CO21BTECH11002
Aayush Kumar
Theory Assignment 3

1)
Early-stopping algorithm for Consensus under Crash Failures:

```
(global constants)
integer: f; // maximum number of crash failures tolerated

(local variables)
integer: x <- local value;
integer: c <- number of consecutive rounds in which the process has received
the value from all the expected processes;
set<integer>: st <- set of process ids from which a message is expected;
set<integer>: received_from <- set of process ids from which a message has
been received;

(initially)
x <- proposed value;
c <- 0;
st <- {all process ids except the current process id};
received_from <- {};

for round from 1 to f +1 do:
   // Every process broadcasts its value and count to all other processes
   broadcast(x, c);

   if message received from j then:
       received_from.insert(j);
       y_j <- message received from j;

   // Update the value of x to the minimum of the received values of x
   for j in received_from do:
       x <- min(x, y_j.x);
```

```
   // Flag to check if every process has received the value from all the
expected processes at least once
   bool: flag <- true;
   for j in received_from do:
       if y_j.c == 0 then:
           flag <- false;
           break;

   // There was a round when none of the processes failed, so the loop can be
terminated
   if flag then:
       break;

   // Use the flag to check if the process has received the value from all
the expected processes at least once
   flag <- true;
   for j in st do:
       if j not in received_from then:
           flag <- false;
           break;

   // If the process has received the value from all the expected processes,
then increment the count
   if flag then:
       c <- c + 1;
   // If the process has not received the value from all the expected
processes, then reset the count
   else:
       c <- 0;

   // Update the set of process ids from which a message is expected
   st <- received_from;
   received_from <- {};
end

output x as the consensus value.
```

Description:

‘c’ Represents the number of consecutive rounds in which the process has received the value from all expected processes.

In each round, every process broadcasts its current value x and its count c to all other processes. Upon receiving a message from process j, the process updates its received_from set and extracts the received value from y_j. The process updates its value x to the minimum of its current value and the received values from all processes.

It checks if every process has received the value from all expected processes at least once in a previous round (flag check). If so, then it is ensured that there was a round when none of the processes failed, so consensus is reached. If not, it continues to the next step. This check is made before checking if the process has received the value from all expected processes at least once in the current round. Since, even if in the current round, some processes crashed, there was a round when none of the processes failed, so every process would have received the value from all expected processes and would have the same value of x. So, the loop can be terminated.

It then checks if the process has received the value from all expected processes in the current round (flag check). If so, it increments the consecutive count c. If not, it resets the count c to 0.

Then it updates the set ‘st’ of expected process IDs to be those from which a message was received in the current round. Every process that sent a message in the current round has not crashed, so it is expected to send a message in the next round. So, the set ‘st’ is updated to the set of process IDs from which a message was received in the current round. The set ‘received_from’ is reset to an empty set for the next round.

2)

Generalizing the Consensus Problem with binary inputs to work with multi-valued inputs:

Let binCon() be the binary consensus algorithm that solves the binary consensus problem. The algorithm binCon takes as input a binary value or no value at all. The algorithm binCon returns a binary value.

A process can propose a binary value, b as binCon(b), or call binCon() to propose no value.

The process receives a binary value, r = binCon(b) or r = binCon(), from the algorithm binCon.

```
multiValCon(x):

   // flag to know if the process will propose a value or not
   bool flag = true;
   // Final consensus value, each bit is initially 0
   // Assuming 32-bit integer
   integer: consensus_value = 0;

   // Run the binary consensus algorithm for each bit of the input value
   for i from 0 to 31 do:
       // If process is proposing a value
       if flag then:
           // Get the i-th bit of the input value
           integer: bit = (x >> i) & 1;
           // Run the binary consensus algorithm to get the i-th bit of the
consensus value
           integer: bit_consensus = binCon(bit);
           // Update the i-th bit of the consensus value
           consensus_value = consensus_value | (bit_consensus << i);
           // If the bit returned by the binary consensus algorithm is
different from the input bit, then the process will no longer propose a value
           if bit_consensus != bit then:
               flag = false;
       // If process is not proposing a value
       else:
           // Run the binary consensus algorithm to get the i-th bit of the
consensus value
           integer: bit_consensus = binCon();
           // Update the i-th bit of the consensus value
           consensus_value = consensus_value | (bit_consensus << i);

   // Return the final consensus value
   return consensus_value;
```

Description:

For each bit of the input value x, the process runs the binary consensus algorithm binCon to get the i-th bit of the consensus value. If the process is proposing a value, it gets the i-th bit of the input value x and runs the binary consensus algorithm to get the i-th bit of the consensus value. If the bit returned by the binary consensus algorithm is different from the input bit, then the process will no longer propose a value. If the process is not proposing a value, it runs the binary consensus algorithm to get the i-th bit of the consensus value. The final consensus value is returned as a 32-bit integer.

Example:

Consider 4 processes with values x1 = 1 (001), x2 = 2 (010), x3 = 3 (011) and x4 = 6 (110).

In the first round, process 1 proposes 1 (001), process 2 proposes 0 (010), process 3 proposes 1 (011) and process 4 proposes 0 (110).
The binary consensus algorithm returns 0. So process 1 and 3 will no longer propose a value. Currently, consensus_value is (000).

In the second round, process 2 proposes 1 (010) and process 4 proposes 1 (110) while process 1 and 3 call binCon() to propose no value.
The binary consensus algorithm returns 1. So process 2 and 4 will continue to propose a value. Currently, consensus_value is (010).

In the third round, process 2 proposes 0 (010) and process 4 proposes 1 (110) while process 1 and 3 call binCon() to propose no value.
The binary consensus algorithm returns 1. So process 2 will no longer propose a value. Currently, consensus_value is (110).

For the rest of the rounds, process 4 will propose 0 (since further bits are 0) and the final consensus value will be (110).

So all the processes reached consensus on the value 6 (110).