The Paxos Algorithm

Leslie Lamport
Microsoft Research

The Paxos Algorithm

or

How to Win a Turing Award

Leslie Lamport
Microsoft Research

The Paxos Algorithm

or

How to Win a Turing Award

and

Why You Should Know Even if You’re not Going to Win One

Leslie Lamport
Microsoft Research

When I visit universities, young researchers ask me:

What's the next big problem?

When I visit universities, young researchers ask me:

What's the next big problem?

What should I work on?

When I visit universities, young researchers ask me:

What's the next big problem?

What should I work on?

My answer:

If I knew, I'd be working on it.

What they really want to ask:

How can I win a Turing award?

What they really want to ask:

How can I win a Turing award?

My answer:

Don't worry about the "big problem".

What they really want to ask:

How can I win a Turing award?

My answer:

Don't worry about the "big problem".

Learn how to think properly.

What they really want to ask:

How can I win a Turing award?

My answer:

Don't worry about the "big problem".

Learn how to think properly.

I learned to think properly by learning math.

Mathematics

Mathematics

By the end of your first year at university you learned almost all the math I've ever found useful.

Mathematics

By the end of your first year at university you learned almost all the math I've ever found useful.

What you probably didn't learn is how to use that math outside a math class.

Mathematics

By the end of your first year at university you learned almost all the math I've ever found useful.

What you probably didn't learn is how to use that math outside a math class.

This lecture is a lesson in how to do that.

The heart of mathematics is abstraction

The heart of mathematics is abstraction — ignoring uninteresting details and finding what's important.

The heart of mathematics is abstraction — ignoring uninteresting details and finding what’s important.

This might not lead to an important problem

The heart of mathematics is abstraction — ignoring uninteresting details and finding what's important.

This might not lead to an important problem, but it will keep you from wasting time on non-problems.

A Formal Language for Writing Math

A Formal Language for Writing Math

You studied math in school without one.

A Formal Language for Writing Math

You studied math in school without one.
Why use one now?

A Formal Language for Writing Math

You studied math in school without one.
Why use one now?

Because you can't use math outside a math class
if only a math teacher can check your math.

A Formal Language for Writing Math

You studied math in school without one.
Why use one now?

Because you can't use math outside a math class
if only a math teacher can check your math.

You need to write math in a formal language
so tools can check it.

The formal language I'll use is called TLA^+.

The formal language I'll use is called TLA^+.

(Don't worry why.)

The formal language I'll use is called TLA$^+$.

It's mostly a precise way to write the math you already know.

The formal language I'll use is called TLA$^+$.

It's mostly a precise way to write the math you already know.

It introduces two concepts you haven't seen before.

The formal language I'll use is called TLA^+.

It's mostly a precise way to write the math you already know.

It introduces two concepts you haven't seen before.

But they're so simple you probably won't notice
that they're new.

CONSENSUS

The Problem Solved by Paxos

The Problem Solved by Paxos

Multiple clients can send requests to a system.

The Problem Solved by Paxos

Multiple clients can send requests to a system.

The system must choose in what order to handle them.

The Problem Solved by Paxos

Multiple clients can send requests to a system.

The system must choose in what order to handle them.

The system is implemented by multiple computers.

The Problem Solved by Paxos

Multiple clients can send requests to a system.

The system must choose in what order to handle them.

The system is implemented by multiple computers.

They must choose a single ordering even if some computers fail.

The Problem Solved by Paxos

Multiple clients can send requests to a system.

The system must choose in what order to handle them.

The system is implemented by multiple computers.

They must choose a single ordering even if some computers fail.

Paxos solves this problem by running multiple solutions to the following problem.

The Consensus Problem

The Consensus Problem

Multiple clients can each send a request to a system.

The Consensus Problem

Multiple clients can each send a request to a system.

The system must choose which one to handle next.

The Consensus Problem

Multiple clients can each send a request to a system.

The system must choose which one to handle next.

The system is implemented by multiple computers.

The Consensus Problem

Multiple clients can each send a request to a system.

The system must choose which one to handle next.

The system is implemented by multiple computers.

They must choose a single request even if some computers fail.

The Consensus Problem

Multiple clients can each send a request to a system.

The system must choose which one to handle next.

The system is implemented by multiple computers.

They must choose a single request even if some computers fail.

Paxos is efficient because it simultaneously executes the first part of all the consensus solutions.

Paxos is efficient because it simultaneously executes the first part of all the consensus solutions.

But we’re interested in why Paxos is correct, not why it’s efficient.

Paxos is efficient because it simultaneously executes the first part of all the consensus solutions.

But we're interested in why Paxos is correct, not why it's efficient.

So we abstract away such implementation details.

The Consensus Problem

Multiple clients can each send a request to a system.

The system must choose which one to handle next.

The system is implemented by multiple computers.

They must choose a single request even if some computers fail.

A More Abstract Consensus Problem

Multiple clients can each send a request to a system.

The system must choose which one to handle next.

The system is implemented by multiple computers.

They must choose a single request even if some computers fail.

A More Abstract Consensus Problem

Forget about clients.

The system must choose which one to handle next.

The system is implemented by multiple computers.

They must choose a single request even if some computers fail.

A More Abstract Consensus Problem

The system must choose which one to handle next.

The system is implemented by multiple computers.

They must choose a single request even if some computers fail.

The computers may choose any request in the set $Value$.

A More Abstract Consensus Problem

The system must choose which one to handle next.

The system is implemented by multiple computers.

They must choose a single request even if some computers fail.

The computers may choose any request in the set $Value$.

A More Abstract Consensus Problem

The system must choose which one to handle next.

The system is implemented by multiple computers.

They must choose a single value even if some computers fail.

The computers may choose any value in the set $Value$.

The Mathematical Model

The Mathematical Model

An execution of the system is represented by a behavior.

The Mathematical Model

An execution of the system is represented by a behavior.

A behavior is a sequence of states.

The Mathematical Model

An execution of the system is represented by a behavior.

A behavior is a sequence of states.

A state is an assignment of values to variables.

The Mathematical Model

An execution of the system is represented by a behavior.

A behavior is a sequence of states.

A state is an assignment of values to variables.

The system is described by a formula about behaviors that's true on behaviors that represent possible system executions.

The Mathematical Model

An execution of the system is represented by a behavior.

A behavior is a sequence of states.

A state is an assignment of values to variables.

The system is described by a formula about behaviors that's true on behaviors that represent possible system executions.

I like this model because it's simple

The Mathematical Model

An execution of the system is represented by a behavior.

A behavior is a sequence of states.

A state is an assignment of values to variables.

The system is described by a formula about behaviors that's true on behaviors that represent possible system executions.

I like this model because it's simple, and it resembles the way physics describes systems.

The Mathematical Model

An execution of the system is represented by a behavior.

A behavior is a sequence of states.

A state is an assignment of values to variables.

The system is described by a formula about behaviors that's true on behaviors that represent possible system executions.

I like this model because it's simple, and it resembles the way physics describes systems.

And years of experience has told me that it works well.

Writing Formulas that Describe Behaviors

Writing Formulas that Describe Behaviors

I'll write them in a real language with a grammar and formal semantics.

Writing Formulas that Describe Behaviors

I'll write them in a real language with a grammar and formal semantics.

Because otherwise computer people won't believe I'm doing something relevant to real systems.

Writing Formulas that Describe Behaviors

I'll write them in a real language with a grammar and formal semantics.

Because otherwise computer people won't believe I'm doing something relevant to real systems.

Perhaps with good reason.

MODULE *Consensus*

MODULE *Consensus*

The specification appears in a module.

MODULE *Consensus*

This is the pretty-printed version.

The actual spec is in ASCII.

MODULE *Consensus*

EXTENDS *Naturals*, *FiniteSets*

Imports definitions from other modules.

MODULE *Consensus*

EXTENDS *Naturals*, *FiniteSets*

Contains definitions of $+$, $*$, etc.

MODULE *Consensus*

EXTENDS *Naturals*, *FiniteSets*

Contains definitions of a couple of operators for finite sets.

MODULE *Consensus*

EXTENDS *Naturals*, *FiniteSets*

CONSTANT *Value*

MODULE *Consensus*

EXTENDS *Naturals*, *FiniteSets*

CONSTANT *Value*

Declares a “variable” that remains constant throughout a behavior.

MODULE *Consensus*

EXTENDS *Naturals*, *FiniteSets*

CONSTANT *Value*

VARIABLE *chosen*

Declares a variable whose value can differ in different states of a behavior.

MODULE *Consensus*

EXTENDS *Naturals*, *FiniteSets*

CONSTANT *Value*

VARIABLE *chosen*

There are lots of ways to model choosing a value.

MODULE *Consensus*

EXTENDS *Naturals*, *FiniteSets*

CONSTANT *Value*

VARIABLE *chosen*

There are lots of ways to model choosing a value.

I decided to let *chosen* be the set of values that have been chosen.

MODULE *Consensus*

EXTENDS *Naturals*, *FiniteSets*

CONSTANT *Value*

VARIABLE *chosen*

There are lots of ways to model choosing a value.

I decided to let *chosen* be the set of values that have been chosen.

The spec should say that (in a correct behavior) at most one value is chosen.

$$TypeOK \triangleq \begin{array}{l} \land\ chosen \subseteq Value \\ \land\ IsFiniteSet(chosen) \end{array}$$

$$TypeOK \triangleq \land\ chosen \subseteq Value$$
$$\qquad\qquad\quad \land\ IsFiniteSet(chosen)$$

Defines $TypeOK$

$$TypeOK \triangleq \boxed{\begin{array}{l} \land\ chosen \subseteq Value \\ \land\ IsFiniteSet(chosen) \end{array}}$$

Defines $TypeOK$ to equal this.

$$TypeOK \triangleq \begin{array}{|l|}\hline \land \\ \land \\ \hline\end{array} \begin{array}{l} chosen \subseteq Value \\ IsFiniteSet(chosen) \end{array}$$

This is usually written

$$TypeOK \triangleq \begin{array}{l} \ chosen \subseteq Value \\ \land\ IsFiniteSet(chosen) \end{array}$$

This is usually written like this.

$$TypeOK \triangleq \begin{array}{l} \land\ chosen \subseteq Value \\ \land\ IsFiniteSet(chosen) \end{array}$$

It's a useful notation for writing large formulas.

$$TypeOK \triangleq \land \boxed{chosen \subseteq Value} \\ \land IsFiniteSet(chosen)$$

Asserts that $chosen$ is a set of values.

$$TypeOK \triangleq \land\ chosen \subseteq Value \\ \land\ \boxed{IsFiniteSet(chosen)}$$

Asserts that it's a finite set.

$$TypeOK \triangleq \land\ chosen \subseteq Value$$
$$\qquad\qquad\quad \land\ \boxed{IsFiniteSet(chosen)}$$

Asserts that it's a finite set.

(There's no reason $Value$ can't be infinite.)

$$TypeOK \triangleq \land\ chosen \subseteq Value$$
$$\qquad\qquad \land\ IsFiniteSet(chosen)$$

Most mathematicians don't talk about types.

$$TypeOK \triangleq \land chosen \subseteq Value$$
$$\qquad \land IsFiniteSet(chosen)$$

Most mathematicians don't talk about types.

In math, having a type just means belonging to some set.

$$TypeOK \triangleq \land chosen \subseteq Value$$
$$\qquad\qquad \land IsFiniteSet(chosen)$$

Most mathematicians don't talk about types.

In math, having a type just means belonging to some set.

$TypeOK$ asserts that $chosen$ is an element of the set of all finite subsets of $Value$.

$$TypeOK \triangleq \land chosen \subseteq Value$$
$$\land IsFiniteSet(chosen)$$

$TypeOK$ should be an invariant of the spec,

$$TypeOK \triangleq \land chosen \subseteq Value$$
$$\land IsFiniteSet(chosen)$$

$TypeOK$ should be an invariant of the spec, meaning its true in every state of every behavior satisfying the spec.

$$TypeOK \triangleq \land chosen \subseteq Value$$
$$\qquad\qquad \land IsFiniteSet(chosen)$$

$TypeOK$ should be an invariant of the spec, meaning its true in every state of every behavior satisfying the spec.

Defining a type invariant helps the reader understand the spec.

Writing Comments

Writing Comments

Formulas are not easy to understand.

Writing Comments

Formulas are not easy to understand.

People need prose explanations of them.

Writing Comments

Formulas are not easy to understand.

People need prose explanations of them.

You should put those explanations in comments in the module.

Writing Comments

Formulas are not easy to understand.

People need prose explanations of them.

You should put those explanations in comments in the module.

I'm telling you the explanations.

Writing Comments

Formulas are not easy to understand.

People need prose explanations of them.

You should put those explanations in comments in the module.

I’m telling you the explanations.

The written versions of the specs have lots of comments.

The Spec Formula

The Spec Formula

Remember that I'm going to represent the system by a mathematical formula.

The Spec Formula

Remember that I'm going to represent the system by a mathematical formula.

The word *spec* is used to mean both that formula and the entire module containing the formula.

The Spec Formula

Remember that I'm going to represent the system by a mathematical formula.

The word *spec* is used to mean both that formula and the entire module containing the formula.

For now, *spec* will mean the formula.

The spec must describe a set of behaviors

The spec must describe a set of behaviors, where a behavior is a sequence of states.

The spec must describe a set of behaviors, where a behavior is a sequence of states.

It does this with two formulas:

The spec must describe a set of behaviors, where a behavior is a sequence of states.

It does this with two formulas:

The Initial Formula

The spec must describe a set of behaviors, where a behavior is a sequence of states.

It does this with two formulas:

The Initial Formula

It describes all possible first states of a behavior.

The spec must describe a set of behaviors, where a behavior is a sequence of states.

It does this with two formulas:

The Initial Formula

It describes all possible first states of a behavior.

The Next-State Formula

The spec must describe a set of behaviors, where a behavior is a sequence of states.

It does this with two formulas:

The Initial Formula

It describes all possible first states of a behavior.

The Next-State Formula

For any state in a behavior, it describes all possible next states in the behavior.

The Initial Formula

The Initial Formula

$$Init \triangleq$$

We can call the formula anything,
but $Init$ is the conventional name.

The Initial Formula

$$Init \triangleq chosen = \{\}$$

The Initial Formula

$$Init \triangleq \boxed{chosen = \{\}}$$

Asserts that the value of $chosen$ in the state is the empty set.

The Initial Formula

$$Init \triangleq \boxed{chosen = \{\}}$$

Asserts that the value of $chosen$ in the state is the empty set.

In the first state of the behavior, no value is chosen.

The Next-State Formula

The Next-State Formula

$$Next \triangleq$$

It's conventional to call it $Next$.

The Next-State Formula

$$Next \triangleq \begin{array}{l} \land \; chosen = \{\} \\ \land \; \exists\, v \in Value : chosen' = \{v\} \end{array}$$

It equals the conjunction of two formulas.

The Next-State Formula

$$Next \triangleq \begin{array}{l} \land \boxed{chosen = \{\}} \\ \land \exists v \in Value : chosen' = \{v\} \end{array}$$

Asserts that in a state with $chosen$ equal to the empty set

The Next-State Formula

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \boxed{\exists\, v \in Value :}\ chosen' = \{v\} \end{array}$$

Asserts that in a state with $chosen$ equal to the empty set, there exists a value v in the set $Value$ such that

The Next-State Formula

$$
\begin{aligned}
Next \triangleq\ & \land\ chosen = \{\} \\
& \land\ \exists\, v \in Value : chosen' = \{v\}
\end{aligned}
$$

Asserts that in a state with $chosen$ equal to the empty set, there exists a value v in the set $Value$ such that the value of $chosen$

The Next-State Formula

$$Next \triangleq \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\}$$

Asserts that in a state with $chosen$ equal to the empty set, there exists a value v in the set $Value$ such that the value of $chosen$ in the next state

The Next-State Formula

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Asserts that in a state with $chosen$ equal to the empty set, there exists a value v in the set $Value$ such that the value of $chosen$ in the next state

$'$ (prime) means *in the next state*.

The Next-State Formula

$$Next \triangleq \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' \boxed{= \{v\}}$$

Asserts that in a state with $chosen$ equal to the empty set, there exists a value v in the set $Value$ such that the value of $chosen$ in the next state equals the set containing the single element v.

The Next-State Formula

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Asserts that in a state with $chosen$ equal to the empty set, there exists a value v in the set $Value$ such that the value of $chosen$ in the next state equals the set containing the single element v.

The Next-State Formula

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

The Next-State Formula

$$Next \triangleq \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\}$$

Formula $Next$ expresses a condition on a pair of states:

The Next-State Formula

$$Next \triangleq \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\}$$

Formula $Next$ expresses a condition on a pair of states:

– A 1st state described by the unprimed variables.

The Next-State Formula

$$Next \triangleq \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\}$$

Formula $Next$ expresses a condition on a pair of states:

– A 1st state described by the unprimed variables.

– A 2nd state described by the primed variables.

The Next-State Formula

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Formula $Next$ expresses a condition on a pair of states:

- A 1st state described by the unprimed variables.
- A 2nd state described by the primed variables.

Think of the pair as describing a **step** in which the system goes from the 1st state to the 2nd state.

The Next-State Formula

$$Next \triangleq \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\}$$

Formula $Next$ expresses a condition on a pair of states:

– A ~~1~~st state described by the unprimed variables.

– A ~~2~~nd state described by the primed variables.

The Next-State Formula

$$Next \triangleq \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\}$$

Formula $Next$ expresses a condition on a pair of states:

– A current state described by the unprimed variables.

– A next state described by the primed variables.

The Next-State Formula

$$
\begin{aligned}
Next \triangleq\ & \land\ chosen = \{\} \\
& \land\ \exists\, v \in Value : chosen' = \{v\}
\end{aligned}
$$

The Next-State Formula

$$
\begin{aligned}
Next \triangleq\ & \land \boxed{chosen = \{\}} \\
& \land \exists v \in Value : chosen' = \{v\}
\end{aligned}
$$

This formula has no primes, so it's a condition on the current state.

The Next-State Formula

$$Next \triangleq \begin{array}{l} \land\ \boxed{chosen = \{\}} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

This formula has no primes, so it's a condition on the current state.

If it equals FALSE, then there is no next state for which $Next$ equals TRUE.

The Next-State Formula

$$Next \triangleq \land\ \boxed{chosen = \{\}} \\ \land\ \exists\, v \in Value : chosen' = \{v\}$$

This formula has no primes, so it's a condition on the current state.

If it equals FALSE, then there is no next state for which $Next$ equals TRUE.

It's an **enabling condition** for the step.

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$\begin{aligned} Next \triangleq\ & \land\ chosen = \{\} \\ & \land\ \exists\, v \in Value : chosen' = \{v\} \end{aligned}$$

I've described the possible behaviors with two formulas:

$$Init \triangleq \boxed{chosen = \{\}}$$

$$\begin{aligned} Next \triangleq\ & \land\ chosen = \{\} \\ & \land\ \exists\, v \in Value : chosen' = \{v\} \end{aligned}$$

A condition on the initial state of a behavior.

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

A condition on the initial state of a behavior.

A condition on all the steps of the behavior.

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Let's construct a behavior satisfying these conditions.

I've described the possible behaviors with two formulas:

$$Init \triangleq \boxed{chosen = \{\}}$$

$$\begin{aligned} Next \triangleq\ & \land\ chosen = \{\} \\ & \land\ \exists\, v \in Value : chosen' = \{v\} \end{aligned}$$

Let's construct a behavior satisfying these conditions.

$$\left[chosen = \{\} \right]$$

The is only one posible initial state.

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Let's construct a behavior satisfying these conditions.

$$\left[chosen = \{\} \right]$$

The is only one posible initial state.

Remember that a state is an assignment of a value to the one variable $chosen$.

I’ve described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Let’s construct a behavior satisfying these conditions.

$[chosen = \{\}]$ $[chosen = \{42\}]$

A possible next state, if 42 $\in$ *Value* .

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land \boxed{chosen = \{\}} \\ \land \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Let's construct a behavior satisfying these conditions.

$$\left[chosen = \{\}\right] \qquad \left[chosen = \{42\}\right]$$

A possible next state, if $42 \in Value$.

The first condition is satisfied

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \land\ \boxed{chosen = \{\}}$$
$$\qquad\quad \land\ \exists\, v \in Value : chosen' = \{v\}$$

Let's construct a behavior satisfying these conditions.

$$\big[chosen = \{\,\}\big] \qquad \big[chosen = \{42\}\big]$$

A possible next state, if $42 \in Value$.

The first condition is satisfied because $chosen$ equals $\{\,\}$ in the current state of the step.

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\}$$

Let's construct a behavior satisfying these conditions.

$$[chosen = \{\}] \qquad [chosen = \{42\}]$$

A possible next state, if $42 \in Value$.

The second condition is satisfied

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \boxed{\exists\, v \in Value : chosen' = \{v\}} \end{array}$$

Let's construct a behavior satisfying these conditions.

$$\left[chosen = \{\}\right] \qquad \left[chosen = \{42\}\right]$$

A possible next state, if $42 \in Value$.

The second condition is satisfied because $chosen$ equals $\{42\}$ in the next state of the step.

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \land\ chosen = \{\} \\ \land\ \boxed{\exists\, v \in Value : chosen' = \{v\}}$$

Let's construct a behavior satisfying these conditions.

$$\left[chosen = \{\}\right] \qquad \left[chosen = \{42\}\right]$$

A possible next state, if $42 \in Value$.

The second condition is satisfied because $chosen$ equals $\{42\}$ in the next state of the step.

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Let's construct a behavior satisfying these conditions.

$$\left[chosen = \{\}\right] \qquad \left[chosen = \{42\}\right]$$

There is no possible next state after this one

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land \boxed{chosen = \{\}} \\ \land \exists v \in Value : chosen' = \{v\} \end{array}$$

Let's construct a behavior satisfying these conditions.

$$\left[chosen = \{\}\right] \qquad \left[chosen = \{42\}\right]$$

There is no possible next state after this one because the enabling condition equals FALSE ,

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Let's construct a behavior satisfying these conditions.

$$\left[chosen = \{\}\right] \qquad \left[chosen = \{42\}\right]$$

There is no possible next state after this one because the enabling condition equals FALSE, so no next state can satisfy formula $Next$.

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Let's construct a behavior satisfying these conditions.

$$\big[chosen = \{\,\}\big] \qquad \big[chosen = \{42\}\big]$$

A behavior of the *Consensus* system can have only two states.

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

Let's construct a behavior satisfying these conditions.

$$\Big[chosen = \{\}\Big] \longrightarrow \Big[chosen = \{42\}\Big]$$

I like to write behaviors like this.

I've described the possible behaviors with two formulas:

$$Init \triangleq chosen = \{\}$$

$$\begin{aligned} Next \triangleq\ & \land\ chosen = \{\} \\ & \land\ \exists\, v \in Value : chosen' = \{v\} \end{aligned}$$

I've described the possible behaviors with two formulas:

$$Init \triangleq \text{a condition on states}$$

$$Next \triangleq \begin{array}{l} \land\ chosen = \{\} \\ \land\ \exists\, v \in Value : chosen' = \{v\} \end{array}$$

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I'll now combine them into a single formula that's a condition on behaviors.

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I'll now combine them into a single formula that's a condition on behaviors.

$Spec \triangleq$

It's a convention to call the formula $Spec$.

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I'll now combine them into a single formula that's a condition on behaviors.

$Spec \triangleq$

A condition on states is interpreted as a condition on the initial state of a behavior.

I've described the possible behaviors with two formulas:

$$Init \triangleq \text{a condition on states}$$

$$Next \triangleq \text{a condition on steps}$$

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init$$

A condition on states is interpreted as a condition on the initial state of a behavior.

So $Spec$ asserts that the initial state of the behavior satisfies $Init$.

I've described the possible behaviors with two formulas:

$$Init \triangleq \text{a condition on states}$$

$$Next \triangleq \text{a condition on steps}$$

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init \land$$

and

I've described the possible behaviors with two formulas:

$$Init \triangleq \text{a condition on states}$$

$$Next \triangleq \text{a condition on steps}$$

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init \land \Box$$

$\Box$ means "at all points in the behavior"

I've described the possible behaviors with two formulas:

$$Init \triangleq \text{a condition on states}$$

$$Next \triangleq \text{a condition on steps}$$

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init \land \Box$$

$\Box$ means "at all points in the behavior"

For example, $\Box\, TypeOK$ asserts that all states of the behavior satisfy $TypeOK$.

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init \land \Box\, Next$$

$\Box$ means "at all points in the behavior"

And $\Box\, Next$ asserts that all steps of the behavior satisfy $Next$.

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init \land \Box Next$$

So $Spec$ is true of a behavior if and only if

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init \land \Box\, Next$$

So $Spec$ is true of a behavior if and only if the behavior's first state satisfies $Init$

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init \land \Box Next$$

So $Spec$ is true of a behavior if and only if the behavior's first state satisfies $Init$ and every step in the behavior satisfies $Next$.

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init \land \Box Next$$

$Spec$ is our specification of the $Consensus$ system

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I'll now combine them into a single formula that's
a condition on behaviors.

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

$Spec$ is our specification of the $Consensus$ system
except this is its actual definition.

I've described the possible behaviors with two formulas:

$$Init \triangleq \text{a condition on states}$$

$$Next \triangleq \text{a condition on steps}$$

I'll now combine them into a single formula that's a condition on behaviors.

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

Don't worry about this stuff; I'll explain it later.

I've described the possible behaviors with two formulas:

$Init \triangleq$ a condition on states

$Next \triangleq$ a condition on steps

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init \wedge \Box[Next]_{chosen}$$

Don't worry about this stuff; I'll explain it later.

For now

I've described the possible behaviors with two formulas:

$$Init \triangleq \text{a condition on states}$$

$$Next \triangleq \text{a condition on steps}$$

I'll now combine them into a single formula that's a condition on behaviors.

$$Spec \triangleq Init \land \Box\, Next$$

Don't worry about this stuff; I'll explain it later.

For now pretend it's not there.

Checking the Spec

Checking the Spec

Our specification is a formula that is true for some behaviors and false for others.

Checking the Spec

Our specification is a formula that is true for some behaviors and false for others.

There's no mathematical sense in which such a formula is correct or incorrect.

Checking the Spec

Our specification is a formula that is true for some behaviors and false for others.

There's no mathematical sense in which such a formula is correct or incorrect.

Our spec is correct if it means what we want it to mean

Checking the Spec

Our specification is a formula that is true for some behaviors and false for others.

There's no mathematical sense in which such a formula is correct or incorrect.

Our spec is correct if it means what we want it to mean, and wanting is not a mathematical concept.

Checking the Spec

Our specification is a formula that is true for some behaviors and false for others.

There's no mathematical sense in which such a formula is correct or incorrect.

Our spec is correct if it means what we want it to mean, and wanting is not a mathematical concept.

So, how do we check that our spec is correct?

We check our spec by checking that it satisfies properties that it should satisfy.

We check our spec by checking that it satisfies properties that it should satisfy. For example, it should be type correct.

We check our spec by checking that it satisfies properties that it should satisfy. For example, it should be type correct.

Type correctness of $Spec$ means:

In every behavior that satisfies $Spec$,
every state of that behavior satisfies $TypeOK$.

We check our spec by checking that it satisfies properties that it should satisfy. For example, it should be type correct.

Type correctness of $Spec$ means:

In every behavior that satisfies $Spec$,
every state of that behavior satisfies $TypeOK$.

In other words:

Every behavior that satisfies $Spec$ satisfies $\Box\, TypeOK$.

We check our spec by checking that it satisfies properties that it should satisfy. For example, it should be type correct.

Type correctness of $Spec$ means:

In every behavior that satisfies $Spec$,
every state of that behavior satisfies $TypeOK$.

In other words:

Every behavior that satisfies $Spec$ satisfies $\Box\, TypeOK$.

In other words:

Every behavior satisfies $Spec \Rightarrow \Box\, TypeOK$.

We check our spec by checking that it satisfies properties that it should satisfy. For example, it should be type correct.

Type correctness of $Spec$ means:

In every behavior that satisfies $Spec$,
every state of that behavior satisfies $TypeOK$.

In other words:

Every behavior that satisfies $Spec$ satisfies $\Box\, TypeOK$.

In other words:

Every behavior satisfies $Spec \Rightarrow \Box\, TypeOK$.

A formula about behaviors that's satisfied by all behaviors
is a theorem, so

We check our spec by checking that it satisfies properties that it should satisfy. For example, it should be type correct.

Type correctness of $Spec$ means:
In every behavior that satisfies $Spec$,
every state of that behavior satisfies $TypeOK$.

In other words:
Every behavior that satisfies $Spec$ satisfies $\Box\, TypeOK$.

In other words:
Every behavior satisfies $Spec \Rightarrow \Box\, TypeOK$.

We write this:
THEOREM $Spec \Rightarrow \Box\, TypeOK$

For any formula F that expresses a condition on states, if $Spec \Rightarrow \Box F$ is a theorem, then we say that F is an **invariant** of $Spec$.

For any formula F that expresses a condition on states, if $Spec \Rightarrow \Box F$ is a theorem, then we say that F is an **invariant** of $Spec$.

Recall that $TypeOK$ is defined by:

$$TypeOK \triangleq \begin{array}{l} \land\ chosen \subseteq Value \\ \land\ IsFiniteSet(chosen) \end{array}$$

For any formula F that expresses a condition on states, if $Spec \Rightarrow \Box F$ is a theorem, then we say that F is an **invariant** of $Spec$.

Recall that $TypeOK$ is defined by:

$$TypeOK \triangleq \begin{array}{l} \land\ chosen \subseteq Value \\ \land\ IsFiniteSet(chosen) \end{array}$$

The $Consensus$ module defines this state condition that is stronger than $TypeOK$, and asserts that it's an invariant:

For any formula F that expresses a condition on states, if $Spec \Rightarrow \Box F$ is a theorem, then we say that F is an **invariant** of $Spec$.

Recall that $TypeOK$ is defined by:

$$TypeOK \triangleq \begin{array}{l} \land\ chosen \subseteq Value \\ \land\ IsFiniteSet(chosen) \end{array}$$

The $Consensus$ module defines this state condition that is stronger than $TypeOK$, and asserts that it's an invariant:

$$Inv \triangleq \begin{array}{l} \land\ TypeOK \\ \land\ Cardinality(chosen) \leq 1 \end{array}$$

For any formula F that expresses a condition on states, if $Spec \Rightarrow \Box F$ is a theorem, then we say that F is an **invariant** of $Spec$.

Recall that $TypeOK$ is defined by:

$$TypeOK \triangleq \land\ chosen \subseteq Value \\ \land\ IsFiniteSet(chosen)$$

The $Consensus$ module defines this state condition that is stronger than $TypeOK$, and asserts that it's an invariant:

$$Inv \triangleq \land\ TypeOK \\ \land\ \boxed{Cardinality}(chosen) \leq 1$$

Defined in the $FiniteSets$ module to be the number of elements in a finite set.

For any formula F that expresses a condition on states, if $Spec \Rightarrow \Box F$ is a theorem, then we say that F is an **invariant** of $Spec$.

Recall that $TypeOK$ is defined by:

$$TypeOK \triangleq \begin{array}{l} \land\ chosen \subseteq Value \\ \land\ IsFiniteSet(chosen) \end{array}$$

The $Consensus$ module defines this state condition that is stronger than $TypeOK$, and asserts that it's an invariant:

$$Inv \triangleq \begin{array}{l} \land\ TypeOK \\ \land\ Cardinality(chosen) \leq 1 \end{array}$$

The invariance of Inv should be obvious.

The invariance of Inv should be obvious.

$Consensus$ should be obviously correct because we'll use it to define what correctness of the $Paxos$ spec means.

The invariance of Inv should be obvious.

$Consensus$ should be obviously correct because we'll use it to define what correctness of the $Paxos$ spec means.

We should check it for two reasons:

The invariance of Inv should be obvious.

$Consensus$ should be obviously correct because we'll use it to define what correctness of the $Paxos$ spec means.

We should check it for two reasons:

– To learn how, because we'll need to check that our $Paxos$ spec correctly describes the Paxos algorithm.

The invariance of Inv should be obvious.

$Consensus$ should be obviously correct because we'll use it to define what correctness of the $Paxos$ spec means.

We should check it for two reasons:

– To learn how, because we'll need to check that our $Paxos$ spec correctly describes the Paxos algorithm.

– When you start writing specs, you can make mistakes even in one as simple as $Consensus$.

The invariance of Inv should be obvious.

$Consensus$ should be obviously correct because we'll use it to define what correctness of the $Paxos$ spec means.

We should check it for two reasons:

- To learn how, because we'll need to check that our $Paxos$ spec correctly describes the Paxos algorithm.
- When you start writing specs, you can make mistakes even in one as simple as $Consensus$.

The easy way to check a spec: use the TLC model checker.

The invariance of Inv should be obvious.

$Consensus$ should be obviously correct because we'll use it to define what correctness of the $Paxos$ spec means.

We should check it for two reasons:

– To learn how, because we'll need to check that our $Paxos$ spec correctly describes the Paxos algorithm.

– When you start writing specs, you can make mistakes even in one as simple as $Consensus$.

The easy way to check a spec: use the TLC model checker.

Just tell TLC what set to use for $Value$.

The invariance of Inv should be obvious.

$Consensus$ should be obviously correct because we'll use it to define what correctness of the $Paxos$ spec means.

We should check it for two reasons:

– To learn how, because we'll need to check that our $Paxos$ spec correctly describes the Paxos algorithm.

– When you start writing specs, you can make mistakes even in one as simple as $Consensus$.

The easy way to check a spec: use the TLC model checker.

Just tell TLC what set to use for $Value$.

TLC computes all possible executions and reports an error if one doesn't satisfy $\Box Inv$.

Proving Invariance

Proving Invariance

I’m not going to prove anything, but let’s see how it’s done.

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \triangleq \boxed{\mathit{Spec} \Rightarrow \Box \mathit{Inv}}$$

The theorem asserting that Inv is an invariant of Spec.

Proving Invariance

$$\text{THEOREM } \boxed{\mathit{Invariance} \quad \triangleq} \quad \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

The theorem asserting that Inv is an invariant of Spec.

This gives the theorem the name $\mathit{Invariance}$.

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \;\triangleq\; \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

The theorem asserting that Inv is an invariant of Spec.

This gives the theorem the name $\mathit{Invariance}$.

Let's look at the proof.

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \;\triangleq\; \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$$

The first step is a condition on states.

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$
$$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$$

The first step is a condition on states.

It says that every state satisfying Init satisfies Inv .

Proving Invariance

$$\text{THEOREM } Invariance \triangleq Spec \Rightarrow \Box Inv$$
$$\langle 1 \rangle 1.\ Init \Rightarrow Inv$$

The first step is a condition on states.

It says that every state satisfying $Init$ satisfies Inv .

It implies Inv is satisfied by the initial state of any behavior satisfying $Spec$.

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \;\triangleq\; \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land [\mathit{Next}]_{\mathit{chosen}} \Rightarrow \mathit{Inv}'$

Here’s the next step,

Proving Invariance

$$\text{THEOREM } Invariance \triangleq Spec \Rightarrow \Box Inv$$

$$\langle 1 \rangle 1.\ Init \Rightarrow Inv$$

$$\langle 1 \rangle 2.\ Inv \wedge [Next]_{chosen} \Rightarrow Inv'$$

Here’s the next step, with this mysterious stuff.

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$$

$$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$$

Here's the next step, with this mysterious stuff.

Which we ignore.

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

The second proof step is a condition on steps.

Proving Invariance

$$\text{THEOREM } Invariance \triangleq Spec \Rightarrow \Box Inv$$

$\langle 1 \rangle 1.\ Init \Rightarrow Inv$

$\langle 1 \rangle 2.\ \boxed{Inv} \land Next \Rightarrow Inv'$

The second proof step is a condition on steps.

It asserts that if the current state satisfies Inv

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$$

$$\langle 1 \rangle 2.\ \mathit{Inv} \land \boxed{\mathit{Next}} \Rightarrow \mathit{Inv}'$$

The second proof step is a condition on steps.

It asserts that if the current state satisfies Inv
and the step satisfies formula Next

Proving Invariance

$$\text{THEOREM } Invariance \triangleq Spec \Rightarrow \Box Inv$$

$\langle 1 \rangle 1.\ Init \Rightarrow Inv$

$\langle 1 \rangle 2.\ Inv \land Next \Rightarrow Inv'$

The second proof step is a condition on steps.

It asserts that if the current state satisfies Inv
and the step satisfies formula $Next$
then the next state

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \boxed{\mathit{Inv}'}$

The second proof step is a condition on steps.

It asserts that if the current state satisfies Inv
and the step satisfies formula Next
then the next state satisfies Inv .

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$$

$$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$$

The second proof step is a condition on steps.

It asserts that if the current state satisfies Inv
and the step satisfies formula Next
then the next state satisfies Inv .

This condition is satisfied by every step of every behavior

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$$

$$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$$

The second proof step is a condition on steps.

It asserts that if the current state satisfies Inv
and the step satisfies formula Next
then the next state satisfies Inv .

This condition is satisfied by every step of every behavior (sequence of states).

$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.\ \text{QED}$

The last step of a proof is a QED step

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.$ QED

The last step of a proof is a QED step

It asserts that the preceding steps prove the theorem.

Proving Invariance

$\text{THEOREM } \mathit{Invariance} \;\triangleq\; \mathit{Spec} \Rightarrow \Box \mathit{Inv}$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \quad \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.\ \text{QED}$

$\quad \text{BY } \langle 1 \rangle 1,\ \langle 1 \rangle 2 \ \ \text{DEF } \mathit{Spec}$

Here's its proof

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.$ QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF Spec

Here's its proof which says that the theorem follows from steps $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$ and the definition of Spec.

$$\text{THEOREM } Invariance \triangleq Spec \Rightarrow \Box Inv$$

$\langle 1 \rangle 1.\ Init \Rightarrow Inv$

$\langle 1 \rangle 2.\ Inv \land Next \Rightarrow Inv'$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF $Spec$

The proof is correct because if a behavior satisfies $Spec$ then

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.$ QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF Spec

The proof is correct because if a behavior satisfies Spec then

1. Its initial state satisfies Inv .

Proving Invariance

$\text{THEOREM } \mathit{Invariance} \;\triangleq\; \mathit{Spec} \Rightarrow \Box \mathit{Inv}$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.\ \text{QED}$

$\quad \text{BY } \langle 1 \rangle 1, \langle 1 \rangle 2 \ \text{ DEF } \mathit{Spec}$

The proof is correct because if a behavior satisfies Spec then

1. Its initial state satisfies Inv.

   Because it satisfies Init

Proving Invariance

$\textsc{theorem}\ \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.\ \textsc{qed}$

$\quad \textsc{by}\ \langle 1 \rangle 1,\ \langle 1 \rangle 2\ \ \textsc{def}\ \mathit{Spec}$

The proof is correct because if a behavior satisfies Spec then

1. Its initial state satisfies Inv .

   Because it satisfies Init (by definition of Spec)

Proving Invariance

THEOREM $Invariance \triangleq Spec \Rightarrow \Box Inv$

$\langle 1 \rangle 1.$ $Init \Rightarrow Inv$

$\langle 1 \rangle 2.$ $Inv \land Next \Rightarrow Inv'$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF $Spec$

The proof is correct because if a behavior satisfies $Spec$ then

1. Its initial state satisfies Inv.

   Because it satisfies $Init$ so by $\langle 1 \rangle 1$ it satisfies Inv.

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \;\triangleq\; \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.$ QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF Spec

The proof is correct because if a behavior satisfies Spec then

1. Its initial state satisfies Inv.
2. If any of its states satisfies Inv then so does its next state.

Proving Invariance

THEOREM $Invariance \triangleq Spec \Rightarrow \Box Inv$

$\langle 1 \rangle 1.$ $Init \Rightarrow Inv$

$\langle 1 \rangle 2.$ $Inv \land Next \Rightarrow Inv'$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF $Spec$

The proof is correct because if a behavior satisfies $Spec$ then

1. Its initial state satisfies Inv .
2. If any of its states satisfies Inv then so does its next state.
   Because the step to the next state satisfies $Next$

Proving Invariance

THEOREM $Invariance \triangleq Spec \Rightarrow \Box Inv$

$\langle 1 \rangle 1.$ $Init \Rightarrow Inv$

$\langle 1 \rangle 2.$ $Inv \wedge Next \Rightarrow Inv'$

$\langle 1 \rangle 3.$ QED
  BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF $Spec$

The proof is correct because if a behavior satisfies $Spec$ then

1. Its initial state satisfies Inv .
2. If any of its states satisfies Inv then so does its next state.
   Because the step to the next state satisfies $Next$ (by definition of $Spec$)

Proving Invariance

$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.\ \text{QED}$

$\quad \text{BY } \langle 1 \rangle 1, \langle 1 \rangle 2 \text{ DEF } \mathit{Spec}$

The proof is correct because if a behavior satisfies Spec then

1. Its initial state satisfies Inv.
2. If any of its states satisfies Inv then so does its next state.
   Because the step to the next state satisfies Next so by $\langle 1 \rangle 2$ its next state satisfies Inv.

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF Spec

The proof is correct because if a behavior satisfies Spec then

1. Its initial state satisfies Inv .
2. If any of its states satisfies Inv then so does its next state.
3. All of its states satisfy Inv .

Proving Invariance

$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \wedge \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.\ \text{QED}$

$\quad \text{BY } \langle 1 \rangle 1,\ \langle 1 \rangle 2 \ \text{ DEF } \mathit{Spec}$

The proof is correct because if a behavior satisfies Spec then

1. Its initial state satisfies Inv .
2. If any of its states satisfies Inv then so does its next state.
3. All of its states satisfy Inv .
   By 1 , 2 , and mathematical induction.

Proving Invariance

$$\text{THEOREM } \mathit{Invariance} \;\triangleq\; \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \wedge \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF Spec

The proof is correct because if a behavior satisfies Spec then

1. Its initial state satisfies Inv.
2. If any of its states satisfies Inv then so does its next state.
3. All of its states satisfy Inv.
4. The behavior satisfies $\Box\, \mathit{Inv}$.

Proving Invariance

$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.\ \text{QED}$

$\quad \text{BY } \langle 1 \rangle 1,\ \langle 1 \rangle 2 \ \text{ DEF } \mathit{Spec}$

The proof is correct because if a behavior satisfies Spec then

1. Its initial state satisfies Inv.
2. If any of its states satisfies Inv then so does its next state.
3. All of its states satisfy Inv.
4. The behavior satisfies $\Box \mathit{Inv}$.
   By 3, and the meaning of $\Box$.

Proving Invariance

THEOREM $Invariance \triangleq Spec \Rightarrow \Box Inv$

$\langle 1 \rangle 1.$ $Init \Rightarrow Inv$

$\langle 1 \rangle 2.$ $Inv \land Next \Rightarrow Inv'$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF $Spec$

The proof is correct because if a behavior satisfies $Spec$ then

1. Its initial state satisfies Inv .
2. If any of its states satisfies Inv then so does its next state.
3. All of its states satisfy Inv .
4. The behavior satisfies $\Box Inv$.

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land \mathit{Next} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF Spec

The proof is correct because if a behavior satisfies Spec then

1. Its initial state satisfies Inv .
2. If any of its states satisfies Inv then so does its next state.
3. All of its states satisfy Inv .
4. The behavior satisfies $\Box \mathit{Inv}$.

This proves the theorem.

$$\text{THEOREM } \mathit{Invariance} \;\triangleq\; \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \wedge [\mathit{Next}]_{\mathit{chosen}} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF Spec

Here's the proof.

THEOREM $Invariance \triangleq Spec \Rightarrow \Box Inv$

$\langle 1 \rangle 1.$ $Init \Rightarrow Inv$

$\langle 1 \rangle 2.$ $Inv \wedge [Next]_{chosen} \Rightarrow Inv'$

$\langle 1 \rangle 3.$ QED
BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF $Spec$

Here's the proof (with the mysterious stuff added back).

$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land [\mathit{Next}]_{chosen} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.\ \text{QED}$

$\quad \boxed{\text{BY } \langle 1 \rangle 1,\ \langle 1 \rangle 2 \ \text{ DEF } \mathit{Spec}}$

The TLA$^+$ proof system can check this proof.

$$\text{THEOREM } Invariance \triangleq Spec \Rightarrow \Box Inv$$

$\langle 1 \rangle 1.$ $Init \Rightarrow Inv$

$\langle 1 \rangle 2.$ $Inv \land [Next]_{chosen} \Rightarrow Inv'$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF $Spec$

A complete proof would have proofs of the other two steps

$\text{THEOREM } Invariance \triangleq Spec \Rightarrow \Box Inv$

$\langle 1 \rangle 1.\ Init \Rightarrow Inv$

$\langle 1 \rangle 2.\ Inv \land [Next]_{chosen} \Rightarrow Inv'$

$\langle 1 \rangle 3.\ \text{QED}$

$\quad \text{BY } \langle 1 \rangle 1, \langle 1 \rangle 2 \ \text{ DEF } Spec$

A complete proof would have proofs of the other two steps, each proof being either

$\text{THEOREM } \mathit{Invariance} \;\triangleq\; \mathit{Spec} \Rightarrow \Box \mathit{Inv}$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land [\mathit{Next}]_{chosen} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.\ \text{QED}$

$\quad \text{BY } \langle 1 \rangle 1,\ \langle 1 \rangle 2 \;\; \text{DEF } \mathit{Spec}$

A complete proof would have proofs of the other two steps, each proof being either

- A BY statement, or
- A sequence of steps $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, … with their proofs

$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \wedge [\mathit{Next}]_{\mathit{chosen}} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.\ \text{QED}$

$\quad \text{BY } \langle 1 \rangle 1,\ \langle 1 \rangle 2 \ \ \text{DEF } \mathit{Spec}$

A complete proof would have proofs of the other two steps, each proof being either

- A BY statement, or
- A sequence of steps $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, ... with their proofs, each proof being either

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land [\mathit{Next}]_{chosen} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.$ QED
  BY $\langle 1 \rangle 1, \langle 1 \rangle 2$ DEF Spec

A complete proof would have proofs of the other two steps, each proof being either

- A BY statement, or
- A sequence of steps $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, ... with their proofs, each proof being either
  - A BY statement, or
  - A sequence of steps $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, ... with their proofs

$$\text{THEOREM } \textit{Invariance} \triangleq \textit{Spec} \Rightarrow \Box \textit{Inv}$$

$\langle 1 \rangle 1.\ \textit{Init} \Rightarrow \textit{Inv}$

$\langle 1 \rangle 2.\ \textit{Inv} \land [\textit{Next}]_{\textit{chosen}} \Rightarrow \textit{Inv}'$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF *Spec*

A complete proof would have proofs of the other two steps, each proof being either

- A BY statement, or
- A sequence of steps $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, ... with their proofs, each proof being either
  - A BY statement, or
  - A sequence of steps $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, ... with their proofs, each proof being either

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \mathit{Inv} \land [\mathit{Next}]_{chosen} \Rightarrow \mathit{Inv}'$

$\langle 1 \rangle 3.$ QED
  BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF Spec

A complete proof would have proofs of the other two steps, each proof being either

- A BY statement, or
- A sequence of steps $\langle 2 \rangle 1$, $\langle 2 \rangle 2$, ... with their proofs, each proof being either
  - A BY statement, or
  - A sequence of steps $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, ... with their proofs, each proof being either

    ⋮

$$\text{THEOREM } Invariance \triangleq Spec \Rightarrow \Box Inv$$

$\langle 1 \rangle 1.$ $Init \Rightarrow Inv$

$\langle 1 \rangle 2.$ $\boxed{Inv \land [Next]_{chosen} \Rightarrow Inv'}$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF $Spec$

An invariant satisfying this condition is called an **inductive** invariant.

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \boxed{\mathit{Inv} \land [\mathit{Next}]_{\mathit{chosen}} \Rightarrow \mathit{Inv}'}$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF Spec

An invariant satisfying this condition is called an **inductive** invariant.

Not all invariants are inductive.

$$\text{THEOREM } \mathit{Invariance} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{Inv}$$

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{Inv}$

$\langle 1 \rangle 2.\ \boxed{\mathit{Inv} \land [\mathit{Next}]_{chosen} \Rightarrow \mathit{Inv}'}$

$\langle 1 \rangle 3.$ QED

BY $\langle 1 \rangle 1$, $\langle 1 \rangle 2$ DEF Spec

An invariant satisfying this condition is called an **inductive** invariant.

Not all invariants are inductive.

To prove a non-inductive invariant, you must find an inductive invariant that implies it.

What a system does next depends only on its current state

What a system does next depends only on its current state, not what it did before.

What a system does next depends only on its current state, not what it did before.

It always does the right thing because it satisfies some inductive invariant.

What a system does next depends only on its current state, not what it did before.

It always does the right thing because it satisfies some inductive invariant.

To understand why the system works right, you need to find that inductive invariant.

THE VOTING ALGORITHM

How I Discovered Paxos

How I Discovered Paxos

Some people in the lab had built a distributed file system.

How I Discovered Paxos

Some people in the lab had built a distributed file system.

I thought what they claimed it accomplished was impossible.

How I Discovered Paxos

Some people in the lab had built a distributed file system.

I thought what they claimed it accomplished was impossible.

I tried to prove it was impossible; instead I discovered the Paxos algorithm.

I don't remember the thought process that led me to Paxos.

I don't remember the thought process that led me to Paxos.

But I knew that execution on computers sending messages to one another was an irrelevant detail.

I don’t remember the thought process that led me to Paxos.

But I knew that execution on computers sending messages to one another was an irrelevant detail.

I was thinking only about a set of processes and what they needed to know about one another.

I don't remember the thought process that led me to Paxos.

But I knew that execution on computers sending messages to one another was an irrelevant detail.

I was thinking only about a set of processes and what they needed to know about one another.

How they could get that information from messages was the easy part that came later.

I don't remember the thought process that led me to Paxos.

But I knew that execution on computers sending messages to one another was an irrelevant detail.

I was thinking only about a set of processes and what they needed to know about one another.

How they could get that information from messages was the easy part that came later.

What I had first was what, many years later, I described as the Voting algorithm that I'll show you next.

I don’t remember the thought process that led me to Paxos.

But I knew that execution on computers sending messages to one another was an irrelevant detail.

I was thinking only about a set of processes and what they needed to know about one another.

How they could get that information from messages was the easy part that came later.

What I had first was what, many years later, I described as the Voting algorithm that I’ll show you next.

But first . . .

How to Write an Algorithm Today

How to Write an Algorithm Today

1. Think up the algorithm.

How to Write an Algorithm Today

1. Think up the algorithm.
2. Formally specify it.

How to Write an Algorithm Today

1. Think up the algorithm.

2. Formally specify it.
   If you can't, go to 1.

How to Write an Algorithm Today

1. Think up the algorithm.
2. Formally specify it.
   If you can't, go to 1.
3. Model check it.

How to Write an Algorithm Today

1. Think up the algorithm.

2. Formally specify it.
   If you can't, go to 1.

3. Model check it.
   If model checking finds an error, go to 1 or 2.

How to Write an Algorithm Today

1. Think up the algorithm.

2. Formally specify it.
   If you can't, go to 1.

3. Model check it.
   If model checking finds an error, go to 1 or 2.
   If you're sufficiently convinced it's correct

How to Write an Algorithm Today

1. Think up the algorithm.

2. Formally specify it.
   If you can't, go to 1.

3. Model check it.
   If model checking finds an error, go to 1 or 2.
   If you're sufficiently convinced it's correct
   and you're not going to publish it, then stop.

How to Write an Algorithm Today

1. Think up the algorithm.

2. Formally specify it.
   If you can't, go to 1.

3. Model check it.
   If model checking finds an error, go to 1 or 2.
   If you're sufficiently convinced it's correct
   and you're not going to publish it, then stop.

4. Write a rigorous correctness proof & keep model checking.

How to Write an Algorithm Today

1. Think up the algorithm.

2. Formally specify it.
   If you can't, go to 1.

3. Model check it.
   If model checking finds an error, go to 1 or 2.
   If you're sufficiently convinced it's correct
   and you're not going to publish it, then stop.

4. Write a rigorous correctness proof & keep model checking.
   If you or the model checker find an error, go to 1 or 2.

How to Write an Algorithm Today

1. Think up the algorithm.

2. Formally specify it.
   If you can't, go to 1.

3. Model check it.
   If model checking finds an error, go to 1 or 2.
   If you're sufficiently convinced it's correct
   and you're not going to publish it, then stop.

4. Write a rigorous correctness proof & keep model checking.
   If you or the model checker find an error, go to 1 or 2.

   Probably not machine checked.

How to Write an Algorithm Today

1. Think up the algorithm.

2. Formally specify it.

   If you can't, go to 1.

3. Model check it.

   If model checking finds an error, go to 1 or 2.

   If you're sufficiently convinced it's correct and you're not going to publish it, then stop.

4. Write a rigorous correctness proof & keep model checking.

   If you or the model checker find an error, go to 1 or 2.

   Probably not machine checked.

   See *How to Write a 21st Century Proof*.

The Informal Idea Behind the Algorithm

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of processes.

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

Acceptors vote on which value to choose.

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

- Acceptors vote on which value to choose.
- A value is chosen if a majority of acceptors vote for it.

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

- Acceptors vote on which value to choose.
- A value is chosen if a majority of acceptors vote for it.

An obvious problem with this obvious approach:

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

- Acceptors vote on which value to choose.
- A value is chosen if a majority of acceptors vote for it.

An obvious problem with this obvious approach:

- Assume $2N + 1$ acceptors.

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

- Acceptors vote on which value to choose.
- A value is chosen if a majority of acceptors vote for it.

An obvious problem with this obvious approach:

- Assume $2N + 1$ acceptors.
- N vote for v_1.

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

- Acceptors vote on which value to choose.
- A value is chosen if a majority of acceptors vote for it.

An obvious problem with this obvious approach:

- Assume $2N + 1$ acceptors.
- N vote for v_1. N vote for v_2.

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

Acceptors vote on which value to choose.

A value is chosen if a majority of acceptors vote for it.

An obvious problem with this obvious approach:

Assume $2N + 1$ acceptors.

N vote for v_1. N vote for v_2.

The other acceptor votes and then fails.

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

Acceptors vote on which value to choose.

A value is chosen if a majority of acceptors vote for it.

An obvious problem with this obvious approach:

Assume $2N + 1$ acceptors.

N vote for v_1. N vote for v_2.

The other acceptor votes and then fails.

There's no way to tell if

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

Acceptors vote on which value to choose.

A value is chosen if a majority of acceptors vote for it.

An obvious problem with this obvious approach:

Assume $2N + 1$ acceptors.

N vote for v_1. N vote for v_2.

The other acceptor votes and then fails.

There's no way to tell if v_1 was chosen,

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

- Acceptors vote on which value to choose.
- A value is chosen if a majority of acceptors vote for it.

An obvious problem with this obvious approach:

- Assume $2N+1$ acceptors.
- N vote for v_1. N vote for v_2.
- The other acceptor votes and then fails.

There's no way to tell if v_1 was chosen, or v_2 was chosen,

The Informal Idea Behind the Algorithm

The algorithm is executed by a set of acceptors.

An obvious approach:

Acceptors vote on which value to choose.

A value is chosen if a majority of acceptors vote for it.

An obvious problem with this obvious approach:

Assume $2N+1$ acceptors.

N vote for v_1. N vote for v_2.

The other acceptor votes and then fails.

There's no way to tell if v_1 was chosen, or v_2 was chosen, or neither was chosen.

Here's the solution to this problem:

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

A value is chosen if a majority of acceptors vote for it in any ballot.

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

A value is chosen if a majority of acceptors vote for it in any ballot.

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

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

A value is chosen if a majority of acceptors vote for it in any ballot.

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

But getting acceptors to agree on a single value is the problem we're trying to solve!

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

A value is chosen if a majority of acceptors vote for it in any ballot.

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

But getting acceptors to agree on a single value is the problem we're trying to solve!

In Paxos, that will be done by having a leader tell them what value to vote for.

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

A value is chosen if a majority of acceptors vote for it in any ballot.

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

But getting acceptors to agree on a single value is the problem we're trying to solve!

In Paxos, that will be done by having a leader tell them what value to vote for.

But what if the leader fails?

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

A value is chosen if a majority of acceptors vote for it in any ballot.

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

But getting acceptors to agree on a single value is the problem we're trying to solve!

In Paxos, that will be done by having a leader tell them what value to vote for.

But what if the leader fails?

There'll be multiple leaders, with a single leader for each ballot.

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

A value is chosen if a majority of acceptors vote for it in any ballot.

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

But there's still this obvious problem:

Here's the solution to this problem:

- Acceptors vote in a sequence of ballots.
- A value is chosen if a majority of acceptors vote for it in any ballot.
- Don't allow different acceptors to vote for different values in the same ballot.

But there's still this obvious problem:

- N acceptors vote for v_1 in one ballot.

Here's the solution to this problem:

- Acceptors vote in a sequence of ballots.
- A value is chosen if a majority of acceptors vote for it in any ballot.
- Don't allow different acceptors to vote for different values in the same ballot.

But there's still this obvious problem:

- N acceptors vote for v_1 in one ballot.
- N acceptors vote for v_2 in another ballot.

Here's the solution to this problem:

- Acceptors vote in a sequence of ballots.
- A value is chosen if a majority of acceptors vote for it in any ballot.
- Don't allow different acceptors to vote for different values in the same ballot.

But there's still this obvious problem:

- N acceptors vote for v_1 in one ballot.
- N acceptors vote for v_2 in another ballot.
- The last acceptor votes and then fails.

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

A value is chosen if a majority of acceptors vote for it in any ballot.

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

But there's still this obvious problem:

N acceptors vote for v_1 in one ballot.

N acceptors vote for v_2 in another ballot.

The last acceptor votes and then fails.

Here's the very non-obvious solution:

Here's the solution to this problem:

- Acceptors vote in a sequence of ballots.
- A value is chosen if a majority of acceptors vote for it in any ballot.
- Don't allow different acceptors to vote for different values in the same ballot.

But there's still this obvious problem:

- N acceptors vote for v_1 in one ballot.
- N acceptors vote for v_2 in another ballot.
- The last acceptor votes and then fails.

Here's the very non-obvious solution:

- Allow an acceptor to vote for value v in ballot b only if

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

A value is chosen if a majority of acceptors vote for it in any ballot.

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

But there's still this obvious problem:

N acceptors vote for v_1 in one ballot.

N acceptors vote for v_2 in another ballot.

The last acceptor votes and then fails.

Here's the very non-obvious solution:

Allow an acceptor to vote for value v in ballot b only if no value other than v

Here's the solution to this problem:

Acceptors vote in a sequence of ballots.

A value is chosen if a majority of acceptors vote for it in any ballot.

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

But there's still this obvious problem:

N acceptors vote for v_1 in one ballot.

N acceptors vote for v_2 in another ballot.

The last acceptor votes and then fails.

Here's the very non-obvious solution:

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

Allow an acceptor to vote for value v in ballot b only if

no value other than v has been or ever will be chosen in any ballot numbered less than b.

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

How do we determine this?

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

How do we determine this?

Especially this?

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

How do we determine this?

Especially this?

If it was obvious, I wouldn't have won a Turing award.

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

How do we determine this?

Especially this?

If it was obvious, I wouldn't have won a Turing award.

Don't worry about it now.

Allow an acceptor to vote for value v in ballot b only if no value other than v has been or ever will be chosen in any ballot numbered less than b.

Define v **safe at** b to mean this.

Allow an acceptor to vote for value v in ballot b only if

v is safe at b.

Allow an acceptor to vote for value v in ballot b only if

v is safe at b.

Where v safe at b means:

No value other than v has been or ever will be chosen in any ballot numbered less than b.

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

Where v safe at b means:

No value other than v has been or ever will be chosen in any ballot numbered less than b.

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

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

Where v safe at b means:

No value other than v has been or ever will be chosen in any ballot numbered less than b.

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

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

Where v safe at b means:
No value other than v has been or ever will be chosen in any ballot numbered less than 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.

Where v safe at b means:
No value other than v has been or ever will be chosen in any ballot numbered less than 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.

Where v safe at b means:
No value other than v has been or ever will be chosen in any ballot numbered less than 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.

Where v safe at b means:
No value other than v has been or ever will be chosen in any ballot numbered less than b.

Theorem: These conditions imply at most one value can be chosen.

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.

Where v safe at b means:
No value other than v has been or ever will be chosen in any ballot numbered less than b.

Theorem: These conditions imply at most one value can be chosen.

Proof: Suppose $v1$ is chosen in ballot $b1$, and $v2$ is chosen in ballot $b2$.

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.

Where v safe at b means:
No value other than v has been or ever will be chosen in any ballot numbered less than b.

Theorem: These conditions imply at most one value can be chosen.

Proof: Suppose $v1$ is chosen in ballot $b1$, and $v2$ is chosen in ballot $b2$.

Case $b1 = b2$: Condition 1 implies $v1 = v2$.

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.

Where v safe at b means:
No value other than v has been or ever will be chosen in any ballot numbered less than b.

Theorem: These conditions imply at most one value can be chosen.

Proof: Suppose $v1$ is chosen in ballot $b1$, and $v2$ is chosen in ballot $b2$.

Case $b1 = b2$: Condition 1 implies $v1 = v2$.

Case $b1 \neq b2$: Condition 2 for b the larger of $b1$ and $b2$ implies no acceptor could have voted in that ballot unless $v1 = v2$.

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.

Where v safe at b means:
No value other than v has been or ever will be chosen in any ballot numbered less than b.

MODULE *Voting*

MODULE *Voting*

EXTENDS *Integers*

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*

the set of acceptors

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

a set of quorums

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

What's a quorum?

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

ASSUME $\land \forall Q \in Quorum : Q \subseteq Acceptor$

Every element of *Quorum* is a subset of *Acceptor*.

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

ASSUME $\land \forall Q \in Quorum : Q \subseteq Acceptor$

Every element of *Quorum* is a subset of *Acceptor* .

Every quorum is a set of acceptors.

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

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

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

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

Any two quorums have at least one common element.

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

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

Example: $Acceptor = \{a1, a2, a3, a4\}$

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

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

Example: $Acceptor = \{a1, a2, a3, a4\}$

$Quorum = \{\{a1, a2, a3\}, \{a1, a2, a4\}, \{a1, a3, a4\}, \{a2, a3, a4\}\}$

all majorities of acceptors

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

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

Example: $Acceptor = \{a1, a2, a3, a4\}$

$Quorum = \{\{a1, a2, a3\}, \{a1, a2, a4\},$
$\{a1, a3, a4\}, \{a2, a3, a4\}\}$

all majorities of acceptors

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

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

Example: $Acceptor = \{a1, a2, a3, a4\}$

$Quorum = \{\{a1, a2\}, \{a1, a2, a4\},$
$\{a1, a3\}, \{a2, a3, a4\}\}$

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *Value*, *Acceptor*, *Quorum*

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

Example: $Acceptor = \{a1, a2, a3, a4\}$

$Quorum = \{\{a1, a2\}, \{a1, a2, a4\}, \{a1, a3\}, \{a2, a3, a4\}\}$

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *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 set of all ballot numbers.

MODULE *Voting*

EXTENDS *Integers*

CONSTANT *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 set of all ballot numbers.

"Ballot" is easier to say than "natural number".

VARIABLES *votes*

VARIABLES $votes$

Describes what votes have been cast.

VARIABLES $votes$

Describes what votes have been cast.

The value of $votes$ is an array indexed by acceptors.

VARIABLES $votes$

Describes what votes have been cast.

The value of $votes$ is an array indexed by acceptors.

$votes[a]$ is the set of votes cast by acceptor a .

$\text{VARIABLES}\ votes$

Describes what votes have been cast.

The value of $votes$ is an array indexed by acceptors.

$votes[a]$ is the set of votes cast by acceptor a .

$\langle b, v \rangle \in votes[a]$ means a voted for value v in ballot b .

VARIABLES $votes$

Describes what votes have been cast.

The value of $votes$ is ~~an array indexed by acceptors~~ a function with domain $Acceptor$.

$votes[a]$ is the set of votes cast by acceptor a.

$\langle b, v \rangle \in votes[a]$ means a voted for value v in ballot b.

$\text{VARIABLES } votes$

$$TypeOK \triangleq$$
$$\land\ votes \in [Acceptor \rightarrow S]$$

$[Acceptor \rightarrow S]$ is the set of all functions with domain $Acceptor$ and values in S.

VARIABLES $votes$

$TypeOK \triangleq$

$\land\ votes \in [Acceptor \rightarrow \text{SUBSET } T]$

SUBSET T is the set of all subsets of T.

$\text{VARIABLES } votes$

$$TypeOK \triangleq$$
$$\land\ votes \in [Acceptor \rightarrow \text{SUBSET } (Ballot \times Value)]$$

$Ballot \times Value$ is the set of all $\langle$ballot, value$\rangle$ pairs.

$\text{VARIABLES } \mathit{votes}, \mathit{maxBal}$

$$\mathit{TypeOK} \triangleq$$
$$\land \mathit{votes} \in [\mathit{Acceptor} \rightarrow \text{SUBSET } (\mathit{Ballot} \times \mathit{Value})]$$

$\text{VARIABLES } votes, \ maxBal$

$$TypeOK \triangleq$$
$$\land\ votes \in [Acceptor \rightarrow \text{SUBSET } (Ballot \times Value)]$$
$$\land\ maxBal \in [Acceptor \rightarrow Ballot \cup \{-1\}]$$

$\text{VARIABLES } \mathit{votes}, \mathit{maxBal}$

$$
\begin{aligned}
&\mathit{TypeOK} \triangleq \\
&\quad \land \mathit{votes} \in [\mathit{Acceptor} \to \text{SUBSET } (\mathit{Ballot} \times \mathit{Value})] \\
&\quad \land \mathit{maxBal} \in [\mathit{Acceptor} \to \mathit{Ballot} \cup \{-1\}]
\end{aligned}
$$

Acceptor a will never vote in any ballot $< \mathit{maxBal}[a]$.

Some more definitions.

$$VotedFor(a, b, v) \triangleq \langle b, v \rangle \in votes[a]$$

$$VotedFor(a, b, v) \triangleq \langle b, v \rangle \in votes[a]$$

asserts that accepter a voted for value v in ballot b.

$$VotedFor(a, b, v) \triangleq \langle b, v \rangle \in votes[a]$$

$$ChosenAt(b, v) \triangleq$$
$$\quad \exists\, Q \in Quorum : \forall\, a \in Q : VotedFor(a, b, v)$$

$$VotedFor(a, b, v) \triangleq \langle b, v \rangle \in votes[a]$$

$$ChosenAt(b, v) \triangleq$$
$$\exists\, Q \in Quorum : \forall\, a \in Q : VotedFor(a, b, v)$$

asserts that every acceptor in some quorum voted for value v in ballot b.

$$VotedFor(a, b, v) \triangleq \langle b, v \rangle \in votes[a]$$

$$ChosenAt(b, v) \triangleq \exists Q \in Quorum : \forall a \in Q : VotedFor(a, b, v)$$

$$chosen \triangleq \{v \in Value : \exists b \in Ballot : ChosenAt(b, v)\}$$

$$VotedFor(a, b, v) \triangleq \langle b, v \rangle \in votes[a]$$

$$ChosenAt(b, v) \triangleq$$
$$\quad \exists\, Q \in Quorum : \forall\, a \in Q : VotedFor(a, b, v)$$

$$chosen \triangleq \{v \in Value : \exists\, b \in Ballot : ChosenAt(b, v)\}$$

defines the set of all chosen values

$$VotedFor(a, b, v) \triangleq \langle b, v \rangle \in votes[a]$$

$$ChosenAt(b, v) \triangleq$$
$$\quad \exists\, Q \in Quorum : \forall\, a \in Q : VotedFor(a, b, v)$$

$$chosen \triangleq \{v \in Value : \exists\, b \in Ballot : ChosenAt(b, v)\}$$

$$DidNotVoteAt(a, b) \triangleq \forall\, v \in Value : \neg VotedFor(a, b, v)$$

$$VotedFor(a, b, v) \triangleq \langle b, v \rangle \in votes[a]$$

$$ChosenAt(b, v) \triangleq \exists Q \in Quorum : \forall a \in Q : VotedFor(a, b, v)$$

$$chosen \triangleq \{v \in Value : \exists b \in Ballot : ChosenAt(b, v)\}$$

$$DidNotVoteAt(a, b) \triangleq \forall v \in Value : \neg VotedFor(a, b, v)$$

asserts that a did not vote in ballot b

The crucial definition.

$$ShowsSafeAt(Q,\ b,\ v)\ \triangleq$$

$ShowsSafeAt(Q, b, v) \triangleq$

An acceptor will vote for value v in ballot b only if this formula is true for some quorum Q.

$$ShowsSafeAt(Q,\ b,\ v)\ \triangleq$$
$$\quad \land\ \forall\, a \in Q : maxBal[a] \geq b$$

$$ShowsSafeAt(Q, b, v) \triangleq$$
$$\land \forall a \in Q : \boxed{maxBal[a]} \geq b$$

from now on, a can never vote in any ballot $c < maxBal[a]$

$$ShowsSafeAt(Q, b, v) \triangleq$$
$$\land \forall a \in Q : maxBal[a] \geq b$$

From now on, no acceptor in Q can ever vote in any ballot $< b$.

$$
\begin{aligned}
&ShowsSafeAt(Q, b, v) \triangleq \\
&\quad \land \forall a \in Q : maxBal[a] \geq b \\
&\quad \land \exists c \in -1 \,..\, (b-1) :
\end{aligned}
$$

From now on, no acceptor in Q can ever vote in any ballot $< b$.

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

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

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

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

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

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

$ShowsSafeAt(Q, b, v) \triangleq$

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

$ShowsSafeAt(Q, b, v) \triangleq$

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

Claim: If this is true for some quorum Q,
then v is safe at b.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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.

Where v safe at b means:
No value other than v has been or ever will be chosen in any ballot numbered less than b.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

We now assume Q is a quorum and prove v is safe at b.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

The Proof

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

⟨1⟩2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

⟨1⟩3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

⟨1⟩2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

⟨1⟩3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

⟨1⟩4. QED

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

⟨1⟩2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

⟨1⟩3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

⟨1⟩4. QED

Proof: No value other than v has been or can be chosen for any $d < b$

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c .

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

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 any ballot d only if v is safe at d .

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

⟨1⟩2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

⟨1⟩3. If $c \neq -1$ then no value other than v has been or will be chosen at c .

⟨1⟩4. QED

Proof: No value other than v has been or can be chosen for any $d < b$
by ⟨1⟩1 if $c < d < b$,

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

$\langle 1 \rangle$3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

$\langle 1 \rangle$4. QED

Proof: No value other than v has been or can be chosen for any $d < b$ by $\langle 1 \rangle$1 if $c < d < b$, by $\langle 1 \rangle$3 if $d = c$,

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

$\langle 1 \rangle$3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

$\langle 1 \rangle$4. QED

Proof: No value other than v has been or can be chosen for any $d < b$ by $\langle 1 \rangle$1 if $c < d < b$, by $\langle 1 \rangle$3 if $d = c$, and by $\langle 1 \rangle$2 if $d < c$.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

$\langle 1 \rangle$3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

$\langle 1 \rangle$4. QED

Proof: No value other than v has been or can be chosen for any $d < b$
by $\langle 1 \rangle$1 if $c < d < b$, by $\langle 1 \rangle$3 if $d = c$, and by $\langle 1 \rangle$2 if $d < c$.
By definition, this proves v is safe at b.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c .

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

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 any ballot d only if v is safe at d .

Where v safe at b means:
No value other than v has been or ever will be chosen in any ballot numbered less than b .

⟨1⟩2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

⟨1⟩3. If $c \neq -1$ then no value other than v has been or will be chosen at c .

⟨1⟩4. QED

Proof: No value other than v has been or can be chosen for any $d < b$
by ⟨1⟩1 if $c < d < b$, by ⟨1⟩3 if $d = c$, and by ⟨1⟩2 if $d < c$.
By definition, this proves v is safe at b .

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

Proof: No acceptor in Q has voted in ballot d.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

Proof: No acceptor in Q has voted in ballot d.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

Proof: No acceptor in Q has voted or can ever vote in ballot d.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

Proof: No acceptor in Q has voted or can ever vote in ballot d.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

Proof: No acceptor in Q has voted or can ever vote in ballot d.
A value can be chosen at d only if a quorum votes for it at d

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

Proof: No acceptor in Q has voted or can ever vote in ballot d.
A value can be chosen at d only if a quorum votes for it at d,
which is impossible.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

Proof: No acceptor in Q has voted or can ever vote in ballot d.
A value can be chosen at d only if a quorum votes for it at d,
which is impossible.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

Proof: An acceptor voted for v in ballot c,

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

Proof: An acceptor voted for v in ballot c,

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

Proof: An acceptor voted for v in ballot c, so v is safe at c.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c .

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

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 any ballot d only if v is safe at d .

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

⟨1⟩2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

Proof: An acceptor voted for v in ballot c , so v is safe at c .

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle 1$. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle 2$. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

Proof: An acceptor voted for v in ballot c, so v is safe at c. So $\langle 1 \rangle 2$ follows from the definition of *safe at*.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Where v safe at c means:
No value other than v has been or ever will be chosen in any ballot numbered less than c.

$\langle 1 \rangle 2$. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

Proof: An acceptor voted for v in ballot c, so v is safe at c.
So $\langle 1 \rangle 2$ follows from the definition of *safe at*.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

$\langle 1 \rangle$3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c .

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

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 any ballot d only if v is safe at d .

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

⟨1⟩2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

⟨1⟩3. If $c \neq -1$ then no value other than v has been or will be chosen at c .

Proof: An acceptor voted for v in ballot c ,

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle 1$. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle 2$. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

$\langle 1 \rangle 3$. If $c \neq -1$ then no value other than v has been or will be chosen at c.

Proof: An acceptor voted for v in ballot c,

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c .

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

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 any ballot d only if v is safe at d .

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

⟨1⟩2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

⟨1⟩3. If $c \neq -1$ then no value other than v has been or will be chosen at c .

Proof: An acceptor voted for v in ballot c , so no acceptor has voted or will vote for any value other than v in ballot c,

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c .

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

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 any ballot d only if
   v is safe at d .

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

⟨1⟩2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

⟨1⟩3. If $c \neq -1$ then no value other than v has been or will be chosen at c .

Proof: An acceptor voted for v in ballot c , so no acceptor has voted or will vote for any value other than v in ballot c,

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

⟨1⟩1. No value has been or ever will be chosen at d if $c < d < b$.

⟨1⟩2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

⟨1⟩3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

Proof: An acceptor voted for v in ballot c, so no acceptor has voted or will vote for any value other than v in ballot c, proving ⟨1⟩3.

From now on, no acceptor in Q can ever vote in any ballot $< b$.

For some c with either $c = -1$ or c is a ballot number $< b$:

Either $c = -1$ or some acceptor in Q voted for v in ballot c.

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

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 any ballot d only if v is safe at d.

Any two quorums have at least one common element.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

$\langle 1 \rangle$3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

$\langle 1 \rangle$4. QED

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

$\langle 1 \rangle$3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

$\langle 1 \rangle$4. QED

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

$\langle 1 \rangle$3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

$\langle 1 \rangle$4. QED

This completes the proof that:

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

$\langle 1 \rangle$3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

$\langle 1 \rangle$4. QED

This completes the proof that:

$ShowsSafeAt(Q, b, v)$ for a quorum Q implies v is safe at b.

$\langle 1 \rangle$1. No value has been or ever will be chosen at d if $c < d < b$.

$\langle 1 \rangle$2. If $c \neq -1$ then no value other than v has been or will be chosen at d for $d < c$.

$\langle 1 \rangle$3. If $c \neq -1$ then no value other than v has been or will be chosen at c.

$\langle 1 \rangle$4. QED

This completes the proof that:

$ShowsSafeAt(Q, b, v)$ for a quorum Q implies v is safe at b.

Which is the heart of the Paxos algorithm.

$ShowsSafeAt(Q, b, v)$ for a quorum Q implies v is safe at b.

$ShowsSafeAt(Q, b, v)$ for a quorum Q implies v is safe at b.

The $Voting$ module contains a theorem that is a precise statement of this result.

The Spec

The Spec

$$Init \triangleq$$

The Spec

$$Init \triangleq \land\ votes \quad = [a \in Acceptor \mapsto \{\}]$$

The Spec

$$Init \triangleq \land\ votes = \boxed{[a \in Acceptor \mapsto \{\}]}$$

The Spec

$$Init \triangleq \land\ votes = \boxed{[a \in Acceptor \mapsto \{\}]}$$

$[x \in S \mapsto exp(x)]$ is the function f with domain S such that $f[x] = exp(x)$ for all $x \in S$.

The Spec

$$Init \triangleq \land\ votes = [a \in Acceptor \mapsto \{\}]$$

$[x \in S \mapsto exp(x)]$ is the function f with domain S such that $f[x] = exp(x)$ for all $x \in S$.

So initially, $votes[a] = \{\}$ for every acceptor a.

The Spec

$$Init \triangleq \begin{array}{l} \land\ votes\ \ \ \ = [a \in Acceptor \mapsto \{\}] \\ \land\ maxBal = [a \in Acceptor \mapsto\ -1] \end{array}$$

Initially, $maxBal[a] = -1$ for every acceptor a .

$$IncreaseMaxBal(a, b) \triangleq \cdots$$

$IncreaseMaxBal(a, b) \triangleq \;\ldots$

Describes a step in which acceptor a increases the value of $maxBal[a]$ to b.

$$IncreaseMaxBal(a, b) \triangleq \ \ldots$$

$$IncreaseMaxBal(a, b) \triangleq \cdots$$

$$VoteFor(a, b, v) \triangleq \cdots$$

$IncreaseMaxBal(a, b) \triangleq \cdots$

$VoteFor(a, b, v) \triangleq \cdots$

Describes a step in which acceptor a votes for v in ballot b.

$$IncreaseMaxBal(a, b) \triangleq \cdots$$

$$VoteFor(a, b, v) \triangleq \cdots$$

$$Next \triangleq$$

$IncreaseMaxBal(a, b) \triangleq \cdots$

$VoteFor(a, b, v) \triangleq \cdots$

$Next \triangleq$

The condition any step of the algorithm must satisfy.

$IncreaseMaxBal(a, b) \triangleq \ \cdots$

$VoteFor(a, b, v) \triangleq \ \cdots$

$Next \ \triangleq \ \exists\, a \in Acceptor, b \in Ballot :$

$IncreaseMaxBal(a, b) \triangleq \cdots$

$VoteFor(a, b, v) \triangleq \cdots$

$Next \triangleq \exists\, a \in Acceptor, b \in Ballot :$

For some acceptor a and some ballot b :

$$IncreaseMaxBal(a, b) \triangleq \cdots$$

$$VoteFor(a, b, v) \triangleq \cdots$$

$$\begin{aligned} Next \triangleq \; & \exists\, a \in Acceptor,\, b \in Ballot : \\ & \quad \lor\, IncreaseMaxBal(a, b) \end{aligned}$$

$IncreaseMaxBal(a, b) \triangleq \cdots$

$VoteFor(a, b, v) \triangleq \cdots$

$$
\begin{aligned}
Next \triangleq \ & \exists\, a \in Acceptor,\ b \in Ballot : \\
& \quad \lor\ IncreaseMaxBal(a, b)
\end{aligned}
$$

either a increases $maxBal[a]$ to b

$$IncreaseMaxBal(a, b) \triangleq \cdots$$

$$VoteFor(a, b, v) \triangleq \cdots$$

$$\begin{aligned} Next \triangleq\ & \exists\, a \in Acceptor,\ b \in Ballot : \\ & \quad \lor\ IncreaseMaxBal(a, b) \\ & \quad \lor\ \exists\, v \in Value : VoteFor(a, b, v) \end{aligned}$$

$$IncreaseMaxBal(a, b) \triangleq \ldots$$

$$VoteFor(a, b, v) \triangleq \ldots$$

$$
\begin{aligned}
Next \triangleq\ & \exists\, a \in Acceptor,\ b \in Ballot : \\
& \quad \lor\ IncreaseMaxBal(a, b) \\
& \quad \lor\ \exists\, v \in Value : VoteFor(a, b, v)
\end{aligned}
$$

or a votes for some value v in ballot b.

$$IncreaseMaxBal(a, b) \triangleq \ \ldots$$

$$VoteFor(a, b, v) \triangleq \ \ldots$$

$$\begin{aligned} Next \ \triangleq \ & \exists\, a \in Acceptor,\ b \in Ballot : \\ & \quad \lor\ IncreaseMaxBal(a, b) \\ & \quad \lor\ \exists\, v \in Value : VoteFor(a, b, v) \end{aligned}$$

$IncreaseMaxBal(a, b) \triangleq$

$IncreaseMaxBal(a, b) \triangleq$

Describes a step in which acceptor a increases the value of $maxBal[a]$ to b.

$IncreaseMaxBal(a, b) \triangleq$

$$IncreaseMaxBal(a,\ b) \triangleq$$
$$\land\ b > maxBal[a]$$

$IncreaseMaxBal(a, b) \triangleq$

$\land\ b > maxBal[a]$

$b >$ current value of $maxBal[a]$

$IncreaseMaxBal(a, b) \triangleq$

$\land b > maxBal[a]$

$b >$ current value of $maxBal[a]$

(an enabling condition)

$$IncreaseMaxBal(a, b) \triangleq$$
$$\land\ b > maxBal[a]$$

$$IncreaseMaxBal(a, b) \triangleq$$
$$\land\ b > maxBal[a]$$
$$\land\ maxBal[a]' = b$$

$IncreaseMaxBal(a, b) \triangleq$

$\land b > maxBal[a]$

$\land maxBal[a]' = b$

and the value of $maxBal[a]$

in the next state is b .

$$
\begin{aligned}
&IncreaseMaxBal(a,\ b) \ \triangleq \\
&\qquad \land\ b > maxBal[a] \\
&\qquad \land\ maxBal[a]' = b \\
&\qquad \land\ \text{UNCHANGED}\ votes
\end{aligned}
$$

$$IncreaseMaxBal(a,\ b)\ \triangleq$$

$$\land\ b > maxBal[a]$$

$$\land\ maxBal[a]' = b$$

$$\land\ \text{UNCHANGED}\ \ votes$$

an abbreviation for $votes' = votes$

$$
\begin{aligned}
&IncreaseMaxBal(a,\ b) \ \triangleq \\
&\quad \land\ b > maxBal[a] \\
&\quad \land\ maxBal[a]' = b \\
&\quad \land\ \text{UNCHANGED}\ \ votes
\end{aligned}
$$

and the value of $votes$ is unchanged.

$$
\begin{aligned}
&IncreaseMaxBal(a,\ b) \ \triangleq \\
&\qquad \land\ b > maxBal[a] \\
&\qquad \land\ maxBal[a]' = b \\
&\qquad \land\ \text{UNCHANGED}\ \ votes
\end{aligned}
$$

$$
\begin{aligned}
&IncreaseMaxBal(a,\ b)\ \triangleq \\
&\qquad \land\ b > maxBal[a] \\
&\qquad \land\ maxBal[a]' = b \\
&\qquad \land\ \text{UNCHANGED}\ \ votes
\end{aligned}
$$

What's wrong with this?

$\wedge\ maxBal[a]' = b$

$\wedge\ maxBal[a]' = b$

and the value of $maxBal[a]$ in the next state is b.

$\wedge \; maxBal[a]' = b$

and the value of $maxBal[a]$
in the next state is b.

What about the value of $maxBal[a2]$
in the next state for an acceptor $a2 \neq a$?

$\wedge\ maxBal[a]' = b$

and the value of $maxBal[a]$
in the next state is b .

What about the value of $maxBal[a2]$
in the next state for an acceptor $a2 \neq a$?

What about the domain of $maxBal$ in the next state?

$\land \ maxBal[a]' = b$

and the value of $maxBal[a]$
in the next state is b .

This is all it says.

$$\land\ maxBal[a]' = b$$

$\wedge\ maxBal' =$

$$\land\ maxBal' = [\,x \in Acceptor \mapsto$$

$$\land\ maxBal' = [\,x \in Acceptor \mapsto \\ \text{IF } x = a \text{ THEN } b \text{ ELSE } maxBal[x]\,]$$

$$\land\ maxBal' = [\,x \in Acceptor \mapsto \\ \quad \text{IF } x = a \text{ THEN } b \text{ ELSE } maxBal[x]\,]$$

An expression like this will appear whenever a step changes “one element of an array”.

$$\land\ maxBal' = [\,x \in Acceptor \mapsto \\ \quad \text{IF } x = a \text{ THEN } b \text{ ELSE } maxBal[x]\,]$$

An expression like this will appear whenever a step changes “one element of an array”.

So we want an abbreviation for it.

$$\land\ maxBal' = [maxBal\ \text{EXCEPT}\ ![a] = b]$$

$$\land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = b]$$

It's a terrible notation.

$\land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = b]$

It’s a terrible notation.

But it’s better than any other that I’ve seen.

$$\land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = b]$$

$$
\begin{aligned}
&IncreaseMaxBal(a,\ b) \triangleq \\
&\quad \land\ b > maxBal[a] \\
&\quad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = b] \\
&\quad \land\ \text{UNCHANGED } votes
\end{aligned}
$$

$$VoteFor(a, b, v) \triangleq$$

$VoteFor(a, b, v) \triangleq$

Describes a step in which acceptor a votes for v in ballot b.

$$VoteFor(a, b, v) \triangleq$$

$$\land \quad maxBal[a] \leq b$$

$$VoteFor(a,\ b,\ v)\ \triangleq$$
$$\land\quad maxBal[a] \leq b$$

Enabling condition that assures a doesn't vote in a ballot $b < maxBal[a]$.

$$VoteFor(a, b, v) \triangleq$$
$$\land \quad maxBal[a] \leq b$$

$$
\begin{aligned}
&VoteFor(a,\ b,\ v)\ \triangleq \\
&\quad \land\quad maxBal[a] \leq b \\
&\quad \land\quad \forall\, vt \in votes[a] : vt[1] \neq b
\end{aligned}
$$

$$VoteFor(a, b, v) \triangleq$$
$$\land \quad maxBal[a] \leq b$$
$$\land \quad \forall vt \in \boxed{votes[a]} : vt[1] \neq b$$

A set of $\langle ballot, value \rangle$ pairs.

$$VoteFor(a,\ b,\ v)\ \triangleq$$
$$\land\ \ maxBal[a] \leq b$$
$$\land\ \ \forall\, vt \in \boxed{votes[a]} : vt[1] \neq b$$

A set of $\langle ballot, value \rangle$ pairs.

A pair is a function, with $\langle x, y \rangle$[1] $= x$.

$VoteFor(a, b, v) \triangleq$

$\land \; maxBal[a] \leq b$

$\land \; \forall vt \in votes[a] : vt[1] \neq b$

A set of $\langle ballot, value \rangle$ pairs.

A pair is a function, with $\langle x, y \rangle[1] = x$.

Asserts that a hasn't already voted in ballot b.

$$
\begin{aligned}
&VoteFor(a,\ b,\ v)\ \triangleq \\
&\quad \land\ \ maxBal[a] \leq b \\
&\quad \land\ \ \forall\, vt \in votes[a] : vt[1] \neq b
\end{aligned}
$$

$$
\begin{aligned}
&VoteFor(a,\ b,\ v)\ \triangleq \\
&\quad \land\ \ maxBal[a] \leq b \\
&\quad \land\ \ \forall\, vt \in votes[a] : vt[1] \neq b \\
&\quad \land\ \ \forall\, c \in Acceptor \setminus \{a\} :
\end{aligned}
$$

$$
\begin{aligned}
&VoteFor(a, b, v) \triangleq \\
&\quad \land \ maxBal[a] \leq b \\
&\quad \land \ \forall vt \in votes[a] : vt[1] \neq b \\
&\quad \land \ \forall c \in Acceptor \setminus \{a\} :
\end{aligned}
$$

For every acceptor c different from a

$$
\begin{aligned}
&VoteFor(a,\ b,\ v)\ \triangleq \\
&\quad \land\ \ maxBal[a] \leq b \\
&\quad \land\ \ \forall\, vt \in votes[a] : vt[1] \neq b \\
&\quad \land\ \ \forall\, c \in Acceptor \setminus \{a\} : \\
&\qquad\quad \forall\, vt \in votes[c] :
\end{aligned}
$$

For every acceptor c different from a

$$
\begin{aligned}
VoteFor(a, b, v) \triangleq \\
\quad \land \;& maxBal[a] \leq b \\
\quad \land \;& \forall vt \in votes[a] : vt[1] \neq b \\
\quad \land \;& \forall c \in Acceptor \setminus \{a\} : \\
& \quad \forall vt \in votes[c] :
\end{aligned}
$$

For every acceptor c different from a

and for every vote vt of c ,

$$
\begin{array}{l}
VoteFor(a,\ b,\ v)\ \triangleq \\
\quad \land\ \ maxBal[a] \leq b \\
\quad \land\ \ \forall\, vt \in votes[a] : vt[1] \neq b \\
\quad \land\ \ \forall\, c \in Acceptor \setminus \{a\} : \\
\qquad\quad \forall\, vt \in votes[c] : (vt[1] = b) \Rightarrow
\end{array}
$$

For every acceptor c different from a

and for every vote vt of c ,

$$
\begin{aligned}
&VoteFor(a,\ b,\ v)\ \triangleq\\
&\quad \land\ \ maxBal[a] \leq b\\
&\quad \land\ \ \forall\, vt \in votes[a] : vt[1] \neq b\\
&\quad \land\ \ \forall\, c \in Acceptor \setminus \{a\} :\\
&\qquad\quad \forall\, vt \in votes[c] : (vt[1] = b) \Rightarrow
\end{aligned}
$$

For every acceptor c different from a

and for every vote vt of c, if vt is a vote in ballot b

$$
\begin{aligned}
&VoteFor(a,\ b,\ v)\ \triangleq \\
&\quad \land\ \ maxBal[a] \leq b \\
&\quad \land\ \ \forall\, vt \in votes[a] : vt[1] \neq b \\
&\quad \land\ \ \forall\, c \in Acceptor \setminus \{a\} : \\
&\qquad\qquad \forall\, vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v)
\end{aligned}
$$

For every acceptor c different from a

and for every vote vt of c, if vt is a vote in ballot b

$$
\begin{aligned}
&VoteFor(a,\ b,\ v)\ \triangleq \\
&\quad \land\ \ maxBal[a] \leq b \\
&\quad \land\ \ \forall\, vt \in votes[a] : vt[1] \neq b \\
&\quad \land\ \ \forall\, c \in Acceptor \setminus \{a\} : \\
&\qquad\quad \forall\, vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v)
\end{aligned}
$$

For every acceptor c different from a

and for every vote vt of c, if vt is a vote in ballot b

then vt is a vote for v.

$$
\begin{aligned}
&VoteFor(a, b, v) \triangleq \\
&\quad \land \ maxBal[a] \leq b \\
&\quad \land \ \forall vt \in votes[a] : vt[1] \neq b \\
&\quad \land \ \forall c \in Acceptor \setminus \{a\} : \\
&\qquad\quad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v)
\end{aligned}
$$

Any vote already cast in ballot b is for value v.

$$
\begin{array}{l}
\mathit{VoteFor}(a,\ b,\ v)\ \triangleq \\
\qquad \land\ \ \mathit{maxBal}[a] \leq b \\
\qquad \land\ \ \forall\, vt \in \mathit{votes}[a] : vt[1] \neq b \\
\qquad \land\ \ \forall\, c \in \mathit{Acceptor} \setminus \{a\} : \\
\qquad\qquad \forall\, vt \in \mathit{votes}[c] : (vt[1] = b) \Rightarrow (vt[2] = v)
\end{array}
$$

$$
\begin{array}{l}
VoteFor(a, b, v) \triangleq \\
\quad \land \; maxBal[a] \leq b \\
\quad \land \; \forall vt \in votes[a] : vt[1] \neq b \\
\quad \land \; \forall c \in Acceptor \setminus \{a\} : \\
\quad \quad \quad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
\quad \land \; \exists Q \in Quorum : ShowsSafeAt(Q, b, v)
\end{array}
$$

$$
\begin{aligned}
&VoteFor(a,\ b,\ v)\ \triangleq \\
&\quad \land\ \ maxBal[a] \leq b \\
&\quad \land\ \ \forall\, vt \in votes[a] : vt[1] \neq b \\
&\quad \land\ \ \forall\, c \in Acceptor \setminus \{a\} : \\
&\qquad\quad \forall\, vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
&\quad \land\ \ \exists\, Q \in Quorum : ShowsSafeAt(Q,\ b,\ v)
\end{aligned}
$$

$ShowsSafeAt(Q, b, v)$ is true for some quorum Q ,

$$
\begin{array}{l}
VoteFor(a, b, v) \triangleq \\
\quad \land \; maxBal[a] \leq b \\
\quad \land \; \forall vt \in votes[a] : vt[1] \neq b \\
\quad \land \; \forall c \in Acceptor \setminus \{a\} : \\
\qquad\quad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
\quad \land \; \exists Q \in Quorum : ShowsSafeAt(Q, b, v)
\end{array}
$$

$ShowsSafeAt(Q, b, v)$ is true for some quorum Q,

which implies v is safe at b.

$$
\begin{aligned}
&VoteFor(a, b, v) \triangleq \\
&\quad \land \ maxBal[a] \leq b \\
&\quad \land \ \forall vt \in votes[a] : vt[1] \neq b \\
&\quad \land \ \forall c \in Acceptor \setminus \{a\} : \\
&\qquad\quad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
&\quad \land \ \exists Q \in Quorum : ShowsSafeAt(Q, b, v)
\end{aligned}
$$

$$
\begin{aligned}
&VoteFor(a, b, v) \triangleq \\
&\quad \land\ maxBal[a] \leq b \\
&\quad \land\ \forall vt \in votes[a] : vt[1] \neq b \\
&\quad \land\ \forall c \in Acceptor \setminus \{a\} : \\
&\qquad\quad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
&\quad \land\ \exists Q \in Quorum : ShowsSafeAt(Q, b, v) \\
&\quad \land\ votes' = [votes \text{ EXCEPT } ![a] = votes[a] \cup \{\langle b, v\rangle\}]
\end{aligned}
$$

$$
\begin{array}{l}
VoteFor(a, b, v) \triangleq \\
\quad \land \; maxBal[a] \leq b \\
\quad \land \; \forall vt \in votes[a] : vt[1] \neq b \\
\quad \land \; \forall c \in Acceptor \setminus \{a\} : \\
\quad \qquad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
\quad \land \; \exists Q \in Quorum : ShowsSafeAt(Q, b, v) \\
\quad \land \; votes' = [votes \text{ EXCEPT } ![a] = votes[a] \cup \{\langle b, v\rangle\}]
\end{array}
$$

$votes[a]$ is set to $votes[a] \cup \{\langle b, v \rangle\}$,

$$
\begin{aligned}
&VoteFor(a, b, v) \triangleq \\
&\quad \land\ maxBal[a] \leq b \\
&\quad \land\ \forall vt \in votes[a] : vt[1] \neq b \\
&\quad \land\ \forall c \in Acceptor \setminus \{a\} : \\
&\quad\qquad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
&\quad \land\ \exists Q \in Quorum : ShowsSafeAt(Q, b, v) \\
&\quad \land\ votes' = [votes \text{ EXCEPT } ![a] = votes[a] \cup \{\langle b, v \rangle\}]
\end{aligned}
$$

$votes[a]$ is set to $votes[a] \cup \{\langle b, v \rangle\}$,

meaning a votes for v in ballot b.

$$
\begin{aligned}
&VoteFor(a, b, v) \triangleq \\
&\quad \land\ maxBal[a] \leq b \\
&\quad \land\ \forall vt \in votes[a] : vt[1] \neq b \\
&\quad \land\ \forall c \in Acceptor \setminus \{a\} : \\
&\qquad\quad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
&\quad \land\ \exists Q \in Quorum : ShowsSafeAt(Q, b, v) \\
&\quad \land\ votes' = [votes \text{ EXCEPT } ![a] = votes[a] \cup \{\langle b, v\rangle\}]
\end{aligned}
$$

$$
\begin{aligned}
&VoteFor(a, b, v) \triangleq \\
&\quad \land\ maxBal[a] \leq b \\
&\quad \land\ \forall vt \in votes[a] : vt[1] \neq b \\
&\quad \land\ \forall c \in Acceptor \setminus \{a\} : \\
&\quad\qquad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
&\quad \land\ \exists Q \in Quorum : ShowsSafeAt(Q, b, v) \\
&\quad \land\ votes' = [votes \text{ EXCEPT } ![a] = votes[a] \cup \{\langle b, v \rangle\}] \\
&\quad \land\ maxBal' = [maxBal \text{ EXCEPT } ![a] = b]
\end{aligned}
$$

$$
\begin{aligned}
VoteFor(a, b, v) \;\triangleq\; & \\
\land\;\; & maxBal[a] \leq b \\
\land\;\; & \forall vt \in votes[a] : vt[1] \neq b \\
\land\;\; & \forall c \in Acceptor \setminus \{a\} : \\
& \quad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
\land\;\; & \exists Q \in Quorum : ShowsSafeAt(Q, b, v) \\
\land\;\; & votes' = [votes \text{ EXCEPT } ![a] = votes[a] \cup \{\langle b, v\rangle\}] \\
\land\;\; & maxBal' = [maxBal \text{ EXCEPT } ![a] = b]
\end{aligned}
$$

$maxBal[a]$ is set to b ,

$$
\begin{aligned}
&VoteFor(a, b, v) \triangleq \\
&\quad \land \; maxBal[a] \leq b \\
&\quad \land \; \forall vt \in votes[a] : vt[1] \neq b \\
&\quad \land \; \forall c \in Acceptor \setminus \{a\} : \\
&\qquad\quad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
&\quad \land \; \exists Q \in Quorum : ShowsSafeAt(Q, b, v) \\
&\quad \land \; votes' = [votes \text{ EXCEPT } ![a] = votes[a] \cup \{\langle b, v\rangle\}] \\
&\quad \land \; maxBal' = [maxBal \text{ EXCEPT } ![a] = b]
\end{aligned}
$$

$maxBal[a]$ is set to b,

announcing that a will never again vote in a ballot $< b$.

$$
\begin{array}{l}
VoteFor(a, b, v) \triangleq \\
\quad \land \ maxBal[a] \leq b \\
\quad \land \ \forall vt \in votes[a] : vt[1] \neq b \\
\quad \land \ \forall c \in Acceptor \setminus \{a\} : \\
\qquad \quad \forall vt \in votes[c] : (vt[1] = b) \Rightarrow (vt[2] = v) \\
\quad \land \ \exists Q \in Quorum : ShowsSafeAt(Q, b, v) \\
\quad \land \ votes' = [votes \text{ EXCEPT } ![a] = votes[a] \cup \{\langle b, v \rangle\}] \\
\quad \land \ maxBal' = [maxBal \text{ EXCEPT } ![a] = b]
\end{array}
$$

The Complete Definition of $Spec$

The Complete Definition of $Spec$

$$Init \triangleq \ldots$$

The Complete Definition of $Spec$

$$Init \triangleq \ldots$$

$$IncreaseMaxBal(a,\ b) \triangleq \ldots$$

The Complete Definition of $Spec$

$$Init \triangleq \ldots$$

$$IncreaseMaxBal(a, b) \triangleq \ldots$$

$$VoteFor(a, b, v) \triangleq \ldots$$

The Complete Definition of *Spec*

$$Init \triangleq \ldots$$

$$IncreaseMaxBal(a, b) \triangleq \ldots$$

$$VoteFor(a, b, v) \triangleq \ldots$$

$$\begin{aligned} Next \triangleq \; & \exists a \in Acceptor, b \in Ballot : \\ & \quad \lor IncreaseMaxBal(a, b) \\ & \quad \lor \exists v \in Value : VoteFor(a, b, v) \end{aligned}$$

The Complete Definition of Spec

$$\mathit{Init} \triangleq \ldots$$

$$\mathit{IncreaseMaxBal}(a, b) \triangleq \ldots$$

$$\mathit{VoteFor}(a, b, v) \triangleq \ldots$$

$$\begin{aligned} \mathit{Next} \triangleq\ & \exists\, a \in \mathit{Acceptor}, b \in \mathit{Ballot} : \\ & \quad \lor\ \mathit{IncreaseMaxBal}(a, b) \\ & \quad \lor\ \exists\, v \in \mathit{Value} : \mathit{VoteFor}(a, b, v) \end{aligned}$$

$$\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\langle \mathit{votes},\, \mathit{maxBal} \rangle}$$

The Complete Definition of $Spec$

$$Init \triangleq \ldots$$

$$IncreaseMaxBal(a, b) \triangleq \ldots$$

$$VoteFor(a, b, v) \triangleq \ldots$$

$$\begin{aligned} Next \triangleq\ & \exists\, a \in Acceptor,\ b \in Ballot : \\ & \quad \lor IncreaseMaxBal(a, b) \\ & \quad \lor \exists\, v \in Value : VoteFor(a, b, v) \end{aligned}$$

$$Spec \triangleq Init \land \Box[Next]_{\langle votes,\, maxBal \rangle}$$