivers (decides) then all processes should deliver (decide)
c) if any correct process delivers (decides) then all correct processes should deliver (decide)
d) if any process delivers (decides) then all correct processes should deliver (decide)

7. Given the register execution in the figure below one can say that the register is:



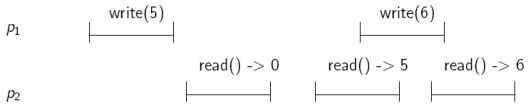
a) regular or atomic
b) regular but not atomic
c) atomic but not regular
d) not regular and not atomic

8. Given the register execution in the figure below one can say that the register is:



a) regular or atomic
b) regular but not atomic
c) atomic but not regular
d) not regular and not atomic

9. Given the register execution in the figure below one can say that the register is:



a) regular or atomic
b) regular but not atomic
c) atomic but not regular
d) not regular and not atomic

10. Which of the following statements are true about View Synchronous Broadcast?

a) all messages broadcast by both correct and faulty processes in view j are delivered by all correct processes in the same view j

b) all messages broadcast by correct processes in view j are delivered by all correct processes in view j and no message broadcast by a faulty process in view j is delivered by any correct process in view j

c) all messages broadcast by correct processes in view j are delivered by all correct processes in view j and all messages broadcast by faulty processes in view j are delivered by correct processes in view j or higher

d) all messages broadcast by correct processes in view j are delivered by all correct processes in view j and all messages broadcast by faulty processes are delivered by all or no correct process in view j

Name: _____
Personal Number: _____

11. Will the marked statement in the Tree Algorithm, in the below figure, be executed by every node? If so, state why! If not, state by which it will be executed!

```
var rec_p[q] for each q ∈ Neigh_p  : boolean init false ;
    (* rec_p[q] is true if p has received a message from q *)

begin while #{q : rec_p[q] is false} > 1 do
            begin receive ⟨tok⟩ from q ; rec_p[q] := true end ;
      (* Now there is one q_0 with rec_p[q_0] is false *)
      send ⟨tok⟩ to q_0 with rec_p[q_0] is false ;
   x: receive ⟨tok⟩ from q_0 ; rec_p[q_0] := true ;
      decide
      (* Inform other processes of decision:
         forall q ∈ Neigh_p, q ≠ q_0 do send ⟨tok⟩ to q *)
end
```

Solution

No, it will not be executed by any of the leaf nodes, as leaf nodes only have one neighbor. Hence, the condition of the while loop is false and consequently the while loop will never executed.

End Solution

12. What is the basic assumption made in the ABD-synchronizer to make it simulate a synchronous system?

Solution

There is an upper bound on the time it takes to send a message from one process to another.

End Solution

Name: _____
Personal Number: _____

13. Explain why a timestamp is needed in the Majority Voting algorithm but not in the Read-One Write-All algorithm. The Read-One Write-All algorithm (4.1) and the Majority Voting algorithm (4.2-4.3) are given on the last pages of this exam.
Solution

In the Read-One Write-All algorithm, the writer imposes the latest value at all processes, but in Majority Voting it only imposes it at a majority and a later read operation contacting a different majority might contain both the latest value and a previous value. The reader cannot distinguish which value is the latest without a timestamp, so in this case even if the operations are not concurrent, the read cannot return the last value written.

End Solution

14. Explain the purpose of the "read impose" idea in implementing any (1, N) atomic register. The Read-Impose Write Majority algorithm (4.9-4.10) is given on the last pages of this exam, in case you need it for reference.
Solution

The "read impose" idea is used to ensure the ordering property of a (1, N) atomic register. That is, a read operation should not return an older value at any process, after a newer value has been returned by a read operation at any process. "Read impose" delays the completion of a read operation until "enough" processes have the current or newer value, so that further reads will not return a value older than the current value. "Enough" processes means all processes if the read operation checks only the local value of a process (in our fail-stop algorithms), and it means a majority of processes if the read operation checks the values of a majority of processes (in our fail-silent algorithm).

End Solution

Name: _____
Personal Number: _____

15. Explain the purpose of the “write consult” idea in implementing any (N, N) atomic register. The Read-Impose Write-Consult-Majority algorithm (4.14-4.15) is given on the last pages of this exam, in case you need it for reference.
Solution

The “write consult” idea is used to satisfy unique, totally ordered sequence numbers for write operations. This is needed for a register with multiple writers as otherwise two different writers might use the same sequence number for two different values. “Write consult” means that a writer first consults the latest timestamp used in a previous write operation, so it can use a larger one for the current write operation. In the Read-Impose Write-Consult-Majority algorithm, a writer first consults a majority of processes to get the latest timestamp used, so it can use a larger one for its current write operation. In the Read-Impose Write-Consult algorithm (fail-stop), the latest timestamp used is already available locally, as a previous operation has contacted all processes.

End Solution

16. Explain why Flooding Consensus does not ensure Uniform Consensus. Give an example execution using a time-space diagram. The Flooding Consensus algorithm (5.1) is given on the last pages of this exam.
Solution

You have 3 processes: p1, p2, and p3 proposing values 1, 2, and 3 respectively. p1 sends it proposal set (1) to p2 and then crashes. p2 receives all the proposal-sets (from p1 and p3) and decides 1 (1=min(1,3)) before detecting the crash of p1, and then crashes. p3 receives no proposal sets, detects the crashes of p1 and p2 and decides differently (3).

Another example of a non-uniform execution you have in slide 29 of consensus.

End Solution

Name: _____
Personal Number: _____

17. Explain why the best-effort broadcast abstraction is good enough to disseminate a decision in the Uniform Consensus based on Abortable Consensus algorithm. Why is the algorithm satisfying uniform agreement even if the process which is beb-broadcasting its decision crashes? The Abortable Consensus algorithm (5.5-5.6) and the Uniform Consensus algorithm built on top of it (5.7) are given on the last pages of this exam.

Solution

Once a decision needs to be disseminated, the respective value has been imposed at a majority of processes. Uniform decision is safe now. Even if the current leader crashes after beb-broadcasting the decision, any other leader is going to decide on the same value anyway given that all further read phases of abortable consensus will return the same value and this value is going to be used in all further write phases.

End Solution

18. What is the difference between a perfect failure detector abstraction and a group membership abstraction?

Solution

By using a group membership abstraction, all processes see the same order of crash notifications while by using a failure detector abstraction different processes might perceive crashes in different orders. A group membership abstraction offers a coordinated view of crashes.

End Solution

Name: _____
Personal Number: _____

19. What guarantees are provided by a Total Order Broadcast abstraction? How about a Uniform Total Order Broadcast abstraction?
Solution

A Total Order Broadcast Abstraction guarantees Reliable Broadcast and the fact that all broadcast messages are delivered in the same order at all correct processes.

A Uniform Total Order Broadcast Abstraction guarantees Uniform Reliable Broadcast and the fact that all broadcast messages are delivered in the same order at all processes.

End Solution

20. Consider the Consensus-Based (Regular) View Synchronous Broadcast algorithm (6.11-6.12) given on the last pages of this exam. This algorithm does not implement Uniform View Synchronous Broadcast. It uses Uniform Consensus, Best-Effort Broadcast and a Perfect Failure Detector. Assume the same algorithm would use Uniform Reliable Broadcast instead of Best-Effort Broadcast. Would it, in that case, implement Uniform View Synchronous Broadcast? If yes, explain why. If no, explain why not.
Solution

No. The guarantee offered by Uniform Reliable Broadcast is that if a process urbDelivers a message, eventually all correct processes will urbDeliver it. So it might happen that p1 urbDelivers m1 in view v1, vsDelivers v1, and crashes. If all other processes detect the crash of p1 and initiate a flush before any of them urbDelivers m1, then these processes will not vsDeliver m1 in the current view v1, as it is not contained in any process's delivered set. After flushing they install view v2. Eventually these processes urbDeliver m1 but as m1 was broadcast in view v1 it is ignored in view v2 so it won't be vsDelivered. Hence, we cannot achieve Uniform View Synchronous Broadcast just by replacing Best-Effort Broadcast with Uniform Reliable Broadcast.

End Solution

Name: _____
Personal Number: _____

Algorithm 4.1 Read-One Write-All

Implements:
(1, N)RegularRegister (on-rreg).

Uses:
BestEffortBroadcast (beb);
PerfectPointToPointLinks (pp2p);
PerfectFailureDetector ($\mathcal{P}$).

upon event ⟨ *Init* ⟩ **do**
  forall r **do**
    value[r] := 0;
    writeSet[r] := $\emptyset$;
  correct := Π;

upon event ⟨ *crash* | p_i ⟩ **do**
  correct := correct \ $\{p_i\}$;

upon event ⟨ *on-rregRead* | reg ⟩ **do**
  trigger ⟨ *on-rregReadReturn* | reg, value[reg] ⟩;

upon event ⟨ *on-rregWrite* | reg, val ⟩ **do**
  trigger ⟨ *bebBroadcast* | [WRITE, reg, val] ⟩;

upon event ⟨ *bebDeliver* | p_j, [WRITE, reg, val] ⟩ **do**
  value[reg] := val;
  trigger ⟨ *pp2pSend* | p_j, [ACK, reg] ⟩;

upon event ⟨ *pp2pDeliver* | p_j, [ACK, reg] ⟩ **do**
  writeSet[reg] := writeSet[reg] ∪ $\{p_j\}$;

upon exists r **such that** correct ⊆ writeSet[r] **do**
  writeSet[r] := $\emptyset$;
  trigger ⟨ *on-rregWriteReturn* | r ⟩;

Name: _____
Personal Number: _____

Algorithm 4.2 Majority Voting (write)

Implements:
(1, N)RegularRegister (on-rreg).
Uses:
BestEffortBroadcast (beb);
PerfectPointToPointLinks (pp2p).

upon event ⟨ *Init* ⟩ **do**
forall r **do**
sn[r] := 0;
v[r] := 0;
acks[r] := 0;
reqid[r] := 0;
readSet[r] := ∅;

upon event ⟨ *on-rregWrite* | r, val ⟩ **do**
sn[r] := sn[r] + 1;
v[r] := val;
acks[r] := 1;
trigger ⟨ *bebBroadcast* | [WRITE, r, sn[r], val] ⟩;

upon event ⟨ *bebDeliver* | p_j, [WRITE, r, tstamp, val] ⟩ **do**
if tstamp > sn[r] **then**
v[r] := val;
sn[r] := tstamp;
trigger ⟨ *pp2pSend* | p_j, [ACK, r, tstamp] ⟩;

upon event ⟨ *pp2pDeliver* | p_j, [ACK, r, ts] ⟩ **do**
if ts=sn[r] **then**
acks[r] := acks[r] + 1;

upon exists r **such that** acks[r] > $N/2$ **do**
trigger ⟨ *on-rregWriteReturn* | r ⟩;

Algorithm 4.3 Majority Voting (read)

upon event ⟨ *on-rregRead* | r ⟩ **do**
reqid[r] := reqid[r] +1;
readSet[r] := ∅;
trigger ⟨ *bebBroadcast* | [READ, r, reqid[r]] ⟩;

upon event ⟨ *bebDeliver* | p_j, [READ, r, id] ⟩ **do**
trigger ⟨ *pp2pSend* | p_j,[READVALUE, r, id, $sn[r], v[r]$] ⟩;

upon event ⟨ *pp2pDeliver* | p_j, [READVALUE, r, id, tstamp, val] ⟩ **do**
if id=reqid[r] **then**
readSet[r] := readSet[r] ∪ {(tstamp, val)};

upon exists r **such that** (|readSet[r]| > $N/2$) **do**
(v, ts) := highest(readSet[r]);
$v[r] := v$;
$sn[r] := ts$;
trigger ⟨ *on-rregReadReturn* | r, v) ⟩;

Name: _____
Personal Number: _____

Algorithm 4.9 Read-Impose Write-Majority (part I)

Implements:
(1, N)AtomicRegister (on-areg).

Uses:
BestEffortBroadcast (beb); PerfectPointToPointLinks (pp2p).

upon event ⟨ *Init* ⟩ **do**
forall r **do**
sn[r] := 0;
v[r] := 0;
acks[r] := 0;
reqid[r] := 0;
readval [r] := 0;
reading[r] := false;
readSet[r] := ∅;

upon event ⟨ *on-aregWrite* | r, val ⟩ **do**
reqid[r] := reqid[r] + 1;
sn[r] := sn[r] + 1;
v[r] := val;
acks[r] := 1;
trigger ⟨ *bebBroadcast* | [WRITE, r, reqid, sn[r], val] ⟩;

upon event ⟨ *bebDeliver* | p_j, [WRITE, r, id, t, val] ⟩ **do**
if t > sn[r] **then**
sn[r]:= t; v[r] := val;
trigger ⟨ *pp2pSend* | p_j, [ACK, r, id] ⟩;

upon event ⟨ *pp2pDeliver* | p_j, [ACK, r, id] ⟩ **do**
if reqid[r] = id **then**
acks[r] := acks[r] + 1;

Algorithm 4.10 Read-Impose Write-Majority (part II)

upon exists r **such that** acks[r] > $N/2$ **do**
if reading[r] = true **then**
reading[r] := false;
trigger ⟨ *on-aregReadReturn* | r, readval[r] ⟩;
else
trigger ⟨ *on-aregWriteReturn* | r ⟩;

upon event ⟨ *on-aregRead* | r ⟩ **do**
reqid[r] := reqid[r] + 1;
readSet[r] := ∅;
trigger ⟨ *bebBroadcast* | [READ, r, reqid[r]] ⟩;

upon event ⟨ *bebDeliver* | p_j, [READ, r, id] ⟩ **do**
trigger ⟨ *pp2pSend* | p_j, [READVALUE, r, id, sn[r], v[r]] ⟩;

upon event ⟨ *pp2pDeliver* | p_j, [READVALUE, r, id, ts, val] ⟩ **do**
if reqid[r] = id **then**
readSet[r] := readSet[r] ∪ { (ts, val) };

upon exists r **such that** |readSet[r]| > $N/2$ **do**
(tstamp,readval[r]) := *highest*(readSet[r]);
acks[r] := 0;
reading[r] := true;
trigger ⟨ *bebBroadcast* | [WRITE, r, reqid[r], tstamp, readval[r]] ⟩;

Name: _____
Personal Number: _____

Algorithm 4.14 Read-Impose Write-Consult-Majority (part I)

Implements:
(N, N)AtomicRegister (nn-areg).

Uses:
BestEffortBroadcast (beb);
PerfectPointToPointLinks (pp2p).

upon event ⟨ *Init* ⟩ **do**
i := *rank* (self);
forall r **do**
writeSet[r] := ∅;
readSet[r] := ∅;
reading[r] := false;
reqid[r] := 0;
v[r] := 0;
ts[r] := 0;
mrank[r] := 0;

upon event ⟨ *nn-aregRead* | r ⟩ **do**
reqid[r] := reqid[r] + 1;
reading[r] := true;
readSet[r] := ∅;
writeSet[r] := ∅;
trigger ⟨ *bebBroadcast* | [READ, r, reqid[r]] ⟩;

upon event ⟨ *nn-aregWrite* | r, val ⟩ **do**
reqid[r] := reqid[r] + 1;
writeval[r] := val;
readSet[r] := ∅;
writeSet[r] := ∅;
trigger ⟨ *bebBroadcast* | [READ, r, reqid[r]] ⟩;

upon event ⟨ *bebDeliver* | p_j, [READ, r, id] ⟩ **do**
trigger ⟨ *pp2pSend* | p_j, [READVALUE, r, id, (ts[r], mrank[r]), v[r]] ⟩;

upon event ⟨ *pp2pDeliver* | p_j, [READVALUE, r, id, (t,rk), val] ⟩ **do**
if id = reqid[r] **then**
readSet[r] := readSet[r] ∪ {((t, rk), val)};

Name: _____
Personal Number: _____

Algorithm 4.15 Read-Impose Write-Consult-Majority (part II)

```
upon exists r such that |readSet[r]| > N/2 do
    ((t,rk), v) := highest (readSet[r]);
    readval[r] := v;
    if reading[r] then
        trigger ⟨ bebBroadcast | [WRITE, r, reqid[r], (t,rk), readval[r]] ⟩;
    else
        trigger ⟨ bebBroadcast | [WRITE, r, reqid[r], (t+1,i), writeval[r]] ⟩;

upon event ⟨ bebDeliver | p_j,[WRITE, r, id, (t,j), val] ⟩ do
    if (t,j) > (ts[r], mrank[r]) then
        v[r] := val;
        ts[r] := t;
        mrank[r] := j;
    trigger ⟨ pp2pSend | p_j, [ACK, r, id] ⟩;

upon event ⟨ pp2pDeliver | p_j, [ACK, r, id] ⟩ do
    if id = reqid[r] then
        writeSet[r] := writeSet[r] ∪ {p_j};

upon exists r such that |writeSet[r]| > N/2 do
    if (reading[r] = true) then
        reading[r] := false;
        trigger ⟨ nn-aregReadReturn | r, readval[r] ⟩;
    else
        trigger ⟨ nn-aregWriteReturn | r ⟩;
```

Algorithm 6.12 Consensus-Based View Synchrony (group change)

```
upon event ⟨ crash | p_i ⟩ do
    correct := correct \{p_i};

upon (correct ⊂ current-view.memb) ∧ (flushing = false) do
    flushing := true;
    trigger ⟨ vsBlock ⟩;

upon event ⟨ vsBlockOk ⟩ do
    blocked := true;
    trigger ⟨ bebBroadcast | [DSET, current-view.id, delivered] ⟩;

upon event ⟨ bebDeliver | src,[DSET, vid, mset] ⟩ ∧ (blocked) do
    dset[vid] := dset[vid] ∪ {(src, mset)};

upon ∀p ∈ correct ∃{(p, mset)} ∈ dset[current-view.id] ∧ (wait = false) do
    trigger ⟨ ucPropose | current-view.id+1, correct, dset[current-view.id] ⟩;
    wait := true;

upon event ⟨ ucDecided | id, memb, vs-dset ⟩ do
    forall {(p, mset)} ∈ vs-dset: p ∈ memb do
        forall (src_m, m) ∈ mset: (src_m, m) ∉ delivered do
            delivered := delivered ∪ { (src_m, m) };
            trigger ⟨ vsDeliver | src_m, m ⟩;
    flushing := blocked := wait := false;
    delivered := ∅;
    current-view := (id, memb);
    trigger ⟨ vsView | current-view ⟩;
```

Name: _____
Personal Number: _____

Algorithm 5.1 Flooding Consensus

Implements:
Consensus (c).

Uses:
BestEffortBroadcast (beb);
PerfectFailureDetector ($\mathcal{P}$).

upon event $\langle$ *Init* $\rangle$ **do**
correct := correct-this-round[0] := Π;
decided := $\perp$; round := 1;
for $i = 1$ **to** N **do**
correct-this-round[i] := proposal-set[i] := $\emptyset$;

upon event $\langle$ *crash* | p_i $\rangle$ **do**
correct := correct $\setminus \{p_i\}$;

upon event $\langle$ *cPropose* | v $\rangle$ **do**
proposal-set[1] := proposal-set[1] $\cup \{v\}$;
trigger $\langle$ *bebBroadcast* | [MySet, 1, proposal-set[1]] $\rangle$;

upon event $\langle$ *bebDeliver* | p_i, [MySet, r, set] $\rangle$ **do**
correct-this-round[r] := correct-this-round[r] $\cup \{p_i\}$;
proposal-set[r] := proposal-set[r] $\cup$ set;

upon correct $\subseteq$ correct-this-round[round] $\wedge$ (decided = $\perp$) **do**
if (correct-this-round[round] = correct-this-round[round-1]) **then**
decided := *min* (proposal-set[round]);
trigger $\langle$ *cDecide* | decided $\rangle$;
trigger $\langle$ *bebBroadcast* | [Decided, decided] $\rangle$;
else
round := round +1;
trigger $\langle$ *bebBroadcast* | [MySet, round, proposal-set[round-1]] $\rangle$;

upon event $\langle$ *bebDeliver* | p_i, [Decided, v] $\rangle$ $\wedge$ $p_i \in$ correct $\wedge$ (decided = $\perp$) **do**
decided := v;
trigger $\langle$ *cDecide* | v $\rangle$;
trigger $\langle$ *bebBroadcast* | [Decided, decided] $\rangle$;

Name: _____
Personal Number: _____

Algorithm 5.5 RW Abortable Consensus: Read Phase

Implements:
Abortable Consensus (ac).

Uses:
BestEffortBroadcast (beb);
PerfectPointToPointLinks (pp2p).

upon event ⟨ *Init* ⟩ **do**
tempValue := val := ⊥;
wAcks := rts := wts := 0;
tstamp := *rank*(self);
readSet := ∅;

upon event ⟨ *acPropose* | *v* ⟩ **do**
tstamp := tstamp+*N*;
tempValue := v;
trigger ⟨ *bebBroadcast* | [READ, tstamp] ⟩;

upon event ⟨ *bebDeliver* | p_j,[READ, ts] ⟩ **do**
if rts ≥ ts **or** wts ≥ ts **then**
trigger ⟨ *pp2pSend* | p_j, [NACK] ⟩;
else
rts := ts;
trigger ⟨ *pp2pSend* | p_j, [READACK, wts, val] ⟩;

upon event ⟨ *pp2pDeliver* | p_j, [NACK] ⟩ **do**
trigger ⟨ *acReturn* | ⊥ ⟩;

upon event ⟨ *p2pDeliver* | p_j, [READACK, ts, v] ⟩ **do**
readSet := readSet ∪ {(*ts*, *v*)}

upon (|readSet| > *N*/2) **do**
(ts, v) := *highest*(readSet);
if v ≠ ⊥ **then** tempValue := v;
trigger ⟨ *bebBroadcast* | [WRITE, tstamp, tempValue] ⟩;

Algorithm 5.6 RW Abortable Consensus: Write Phase

Implements:
Abortable Consensus (ac).

upon event ⟨ *bebDeliver* | p_j, [WRITE, *ts*, *v*] ⟩ **do**
if rts > ts **or** wts > ts **then**
trigger ⟨ *pp2pSend* | p_j,[NACK] ⟩;
else
val := v;
wts := ts;
trigger ⟨ *pp2pSend* | p_j, [WRITEACK] ⟩;

upon event ⟨ *pp2pDeliver* | p_j, [NACK] ⟩ **do**
trigger ⟨ *acReturn* | ⊥ ⟩;

upon event ⟨ *pp2pDeliver* | p_j, [WRITEACK] ⟩ **do**
wAcks := wAcks+1;

upon (wAcks > *N*/2) **do**
readSet := ∅;
wAcks := 0;
trigger ⟨ *acReturn* | tempValue ⟩;

Name: _____
Personal Number: _____

Algorithm 5.7 From Abortable Consensus to Consensus

Implements:
UniformConsensus (uc).

Uses:
AbortableConsensus (ac);
BestEffortBroadcast (beb);
EventualLeaderDetector (Ω).

upon event ⟨ *Init* ⟩ **do**
proposal := $\perp$;
leader := proposed := decided := false;

upon event ⟨ *trust* | p_i ⟩ **do**
if p_i = self **then** leader := true;
else leader := false;

upon event ⟨ *ucPropose* | v ⟩ **do**
proposal := v;

upon (leader = true) ∧ (proposed = false) ∧ (proposal ≠ $\perp$) **do**
proposed := true;
trigger ⟨ *acPropose* | proposal ⟩;

upon event ⟨ *acReturn* | result ⟩ **do**
if result ≠ $\perp$ **then**
trigger ⟨ *bebBroadcast* | [DECIDED, result] ⟩;
else
proposed := false;

upon event ⟨ *bebDeliver* | p_i, [DECIDED, v] ⟩ ∧ (decided = false) **do**
decided := true;
trigger ⟨ *ucDecide* | v ⟩;

Algorithm 6.11 Consensus-Based View Synchrony (data transmission)

Implements:
ViewSynchrony (vs).

Uses:
UniformConsensus (uc);
BestEffortBroadcast (beb);
PerfectFailureDetector ($\mathcal{P}$).

upon event ⟨ *Init* ⟩ **do**
current-view := (0, Π);
correct := Π;
flushing := blocked := wait := false;
delivered := $\emptyset$;
forall i **do** dset[i] := $\emptyset$;

upon event ⟨ *vsBroadcast* | m ⟩ ∧ (blocked = false) **do**
delivered := delivered ∪ {(self, m)};
trigger ⟨ *vsDeliver* | self, m ⟩;
trigger ⟨ *bebBroadcast* | [DATA, current-view.id, m] ⟩;

upon event ⟨ *bebDeliver* | src_m,[DATA, vid, m] ⟩ ∧
current-view.id = vid) ∧ (blocked = false)**do**
if ⟨ (src_m, m) ∉ delivered) **do**
delivered := delivered ∪ {(src_m, m)};
trigger ⟨ *vsDeliver* | src_m, m ⟩;