# COM3026_sl01347_CW2

## Originality Declaration

[illegible]

## Pseudocode

```
Implements:
     PaxosConsensusAlgorithm, instance pca.

Uses:
     EventualLeaderDetector, instance Ω.
     PerfectPointToPointLinks, instance pl.

Upon event <pca, Init | name, participants, client> do
     name := name;
     participants: participants;
     client := client;
     leader := ⊥;
     decided := False;
     isAborted := False;
     messages := [];
     valueToPropose := ⊥;
     quorum_counter := 0;
     acceptedV := 0;
     bal := 0;
     a_bal := 0;
     a_val := ⊥;

     if (file) exists then
          read_from_file(decided, valueToPropose, bal);

Upon event <pca, Propose | val> do
     valueToPropose := val;

Upon event <Ω, Trust | p> do
     if (name = p ∧ decided = False) then
          bal := bal + 1;
          save_state();
          # in the beb_broadcast, we are broadcasting the ballot
          value (which is incremented by 1) along with the process
          name, which makes the message sent by beb unique
          beb_broadcast([Prepare, name, bal], participants);
          isAborted := False;
          leader := p;
     else
          leader := p;
```

```
Upon event <pca, Prepare | name, b> do
    if (b > bal) then
        bal := b;
        trigger <pl, Prepared | name, [b, a_bal, a_val]>;
    else
        trigger <pl, Nack | name, b>;

Upon event quorum S of (Prepared, b, a_bal, a_val) do:
    valueToPropose := a_val with the highest a_bal ≠ 0 in S;
    If no such a_val exists in S, valueToPropose := 0
    save_state();
    beb_broadcast([Accept, name, b, valueToPropose],
    participants);
    acceptedV := valueToPropose;

Upon event <pl, Accept | p, [b, v]> do:
    if b ⩾ bal then
        bal := b;
        a_bal := b;
        a_val := v;
        trigger <pl, Send | p, [Accepted, b]>;
    else
        trigger <pl, Send | p, [b, Nack]>;

Upon event <pca, Decide | val> do:
    if (decided = False) then
        decided := True;
        save_state();
        trigger <pl, Send | client, val>;
        if (isAborted = True) then
            messages := [];

Upon event quorum S of (Accepted, b) do:
    beb_broadcast([Decide, acceptedV], participants);

Upon event <pl, Nack | b> do
    isAborted:= True;


procedure unicast(m, p) is
    trigger <pl, Send | p, m>;

procedure save_state() is
    write to file (decided, valueToPropose, bal);
```

```
procedure read_from_file(file_name) is
    read_from_file(file_name);

procedure beb_broadcast(m, dest) is
    for (p ∈ dest) do
        unicast(m, p);
```

## How The Pseudocode Satisfies Uniform Consensus Properties

### Termination

*Every correct process eventually decides some value*

We know that some correct process *p* will be permanently trusted to be the leader by all correct processes. This means that *p* will be the only process triggering the (*name = p ∧ decided = False*) code. Since the *bal* value keeps increasing, process *p* will call the abortable consensus with a ballot which is greater than the value of *bal* of any process. When this happens, we know that the abortable consensus algorithm will return an accepted value (*acceptedV*) not equal to abort, which causes *p* to decide the accepted value and broadcast it to all processes. Since *p* is correct, the accepted value will reach all correct processes which causes them to accept the accepted value too.

### Validity

*If a process decides v, then v was proposed by some process*

We know that a process can only decide on a value that was proposed by the leader. We also know that a leader can only propose a value that it set, or a value that a process accepted that had a higher ballot number. These acceptor processes can only accept this proposed value when the leader has said to accept the value, so if a process does decide on a value, that value had to have come from a process.

### Integrity

*No process decides twice*

No process will be able to decide twice because if a process has decided, then we set the decided variable to true (in the *decide* event) which means that the trust event will not trigger, so a process will not be able to decide twice.

## Uniform Agreement

*No two processes decide differently*

In the prepared phase, the leader waits for a quorum of processes that have accepted the value it has proposed to them. The value the leader has proposed must be the highest value in the quorum. This is because we check to find the highest a_val which corresponds to the highest a_bal. Since this is true, we can say that either:

- No process in the quorum has accepted any proposal with a ballot less than the ballot the value was proposed at, or
- The value proposed is the value accepted at the highest-numbered ballot among all proposals accepted by the processes in the quorum at ballots < the ballot where the value is issued

This is agreement P2c, and we know that P2c ⇒ P2b ⇒ P2a ⇒ P2. If a value v is chosen at ballot b, v is also chosen at all ballots greater than b. If B is the lowest ballot where some value V is chosen, then V would also be chosen at all ballots greater than B. Since only chosen values can be decided, then if a process decides, it must decide V.

# Pseudocode Translated Into Elixir Code

## Crash Recovery

In my pseudocode, if there is a file that exists with state (i.e. if the processes crashed) then its state would be read and used in the round. I translated that into Elixir by initially creating a function to write the needed variables to a file. These needed variables were *`decided`*, *`bal`* and *`valueToPropose`* because all the other information could be gathered from other processes. This function is called *`save_state()`* and it uses a function *`File.write()`* to write these variables to a file. The filename corresponds to the process name to which these variables belong to. I saved state in the *`trust`*, *`prepared`* and *`decide`* events by calling *`save_state()`* before I return state (otherwise the appropriate state would not be saved to the files).

I also had a function *`read_from_file`* which reads a file and converts it's binary contents to elixir code. This is used to recover the state after a crash. This function checks if the file exists (*`:ok`* will be returned if this is the case) and converts binary to code. If the file does not exist because the processes did not crash (ie result from *`File.read`* is *`:error`*) then an empty map is returned to "do nothing about it" and handle the error. This function is used in the *`Init`* event where you initially want to read in the state. If the files do exist, you know a process has crashed so you will merge your current state with what is in the files. If the files do not exist then carry on as normal.

## Quorum of Processes

To get a quorum of processes in the *prepared* event, I made a local variable *messagesLocal* and appended any incoming messages that are received by the leader to that variable. Then, I took the length of that variable and divided it by the number of participants. If that number was more than 0.5, then a quorum was achieved. Similarly for the *accepted* event, I kept a *quorum_counter* which was incremented every time a value was accepted. Then I divided the *quorum_counter* value with the number of participants and checked if that number was greater than 0.5 to see if a quorum was met.

## Find a_val Which Corresponds To The Highest a_bal

In order to find *a_val* which corresponds to the highest *a_bal*, I used *Enum.reduce* which can be used to find a single value. I passed in my *messagesLocal* variable, and checked if the value in index 1 of the accumulator was less than the value in index 1 of x (we compare *a_bal* values and find the greatest one) and we find the *a_val* that corresponds to that value. We also check to see if the *a_bal* value is 0, and if so, we propose the initially set value of *valueToPropose*.

## Not Deciding Twice

In order for a process not to decide twice, I have a *decide* event which, when reached, sets the decided value to *true*. This event will only be reached once a quorum of processes have accepted the value. This means that in the *trust* event, if the decided value is *true*, there will not be another round of proposals because the processes have decided on a value.