This is a specification of a variant of the classic Paxos consensus algorithm described in

volume = 16, Number = 2, Month = may, Year = 1998, pages = “133–169”

This algorithm was also described without proof in Brian Oki’s Ph.D. thesis.

It describes the actions that can be performed by leaders, but does not introduce explicit leader processes. More precisely, the specification is written as if there were a separate leader for each ballot.

This variant of the classic Paxos algorithm is an abstraction of an algorithm that is used in

editor = {Srikanta Tirthapura and Lorenzo Alvisi},
publisher = {ACM}, YEAR = 2009, PAGES = “312–313”

and in

Cheap paxos United States Patent 7249280 Inventors: Lamport, Leslie B.
Massa, Michael T.
Filing Date:06/18/2004

In the classic Paxos algorithm, the leader sends a phase 2a message for a ballot b and value v that instructs acceptors to vote for v in ballot b. In terms of implementing the voting algorithm of module VoteProof, that 2a message serves two functions:

- It asserts that value v is safe at ballot b, so the acceptor can vote for it without violating invariant VInv2
- It tells the acceptors which single safe value they can vote for in ballot b, so they can vote for that value without violating VInv3.

The variant of the algorithm we specify here introduces phase 1c messages that perform the first function. The phase 2a message serves only the first function, being sent only if a 1c message had been sent for the value.

This variant of the algorithm is useful when reconfiguration is performed by using different sets of acceptors for different ballots. The leader propagates knowledge of what values are safe at ballot b so that the acceptors in the current configuration are no longer needed to determine that information. If the ballot b leader determines that all values are safe at b, then it sends a 1c message for every value and sends a phase 2a message only when it has a value to propose. The presence of the 1c messages removes dependency on the acceptors of ballots numbered b or lower for progress. (If the leader determines that only a single value is safe at b, then it sends the 1c and 2a messages together.)

In the algorithm described here, we do not include reconfiguration. Therefore, the sending of a 1c message serves only as a precondition for the sending of a 2a message with that value.

Classic Paxos and its variants maintain consensus in the presence of omission faults—faults in which a process fails to perform some enabled action or a message that is sent fails to be received. The safety specification, which is given by the PlusCal code, does not require that any action need ever be performed. A process need not execute an enabled action. Receipt of a message is modeled by a process performing the action enabled by that message having been sent, so message loss is also represented by a process not performing an enabled action. Thus, failures are never mentioned in the description of the algorithm.
The constant parameters and the set Ballots are the same as in the voting algorithm.

\textbf{CONSTANT Value, Acceptor, Quorum}

Assume $QA \doteq \forall Q \in \text{Quorum} : Q \subseteq \text{Acceptor}$

\[ \land \forall Q_1, Q_2 \in \text{Quorum} : Q_1 \cap Q_2 \neq \{\} \]

\textbf{Ballot} \doteq \text{Nat}

We are going to have a leader process for each ballot and an acceptor process for each acceptor. So we can use the ballot numbers and the acceptors themselves as the identifiers for these processes, we assume that the set of ballots and the set of acceptors are disjoint. For good measure, we also assume that $-1$ is not an acceptor, although that is probably not necessary.

Assume $\text{BallotAssump} \doteq (\text{Ballot} \cup \{-1\}) \cap \text{Acceptor} = \{\}$

We define $\text{None}$ to be an unspecified value that is not in the set $\text{Value}$.

$\text{None} \doteq \text{CHOOSE } v : v \notin \text{Value}$

This is a message-passing algorithm, so we begin by defining the set \text{Message} of all possible messages. The messages are explained below with the actions that send them. A message $m$ with $m\.type = "1a"$ is called a 1a message, and similarly for the other message types.

\textbf{Message} \doteq \begin{align*}
\{ & \text{type} : \{"1a"\}, \text{bal} : \text{Ballot} \\
\cup & \{ \text{type} : \{"1b"\}, \text{acc} : \text{Acceptor}, \text{bal} : \text{Ballot}, \\
& \text{mbal} : \text{Ballot} \cup \{-1\}, \text{mval} : \text{Value} \cup \{\text{None}\} \} \\
\cup & \{ \text{type} : \{"1c"\}, \text{bal} : \text{Ballot}, \text{val} : \text{Value} \} \\
\cup & \{ \text{type} : \{"2a"\}, \text{bal} : \text{Ballot}, \text{val} : \text{Value} \} \\
\cup & \{ \text{type} : \{"2b"\}, \text{acc} : \text{Acceptor}, \text{bal} : \text{Ballot}, \text{val} : \text{Value} \} \\
\} \end{align*}

The algorithm is easiest to understand in terms of the set \text{msgs} of all messages that have ever been sent. A more accurate model would use one or more variables to represent the messages actually in transit, and it would include actions representing message loss and duplication as well as message receipt.

In the current spec, there is no need to model message loss explicitly. The safety part of the spec says only what messages may be received and does not assert that any message actually is received. Thus, there is no difference between a lost message and one that is never received. The liveness property of the spec will make it clear what messages must be received (and hence either not lost or successfully retransmitted if lost) to guarantee progress.

Another advantage of maintaining the set of all messages that have ever been sent is that it allows us to define the state function \text{votes} that implements the variable of the same name in the voting algorithm without having to introduce a history variable.

\textbf{**********}

In addition to the variable \text{msgs}, the algorithm uses four variables whose values are arrays indexed by acceptor, where for any acceptor $a$:

$\text{maxBal}[a]$ The largest ballot number in which $a$ has participated

$\text{maxVBal}[a]$ The largest ballot number in which $a$ has voted, or $-1$ if it has never voted.
maxVVal[a] If a has voted, then this is the value it voted for in ballot maxVBal; otherwise it
equals None.

As in the voting algorithm, an execution of the algorithm consists of an execution of zero or more
ballots. Different ballots may be in progress concurrently, and ballots may not complete (and
need not even start). A ballot b consists of the following actions (which need not all occur in the
indicated order).

Phase 1a: The leader sends a 1a message for ballot b

Phase 1b: If maxBal[a] < b, an acceptor a responds to the 1a message by setting maxBal[a] to
b and sending a 1b message to the leader containing the values of maxVBal[a] and
maxVVal[a].

Phase 1c: When the leader has received ballot-b 1b messages from a quorum, it determines some
set of values that are safe at b and sends 1c messages for them.

Phase 2a: The leader sends a 2a message for some value for which it has already sent a ballot-
b 1c message.

Phase 2b: Upon receipt of the 2a message, if maxBal[a] ≤ b, an acceptor a sets maxBal[a] and
maxVBal[a] to b, sets maxVVal[a] to the value in the 2a message, and votes for that
value in ballot b by sending the appropriate 2b message.

Here is the PlusCal code for the algorithm, which we call PCon.

```
--algorithm PCon{
  variables maxBal = [a ∈ Acceptor ↦ -1],
  maxVBal = [a ∈ Acceptor ↦ -1],
  maxVVal = [a ∈ Acceptor ↦ None],
  msgs = {}
  define {
    sentMsgs(t, b) ∆= \{ m ∈ msgs : \( m.type = \text{t} \) ∧ \( m.bal = b \) \}
    ShowsSafeAt(Q, b, v) ∆= \{ m ∈ sentMsgs("1b", b) : m.acc ∈ Q \}
    \( \forall a ∈ Q : \exists m ∈ Q1b : m.acc = a \)
    \( \forall m ∈ Q1b : m.mbal = -1 \)
    \( \exists m1c ∈ msgs : \)
    \( \exists m1c = \{ \text{type} ↦ "1c", \text{bal} ↦ m1c.bal, \text{val} ↦ v \} \)
    \( \forall m ∈ Q1b : m1c.bal ≥ m.mbal \)
    \( m1c.bal = m.mbal \Rightarrow (m.mval = v) \)
  }
}
```

The following two macros send a message and a set of messages, respectively. These macros are
so simple that they’re hardly worth introducing, but they do make the processes a little easier
to read.

```
macro SendMessage(m)\{msgs := msgs ∪ \{m\}\}
macro SendSetOfMessages(S)\{msgs := msgs ∪ S\}
```

The Actions
As before, we describe each action as a macro.

The leader for process self can execute a Phase1a() action, which sends the ballot self 1a message.

```
macro Phase1a() { SendMessage({type ↦ "1a", bal ↦ self}) }
```

Acceptor self can perform a Phase1b(b) action, which is enabled iff \( b > \text{maxBal}[\text{self}] \). The action sets \( \text{maxBal}[\text{self}] \) to \( b \) and sends a phase 1b message to the leader containing the values of \( \text{maxVBal}[\text{self}] \) and \( \text{maxVVal}[\text{self}] \).

```
macro Phase1b(b) {
    when (b > \text{maxBal}[\text{self}]) ∧ (\text{sentMsgs("1a"), b} ≠ {}) ;
    \text{maxBal}[\text{self}] := b ;
    \text{SendMessage}({\text{type} ↦ "1b", acc ↦ \text{self}, bal ↦ b,
    mbal ↦ \text{maxVBal}[\text{self}], mval ↦ \text{maxVVal}[\text{self}]) ;
}
```

The ballot self leader can perform a Phase1c(S) action, which sends a set \( S \) of 1c messages indicating that the value in the val field of each of them is safe at ballot \( b \). In practice, \( S \) will either contain a single message, or else will have a message for each possible value, indicating that all values are safe. In the first case, the leader will immediately send a 2a message with the value contained in that single message. (Both logical messages will be sent in the same physical message.) In the latter case, the leader is informing the acceptors that all values are safe. (All those logical messages will, of course, be encoded in a single physical message.)

```
macro Phase1c(S) {
    when \( \forall v \in S : \exists Q \in \text{Quorum} : \text{ShowsSafeAt}(Q, \text{self}, v) \);
    \text{SendSetOfMessages}({{\text{type} ↦ "1c", bal ↦ \text{self}, val ↦ v} : v \in S})
}
```

The ballot self leader can perform a Phase2a(v) action, sending a 2a message for value \( v \), if it has not already sent a 2a message (for this ballot) and it has sent a ballot self 1c message with val field \( v \).

```
macro Phase2a(v) {
    when \( \land \text{sentMsgs("2a", self)} = {} \)
    \land [{\text{type} ↦ "1c", bal ↦ \text{self}, val ↦ v}] \in \text{msgs} ;
    \text{SendMessage}({\text{type} ↦ "2a", bal ↦ \text{self}, val ↦ v})
}
```

The Phase2b(b) action is executed by acceptor self in response to a ballot-b 2a message. Note this action can be executed multiple times by the acceptor, but after the first one, all subsequent executions are stuttering steps that do not change the value of any variable.

```
macro Phase2b(b) {
    when \( b \geq \text{maxBal}[\text{self}] \) ;
    with (m \in \text{sentMsgs("2a", b)}){
        \text{maxBal}[\text{self}] := b ;
        \text{maxVBal}[\text{self}] := b ;
        \text{maxVVal}[\text{self}] := m.val ;
        \text{SendMessage}({\text{type} ↦ "2b", acc ↦ \text{self}, bal ↦ b, val ↦ m.val})
    }
}
```
An acceptor performs the body of its while loop as a single atomic action by nondeterministically choosing a ballot in which its Phase1b or Phase2b action is enabled and executing that enabled action. If no such action is enabled, the acceptor does nothing.

```plaintext
process (acceptor ∈ Acceptor)
{
  acc: while (TRUE)
  {
    with (b ∈ Ballot)
    {
      either Phase1b(b) or Phase2b(b)
    }
  }
}
```

The leader of a ballot nondeterministically chooses one of its actions that is enabled (and the argument for which it is enabled) and performs it atomically. It does nothing if none of its actions is enabled.

```plaintext
process (leader ∈ Ballot)
{
  ldr: while (TRUE)
  {
    either Phase1a() or with (S ∈ subset Value) {Phase1c(S)}
    or with (v ∈ Value) {Phase2a(v)}
  }
}
```

The translator produces the following TLA+ specification of the algorithm. Some blank lines have been deleted.

```
BEGIN TRANSLATION
VARIABLES maxBal, maxVBal, maxVVal, msgs

define statement
  sentMsgs(t, b) \doteq \{ m ∈ msgs : (m.type = t) \land (m.bal = b) \}

  ShowsSafeAt(Q, b, v) \doteq
  let Q1b \doteq \{ m ∈ sentMsgs("1b", b) : m.acc ∈ Q \}
  in
  \forall a ∈ Q : \exists m ∈ Q1b : m.acc = a
  \lor \forall m ∈ Q1b : m.mbal = -1
  \lor \exists m1c ∈ msgs :
    \land m1c = [type \mapsto "1c", bal \mapsto m1c.bal, val \mapsto v]
  \land \forall m ∈ Q1b :
    \land m1c.bal ≥ m.mbal
    \land (m1c.bal = m.mbal) ⇒ (m.mval = v)

vars \doteq (maxBal, maxVBal, maxVVal, msgs)

ProcSet \doteq (Acceptor) \cup (Ballot)

Init \doteq Global variables
```

5
\(\land \text{maxBal} = [a \in \text{Acceptor} \mapsto -1]\)
\(\land \text{maxVBal} = [a \in \text{Acceptor} \mapsto -1]\)
\(\land \text{maxVVal} = [a \in \text{Acceptor} \mapsto \text{None}]\)
\(\land \text{msgs} = \{\}\)

\text{acceptor}(self) \triangleq \exists b \in \text{Ballot}:
\begin{align*}
\lor (b > \text{maxBal}(self)) & \land (\text{sentMsgs}("1a", b) \neq \{\}) \\
\land \text{maxBal}' = [\text{maxBal} \text{EXCEPT} ![self] = b] & \\
\land \text{msgs}' = (\text{msgs} \cup \{([\text{type} \mapsto "1b", \text{acc} \mapsto self, \text{bal} \mapsto b, \\
\text{mbal} \mapsto \text{maxVBal}[self], \text{mval} \mapsto \text{maxVVal}[self]])\}) \\
\land \text{UNCHANGED} (\text{maxVBal}, \text{maxVVal}) & \\
\lor b \geq \text{maxBal}(self)
\end{align*}

\text{leader}(self) \triangleq \land \lor \exists S \in \text{subset Value}:
\begin{align*}
\land \forall v \in S : \exists Q \in \text{Quorum} : \text{ShowsSafeAt}(Q, self, v) & \\
\land \text{msgs}' = (\text{msgs} \cup \{([\text{type} \mapsto "1c", \text{bal} \mapsto self, \text{val} \mapsto v] : v \in S)\}) \\
\lor \exists v \in \text{Value}:
\begin{align*}
\land \land \text{sentMsgs}("2a", self) = \{\} & \\
\land [\text{type} \mapsto "1c", \text{bal} \mapsto self, \text{val} \mapsto v] \in \text{msgs} & \\
\land \text{msgs}' = (\text{msgs} \cup \{([\text{type} \mapsto "2a", \text{bal} \mapsto self, \text{val} \mapsto v}]\}) \\
\land \text{UNCHANGED} (\text{maxBal}, \text{maxVBal}, \text{maxVVal})
\end{align*}
\end{align*}
∧ UNCHANGED (\textit{maxBal}, \textit{maxVBal}, \textit{maxVVal})

\textbf{Phase2a}(\textit{self}, v) \triangleq
∧ \text{sentMsgs}("2a", \textit{self}) = \{
∧ \{\text{type} \mapsto \text{"1c"}, \text{bal} \mapsto \textit{self}, \text{val} \mapsto v\} \in \text{msgs}
∧ \text{msgs'} = (\text{msgs} \cup \{\{\text{type} \mapsto \text{"2a"}, \text{bal} \mapsto \textit{self}, \text{val} \mapsto v\}\})
∧ UNCHANGED (\textit{maxBal}, \textit{maxVBal}, \textit{maxVVal})

\textbf{Phase1b}(\textit{self}, b) \triangleq
∧ b > \textit{maxBal}[\textit{self}]
∧ \text{sentMsgs}("1a", b) \not\subseteq \{
∧ \textit{maxBal}' = [\textit{maxBal} \text{ EXCEPT } ![\textit{self}] = b]
∧ \text{msgs}' = \text{msgs} \cup \{\{\text{type} \mapsto \text{"1b"}, \text{acc} \mapsto \textit{self}, \text{bal} \mapsto b,
\text{mbal} \mapsto \textit{maxVBal}[\textit{self}], \text{mval} \mapsto \textit{maxVVal}[\textit{self}]\}\}
∧ UNCHANGED (\textit{maxVBal}, \textit{maxVVal})

\textbf{Phase2b}(\textit{self}, b) \triangleq
∧ b \geq \textit{maxBal}[\textit{self}]
∧ \exists m \in \text{sentMsgs}("2a", b):
∧ \textit{maxBal}' = [\textit{maxBal} \text{ EXCEPT } ![\textit{self}] = b]
∧ \textit{maxVBal}' = [\textit{maxVBal} \text{ EXCEPT } ![\textit{self}] = b]
∧ \textit{maxVVal}' = [\textit{maxVVal} \text{ EXCEPT } ![\textit{self}] = \textit{m.val}]
∧ \text{msgs}' = (\text{msgs} \cup \{\{\text{type} \mapsto \text{"2b"}, \text{acc} \mapsto \textit{self},
\text{bal} \mapsto b, \text{val} \mapsto \textit{m.val}\}\})

\textbf{TLANext} \triangleq
\lor \exists \textit{self} \in \text{Acceptor}:
\exists b \in \text{Ballot} : \lor \text{Phase1b}(\textit{self}, b)
\lor \text{Phase2b}(\textit{self}, b)
\lor \exists \textit{self} \in \text{Ballot} :
\lor \text{Phase1a}(\textit{self})
\lor \exists S \in \text{subset Value} : \text{Phase1c}(\textit{self}, S)
\lor \exists v \in \text{Value} : \text{Phase2a}(\textit{self}, v)

The following theorem specifies the relation between the next-state relation \textit{Next} obtained by translating the \textit{PlusCal} code and the next-state relation \textit{TLANext}.

\textbf{THEOREM NextDef} \triangleq (\textit{Next} \equiv \textit{TLANext})
\langle 1 \rangle 2. \textbf{ASSUME NEW} \textit{self} \in \text{Acceptor}
\textbf{PROVE} accceptor(\textit{self}) \equiv \textit{TLANext}!!'(\textit{self})
\textbf{BY} \langle 1 \rangle 2, \textbf{BallotAssump def accператор, ProcSet, Phase1b, Phase2b}
\langle 1 \rangle 3. \textbf{ASSUME NEW} \textit{self} \in \text{Ballot}
\textbf{PROVE} leader(\textit{self}) \equiv \textit{TLANext}!!'!(\textit{self})
\textbf{BY} \langle 1 \rangle 3, \textbf{BallotAssump, Zenon def leader, ProcSet, Phase1a, Phase1c, Phase2a}
\langle 1 \rangle 4. \textbf{QED}
\textbf{BY} \langle 1 \rangle 2, \langle 1 \rangle 3 \textbf{ DEF Next, TLANext}
The type invariant.

\[
\text{TypeOK} \triangleq \land \text{maxBal} \in [\text{Acceptor} \rightarrow \text{Ballot} \cup \{-1\}]
\land \text{maxVBal} \in [\text{Acceptor} \rightarrow \text{Ballot} \cup \{-1\}]
\land \text{maxVVal} \in [\text{Acceptor} \rightarrow \text{Value} \cup \{\text{None}\}]
\land \text{msgs} \subseteq \text{Message}
\]

Here is the definition of the state-function \(\text{chosen}\) that implements the state-function of the same name in the voting algorithm.

\[
\text{chosen} \triangleq \{ v \in \text{Value} : \exists Q \in \text{Quorum}, \ b \in \text{Ballot} : \\
\forall a \in Q : \exists m \in \text{msgs} : \land m.\text{type} = \text{"2b"} \\
\land m.\text{acc} = a \\
\land m.\text{bal} = b \\
\land m.\text{val} = v \}
\]

We now define the refinement mapping under which this algorithm implements the specification in module Voting.

As we observed, votes are registered by sending phase 2b messages. So the array \(\text{votes}\) describing the votes cast by the acceptors is defined as follows.

\[
\text{votes} \triangleq [a \in \text{Acceptor} \mapsto \{ (m.\text{bal}, m.\text{val}) : m \in \{ mm \in \text{msgs} : \land mm.\text{type} = \text{"2b"} \\
\land mm.\text{acc} = a \} \}]
\]

We now instantiate module Voting, substituting:

- The constants \(\text{Value, Acceptor, and Quorum}\) declared in this module for the corresponding constants of that module Voting.

- The variable \(\text{maxBal}\) and the defined state function \(\text{votes}\) for the correspondingly-named variables of module Voting.

\[
V \triangleq \text{instance VoteProof}
\]

We now define \(P_{\text{inv}}\) to be what I believe to be an inductive invariant and assert the theorems for proving that this algorithm implements the voting algorithm under the refinement mapping specified by the instance statement. Whether \(P_{\text{inv}}\) really is an inductive invariant will be determined only by a rigorous proof.

\[
P_{\text{AccInv}} \triangleq \forall a \in \text{Acceptor} : \\
\land \text{maxBal}[a] \geq \text{maxVBal}[a] \\
\land \forall b \in (\text{maxVBal}[a] + 1) .. (\text{maxBal}[a] - 1) : V!\text{DidNotVoteIn}(a, b) \\
\land (\text{maxVBal}[a] \neq -1) \Rightarrow V!\text{VotedFor}(a, \text{maxVBal}[a], \text{maxVVal}[a])
\]

\[
P_{\text{1bInv}} \triangleq \forall m \in \text{msgs} : \\
(m.\text{type} = \text{"1b"}) \Rightarrow \\
\land (\text{maxBal}[m.\text{acc}] \geq m.\text{bal}) \land (m.\text{bal} > m.mbal) \\
\land \forall b \in (m.mbal + 1) .. (m.\text{bal} - 1) : V!\text{DidNotVoteIn}(m.\text{acc}, b)
\]

\[
P_{\text{1cInv}} \triangleq \forall m \in \text{msgs} : (m.\text{type} = \text{"1c"}) \Rightarrow V!\text{SafeAt}(m.\text{bal}, m.\text{val})
\]
\[ P2aInv \triangleq \forall m \in \text{msgs} : \]
\[ (m.\text{type} = "2a") \Rightarrow \exists m1c \in \text{msgs} : \]
\[ m1c.\text{type} = "1c" \]
\[ m1c.\text{bal} = m.\text{bal} \]
\[ m1c.\text{val} = m.\text{val} \]

The following theorem is interesting in its own right. It essentially asserts the correctness of the definition of ShowsSafeAt.

**THEOREM** \( PT1 \triangleq \text{TypeOK} \land P1bInv \land P1cInv \Rightarrow \)
\[ \forall Q \in \text{Quorum}, b \in \text{Ballot}, v \in \text{Value} : \]
\[ \text{ShowsSafeAt}(Q, b, v) \Rightarrow \text{V!SafeAt}(b, v) \]

\( PInv \triangleq \text{TypeOK} \land P\text{AccInv} \land P1bInv \land P1cInv \land P2aInv \)

**THEOREM** \( \text{Invariance} \triangleq \text{Spec} \Rightarrow \square PInv \)

**THEOREM** \( \text{Implementation} \triangleq \text{Spec} \Rightarrow \text{V!Spec} \)

The following result shows that our definition of \( \text{chosen} \) is the correct one, because it implements the state-function \( \text{chosen} \) of the voting algorithm.

**THEOREM** \( \text{Spec} \Rightarrow \square(\text{chosen} = \text{V!chosen}) \)

The four theorems above have been checked by \( \text{TLC} \) for a model with 3 acceptors, 2 values, and 3 ballot numbers. Theorem \( PT1 \) was checked as an invariant, therefore checking only that it is true for all reachable states. This model is large enough that it would most likely have revealed any “coding” errors in the algorithm. We believe that the algorithm is well-enough understood that it is unlikely to contain any fundamental errors.
(* then some value is eventually chosen. *)
(* Note that Phase2a(b) is enabled if msgs contains a ballot b phase 1b *)
(* message from every acceptor in Q. Hence, 4 implies that if the leader *)
(* eventually receives those messages, then it must perform its Phase2a(b) *)
(* action. It might perform that action before it receives those *)
(* messages if it has received phase 1b messages from all the acceptors in *)
(* a different quorum. *)

\[
\text{THEOREM Liveness } = \forall b \in \text{Ballot}, Q \in \text{Quorum} : \begin{align*}
&\forall m \in \text{msgs} : (m.\text{type} = "1a") \Rightarrow (m.\text{bal} < b) \land \\
&\forall c \in \text{Ballot} : (c > b) \Rightarrow \neg \text{Phase1a}(c) \land \\
&\text{WF-vars}(\text{Phase1a}(b)) \land \\
&\text{WF-vars}(\exists v \in \text{Value} : \text{Phase2a}(b, v)) \land \\
&\forall a \in Q : \land \text{WF-vars}(\text{Phase1b}(a, b)) \land \\
&\text{WF-vars}(\text{Phase2b}(a, b)) \land \neg (\text{chosen} \neq \{\})
\end{align*}
\]

\[
\text{CSpec } = \land \text{Init} \land \land \text{Next} \land \forall c \in \text{Ballot} : (c > b) \Rightarrow \neg \text{Phase1a}(c) \land \\
\land \text{WF-vars}(\text{Phase1a}(bb)) \land \\
\land \text{WF-vars}(\exists v \in \text{Value} : \text{Phase2a}(bb, v)) \land \\
\forall a \in Q : \land \text{WF-vars}(\text{Phase1bForBallot}(a, bb)) \land \\
\land \text{WF-vars}(\text{Phase2bForBallot}(a, bb))
\]

\[
\text{CLiveness } = \forall m \in \text{msgs} : (m.\text{type} = "1a") \Rightarrow (m.\text{bal} < bb) \land \neg (\text{chosen} \neq \{\})
\]