The Paxos Consensus Algorithm

The Paxos algorithm implements the voting algorithm.

The Paxos algorithm implements the voting algorithm.

Remember that the voting algorithm is based on two rules:

The Paxos algorithm implements the voting algorithm.

Remember that the voting algorithm is based on two rules:

1. Don’t allow different acceptors to vote for different values in the same ballot.

The Paxos algorithm implements the voting algorithm.

Remember that the voting algorithm is based on two rules:

1. Don't allow different acceptors to vote for different values in the same ballot.
2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

1. Don't allow different acceptors to vote for different values in the same ballot.
2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

1. Don’t allow different acceptors to vote for different values in the same ballot.
2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

1. Don’t allow different acceptors to vote for different values in the same ballot.
2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

Each ballot has a unique leader process.

1. Don’t allow different acceptors to vote for different values in the same ballot.
2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

Each ballot has a unique leader process.

We don’t care who the leaders are.

1. Don’t allow different acceptors to vote for different values in the same ballot.
2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

Each ballot has a unique leader process.

We don’t care who the leaders are.

The same process may be the leader for many different ballots.

1. Don't allow different acceptors to vote for different values in the same ballot.
2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

Each ballot has a unique leader process.

The leader of a ballot tells acceptors what value to vote for in that ballot.

1. Don’t allow different acceptors to vote for different values in the same ballot.
2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

1. Don’t allow different acceptors to vote for different values in the same ballot.

2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

The leader begins the ballot by asking acceptors about votes they have already cast.

1. Don't allow different acceptors to vote for different values in the same ballot.
2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

The leader begins the ballot by asking acceptors about votes they have already cast.

If it receives responses from a quorum of acceptors,

1. Don’t allow different acceptors to vote for different values in the same ballot.

2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

The leader begins the ballot by asking acceptors about votes they have already cast.

If it receives responses from a quorum of acceptors, it will be able to choose a safe value and tell them to vote for it.

1. Don’t allow different acceptors to vote for different values in the same ballot.
2. Allow an acceptor to vote for value v in ballot b only if v is safe at b.

The leader begins the ballot by asking acceptors about votes they have already cast.

If it receives responses from a quorum of acceptors, it will be able to choose a safe value and tell them to vote for it.

Let’s look at the spec.

MODULE *Paxos*

MODULE *Paxos*

EXTENDS *Integers*

The module begins just like module *Voting* .

MODULE *Paxos*

EXTENDS *Integers*

CONSTANTS *Value*, *Acceptor*, *Quorum*

The module begins just like module *Voting* .

MODULE *Paxos*

EXTENDS *Integers*

CONSTANTS *Value*, *Acceptor*, *Quorum*

ASSUME $\land \forall\, Q \in Quorum : Q \subseteq Acceptor$
$\land \forall\, Q1,\, Q2 \in Quorum : Q1 \cap Q2 \neq \{\}$

The module begins just like module *Voting* .

MODULE *Paxos*

EXTENDS *Integers*

CONSTANTS *Value*, *Acceptor*, *Quorum*

ASSUME $\land \forall\, Q \in Quorum : Q \subseteq Acceptor$
$\land \forall\, Q1,\, Q2 \in Quorum : Q1 \cap Q2 \neq \{\}$

$Ballot \triangleq Nat$

The module begins just like module *Voting* .

MODULE *Paxos*

EXTENDS *Integers*

CONSTANTS *Value*, *Acceptor*, *Quorum*

ASSUME $\land \forall\, Q \in Quorum : Q \subseteq Acceptor$
$\land \forall\, Q1,\, Q2 \in Quorum : Q1 \cap Q2 \neq \{\}$

$Ballot \triangleq Nat$

The module begins just like module *Voting* .

MODULE *Paxos*

EXTENDS *Integers*

CONSTANTS *Value*, *Acceptor*, *Quorum*

ASSUME $\land \forall\, Q \in Quorum : Q \subseteq Acceptor$
$\land \forall\, Q1,\, Q2 \in Quorum : Q1 \cap Q2 \neq \{\}$

$Ballot \triangleq Nat$

The module begins just like module *Voting* .

$$None \triangleq \text{CHOOSE } v : v \notin Ballot$$

$$None \triangleq \text{CHOOSE } v : v \notin Ballot$$

This defines $None$ to be some value that is not an element of the set $Ballot$.

$$None \triangleq \text{CHOOSE } v : v \notin Ballot$$

This defines $None$ to be some value that is not an element of the set $Ballot$.

We don't know (or care) what value.

$$None \triangleq \boxed{\text{CHOOSE } v : v \notin Ballot}$$

This defines $None$ to be some value that is not an element of the set $Ballot$.

We don't know (or care) what value.

This is a mathematical expression,

$$None \triangleq \boxed{\text{CHOOSE } v : v \notin Ballot}$$

This defines $None$ to be some value that is not an element of the set $Ballot$.

We don't know (or care) what value.

This is a mathematical expression, so evaluating it for the same value of $Ballot$ always yields the same result.

$$None \triangleq \boxed{\text{CHOOSE } v : v \notin Ballot}$$

This defines $None$ to be some value that is not an element of the set $Ballot$.

We don't know (or care) what value.

This is a mathematical expression, so evaluating it for the same value of $Ballot$ always yields the same result.

And $Ballot$ is a constant, so its value doesn't change,

$$None \triangleq \boxed{\text{CHOOSE } v : v \notin Ballot}$$

This defines $None$ to be some value that is not an element of the set $Ballot$.

We don't know (or care) what value.

This is a mathematical expression, so evaluating it for the same value of $Ballot$ always yields the same result.

And $Ballot$ is a constant, so its value doesn't change, and therefore, neither does the value of $None$.

$$Message \triangleq$$

$$Message \triangleq$$

The leaders and acceptors will communicate with messages.

$Message \triangleq$

The leaders and acceptors will communicate with messages.

$Message$ is defined to be the set of all messages that can ever be sent.

$Message \triangleq$

The leaders and acceptors will communicate with messages.

$Message$ is defined to be the set of all messages that can ever be sent.

A message is represented by a record.

Records

Records

A record is similar to a *struct* in C,

Records

A record is similar to a *struct* in C, or an *object* in an OO language,

Records

A record is similar to a *struct* in C, or an *object* in an OO language, with fields but no concept of methods.

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“2}a\text{”}, \qquad\qquad\qquad\qquad\qquad\qquad]$$

with $type$ field equal to “2a”

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42, \qquad\qquad]$$

with $type$ field equal to “$2a$”, bal field equal to 42

Records

It is defined mathematically to be a function
whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]$$

with $type$ field equal to “$2a$”, bal field equal to 42,
and val field equal to $\sqrt{7}$

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]$$

is the function

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]$$

is the function

$$[\, x \in \{\text{“}type\text{”}, \text{“}bal\text{”}, \text{“}val\text{”}\} \mapsto$$

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]$$

is the function

$$[\, x \in \{\text{“}type\text{”}, \text{“}bal\text{”}, \text{“}val\text{”}\} \mapsto$$

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]$$

is the function

$$[\, x \in \{\text{“}type\text{”}, \text{“}bal\text{”}, \text{“}val\text{”}\} \mapsto$$

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]$$

is the function

$$[\, x \in \{\text{“}type\text{”}, \text{“}bal\text{”}, \text{“}val\text{”}\} \mapsto$$
$$\quad \text{IF } x = \text{“}type\text{”} \text{ THEN } \text{“}2a\text{”}$$

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]$$

is the function

$$[\, x \in \{\text{“}type\text{”}, \text{“}bal\text{”}, \text{“}val\text{”}\} \mapsto$$
$$\text{IF } x = \text{“}type\text{”} \text{ THEN “}2a\text{”}$$
$$\text{ELSE IF } x = \text{“}bal\text{”} \text{ THEN } 42$$

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]$$

is the function

$$\begin{array}{l}[\, x \in \{\text{“}type\text{”}, \text{“}bal\text{”}, \text{“}val\text{”}\} \mapsto \\ \quad \text{IF } x = \text{“}type\text{”} \text{ THEN } \text{“}2a\text{”} \\ \qquad\qquad \text{ELSE IF } x = \text{“}bal\text{”} \text{ THEN } 42 \\ \qquad\qquad\qquad\qquad \text{ELSE } \sqrt{7}\ ]\end{array}$$

Records

It is defined mathematically to be a function whose domain is a finite set of strings.

For example this record

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]$$

is the function

$$\begin{array}{l}[\, x \in \{\text{“}type\text{”}, \text{“}bal\text{”}, \text{“}val\text{”}\} \mapsto \\ \quad \text{IF } x = \text{“}type\text{”} \text{ THEN } \text{“}2a\text{”} \\ \qquad\qquad \text{ELSE IF } x = \text{“}bal\text{”} \text{ THEN } 42 \\ \qquad\qquad\qquad\qquad \text{ELSE } \sqrt{7}\ ]\end{array}$$

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]$$

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]\,[\text{“}bal\text{”}] \;=\; 42$$

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]\,[\text{“}bal\text{”}]$$

$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]\,[\text{“}bal\text{”}]$

We abbreviation $[\text{“}bal\text{”}]$

$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]\,.\,bal$

We abbreviation $[\text{“}bal\text{”}]$ as $.bal$

$$[\, type \mapsto \text{“}2a\text{”},\ bal \mapsto 42,\ val \mapsto \sqrt{7}\,]\,.\,bal \;=\; 42$$

We abbreviation $[\text{“}bal\text{”}]$ as $.bal$

$$Message \triangleq$$

Defined to be a set of records.

$$
\begin{array}{lll}
Message \triangleq & & \\
 & [type : \{\text{“1a”}\},\ bal : Ballot] & \\
\cup & [type : \{\text{“1b”}\},\ acc : Acceptor,\ bal : Ballot, & \\
 & \ mbal : Ballot \cup \{-1\},\ mval : Value \cup \{None\}] & \\
\cup & [type : \{\text{“2a”}\},\ bal : Ballot,\ val : Value] & \\
\cup & [type : \{\text{“2b”}\},\ acc : Acceptor,\ bal : Ballot,\ val : Value] &
\end{array}
$$

Defined to be a set of records.

$$
\begin{array}{ll}
Message \triangleq & \\
 & [type : \{\text{“1a”}\},\ bal : Ballot] \\
\cup & [type : \{\text{“1b”}\},\ acc : Acceptor,\ bal : Ballot, \\
 & \ mbal : Ballot \cup \{-1\},\ mval : Value \cup \{None\}] \\
\cup & [type : \{\text{“2a”}\},\ bal : Ballot,\ val : Value] \\
\cup & [type : \{\text{“2b”}\},\ acc : Acceptor,\ bal : Ballot,\ val : Value]
\end{array}
$$

Defined to be a set of records.

The union of four sets.

$$
\begin{aligned}
Message \;\triangleq\;\; & [\quad] \\
\cup\;\; & [\quad] \\
\cup\;\; & [\quad] \\
\cup\;\; & [\quad]
\end{aligned}
$$

$$
\begin{aligned}
Message \triangleq\ & [\qquad\qquad\qquad\qquad\qquad\qquad] \\
\cup\ & [\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad] \\
\cup\ & [type : \{\text{"2a"}\},\ bal : Ballot,\ val : Value] \\
\cup\ & [\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad]
\end{aligned}
$$

$$
\begin{array}{rl}
Message \triangleq & \\
 & [\qquad\qquad\qquad\qquad\qquad] \\
\cup & [\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad] \\
\cup & [type : \qquad\qquad, \ bal : \qquad\qquad, \ val : \qquad\qquad] \\
\cup & [\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad]
\end{array}
$$

The set of all records R having three fields $type$, bal, and val

$$Message \triangleq$$

$$\begin{array}{ll} & [\quad] \\ \cup & [\quad] \\ \cup & [type : \{\text{"2a"}\}, \quad , \quad] \\ \cup & [\quad] \end{array}$$

The set of all records R having three fields $type$, bal, and val with:

$R.type$ in the set $\{\text{"}2a\text{"}\}$

$Message \triangleq$

$[\quad\quad\quad\quad\quad\quad\quad\quad]$

$\cup \quad [$

$]$

$\cup \quad [\quad\quad\quad , \; bal : Ballot, \quad\quad]$

$\cup \quad [\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad]$

The set of all records R having three fields $type$, bal, and val with:

$R.type$ in the set $\{\text{“}2a\text{”}\}$

$R.bal$ in the set $Ballot$

$$
\begin{aligned}
Message \triangleq\ & [\qquad\qquad\qquad\qquad\qquad\qquad] \\
\cup\ & [\\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad] \\
\cup\ & [\qquad\qquad\qquad, \qquad\qquad\qquad, val : Value] \\
\cup\ & [\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad]
\end{aligned}
$$

The set of all records R having three fields $type$, bal, and val with:

$R.type$ in the set $\{\text{"2}a\text{"}\}$

$R.bal$ in the set $Ballot$

$R.val$ in the set $Value$

$$Message \triangleq$$

$$[\qquad]$$

$$\cup \quad [\qquad]$$

$$\cup \quad [type : \{\text{"2a"}\}, \quad , \quad]$$

$$\cup \quad [\qquad]$$

The set of all records R having three fields $type$, bal, and val with:

$R.type$ in the set $\{\text{"2}a\text{"}\}$

$R.bal$ in the set $Ballot$

$R.val$ in the set $Value$

$Message \triangleq$

$[\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad]$

$\cup\ [\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad]$

$\cup\ [type : \{\text{"2a"}\}, \quad\quad\quad\quad , \quad\quad\quad\quad]$

$\cup\ [\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad]$

The set of all records R having three fields $type$, bal, and val with:

$R.type = \text{"}2a\text{"}$

$R.bal$ in the set $Ballot$

$R.val$ in the set $Value$

$$
\begin{aligned}
Message \triangleq\ & [\qquad\qquad\qquad\qquad\qquad\qquad\qquad] \\
\cup\ & [\\
& \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad] \\
\cup\ & [type : \{\text{"2a"}\},\ bal : Ballot,\ val : Value] \\
\cup\ & [\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad]
\end{aligned}
$$

$$
\begin{aligned}
Message \triangleq\ & [type : \{\text{“1a”}\},\ bal : Ballot] \\
\cup\ & [type : \{\text{“1b”}\},\ acc : Acceptor,\ bal : Ballot, \\
& \ mbal : Ballot \cup \{-1\},\ mval : Value \cup \{None\}] \\
\cup\ & [type : \{\text{“2a”}\},\ bal : Ballot,\ val : Value] \\
\cup\ & [type : \{\text{“2b”}\},\ acc : Acceptor,\ bal : Ballot,\ val : Value]
\end{aligned}
$$

VARIABLES

VARIABLES $maxBal$

Implements variable $maxBal$ of the $Voting$ spec.

VARIABLES $maxBal$

Implements variable $maxBal$ of the $Voting$ spec.

$maxBal[a]$ is the number of the highest-numbered ballot in which acceptor a has participated.

VARIABLES $maxBal$

Implements variable $maxBal$ of the $Voting$ spec.

$maxBal[a]$ is the number of the highest-numbered ballot in which acceptor a has participated.

Or -1 if a hasn't participated in any ballots.

VARIABLES $maxBal$, $maxVBal$

VARIABLES $maxBal,\ maxVBal$

$maxVBal[a]$ is the number of the highest-numbered ballot in which acceptor a has voted.

VARIABLES $maxBal$, $maxVBal$

$maxVBal[a]$ is the number of the highest-numbered ballot in which acceptor a has voted.

Or -1 if a hasn't voted in any ballots.

VARIABLES $maxBal$, $maxVBal$, $maxVal$

VARIABLES $maxBal,\ maxVBal,\ maxVal$

$maxVal[a]$ is the value a voted for in ballot $maxVBal[a]$.

VARIABLES $maxBal$, $maxVBal$, $maxVal$

$maxVal[a]$ is the value a voted for in ballot $maxVBal[a]$.

Or $None$ if a has never voted.

VARIABLES $maxBal$, $maxVBal$, $maxVal$, $msgs$

VARIABLES $maxBal,\ maxVBal,\ maxVal,\ msgs$

The set of all messages that have been sent.

VARIABLES $maxBal,\ maxVBal,\ maxVal,\ msgs$

The set of all messages that have been sent.

$msgs$

The set of all messages that have been sent.

$msgs$

The set of all messages that have been sent.

$msgs$

The set of all messages that have been sent.

A message is sent by adding it to $msgs$.

$msgs$

The set of all messages that have been sent.

A message is sent by adding it to $msgs$.

Messages are never removed from $msgs$.

$msgs$

The set of all messages that have been sent.

A message is sent by adding it to $msgs$.

Messages are never removed from $msgs$.

This is the simplest, most abstract way I know to describe message passing.

$msgs$

It can be used to model the Paxos algorithm because:

$msgs$

It can be used to model the Paxos algorithm because:

– Paxos allows messages to be received in any order.

$msgs$

It can be used to model the Paxos algorithm because:

- Paxos allows messages to be received in any order.
- Since we're not specifying liveness, there's no need to explicitly describe message loss.

$msgs$

It can be used to model the Paxos algorithm because:

– Paxos allows messages to be received in any order.

– Since we're not specifying liveness, there's no need to explicitly describe message loss.

A spec of safety doesn't require any message to be received,

$msgs$

It can be used to model the Paxos algorithm because:

- Paxos allows messages to be received in any order.
- Since we're not specifying liveness, there's no need to explicitly describe message loss.

  A spec of safety doesn't require any message to be received, and there's no difference between a lost message and one that's never received.

$msgs$

The set of all messages that have been sent.

$msgs$

The set of all messages that have been sent.

This is a natural, obvious way to describe message passing

$msgs$

The set of all messages that have been sent.

This is a natural, obvious way to describe message passing if you think mathematically.

$msgs$

The set of all messages that have been sent.

This is a natural, obvious way to describe message passing if you think mathematically.

It doesn't occur to most people because they think in terms of programming languages, not math.

$$vars \triangleq \langle maxBal,\ maxVBal,\ maxVal,\ msgs \rangle$$

$$vars \triangleq \langle maxBal,\ maxVBal,\ maxVal,\ msgs \rangle$$

It's easier to write $[Next]_{vars}$

$$vars \triangleq \langle maxBal,\ maxVBal,\ maxVal,\ msgs \rangle$$

It's easier to write $[Next]_{vars}$

than $[Next]_{\langle maxBal,\ maxVBal,\ maxVal,\ msgs \rangle}$.

$TypeOK \triangleq$

$$TypeOK \triangleq \land maxBal \quad \in [Acceptor \to Ballot \cup \{-1\}]$$

$maxBal$ is a function from acceptors to ballot numbers or -1.

$$\begin{aligned} TypeOK \triangleq\ & \wedge maxBal \in [Acceptor \to Ballot \cup \{-1\}] \\ & \wedge maxVBal \in [Acceptor \to Ballot \cup \{-1\}] \end{aligned}$$

$maxBal$ is a function from acceptors to ballot numbers or -1.

$maxVBal$ is a function from acceptors to ballot numbers or -1.

$$
\begin{array}{lll}
TypeOK \triangleq & \wedge\, maxBal & \in [Acceptor \to Ballot \cup \{-1\}] \\
& \wedge\, maxVBal & \in [Acceptor \to Ballot \cup \{-1\}] \\
& \wedge\, maxVal & \in [Acceptor \to Value \cup \{None\}]
\end{array}
$$

$maxBal$ is a function from acceptors to ballot numbers or -1.

$maxVBal$ is a function from acceptors to ballot numbers or -1.

$maxVal$ is a function from acceptors to values or $None$.

$$
\begin{aligned}
TypeOK \triangleq\ & \land\ maxBal \in [Acceptor \to Ballot \cup \{-1\}] \\
& \land\ maxVBal \in [Acceptor \to Ballot \cup \{-1\}] \\
& \land\ maxVal \in [Acceptor \to Value \cup \{None\}] \\
& \land\ msgs \subseteq Message
\end{aligned}
$$

$maxBal$ is a function from acceptors to ballot numbers or -1.

$maxVBal$ is a function from acceptors to ballot numbers or -1.

$maxVal$ is a function from acceptors to values or $None$.

$msgs$ is a subset of the set of all possible messages.

$Init \triangleq$

$$Init \triangleq \land maxBal = [a \in Acceptor \mapsto -1]$$

For all a : $maxBal[a] = -1$

$$Init \triangleq \begin{array}{ll} \wedge\, maxBal &= [a \in Acceptor \mapsto -1] \\ \wedge\, maxVBal &= [a \in Acceptor \mapsto -1] \end{array}$$

For all a : $maxBal[a] = -1$

$maxVBal[a] = -1$

$$
\begin{aligned}
Init \triangleq\ & \wedge\ maxBal = [a \in Acceptor \mapsto -1] \\
& \wedge\ maxVBal = [a \in Acceptor \mapsto -1] \\
& \wedge\ maxVal = [a \in Acceptor \mapsto None]
\end{aligned}
$$

For all a : $maxBal[a] = -1$

$maxVBal[a] = -1$

$maxVal[a] = None$

$$
\begin{aligned}
Init \triangleq\ & \land\ maxBal = [a \in Acceptor \mapsto -1] \\
& \land\ maxVBal = [a \in Acceptor \mapsto -1] \\
& \land\ maxVal = [a \in Acceptor \mapsto None] \\
& \land\ msgs = \{\}
\end{aligned}
$$

For all a : $maxBal[a] = -1$

$maxVBal[a] = -1$

$maxVal[a] = None$

$msgs = \{\}$

The Actions

The Actions

$Phase1a(b)$

The Actions

$Phase1a(b)$

The ballot b leader sends a message asking acceptors to participate in that ballot.

The Actions

$Phase1a(b)$

$Phase1b(a)$

The Actions

$Phase1a(b)$

$Phase1b(a)$

If acceptor a has not agreed to participate in a ballot numbered $\geq b$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

The Actions

$Phase1a(b)$

$Phase1b(a)$

$Phase2a(b, v)$

The Actions

$Phase1a(b)$

$Phase1b(a)$

$Phase2a(b, v)$

If the ballot b leader receives those messages from a quorum, it chooses a value v safe at b and sends a message asking acceptors to vote for v in ballot b.

The Actions

$Phase1a(b)$

$Phase1b(a)$

$Phase2a(b, v)$

$Phase2b(a)$

The Actions

$Phase1a(b)$

$Phase1b(a)$

$Phase2a(b, v)$

$Phase2b(a)$

If acceptor a has not agreed to participate in a ballot numbered $> b$, then it votes for v in ballot b.

The Actions

$Phase1a(b)$

$Phase1b(a)$

$Phase2a(b, v)$

$Phase2b(a)$

If acceptor a has not agreed to participate in a ballot numbered $> b$, then it votes for v in ballot b.

It does this by sending a message.

The Actions

$Phase1a(b)$

$Phase1b(a)$

$Phase2a(b, v)$

$Phase2b(a)$

If acceptor a has not agreed to participate in a ballot numbered $> b$, then it votes for v in ballot b.

It does this by sending a message.

We don't care who receives that message.

$Send(m) \triangleq$

$Send(m) \triangleq$

Describes the sending of a message m .

$$Send(m) \triangleq msgs' = msgs \cup \{m\}$$

Describes the sending of a message m .

$Phase1a(b) \triangleq$

$Phase1a(b) \triangleq$

The ballot b leader sends a message asking acceptors to participate in that ballot.

$Phase1a(b) \triangleq$

$$Phase1a(b) \triangleq \ \land Send([type \mapsto \text{“1a”}, bal \mapsto b])$$

$$Phase1a(b) \triangleq \ \wedge Send([type \mapsto \text{“1a”}, bal \mapsto b])$$

Sends a type $1a$ message containing the ballot number.

$$
\begin{aligned}
Phase1a(b) \triangleq\ & \land\ Send([type \mapsto \text{“1a”},\ bal \mapsto b]) \\
& \land\ \text{UNCHANGED}\ \langle maxBal,\ maxVBal,\ maxVal \rangle
\end{aligned}
$$

$$Phase1a(b) \triangleq \begin{array}{l} \land\ Send([type \mapsto \text{“1a”},\ bal \mapsto b]) \\ \land\ \text{UNCHANGED}\ \langle maxBal,\ maxVBal,\ maxVal\rangle \end{array}$$

Leaves all variables except $msgs$ unchanged.

$Phase1b(a) \triangleq$

$Phase1b(a) \triangleq$

If acceptor a has not agreed to participate in a ballot numbered $\geq b$,

$Phase1b(a) \triangleq$

If acceptor a has not agreed to participate in a ballot numbered $\geq b$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$Phase1b(a) \triangleq$

Upon receipt of a $1a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $\geq b$, then it agrees and sends the leader
$maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$Phase1b(a) \triangleq$

Upon receipt of a $1a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $\geq m.bal$, then it agrees and sends the leader
$maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$Phase1b(a) \triangleq$

Upon receipt of a $1a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $\geq m.bal$, then it agrees and sends the leader
$maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$Phase1b(a) \triangleq$$
$$\land \exists\, m \in msgs :$$
$$\land\, m.type = \text{“1a”}$$

Upon receipt of a $1a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $\geq m.bal$, then it agrees and sends the leader
$maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a]
\end{array}
$$

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a]
\end{array}
$$

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$Phase1b(a) \triangleq$$
$$\land \exists\, m \in msgs :$$
$$\land\ m.type = \text{“1a”}$$
$$\land\ m.bal > \boxed{maxBal[a]}$$

The largest ballot number for which a has agreed to participate.

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a]
\end{array}
$$

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal]
\end{array}
$$

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal]
\end{array}
$$

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal]
\end{array}
$$

Setting $maxBal[a]$ to $m.bal$.

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal]
\end{array}
$$

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ Send([type \mapsto \text{“1b”},\ acc \mapsto a,\ bal \mapsto m.bal, \\
\qquad\qquad\qquad mbal \mapsto maxVBal[a],\ mval \mapsto maxVal[a]])
\end{array}
$$

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ Send([type \mapsto \text{“1b”},\ acc \mapsto a,\ bal \mapsto \boxed{m.bal}, \\
\qquad\qquad\qquad mbal \mapsto \boxed{maxVBal[a]},\ mval \mapsto \boxed{maxVal[a]}])
\end{array}
$$

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ Send([type \mapsto \text{“1b”},\ acc \mapsto a,\ bal \mapsto m.bal, \\
\qquad\qquad\qquad mbal \mapsto maxVBal[a],\ mval \mapsto maxVal[a]])
\end{array}
$$

Upon receipt of a $1a$ message m for ballot $m.bal$, if acceptor a has not agreed to participate in a ballot numbered $\geq m.bal$, then it agrees and sends the leader $maxBal[a]$, $maxVBal[a]$, and $maxVal[a]$.

the new value

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ Send([type \mapsto \text{“1b”},\ acc \mapsto a,\ bal \mapsto m.bal, \\
\qquad\qquad\qquad mbal \mapsto maxVBal[a],\ mval \mapsto maxVal[a]])
\end{array}
$$

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ Send([type \mapsto \text{“1b”},\ acc \mapsto a,\ bal \mapsto m.bal, \\
\qquad\qquad\qquad mbal \mapsto maxVBal[a],\ mval \mapsto maxVal[a]]) \\
\quad \land \text{UNCHANGED } \langle maxVBal,\ maxVal \rangle
\end{array}
$$

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ Send([type \mapsto \text{“1b”},\ acc \mapsto a,\ bal \mapsto m.bal, \\
\qquad\qquad\qquad mbal \mapsto maxVBal[a],\ mval \mapsto maxVal[a]]) \\
\quad \land \text{UNCHANGED } \langle maxVBal,\ maxVal \rangle
\end{array}
$$

And it leaves the other variables unchanged.

$$
\begin{array}{l}
Phase1b(a) \triangleq \\
\quad \land \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“1a”} \\
\qquad \land\ m.bal > maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ Send([type \mapsto \text{“1b”},\ acc \mapsto a,\ bal \mapsto m.bal, \\
\qquad\qquad\qquad mbal \mapsto maxVBal[a],\ mval \mapsto maxVal[a]]) \\
\quad \land \text{UNCHANGED } \langle maxVBal,\ maxVal \rangle
\end{array}
$$

$Phase2a(b,\ v) \triangleq$

$Phase2a(b,\ v) \triangleq$

If the ballot b leader receives those (1b) messages from a quorum, it chooses a value v safe at b and sends a message asking acceptors to vote for v in ballot b.

$Phase2a(b,\ v) \triangleq$

$Phase2a(b,\ v) \triangleq$

$$Phase2a(b,\ v) \triangleq$$
$$\land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b$$

$$Phase2a(b,\ v) \triangleq$$
$$\land \neg\exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b$$

No $2a$ message for ballot b has already been sent.

$$Phase2a(b,\ v) \triangleq$$
$$\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b$$
$$\quad \land \exists\, Q \in Quorum :$$

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \\
\\
\\
\\
\qquad \text{IN}
\end{array}
$$

$$Phase2a(b,\ v) \triangleq$$
$$\land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b$$
$$\land \exists\, Q \in Quorum :$$
$$\text{LET } Q1b \triangleq$$

Locally defines $Q1b$.

IN

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\\
\qquad \text{IN}
\end{array}
$$

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad \text{IN}
\end{array}
$$

The set of ballot b type $1b$ messages sent by acceptors in Q.

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal\ = b\} \\
\qquad\qquad\quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN}
\end{array}
$$

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad\qquad\quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN}
\end{array}
$$

The set of messages in $Q1b$ sent by acceptors who reported already voting in some ballot.

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal\ = b\} \\
\qquad\qquad\ Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a
\end{array}
$$

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad\qquad\ Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a
\end{array}
$$

There is a message in $Q1b$ from every acceptor in Q.

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal\ = b\} \\
\qquad\qquad\ Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\ \land\ \lor\, Q1bv = \{\}
\end{array}
$$

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal\ = b\} \\
\qquad\qquad\ Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\ \land\ \lor\, Q1bv = \{\}
\end{array}
$$

Either no acceptor in Q reported
that it had voted,

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal\ = b\} \\
\qquad\qquad\quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\quad \land\ \lor\ Q1bv = \{\} \\
\qquad\qquad\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal\ = b\} \\
\qquad\qquad\ Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\ \land\ \lor\ Q1bv = \{\} \\
\qquad\qquad\qquad\ \lor \exists\, m \in Q1bv : \\
\qquad\qquad\qquad\qquad \land\, m.mval = v \\
\qquad\qquad\qquad\qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

or the message in $Q1bv$ reporting a vote in the highest-numbered ballot reports a vote for v .

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal\ = b\} \\
\qquad\qquad\quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\quad \land\ \lor\ Q1bv = \{\} \\
\qquad\qquad\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land m.bal = b\} \\
\qquad\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\quad \land \lor Q1bv = \{\} \\
\qquad\qquad\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\qquad\qquad\quad \land m.mval = v \\
\qquad\qquad\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

This condition implies that v is safe at b

$$
\begin{array}{l}
Phase2a(b,\ v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal\ = b\} \\
\qquad\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad \land\ \lor\ Q1bv = \{\} \\
\qquad\qquad\qquad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\qquad\qquad \land\, m.mval = v \\
\qquad\qquad\qquad\qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

This condition implies that v is safe at b
because this condition implies $ShowsSafeAt(Q, b, v)$.

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \wedge\ \vee\, Q1bv = \{\} \\
& \quad \vee \exists\, m \in Q1bv : \\
& \qquad \wedge\, m.mval = v \\
& \qquad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& \quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \quad \wedge\ \vee\, Q1bv = \{\} \\
& \qquad \vee \exists\, m \in Q1bv : \\
& \qquad\quad \wedge\, m.mval = v \\
& \qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

This implies $ShowsSafeAt(Q, b, v)$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \land \lor\, Q1bv = \{\} \\
& \quad \lor \exists\, m \in Q1bv : \\
& \qquad \land\, m.mval = v \\
& \qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

This implies $ShowsSafeAt(Q, b, v)$

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad \land \lor\, Q1bv = \{\} \\
\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

This implies $ShowsSafeAt(Q, b, v)$ where these are defined in terms of messages that have been sent.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land (c \neq -1) \Rightarrow \exists\, a \in Q : \boxed{VotedFor(a,\, c,\, v)} \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : \boxed{DidNotVoteAt(a,\, d)}
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& \quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \quad \wedge \ \vee\, Q1bv = \{\} \\
& \qquad \vee \exists\, m \in Q1bv : \\
& \qquad\quad \wedge\, m.mval = v \\
& \qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge \boxed{\forall\, a \in Q : maxBal[a] \geq b} \\
\quad \wedge \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \land \boxed{\forall\, a \in Q : \exists\, m \in Q1b : m.acc = a} \\
\qquad \land\ \lor\ Q1bv = \{\} \\
\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

For every a in Q, $maxBal[a] = b$ was true right after it sent its message.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \boxed{\forall\, a \in Q : maxBal[a] \geq b} \\
\quad \land \exists\, c \in -1\,..\,(b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1)\,..\,(b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \wedge \boxed{\forall\, a \in Q : \exists\, m \in Q1b : m.acc = a} \\
\qquad \wedge\ \vee\, Q1bv = \{\} \\
\qquad\quad \vee \exists\, m \in Q1bv : \\
\qquad\qquad\quad \wedge\, m.mval = v \\
\qquad\qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

And $maxBal[a]$ never decreases.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge \boxed{\forall\, a \in Q : maxBal[a] \geq b} \\
\quad \wedge \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \wedge \ \vee\, Q1bv = \{\} \\
& \quad \vee \exists\, m \in Q1bv : \\
& \qquad \wedge\, m.mval = v \\
& \qquad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \wedge \ \vee\, Q1bv = \{\} \\
& \quad \vee \exists\, m \in Q1bv : \\
& \qquad \wedge\, m.mval = v \\
& \qquad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& \quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \quad \wedge \ \vee\, Q1bv = \{\} \\
& \qquad \vee \exists\, m \in Q1bv : \\
& \qquad\quad \wedge\, m.mval = v \\
& \qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \wedge \ \vee\, Q1bv = \{\} \\
& \quad \vee \exists\, m \in Q1bv : \\
& \qquad \wedge\, m.mval = v \\
& \qquad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

When an acceptor $a \in Q$ sent its $1b$ message mm for ballot b,

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \wedge \ \vee\, Q1bv = \{\} \\
& \quad \vee \exists\, m \in Q1bv : \\
& \qquad \wedge\, m.mval = v \\
& \qquad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

When an acceptor $a \in Q$ sent its $1b$ message mm for ballot b,
it hadn't voted in any ballot d with $mm.mbal < d < b$.

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& \quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \quad \wedge \ \vee\, Q1bv = \{\} \\
& \qquad \vee \exists\, m \in Q1bv : \\
& \qquad\quad \wedge\, m.mval = v \\
& \qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

When an acceptor $a \in Q$ sent its $1b$ message mm for ballot b, it hadn't voted in any ballot d with $mm.mbal < d < b$.

And because it set $mBal[a]$ to b, it still hasn't voted in those ballots.

When an acceptor $a \in Q$ sent its $1b$ message mm for ballot b, it hadn't voted in any ballot d with $mm.mbal < d < b$.

And because it set $mBal[a]$ to b, it still hasn't voted in those ballots.

IN
$$
\begin{array}{l}
\land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\land \lor Q1bv = \{\} \\
\quad \lor \exists\, m \in Q1bv : \\
\qquad \land\, m.mval = v \\
\qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& \quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \quad \wedge \ \vee\, Q1bv = \{\} \\
& \qquad \vee \exists\, m \in Q1bv : \\
& \qquad\quad \wedge\, m.mval = v \\
& \qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& \quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \quad \wedge \ \vee\, Q1bv = \{\} \\
& \qquad \vee \exists\, m \in Q1bv : \\
& \qquad\quad \wedge\, m.mval = v \\
& \qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad \wedge \ \vee\, Q1bv = \{\} \\
\qquad\quad \vee \exists\, m \in Q1bv : \\
\qquad\qquad\quad \wedge\, m.mval = v \\
\qquad\qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \wedge\, \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad \wedge\ \vee\, \boxed{Q1bv = \{\}} \\
\qquad\quad \vee\, \exists\, m \in Q1bv : \\
\qquad\qquad\quad \wedge\, m.mval = v \\
\qquad\qquad\quad \wedge\, \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Case 1: No acceptor in Q has voted.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge\, \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge\, \boxed{\exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge\, \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)}
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad \wedge\ \vee\, \boxed{Q1bv = \{\}} \\
\qquad\quad \vee \exists\, m \in Q1bv : \\
\qquad\qquad\quad \wedge\, m.mval = v \\
\qquad\qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Let c equal -1.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge \boxed{\exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)}
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad \land \lor \boxed{Q1bv = \{\}} \\
\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Let c equal -1. This is vacuously true.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land \boxed{(c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v)} \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \wedge\, \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad \wedge\ \vee\, \boxed{Q1bv = \{\}} \\
\qquad\quad \vee\, \exists\, m \in Q1bv : \\
\qquad\qquad\quad \wedge\, m.mval = v \\
\qquad\qquad\quad \wedge\, \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge\, \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge\, \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge\, \boxed{\forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)}
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad \land \lor \boxed{Q1bv = \{\}} \\
\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

No acceptor a in Q had voted when it sent its message.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \boxed{\forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)}
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& \quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \quad \wedge \ \vee \boxed{Q1bv = \{\}} \\
& \qquad \vee \exists\, m \in Q1bv : \\
& \qquad\quad \wedge\, m.mval = v \\
& \qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

No acceptor a in Q had voted when it sent its message.

And $mBal[a]$ then equaled b, so it couldn't later have voted in any ballot $< b$.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge \boxed{\forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)}
\end{array}
$$

When an acceptor $a \in Q$ sent its $1b$ message mm for ballot b, it hadn't voted in any ballot d with $mm.mbal < d < b$.

And because it set $mBal[a]$ to b, it still hasn't voted in those ballots.

$$
\begin{array}{ll}
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \land \lor \boxed{Q1bv = \{\}} \\
& \quad \lor \exists\, m \in Q1bv : \\
& \qquad \land\, m.mval = v \\
& \qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

No acceptor a in Q had voted when it sent its message.

And $mBal[a]$ then equaled b, so it couldn't later have voted in any ballot $< b$.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \boxed{\forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)}
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad \land \lor\, Q1bv = \{\} \\
\qquad\quad \lor \boxed{\exists\, m \in Q1bv : \\
\qquad\qquad \land\, m.mval = v \\
\qquad\qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal}
\end{array}
$$

Case 2:

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \boxed{\exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)}
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
 & \qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
 & \qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
 & Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
 & \land \lor\, Q1bv = \{\} \\
 & \quad \lor \exists\, m \in Q1bv : \\
 & \qquad \land\, m.mval = v \\
 & \qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Case 2: Choose such a message m .

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad \land \lor\, Q1bv = \{\} \\
\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Let $c = m.mbal$.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \land \lor\, Q1bv = \{\} \\
& \quad \lor \exists\, m \in Q1bv : \\
& \qquad \land\, m.mval = v \\
& \qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Let $c = m.mbal$.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \boxed{\exists\, a \in Q : VotedFor(a,\, c,\, v)} \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{l}
\text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad \land \lor\, Q1bv = \{\} \\
\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\quad \land \boxed{m.mval = v} \\
\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Let $c = m.mbal$.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land (c \neq -1) \Rightarrow \boxed{\exists\, a \in Q : VotedFor(a,\, c,\, v)} \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \wedge \ \vee\, Q1bv = \{\} \\
& \quad \vee \exists\, m \in Q1bv : \\
& \qquad \wedge \boxed{m.mval = v} \\
& \qquad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Let $c = m.mbal$. $\quad VotedFor(m.acc, c, v)$

$$
\begin{array}{l}
ShowsSafeAt(Q,\ b,\ v) \triangleq \\
\quad \wedge \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge \exists\, c \in -1\,..\,(b-1) : \\
\qquad \wedge (c \neq -1) \Rightarrow \boxed{\exists\, a \in Q : VotedFor(a,\ c,\ v)} \\
\qquad \wedge \forall\, d \in (c+1)\,..\,(b-1),\ a \in Q : DidNotVoteAt(a,\ d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& \quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \quad \wedge \ \vee\, Q1bv = \{\} \\
& \qquad \vee \exists\, m \in Q1bv : \\
& \qquad\quad \wedge\, m.mval = v \\
& \qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Let $c = m.mbal$.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \land \lor\, Q1bv = \{\} \\
& \quad \lor \exists\, m \in Q1bv : \\
& \qquad \land\, m.mval = v \\
& \qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Let $c = m.mbal$.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \boxed{\forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)}
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \land \lor\, Q1bv = \{\} \\
& \quad \lor \exists\, m \in Q1bv : \\
& \qquad \land\, m.mval = v \\
& \qquad \land \boxed{\forall\, mm \in Q1bv : m.mbal \geq mm.mbal}
\end{array}
$$

Let $c = m.mbal$.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \land \lor\, Q1bv = \{\} \\
& \quad \lor \exists\, m \in Q1bv : \\
& \qquad \land\, m.mval = v \\
& \qquad \land \boxed{\forall\, mm \in Q1bv : m.mbal \geq mm.mbal}
\end{array}
$$

Let $c = m.mbal$.

c is the highest numbered ballot in which any $a \in Q$ voted.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

When an acceptor $a \in Q$ sent its $1b$ message mm for ballot b, it hadn't voted in any ballot d with $mm.mbal < d < b$.

And because it set $mBal[a]$ to b, it still hasn't voted in those ballots.

$$
\begin{array}{ll}
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \land \lor Q1bv = \{\} \\
& \quad \lor \exists\, m \in Q1bv : \\
& \qquad \land\, m.mval = v \\
& \qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Let $c = m.mbal$.

c is the highest numbered ballot in which any $a \in Q$ voted.

No $a \in Q$ voted in any ballot d with $c < d < b$.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \land \lor\, Q1bv = \{\} \\
& \quad \lor \exists\, m \in Q1bv : \\
& \qquad \land\, m.mval = v \\
& \qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

Let $c = m.mbal$.

c is the highest numbered ballot in which any $a \in Q$ voted.

No $a \in Q$ voted in any ballot d with $c < d < b$.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \boxed{\forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)}
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \land \lor\, Q1bv = \{\} \\
& \quad \lor \exists\, m \in Q1bv : \\
& \qquad \land\, m.mval = v \\
& \qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

This is an explanation, not a proof.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
 & \qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
 & \qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
 & \quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
 & \quad \land \lor\, Q1bv = \{\} \\
 & \qquad \lor \exists\, m \in Q1bv : \\
 & \qquad\quad \land\, m.mval = v \\
 & \qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

This is an explanation, not a proof.

There are more rigorous proofs

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
& Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \land \lor\, Q1bv = \{\} \\
& \quad \lor \exists\, m \in Q1bv : \\
& \qquad \land\, m.mval = v \\
& \qquad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

This is an explanation, not a proof.

There are more rigorous proofs of incorrect algorithms.

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \land \forall\, a \in Q : maxBal[a] \geq b \\
\quad \land \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \land\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \land \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{ll}
\text{LET} & Q1b \triangleq \{m \in msgs : \ \wedge\, m.type = \text{“1b”} \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.acc \in Q \\
& \qquad\qquad\qquad\qquad\quad \wedge\, m.bal = b\} \\
& \quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\text{IN} & \quad \wedge \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
& \quad \wedge \ \vee\, Q1bv = \{\} \\
& \qquad \vee \exists\, m \in Q1bv : \\
& \qquad\quad \wedge\, m.mval = v \\
& \qquad\quad \wedge \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{l}
ShowsSafeAt(Q,\, b,\, v) \triangleq \\
\quad \wedge \forall\, a \in Q : maxBal[a] \geq b \\
\quad \wedge \exists\, c \in -1 \,..\, (b-1) : \\
\qquad \wedge\, (c \neq -1) \Rightarrow \exists\, a \in Q : VotedFor(a,\, c,\, v) \\
\qquad \wedge \forall\, d \in (c+1) \,..\, (b-1),\, a \in Q : DidNotVoteAt(a,\, d)
\end{array}
$$

$$
\begin{array}{l}
Phase2a(b,\, v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad\qquad\quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\quad \land \lor Q1bv = \{\} \\
\qquad\qquad\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal
\end{array}
$$

$$
\begin{array}{l}
Phase2a(b,\, v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad\qquad\quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\quad \land \lor Q1bv = \{\} \\
\qquad\qquad\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal \\
\quad \land Send([type \mapsto \text{“2a”},\, bal \mapsto b,\, val \mapsto v])
\end{array}
$$

$$
\begin{array}{l}
Phase2a(b,\, v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad\qquad\quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\quad \land \lor Q1bv = \{\} \\
\qquad\qquad\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal \\
\quad \land Send([type \mapsto \text{“2a”},\, bal \mapsto b,\, val \mapsto v])
\end{array}
$$

Sends the $2a$ message to (all) the acceptors.

$$
\begin{array}{l}
Phase2a(b,\, v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad\qquad\quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\quad \land \lor Q1bv = \{\} \\
\qquad\qquad\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal \\
\quad \land Send([type \mapsto \text{“2a”},\, bal \mapsto b,\, val \mapsto v]) \\
\quad \land \text{UNCHANGED } \langle maxBal,\, maxVBal,\, maxVal \rangle
\end{array}
$$

$$
\begin{array}{l}
Phase2a(b,\, v) \triangleq \\
\quad \land \neg \exists\, m \in msgs : m.type = \text{“2a”} \land m.bal = b \\
\quad \land \exists\, Q \in Quorum : \\
\qquad \text{LET } Q1b \triangleq \{m \in msgs : \land\, m.type = \text{“1b”} \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.acc \in Q \\
\qquad\qquad\qquad\qquad\qquad\quad \land\, m.bal = b\} \\
\qquad\qquad\quad Q1bv \triangleq \{m \in Q1b : m.mbal \geq 0\} \\
\qquad \text{IN} \quad \land \forall\, a \in Q : \exists\, m \in Q1b : m.acc = a \\
\qquad\qquad\quad \land \lor Q1bv = \{\} \\
\qquad\qquad\qquad\quad \lor \exists\, m \in Q1bv : \\
\qquad\qquad\qquad\qquad\quad \land\, m.mval = v \\
\qquad\qquad\qquad\qquad\quad \land \forall\, mm \in Q1bv : m.mbal \geq mm.mbal \\
\quad \land Send([type \mapsto \text{“2a”},\, bal \mapsto b,\, val \mapsto v]) \\
\quad \land \text{UNCHANGED } \langle maxBal,\, maxVBal,\, maxVal \rangle
\end{array}
$$

Leaves all variables except $msgs$ unchanged.

$Phase2b(a) \triangleq$

$Phase2b(a) \triangleq$

If acceptor a has not agreed to participate in a ballot numbered $> b$, then it votes for v in ballot b.

$Phase2b(a) \triangleq$

If acceptor a has not agreed to participate in a ballot numbered $> b$, then it votes for v in ballot b.

It does this by sending a message.

$Phase2b(a) \triangleq$

Upon receipt of a $2a$ message m for ballot $m.bal$,
If acceptor a has not agreed to participate in a ballot numbered $> b$, then it votes for v in ballot b.

It does this by sending a message.

$Phase2b(a) \triangleq$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$Phase2b(a) \triangleq$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$Phase2b(a) \triangleq$$
$$\exists\, m \in msgs :$$
$$\wedge\, m.type = \text{“2a”}$$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$Phase2b(a) \triangleq$$
$$\exists\, m \in msgs :$$
$$\quad \wedge\, m.type = \text{“2a”}$$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$Phase2b(a) \triangleq$$
$$\exists\, m \in msgs :$$
$$\quad \land\, m.type = \text{“2a”}$$
$$\quad \land\, m.bal \geq maxBal[a]$$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$Phase2b(a) \triangleq$$
$$\exists\, m \in msgs :$$
$$\quad \land\, m.type = \text{“2a”}$$
$$\quad \land\, m.bal \geq maxBal[a]$$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$Phase2b(a) \triangleq$$
$$\exists\, m \in msgs :$$
$$\quad \land\, m.type = \text{“2a”}$$
$$\quad \land\, m.bal \geq maxBal[a]$$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$
\begin{array}{l}
Phase2b(a) \triangleq \\
\quad \exists\, m \in msgs : \\
\qquad \wedge\, m.type = \text{“2a”} \\
\qquad \wedge\, m.bal \geq maxBal[a] \\
\qquad \wedge\, maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal]
\end{array}
$$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$Phase2b(a) \triangleq$$
$$\exists\, m \in msgs :$$
$$\wedge\, m.type = \text{“2a”}$$
$$\wedge\, m.bal \geq maxBal[a]$$
$$\wedge\, maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal]$$

Set $maxBal[a]$ to $m.bal$.

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$
\begin{array}{l}
Phase2b(a) \triangleq \\
\quad \exists\, m \in msgs : \\
\qquad \land\, m.type = \text{“2a”} \\
\qquad \land\, m.bal \geq maxBal[a] \\
\qquad \land\, maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\, maxVBal' = [maxVBal \text{ EXCEPT } ![a] = m.bal]
\end{array}
$$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$Phase2b(a) \triangleq$$
$$\exists\, m \in msgs :$$
$$\land\, m.type = \text{“2a”}$$
$$\land\, m.bal \geq maxBal[a]$$
$$\land\, maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal]$$
$$\land\, maxVBal' = [maxVBal \text{ EXCEPT } ![a] = m.bal]$$

Set $maxVBal[a]$ to $m.bal$.

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$
\begin{array}{l}
Phase2b(a) \triangleq \\
\quad \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“2a”} \\
\qquad \land\ m.bal \geq maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ maxVBal' = [maxVBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ maxVal' = [maxVal \text{ EXCEPT } ![a] = m.val]
\end{array}
$$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$
\begin{array}{l}
Phase2b(a) \triangleq \\
\quad \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“2a”} \\
\qquad \land\ m.bal \geq maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ maxVBal' = [maxVBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ maxVal' = [maxVal \text{ EXCEPT } ![a] = m.val]
\end{array}
$$

Set $maxVal[a]$ to $m.val$.

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$
\begin{array}{l}
Phase2b(a) \triangleq \\
\quad \exists\, m \in msgs : \\
\qquad \wedge\, m.type = \text{“2a”} \\
\qquad \wedge\, m.bal \geq maxBal[a] \\
\qquad \wedge\, maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \wedge\, maxVBal' = [maxVBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \wedge\, maxVal' = [maxVal \text{ EXCEPT } ![a] = m.val]
\end{array}
$$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$
\begin{array}{l}
Phase2b(a) \triangleq \\
\quad \exists\, m \in msgs : \\
\qquad \land\ m.type = \text{“2a”} \\
\qquad \land\ m.bal \geq maxBal[a] \\
\qquad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ maxVBal' = [maxVBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\ maxVal' = [maxVal \text{ EXCEPT } ![a] = m.val] \\
\qquad \land\ Send([type \mapsto \text{“2b”},\ acc \mapsto a, \\
\qquad\qquad\qquad bal \mapsto m.bal,\ val \mapsto m.val])
\end{array}
$$

Upon receipt of a $2a$ message m for ballot $m.bal$,
if acceptor a has not agreed to participate in a ballot
numbered $> m.bal$, then it votes for $m.val$ in ballot $m.bal$.

It does this by sending a message.

$$
\begin{array}{l}
Phase2b(a) \triangleq \\
\quad \exists\, m \in msgs : \\
\qquad \land\, m.type = \text{“2a”} \\
\qquad \land\, m.bal \geq maxBal[a] \\
\qquad \land\, maxBal' = [maxBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\, maxVBal' = [maxVBal \text{ EXCEPT } ![a] = m.bal] \\
\qquad \land\, maxVal' = [maxVal \text{ EXCEPT } ![a] = m.val] \\
\qquad \land\, Send([type \mapsto \text{“2b”},\ acc \mapsto a, \\
\qquad\qquad\qquad bal \mapsto m.bal,\ val \mapsto m.val])
\end{array}
$$

$$
\begin{array}{lll}
Next \triangleq & \lor \exists\, b \in Ballot : & \lor\ Phase1a(b) \\
 & & \lor \exists\, v \in Value : Phase2a(b,\ v) \\
 & \lor \exists\, a \in Acceptor : Phase1b(a) \lor Phase2b(a) &
\end{array}
$$

$$
\begin{aligned}
Next \triangleq\ & \lor \exists\, b \in Ballot : \lor\ Phase1a(b) \\
& \qquad\qquad\qquad\qquad \lor \exists\, v \in Value : Phase2a(b,\, v) \\
& \lor \exists\, a \in Acceptor : Phase1b(a) \lor Phase2b(a)
\end{aligned}
$$

$$
Spec \triangleq\ Init \land \Box[Next]_{vars}
$$

The Paxos Consensus Algorithm Implements The Voting Algorithm

The parameters of the $Voting$ spec are:

CONSTANTS $Value$ $Acceptor$ $Quorum$

The parameters of the $Voting$ spec are:

CONSTANTS $Value$ $Acceptor$ $Quorum$

VARIABLES $maxBal$ $votes$

The parameters of the $Voting$ spec are:

CONSTANTS $Value$ $Acceptor$ $Quorum$

VARIABLES $maxBal$ $votes$

They are implemented in the $Paxos$ spec by:

The parameters of the $Voting$ spec are:

CONSTANTS $Value$ $Acceptor$ $Quorum$

VARIABLES $maxBal$ $votes$

They are implemented in the $Paxos$ spec by:

CONSTANTS $Value$ $Acceptor$ $Quorum$

The parameters of the $Voting$ spec are:

CONSTANTS $Value$ $Acceptor$ $Quorum$

VARIABLES $maxBal$ $votes$

They are implemented in the $Paxos$ spec by:

CONSTANTS $Value$ $Acceptor$ $Quorum$

VARIABLE $maxBal$

The parameters of the $Voting$ spec are:

CONSTANTS $Value$ $Acceptor$ $Quorum$

VARIABLES $maxBal$ $votes$

They are implemented in the $Paxos$ spec by:

CONSTANTS $Value$ $Acceptor$ $Quorum$

VARIABLE $maxBal$ a defined expression $votes$

The parameters of the $Voting$ spec are:

CONSTANTS $Value$ $Acceptor$ $Quorum$

VARIABLES $maxBal$ $votes$

They are implemented in the $Paxos$ spec by:

CONSTANTS $Value$ $Acceptor$ $Quorum$

VARIABLE $maxBal$ a defined expression $votes$

a defined expression $votes$

$votes \triangleq$

$$votes \triangleq$$
$$[a \in Acceptor \mapsto$$

$$]$$

$$votes \triangleq$$
$$[a \in Acceptor \mapsto$$

$$]$$

A function with domain the set of acceptors.

$votes \triangleq$

$[a \in Acceptor \mapsto$ The set of votes cast by acceptor a .

$]$

A function with domain the set of acceptors.

$$votes \triangleq$$
$$[a \in Acceptor \mapsto$$
$$\{\langle m.bal,\ m.val\rangle : m \in$$
$$\}]$$

$$votes \triangleq$$
$$[a \in Acceptor \mapsto$$
$$\{\langle m.bal,\ m.val\rangle : m \in$$

The set of $2b$ messages sent by a.

$$\}]$$

$$votes \triangleq$$
$$[a \in Acceptor \mapsto$$
$$\{\langle m.bal,\ m.val\rangle : m \in \{mm \in msgs : \land\ mm.type = \text{“2b”}$$
$$\land\ mm.acc = a\}\}]$$

The set of $2b$ messages sent by a.

$$votes \triangleq$$
$$[a \in Acceptor \mapsto$$
$$\{\langle m.bal,\ m.val\rangle : m \in \{mm \in msgs : \land\ mm.type = \text{“2b”}$$
$$\land\ mm.acc = a\}\}]$$

$V \triangleq$ INSTANCE $Voting$

$V \triangleq$ INSTANCE $Voting$

WITH $Value \leftarrow Value,\ Acceptor \leftarrow Acceptor,\ Quorum \leftarrow Quorum,$
$maxBal \leftarrow maxBal,\ votes \leftarrow votes$

Remember that these substitutions are implied.

$V \triangleq$ INSTANCE $Voting$

$V \triangleq$ INSTANCE $Voting$

THEOREM $Spec \Rightarrow V!Spec$

$V \triangleq$ INSTANCE $Voting$

THEOREM $Spec \Rightarrow V!Spec$

The model checker can check this theorem.

I won’t discuss how the Paxos consensus algorithm is used to build fault-tolerant systems.

I won’t discuss how the Paxos consensus algorithm is used to build fault-tolerant systems.

See *Paxos Made Simple*.

What if you don’t win a Turing award?

Will learning to think mathematically be a waste of time?

Will learning to think mathematically be a waste of time?

No.

Will learning to think mathematically be a waste of time?

Will it improve your coding?

Will learning to think mathematically be a waste of time?

Will it improve your coding?

Probably not.

Will learning to think mathematically be a waste of time?

Will it improve your coding?

So why bother?

Will learning to think mathematically be a waste of time?

Will it improve your coding?

So why bother?

Coding should be the easiest, least important part of programming.

Will learning to think mathematically be a waste of time?

Will it improve your coding?

So why bother?

Coding should be the easiest, least important part of programming.

If you’re having trouble writing a piece of code, you’re doing something wrong.

Rosetta

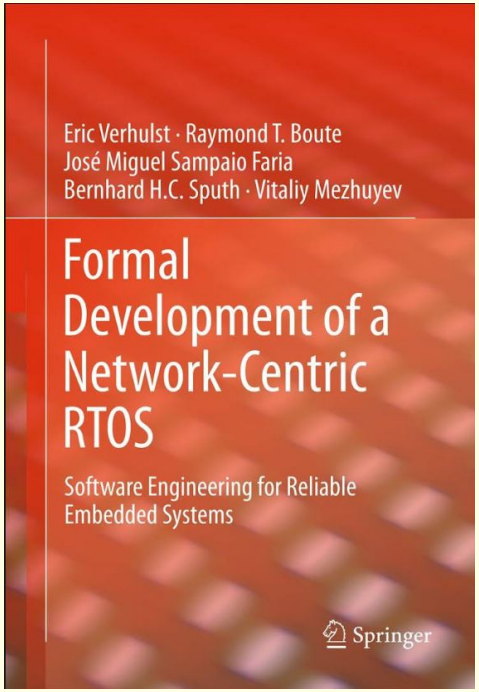
Rosetta

ESA Spacecraft that explored a comet.

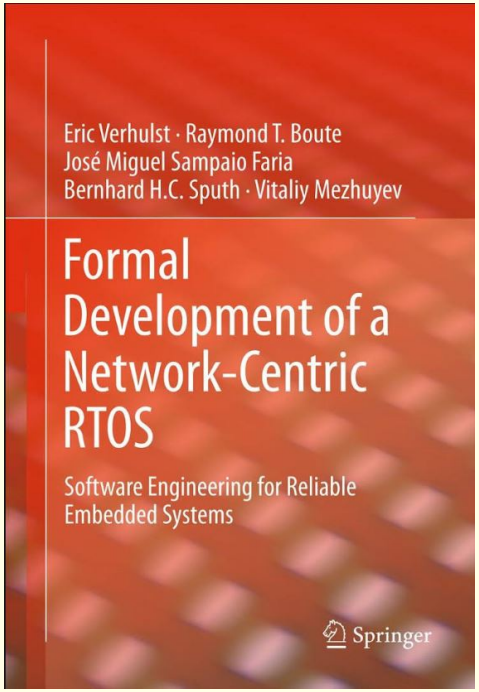
Rosetta

ESA Spacecraft that explored a comet.

Several of its instruments were controlled by the Virtuoso real-time operating system.

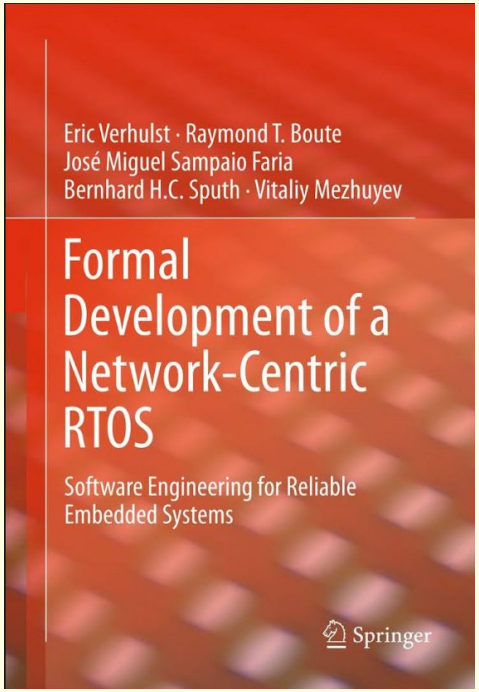


The next version of Virtuoso.



The next version of Virtuoso.

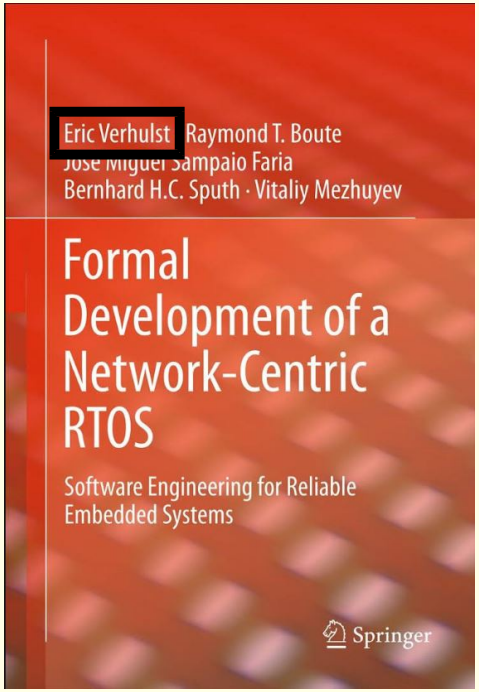
Its high-level design is described in TLA$^+$.



The next version of Virtuoso.

Its high-level design is described in TLA$^+$.

Here's an email from Eric Verhulst,
the head of the development team.

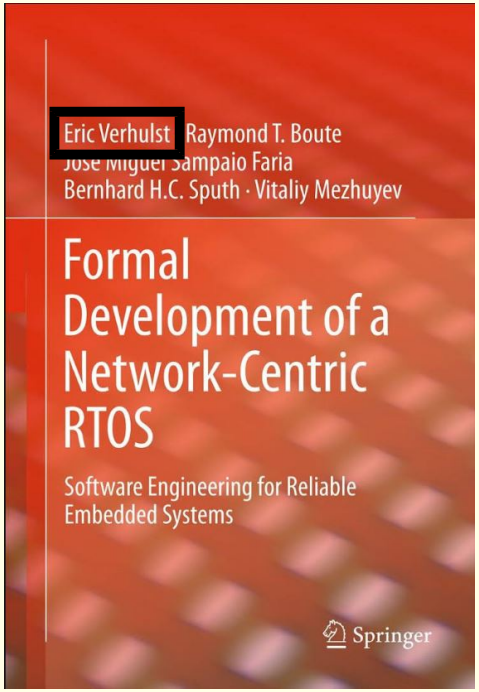


The next version of Virtuoso.

Its high-level design is described in TLA$^+$.

Here's an email from Eric Verhulst,
the head of the development team.

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture

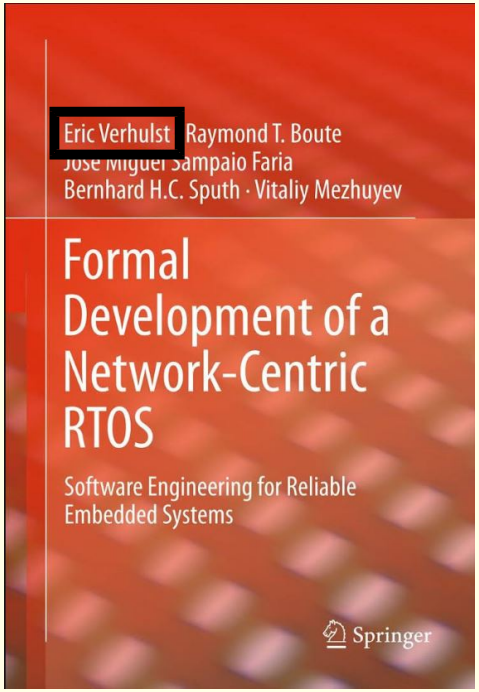


The next version of Virtuoso.

Its high-level design is described in TLA$^+$.

Here's an email from Eric Verhulst,
the head of the development team.

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming).



The next version of Virtuoso.

Its high-level design is described in TLA$^+$.

Here's an email from Eric Verhulst, the head of the development team.

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10$\times$ less than in [Virtuoso].

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that

the code size is about 10$\times$ less than in [Virtuoso].

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10$\times$ less than in [Virtuoso].

You don't produce 10$\times$ less code by better coding.

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10$\times$ less than in [Virtuoso].

You don't produce 10$\times$ less code by better coding.

You do it with a cleaner architecture

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10$\times$ less than in [Virtuoso].

You don't produce 10$\times$ less code by better coding.

better algorithm

You do it with a ~~cleaner architecture~~

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10$\times$ less than in [Virtuoso].

You don't produce 10$\times$ less code by better coding.

better algorithm

You do it with a ~~cleaner architecture~~, which comes from mathematical thinking.

The [TLA$^+$] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first-hand the brainwashing done by years of C programming). One of the results was that the code size is about 10$\times$ less than in [Virtuoso].

You don't produce 10$\times$ less code by better coding.

better algorithm

You do it with a ~~cleaner architecture~~, which comes from mathematical thinking.

It doesn't come from thinking in a programming language.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- **Formal methods find bugs in system designs that cannot be found through any other technique we know of.**
- **Formal methods are surprisingly feasible for mainstream software development and give good return on investment.**
- **At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.**

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other

They build Amazon's cloud infrastructure.

to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- **Formal methods find bugs in system designs that cannot be found through any other technique we know of.**
- **Formal methods are surprisingly feasible for mainstream software development and give good return on investment.**
- **At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.**

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary com-

The formal method they use is TLA+.

contracts on which our customers depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- **Formal methods find bugs in system designs that cannot be found through any other technique we know of.**
- **Formal methods are surprisingly feasible for mainstream software development and give good return on investment.**
- **At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.**

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just
vices that stor
customers hav
safeguard that
service relies
tributed algor
consistency, c
to-scaling, loa
coordination
such algorithm
combining the
tem is a challe
must usually
properly in a
addition, we h
to invent algo
work hard to a
plexity, but the
the task remai

Complexity
ity of human
and operation
the system cou
tion of data, o
contracts on w
pend. So, befo
we need to rea
fidence that th
correct. We ha
verification tec
necessary but
tinely use dee
reviews, static
testing, and fa
still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just
vices that stor
customers ha
safeguard that
service relies
tributed algo
consistency, c
to-scaling, loa
coordination
such algorithm
combining the
tem is a challe
must usually
properly in a
addition, we h
to invent algo
work hard to a
plexity, but the
the task remai

Complexity
ity of human
and operation
the system cou
tion of data, o
contracts on w
pend. So, befo
we need to rea
fidence that th
correct. We ha
verification tec
necessary but
tinely use dee
reviews, static
testing, and fa
still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

» key insights

- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one trillion objects.[3] Less than a year later it had grown to two trillion objects and was regularly handling 1.1 million requests per second.[4]

S3 is just
vices that stor
customers hav
safeguard that
service relies
tributed algor
consistency, c
to-scaling, loa
coordination
such algorithm
combining the
tem is a challe
must usually
properly in a
addition, we h
to invent algo
work hard to a
plexity, but the
the task remai

Complexity
ity of human
and operation
the system cou
tion of data, o
contracts on w
pend. So, befo
we need to rea
fidence that th
correct. We ha
verification tec
necessary but
tinely use dee
reviews, static
testing, and fa
still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- Formal methods find bugs in system designs that cannot be found through any other technique we know of.
- Formal methods are surprisingly feasible for mainstream software development and give good return on investment.
- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

» key insights

- At Amazon, formal methods are routinely applied to the design of complex real-world software, including public cloud services.

DOI:10.1145/2699417

Engineers use TLA+ to prevent serious but subtle bugs from reaching production.

BY CHRIS NEWCOMBE, TIM RATH, FAN ZHANG, BOGDAN MUNTEANU, MARC BROOKER, AND MICHAEL DEARDEUFF

How Amazon Web Services Uses Formal Methods

SINCE 2011, ENGINEERS at Amazon Web Services (AWS) have used formal specification and model checking to help solve difficult design problems in critical systems. Here, we describe our motivation and experience, what has worked well in our problem domain, and what has not. When discussing personal experience we refer to the authors by their initials.

At AWS we strive to build services that are simple for customers to use. External simplicity is built on a hidden substrate of complex distributed systems. Such complex internals are required to achieve high availability while running on cost-efficient infrastructure and cope with relentless business growth. As an example of this growth, in 2006, AWS launched S3, its Simple Storage Service. In the following six years, S3 grew to store one

S3 is just one of many AWS services that store and process data our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks. There are many such algorithms in the literature, but combining them into a cohesive system is a challenge, as the algorithms must usually be modified to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.

Complexity increases the probability of human error in design, code, and operations. Errors in the core of the system could cause loss or corruption of data, or violate other interface contracts on which our customers depend. So, before launching a service, we need to reach extremely high confidence that the core of the system is correct. We have found the standard verification techniques in industry are necessary but not sufficient. We routinely use deep design reviews, code reviews, static code analysis, stress testing, and fault-injection testing but still find that subtle bugs can hide in complex concurrent fault-tolerant systems. One reason they do is that human intuition is poor at estimating the true probability of supposedly "extremely rare" combinations of events in systems operating at a scale of millions of requests per second.

» key insights

- **Formal methods find bugs in system designs that cannot be found through any other technique we know of.**
- **Formal methods are surprisingly feasible for mainstream software development and give good return on investment**

66 COMMUNICATIONS OF THE ACM | APRIL 2015 | VOL. 58 | NO. 4

What Chris Newcombe, senior engineer at Oracle, wrote.

What Chris Newcombe, senior engineer at Oracle, wrote.

TLA+ is the most valuable thing that I've learned in my professional career.

What Chris Newcombe, senior engineer at Oracle, wrote.

TLA+ is the most valuable thing that I've learned in my professional career.

It has changed how I work

What Chris Newcombe, senior engineer at Oracle, wrote.

TLA+ is the most valuable thing that I've learned in my professional career.

It has changed how I work
by giving me an immensely powerful tool
to find subtle flaws in system designs.

What Chris Newcombe, senior engineer at Oracle, wrote.

TLA+ is the most valuable thing that I've learned in my professional career.

It has changed how I work

It has changed how I think

What Chris Newcombe, senior engineer at Oracle, wrote.

TLA+ is the most valuable thing that I've learned in my professional career.

It has changed how I work

It has changed how I think
by giving me a framework for constructing new kinds of mental-models

What Chris Newcombe, senior engineer at Oracle, wrote.

TLA+ is the most valuable thing that I've learned in my professional career.

It has changed how I work

It has changed how I think

- by giving me a framework for constructing new kinds of mental-models
- by revealing the precise relationship between correctness properties and system designs

What Chris Newcombe, senior engineer at Oracle, wrote.

TLA+ is the most valuable thing that I've learned in my professional career.

It has changed how I work

It has changed how I think

by giving me a framework for constructing new kinds of mental-models by revealing the precise relationship between correctness properties and system designs and by allowing me to move from 'plausible prose' to precise statements much earlier in the software development process.

What Chris Newcombe, senior engineer at Oracle, wrote.

TLA+ is the most valuable thing that I've learned in my professional career.

It has changed how I work

It has changed how I think

What Chris Newcombe, senior engineer at Oracle, wrote.

Mathematical thinking

~~TLA+~~ is the most valuable thing that I've learned in my professional career.

It has changed how I work

It has changed how I think