The Voting Algorithm Implements Consensus

Defining $chosen$

Defining $chosen$

To say what it means for the $Voting$ algorithm to implement the $Consensus$ spec,

Defining $chosen$

To say what it means for the $Voting$ algorithm to implement the $Consensus$ spec, we have to say what it means for the algorithm to choose a value.

Defining $chosen$

To say what it means for the $Voting$ algorithm to implement the $Consensus$ spec, we have to say what it means for the algorithm to choose a value.

More precisely, we have to say how the algorithm implements the variable $chosen$ of the spec.

Defining $chosen$

To say what it means for the $Voting$ algorithm to implement the $Consensus$ spec, we have to say what it means for the algorithm to choose a value.

More precisely, we have to say how the algorithm implements the variable $chosen$ of the spec.

This is stated in the definition of the expression $chosen$ in module $Voting$.

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

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

True iff acceptor a has voted for value v in ballot b.

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

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

$$ChosenAt(b, v) \triangleq$$

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

$$\begin{array}{l} ChosenAt(b, v) \triangleq \\ \quad \exists\, Q \in Quorum : \end{array}$$

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

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

True iff there is a quorum

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

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

True iff there is a quorum

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

$$\begin{aligned} &ChosenAt(b, v) \triangleq \\ &\quad \exists\, Q \in Quorum : \forall\, a \in Q : \end{aligned}$$

True iff there is a quorum all of whose acceptors

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

$$\begin{aligned}
&ChosenAt(b,\ v)\ \triangleq\\
&\quad \exists\, Q \in Quorum : \forall\, a \in Q : VotedFor(a,\ b,\ v)
\end{aligned}$$

True iff there is a quorum all of whose acceptors

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

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

True iff there is a quorum all of whose acceptors voted for value v in ballot b.

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

$$\begin{aligned} &ChosenAt(b, v) \triangleq \\ &\quad \exists\, Q \in Quorum : \forall\, a \in Q : VotedFor(a, b, v) \end{aligned}$$

True iff there is a quorum all of whose acceptors voted for value v in ballot b.

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

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

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

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

$$chosen \triangleq$$

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

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

$$chosen \triangleq \{v \in Value :$$

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

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

$$chosen \triangleq \{v \in Value :$$

The set of all values v

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

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

$$chosen \triangleq \{v \in Value : \exists\, b \in Ballot :$$

The set of all values v

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

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

$$chosen \triangleq \{v \in Value : \exists b \in Ballot :$$

The set of all values v for which there is a ballot b

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

$$\begin{aligned} &ChosenAt(b, v) \triangleq \\ &\quad \exists\, Q \in Quorum : \forall\, a \in Q : VotedFor(a, b, v) \end{aligned}$$

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

The set of all values v for which there is a ballot b

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

$$\begin{aligned} &ChosenAt(b, v) \triangleq \\ &\quad \exists\, Q \in Quorum : \forall\, a \in Q : VotedFor(a, b, v) \end{aligned}$$

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

The set of all values v for which there is a ballot b
such that v is chosen in ballot b .

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

$$\begin{aligned} &ChosenAt(b, v) \triangleq \\ &\quad \exists\, Q \in Quorum : \forall\, a \in Q : VotedFor(a, b, v) \end{aligned}$$

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

The set of all values v for which there is a ballot b such that v is chosen in ballot b.

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

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

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

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

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

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

$\{var \in Set :$ formula containing $var\}$

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

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

$$chosen \triangleq \boxed{\{v \in Value : \qquad\qquad\qquad\qquad\qquad\qquad \}}$$

$$\{var \in Set : \qquad\qquad\qquad\qquad \}$$

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

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

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

$$\{ \qquad\qquad\qquad \text{formula containing } var\}$$

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

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

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

$\{var \in Set$: formula containing $var\}$

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

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

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

What it means for the $Voting$ algorithm to implement the $Consensus$ spec with this definition of $chosen$.

What it means for the $Voting$ algorithm to implement the $Consensus$ spec with this definition of $chosen$.

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \rightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \rightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \rightarrow \cdots$$

Consider any behavior of the $Voting$ algorithm.

What it means for the $Voting$ algorithm to implement the $Consensus$ spec with this definition of $chosen$.

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$chosen = \cdots$$

Consider any behavior of the $Voting$ algorithm.

Consider the value of $chosen$ in each state.

What it means for the $Voting$ algorithm to implement the $Consensus$ spec with this definition of $chosen$.

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$chosen = \cdots \qquad\qquad chosen = \cdots$$

Consider any behavior of the $Voting$ algorithm.

Consider the value of $chosen$ in each state.

What it means for the $Voting$ algorithm to implement the $Consensus$ spec with this definition of $chosen$.

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$chosen = \cdots \qquad\qquad chosen = \cdots \qquad\qquad chosen = \cdots$$

Consider any behavior of the $Voting$ algorithm.

Consider the value of $chosen$ in each state.

What it means for the $Voting$ algorithm to implement the $Consensus$ spec with this definition of $chosen$.

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \ldots$$

$$chosen = \cdots \qquad chosen = \cdots \qquad chosen = \cdots \qquad \ldots$$

Consider any behavior of the $Voting$ algorithm.

Consider the value of $chosen$ in each state.

What it means for the $Voting$ algorithm to implement the $Consensus$ spec with this definition of $chosen$.

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

Consider any behavior of the $Voting$ algorithm.

Consider the value of $chosen$ in each state.

Those values of $chosen$ should produce a behavior allowed by the $Consensus$ specification.

What it means for the $Voting$ algorithm to implement the $Consensus$ spec with this definition of $chosen$.

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

Consider any behavior of the $Voting$ algorithm.

Consider the value of $chosen$ in each state.

Those values of $chosen$ should produce a behavior allowed by the $Consensus$ specification.

But that's absurd.

What it means for the $Voting$ algorithm to implement the $Consensus$ spec with this definition of $chosen$.

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

Consider any behavior of the $Voting$ algorithm.

Consider the value of $chosen$ in each state.

Those values of $chosen$ should produce a behavior allowed by the $Consensus$ specification.

But that's absurd.

A behavior of the $Voting$ algorithm has lots of steps.

What it means for the $Voting$ algorithm to implement the $Consensus$ spec with this definition of $chosen$.

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

Consider any behavior of the $Voting$ algorithm.

Consider the value of $chosen$ in each state.

Those values of $chosen$ should produce a behavior allowed by the $Consensus$ specification.

But that's absurd.

A behavior of the $Voting$ algorithm has lots of steps.
The $Consensus$ spec only allows a single step.

Another Example

Another Example

Suppose we want to buy a clock that displays the hour,

Another Example

Suppose we want to buy a clock that displays the hour,
and suppose we don't care if the clock shows the actual time.

Another Example

Suppose we want to buy a clock that displays the hour,
and suppose we don’t care if the clock shows the actual time.

We could specify such a clock like this:

Another Example

Suppose we want to buy a clock that displays the hour,
and suppose we don't care if the clock shows the actual time.

We could specify such a clock like this:

When the clock is plugged in, the display shows 12.

Another Example

Suppose we want to buy a clock that displays the hour,
and suppose we don't care if the clock shows the actual time.

We could specify such a clock like this:

When the clock is plugged in, the display shows 12.

The display then changes to 1

Another Example

Suppose we want to buy a clock that displays the hour,
and suppose we don't care if the clock shows the actual time.

We could specify such a clock like this:

When the clock is plugged in, the display shows 12.
The display then changes to 1, then to 2

Another Example

Suppose we want to buy a clock that displays the hour,
and suppose we don't care if the clock shows the actual time.

We could specify such a clock like this:

When the clock is plugged in, the display shows 12.
The display then changes to 1, then to 2, and so on.

Another Example

Suppose we want to buy a clock that displays the hour,
and suppose we don't care if the clock shows the actual time.

We could specify such a clock like this:

When the clock is plugged in, the display shows 12.

The display then changes to 1, then to 2, and so on.

We go into the store and a salesman shows us a clock
that displays the hour and also the minute.

Another Example

Suppose we want to buy a clock that displays the hour, and suppose we don't care if the clock shows the actual time.

We could specify such a clock like this:

When the clock is plugged in, the display shows 12.

The display then changes to 1, then to 2, and so on.

We go into the store and a salesman shows us a clock that displays the hour and also the minute.

Since we didn't specify that the clock doesn't display the minute, most people would say the hour-minute clock satisfies our spec.

But a computer scientist would say that’s wrong.

But a computer scientist would say that's wrong.

He'd say that the hour clock spec talks about a universe containing only an hour display.

But a computer scientist would say that's wrong.

He'd say that the hour clock spec talks about a universe containing only an hour display.

A spec of an hour-minute clock talks about a different universe that contains an hour display and a minute display.

But a computer scientist would say that's wrong.

He'd say that the hour clock spec talks about a universe containing only an hour display.

A spec of an hour-minute clock talks about a different universe that contains an hour display and a minute display.

Comparing the two specs isn't so simple.

If Math was Invented by Computer Scientists

If Math was Invented by Computer Scientists

Supposed you proved:

Theorem 1. If x is an integer, then $x + 1 > x$.

If Math was Invented by Computer Scientists

Supposed you proved:

Theorem 1. If x is an integer, then $x + 1 > x$.

And suppose you then wanted to prove:

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

If Math was Invented by Computer Scientists

Supposed you proved:

Theorem 1. If x is an integer, then $x + 1 > x$.

And suppose you then wanted to prove:

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

You'd like to use Theorem 1 to prove Theorem 2.

If Math was Invented by Computer Scientists

Supposed you proved:

Theorem 1. If x is an integer, then $x + 1 > x$.

And suppose you then wanted to prove:

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

You'd like to use Theorem 1 to prove Theorem 2.

But you couldn't because Theorem 1 is about a universe containing only one integer x, while Theorem 2 is about a different universe that contains the two integers x and y.

Why Math is So Simple

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x + 1 > x$.

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x + 1 > x$.

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

are about the same universe.

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x + 1 > x$.

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

are about the same universe.

That universe contains the variables x , y

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x+1>x$.

Theorem 2. If x and y are integers, then $y+(x+1)>y+x$.

are about the same universe.

That universe contains the variables x, y, z,

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x + 1 > x$.

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

are about the same universe.

That universe contains the variables x , y , z , q ,

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x + 1 > x$.

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

are about the same universe.

That universe contains the variables x , y , z , q , α ,

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x + 1 > x$.

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

are about the same universe.

That universe contains the variables x, y, z, q, α, β,

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x + 1 > x$.

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

are about the same universe.

That universe contains the variables x, y, z, q, α, β, $maxBal$,

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x + 1 > x$.

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

are about the same universe.

That universe contains the variables x , y , z , q , α , β , $maxBal$, $chosen$,

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x + 1 > x$.

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

are about the same universe.

That universe contains the variables x, y, z, q, α, β, $maxBal$, $chosen$, ...

Why Math is So Simple

These two theorems

Theorem 1. If x is an integer, then $x + 1 > x$.

Theorem 2. If x and y are integers, then $y + (x + 1) > y + x$.

are about the same universe.

That universe contains the variables x, y, z, q, α, β, $maxBal$, $chosen$, $\ldots$

Those theorems only say something about two of those variables.

We’re describing systems with math.

We're describing systems with math.

A spec of an hour clock doesn't describe a universe containing only a single variable $hour$.

We’re describing systems with math.

A spec of an hour clock doesn’t describe a universe containing only a single variable $hour$.

It describes a universe containing infinitely many variables.

We're describing systems with math.

A spec of an hour clock doesn't describe a universe containing only a single variable $hour$.

It describes a universe containing infinitely many variables.

A state is an assignment of values to all those variables.

We're describing systems with math.

A spec of an hour clock doesn't describe a universe containing only a single variable $hour$.

It describes a universe containing infinitely many variables.

A state is an assignment of values to all those variables.

The hour clock's spec says nothing about the values of any variable other than $hour$.

When we say that the hour clock just has this single behavior:

$$\left[hour = 12\right] \longrightarrow \left[hour = 1\right] \longrightarrow \left[hour = 2\right] \longrightarrow \cdots$$

When we say that the hour clock just has this single behavior:

$$\left[hour = 12\right] \longrightarrow \left[hour = 1\right] \longrightarrow \left[hour = 2\right] \longrightarrow \cdots$$

we mean that it allows infinitely many behaviors, such as:

$$\begin{bmatrix} hour & = 12 \\ chosen & = \{\,\} \\ maxBal & = -72 \\ min & = 32 \\ x & = \sqrt{7} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 1 \\ chosen & = \{0\} \\ maxBal & = -7 \\ min & = 16 \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 2 \\ chosen & = \{\,\} \\ maxBal & = 1/2 \\ min & = \langle 1, 2 \rangle \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \cdots$$

When we say that the hour clock just has this single behavior:

$$[hour = 12] \longrightarrow [hour = 1] \longrightarrow [hour = 2] \longrightarrow \cdots$$

we mean that it allows infinitely many behaviors, such as:

$$\begin{bmatrix} hour & = 12 \\ chosen & = \{\,\} \\ maxBal & = -72 \\ min & = 32 \\ x & = \sqrt{7} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 1 \\ chosen & = \{0\} \\ maxBal & = -7 \\ min & = 16 \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 2 \\ chosen & = \{\,\} \\ maxBal & = 1/2 \\ min & = \langle 1, 2 \rangle \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \cdots$$

All behaviors in which $hour$ has these values

When we say that the hour clock just has this single behavior:

$$\begin{bmatrix} hour = 12 \end{bmatrix} \longrightarrow \begin{bmatrix} hour = 1 \end{bmatrix} \longrightarrow \begin{bmatrix} hour = 2 \end{bmatrix} \longrightarrow \cdots$$

we mean that it allows infinitely many behaviors, such as:

$$\begin{bmatrix} hour & = 12 \\ chosen & = \{\,\} \\ maxBal & = -72 \\ min & = 32 \\ x & = \sqrt{7} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 1 \\ chosen & = \{0\} \\ maxBal & = -7 \\ min & = 16 \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 2 \\ chosen & = \{\,\} \\ maxBal & = 1/2 \\ min & = \langle 1, 2 \rangle \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \cdots$$

All behaviors in which $hour$ has these values
and the other variables can have any values.

When we say that the hour clock just has this single behavior:

$$\left[hour = 12\right] \quad \longrightarrow \quad \left[hour = 1\right] \quad \longrightarrow \quad \left[hour = 2\right] \quad \longrightarrow \ldots$$

When we say that the hour clock just has this single behavior:

$$\left[hour = 12\right] \longrightarrow \left[hour = 1\right] \longrightarrow \left[hour = 2\right] \longrightarrow \cdots$$

we also mean that it allows **only** behaviors such as:

$$\begin{bmatrix} hour & = 12 \\ chosen & = \{\,\} \\ maxBal & = -72 \\ min & = 32 \\ x & = \sqrt{7} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 1 \\ chosen & = \{0\} \\ maxBal & = -7 \\ min & = 16 \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 2 \\ chosen & = \{\,\} \\ maxBal & = 1/2 \\ min & = \langle 1, 2\rangle \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \cdots$$

When we say that the hour clock just has this single behavior:

$$\left[hour = 12\right] \longrightarrow \left[hour = 1\right] \longrightarrow \left[hour = 2\right] \longrightarrow \cdots$$

we also mean that it allows **only** behaviors such as:

$$\begin{bmatrix} hour & = 12 \\ chosen & = \{\,\} \\ maxBal & = -72 \\ min & = 32 \\ x & = \sqrt{7} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 1 \\ chosen & = \{0\} \\ maxBal & = -7 \\ min & = 16 \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 2 \\ chosen & = \{\,\} \\ maxBal & = 1/2 \\ min & = \langle 1, 2 \rangle \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \cdots$$

in which the $Voting$ algorithm can change $maxBal$ or $votes$ only once an hour.

When we say that the hour clock just has this single behavior:

$$[hour = 12] \longrightarrow [hour = 1] \longrightarrow [hour = 2] \longrightarrow \cdots$$

we also mean that it allows **only** behaviors such as:

$$\begin{bmatrix} hour & = 12 \\ chosen & = \{\} \\ maxBal & = -72 \\ min & = 32 \\ x & = \sqrt{7} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 1 \\ chosen & = \{0\} \\ maxBal & = -7 \\ min & = 16 \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \begin{bmatrix} hour & = 2 \\ chosen & = \{\} \\ maxBal & = 1/2 \\ min & = \langle 1, 2 \rangle \\ x & = \sqrt{-1} \\ & \vdots \end{bmatrix} \longrightarrow \cdots$$

in which the $Voting$ algorithm can change $maxBal$ or $votes$ only once an hour.

This is silly.

The spec of an hour clock should not allow just this behavior:

$$\big[hour = 12\big] \quad \longrightarrow \quad \big[hour = 1\big] \quad \longrightarrow \quad \big[hour = 2\big] \quad \longrightarrow \; \ldots$$

The spec of an hour clock should not allow just this behavior:

$$\left[hour = 12 \right] \longrightarrow \left[hour = 1 \right] \longrightarrow \left[hour = 2 \right] \longrightarrow \cdots$$

It should also allow behaviors such as:

$$\left[hour = 12 \right] \longrightarrow \left[hour = 12 \right] \longrightarrow \left[hour = 1 \right] \longrightarrow \left[hour = 1 \right] \longrightarrow$$

$$\left[hour = 1 \right] \longrightarrow \left[hour = 2 \right] \longrightarrow \left[hour = 3 \right] \longrightarrow \left[hour = 3 \right] \longrightarrow \cdots$$

The spec of an hour clock should not allow just this behavior:

$$\left[hour = 12\right] \longrightarrow \left[hour = 1\right] \longrightarrow \left[hour = 2\right] \longrightarrow \ldots$$

It should also allow behaviors such as:

$$\left[hour = 12\right] \longrightarrow \left[hour = 12\right] \longrightarrow \left[hour = 1\right] \longrightarrow \left[hour = 1\right] \longrightarrow$$

$$\left[hour = 1\right] \longrightarrow \left[hour = 2\right] \longrightarrow \left[hour = 3\right] \longrightarrow \left[hour = 3\right] \longrightarrow \ldots$$

that include steps that don't change the value of $hour$.

The spec of an hour clock should not allow just this behavior:

$$[hour = 12] \longrightarrow [hour = 1] \longrightarrow [hour = 2] \longrightarrow \cdots$$

It should also allow behaviors such as:

$$[hour = 12] \longrightarrow [hour = 12] \longrightarrow [hour = 1] \longrightarrow [hour = 1] \longrightarrow$$
$$[hour = 1] \longrightarrow [hour = 2] \longrightarrow [hour = 3] \longrightarrow [hour = 3] \longrightarrow \cdots$$

that include steps that don't change the value of $hour$.

The spec of an hour clock should not allow just this behavior:

$$\left[hour = 12\right] \longrightarrow \left[hour = 1\right] \longrightarrow \left[hour = 2\right] \longrightarrow \ldots$$

It should also allow behaviors such as:

$$\left[hour = 12\right] \longrightarrow \left[hour = 12\right] \longrightarrow \left[hour = 1\right] \longrightarrow \left[hour = 1\right] \longrightarrow$$

$$\left[hour = 1\right] \longrightarrow \left[hour = 2\right] \longrightarrow \left[hour = 3\right] \longrightarrow \left[hour = 3\right] \longrightarrow \ldots$$

that include steps that don't change the value of $hour$.

The spec of an hour clock should not allow just this behavior:

$$[hour = 12] \longrightarrow [hour = 1] \longrightarrow [hour = 2] \longrightarrow \ldots$$

It should also allow behaviors such as:

$$[hour = 12] \longrightarrow [hour = 12] \longrightarrow [hour = 1] \longrightarrow [hour = 1] \longrightarrow$$
$$[hour = 1] \longrightarrow [hour = 2] \longrightarrow [hour = 3] \longrightarrow [hour = 3] \longrightarrow \ldots$$

that include steps that don't change the value of $hour$.

The spec of an hour clock should not allow just this behavior:

$$[hour = 12] \longrightarrow [hour = 1] \longrightarrow [hour = 2] \longrightarrow \cdots$$

It should also allow behaviors such as:

$$[hour = 12] \longrightarrow [hour = 12] \longrightarrow [hour = 1] \longrightarrow [hour = 1] \longrightarrow$$

$$[hour = 1] \longrightarrow [hour = 2] \longrightarrow [hour = 3] \longrightarrow [hour = 3] \longrightarrow \cdots$$

that include steps that don't change the value of $hour$.

The spec of an hour clock should not allow just this behavior:

$$[hour = 12] \longrightarrow [hour = 1] \longrightarrow [hour = 2] \longrightarrow \ldots$$

It should also allow behaviors such as:

$$[hour = 12] \longrightarrow [hour = 12] \longrightarrow [hour = 1] \longrightarrow [hour = 1] \longrightarrow$$

$$[hour = 1] \longrightarrow [hour = 2] \longrightarrow [hour = 3] \longrightarrow [hour = 3] \longrightarrow \ldots$$

stuttering steps

that include ~~steps that don't change the value of $hour$~~ .

Our $Consensus$ spec should not allow just this behavior:

$$\left[chosen = \{\,\} \right] \longrightarrow \left[chosen = \{42\} \right]$$

Our $Consensus$ spec should not allow just this behavior:

$$\left[chosen = \{\,\}\right] \longrightarrow \left[chosen = \{42\}\right]$$

It should also allow these behaviors:

Our $Consensus$ spec should not allow just this behavior:

$$\left[chosen = \{\,\} \right] \longrightarrow \left[chosen = \{42\} \right]$$

It should also allow these behaviors:

$$\left[chosen = \{\,\} \right] \rightarrow \left[chosen = \{\,\} \right] \rightarrow \left[chosen = \{42\} \right]$$

Our $Consensus$ spec should not allow just this behavior:

$$\left[chosen = \{\,\}\right] \longrightarrow \left[chosen = \{42\}\right]$$

It should also allow these behaviors:

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right]$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right]$$

Our $Consensus$ spec should not allow just this behavior:

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

It should also allow these behaviors:

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

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

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

Our $Consensus$ spec should not allow just this behavior:

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

It should also allow these behaviors:

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

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

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

$$\vdots$$

Our $Consensus$ spec should not allow just this behavior:

$$\left[chosen = \{\,\}\right] \longrightarrow \left[chosen = \{42\}\right]$$

It should also allow these behaviors:

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right]$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right]$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right]$$

$$\vdots$$

And it does.

Remember our $Consensus$ specification:

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

Remember our $Consensus$ specification:

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

Now we find out what this is all about.

Remember our $Consensus$ specification:

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

$[A]_b$ is an abbreviation for $A \lor (b' = b)$.

Remember our $Consensus$ specification:

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

$[A]_b$ is an abbreviation for $A \lor (b' = b)$.

So $Spec$ equals

Remember our $Consensus$ specification:

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

$[A]_b$ is an abbreviation for $A \lor (b' = b)$.

So $Spec$ equals

$$Init \land \Box(Next \lor (chosen' = chosen))$$

Remember our $Consensus$ specification:

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

$[A]_b$ is an abbreviation for $A \lor (b' = b)$.

So $Spec$ equals

$$Init \land \Box(Next \lor (chosen' = chosen))$$

which means that as well as allowing steps satisfying $Next$

Remember our $Consensus$ specification:

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

$[A]_b$ is an abbreviation for $A \vee (b' = b)$.

So $Spec$ equals

$$Init \wedge \Box(Next \vee (chosen' = chosen))$$

which means that as well as allowing steps satisfying $Next$

it allows stuttering steps that leave $chosen$ unchanged.

Remember the $Voting$ specification:

$$Spec \triangleq Init \land \Box[Next]_{\langle votes,\, maxBal \rangle}$$

Remember the $Voting$ specification:

$$Spec \triangleq Init \wedge \Box \boxed{[} Next] \boxed{_{\langle votes, maxBal \rangle}}$$

We now know that this means $Spec$ equals

$$Init \wedge \Box(Next \vee (\langle votes, maxBal \rangle' = \langle votes, maxBal \rangle))$$

Remember the $Voting$ specification:

$$Spec \triangleq Init \land \Box[Next]_{\langle votes,\, maxBal \rangle}$$

We now know that this means $Spec$ equals

$$Init \land \Box(Next \lor (\langle votes, maxBal \rangle' = \langle votes, maxBal \rangle))$$

$'$ means *in the next state*

Remember the $Voting$ specification:

$$Spec \triangleq Init \land \Box[Next]_{\langle votes,\, maxBal \rangle}$$

We now know that this means $Spec$ equals

$$Init \land \Box(Next \lor (\langle votes, maxBal \rangle' = \langle votes, maxBal \rangle))$$

$'$ means *in the next state*, so this

Remember the $Voting$ specification:

$$Spec \triangleq Init \land \Box[Next]_{\langle votes,\, maxBal \rangle}$$

We now know that this means $Spec$ equals

$$Init \land \Box(Next \lor (\langle votes, maxBal \rangle' = \langle votes, maxBal \rangle))$$

$'$ means *in the next state*, so this allows steps that leave $\langle votes, maxBal \rangle$ unchanged,

Remember the $Voting$ specification:

$$Spec \triangleq Init \land \Box[Next]_{\langle votes,\, maxBal \rangle}$$

We now know that this means $Spec$ equals

$$Init \land \Box(Next \lor (\langle votes, maxBal \rangle' = \langle votes, maxBal \rangle))$$

$'$ means *in the next state*, so this allows steps that leave $\langle votes, maxBal \rangle$ unchanged, which are steps that leave both $votes$ and $maxBal$ unchanged,

Remember the $Voting$ specification:

$$Spec \triangleq Init \land \Box[Next]_{\langle votes,\, maxBal \rangle}$$

We now know that this means $Spec$ equals

$$Init \land \Box(Next \lor (\langle votes, maxBal \rangle' = \langle votes, maxBal \rangle))$$

$'$ means *in the next state*, so this allows steps that leave $\langle votes, maxBal \rangle$ unchanged, which are steps that leave both $votes$ and $maxBal$ unchanged, which are stuttering steps.

Remember the $Voting$ specification:

$$Spec \triangleq Init \land \Box[Next]_{\langle votes,\, maxBal \rangle}$$

We now know that this means $Spec$ equals

$$Init \land \Box(Next \lor (\langle votes, maxBal \rangle' = \langle votes, maxBal \rangle))$$

$'$ means *in the next state*, so this allows steps that leave $\langle votes, maxBal \rangle$ unchanged, which are steps that leave both $votes$ and $maxBal$ unchanged, which are stuttering steps.

So the $Voting$ spec allows stuttering steps.

Remember the $Voting$ specification:

$$Spec \triangleq Init \land \Box[Next]_{\langle votes,\, maxBal \rangle}$$

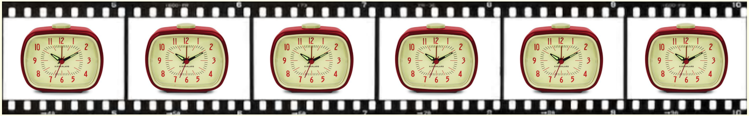
We now know that this means $Spec$ equals

$$Init \land \Box(Next \lor (\langle votes, maxBal \rangle' = \langle votes, maxBal \rangle))$$

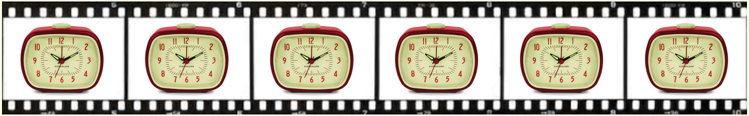
$'$ means *in the next state*, so this allows steps that leave $\langle votes, maxBal \rangle$ unchanged, which are steps that leave both $votes$ and $maxBal$ unchanged, which are stuttering steps.

All TLA$^+$ specs allow stuttering steps.

Think of a state in a behavior as a frame in a movie of the system

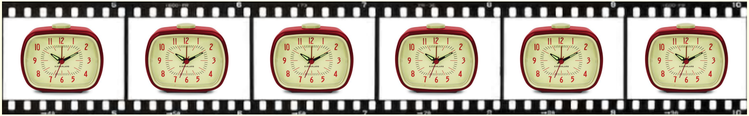


Think of a state in a behavior as a frame in a movie of the system



where the camera ran at an arbitrarily changing rate.

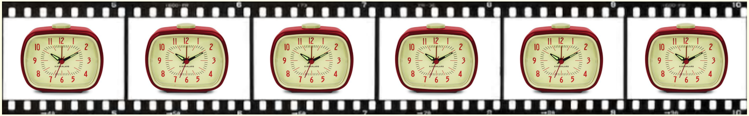
Think of a state in a behavior as a frame in a movie of the system



where the camera ran at an arbitrarily changing rate.

Speeding up the camera adds duplicate frames to the movie;

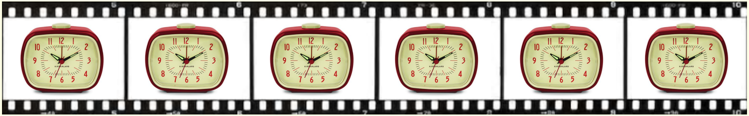
Think of a state in a behavior as a frame in a movie of the system



where the camera ran at an arbitrarily changing rate.

Speeding up the camera adds duplicate frames to the movie;
it doesn't change the actual execution of the system.

Think of a state in a behavior as a frame in a movie of the system

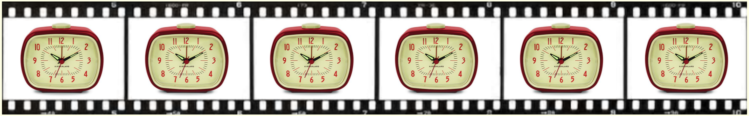


where the camera ran at an arbitrarily changing rate.

Speeding up the camera adds duplicate frames to the movie;
it doesn't change the actual execution of the system.

A step of a behavior could represent the passing of an hour or of a femtosecond.

Think of a state in a behavior as a frame in a movie of the system



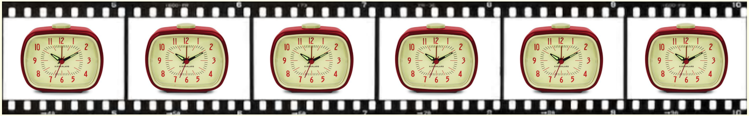
where the camera ran at an arbitrarily changing rate.

Speeding up the camera adds duplicate frames to the movie;
it doesn't change the actual execution of the system.

A step of a behavior could represent the passing of an hour or of a femtosecond.

If you want to describe the passage of real time, do what
real scientists do:

Think of a state in a behavior as a frame in a movie of the system



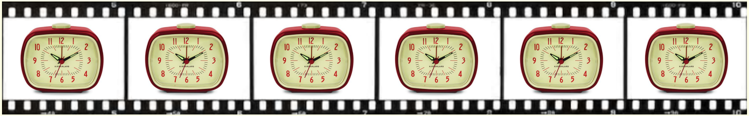
where the camera ran at an arbitrarily changing rate.

Speeding up the camera adds duplicate frames to the movie;
it doesn't change the actual execution of the system.

A step of a behavior could represent the passing of an hour or of a femtosecond.

If you want to describe the passage of real time, do what
real scientists do: add a variable that represents time.

Think of a state in a behavior as a frame in a movie of the system



where the camera ran at an arbitrarily changing rate.

Speeding up the camera adds duplicate frames to the movie;
it doesn't change the actual execution of the system.

A step of a behavior could represent the passing of an hour or of a femtosecond.

If you want to describe the passage of real time, do what
real scientists do: add a variable that represents time.

We needn't do that because Paxos is an asynchronous algorithm.

We now return to our goal:

Showing that the voting algorithm implements the consensus spec

We now return to our goal:

Showing that the voting algorithm implements the consensus spec, where the variable $chosen$ of the $Consensus$ spec is implemented with the expression $chosen$ defined in the $Voting$ spec by:

We now return to our goal:

Showing that the voting algorithm implements the consensus spec, where the variable $chosen$ of the $Consensus$ spec is implemented with the expression $chosen$ defined in the $Voting$ spec by:

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

We now return to our goal:

Showing that the voting algorithm implements the consensus spec, where the variable $chosen$ of the $Consensus$ spec is implemented with the expression $chosen$ defined in the $Voting$ spec by:

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

$$\begin{aligned} & ChosenAt(b, v) \triangleq \\ & \quad \exists Q \in Quorum : \forall a \in Q : VotedFor(a, b, v) \end{aligned}$$

We now return to our goal:

Showing that the voting algorithm implements the consensus spec, where the variable $chosen$ of the $Consensus$ spec is implemented with the expression $chosen$ defined in the $Voting$ spec by:

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

$$\begin{aligned} &ChosenAt(b, v) \triangleq \\ &\quad \exists\, Q \in Quorum : \forall\, a \in Q : VotedFor(a, b, v) \end{aligned}$$

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

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \ \ldots$$

$$chosen = \cdots \qquad chosen = \cdots \qquad chosen = \cdots \qquad \ldots$$

If we take the values of $chosen$ in each state

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \ \cdots$$

$$chosen = \cdots \qquad\qquad chosen = \cdots \qquad\qquad chosen = \cdots \qquad\qquad \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that the original behavior

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that the original behavior satisfies the formula obtained by substituting the definition of $chosen$ in $Voting$ for the variable $chosen$ in formula $Spec$ of $Consensus$.

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that the original behavior satisfies the formula obtained by substituting the definition of $chosen$ in $Voting$ for the variable $chosen$ in formula $Spec$ of $Consensus$.

I don't expect you to see why these two conditions are equivalent.

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that the original behavior satisfies the formula obtained by substituting the definition of $chosen$ in $Voting$ for the variable $chosen$ in formula $Spec$ of $Consensus$.

I don't expect you to see why these two conditions are equivalent.
That requires a lot of thinking.

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that the original behavior satisfies the formula obtained by substituting the definition of $chosen$ in $Voting$ for the variable $chosen$ in formula $Spec$ of $Consensus$.

I don't expect you to see why these two conditions are equivalent.
That requires a lot of thinking. For now, you'll have to believe me.

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \rightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \rightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \rightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \rightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \rightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \rightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that the original behavior satisfies the formula obtained by substituting the definition of $chosen$ in $Voting$ for the variable $chosen$ in formula $Spec$ of $Consensus$.

Let's call this formula $Spec_C^{sub}$.

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \rightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \rightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \rightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \rightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \rightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \rightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that the original behavior satisfies $Spec_C^{sub}$.

Let's call this formula $Spec_C^{sub}$.

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that the original behavior satisfies $Spec_C^{sub}$.

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that the original behavior satisfies $Spec_C^{sub}$.

We have to show that for any behavior of the $Voting$ algorithm:

$$\begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} votes & = \cdots \\ maxBal & = \cdots \end{bmatrix} \longrightarrow \cdots$$

$$\begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \begin{bmatrix} chosen = \cdots \end{bmatrix} \longrightarrow \cdots$$

If we take the values of $chosen$ in each state
and assign them to the variable $chosen$
we get a behavior that satisfies the $Consensus$ spec.

This condition is equivalent to the condition that the original behavior satisfies $Spec_C^{sub}$.

We have to show:

Any behavior of the $Voting$ algorithm satisfies $Spec_C^{sub}$.

We have to show:

Any behavior of the $Voting$ algorithm satisfies $Spec_C^{sub}$.

Let $Spec_V$ be formula $Spec$ of module $Voting$.

We have to show:

Any behavior of the $Voting$ algorithm satisfies $Spec_C^{sub}$.

Let $Spec_V$ be formula $Spec$ of module $Voting$.

We have to show:

Any behavior satisfying $Spec_V$ satisfies $Spec_C^{sub}$.

Let $Spec_V$ be formula $Spec$ of module $Voting$.

We have to show:

Any behavior satisfying $Spec_V$ satisfies $Spec_C^{sub}$.

We have to show:

~~Any behavior satisfying $Spec_V$ satisfies $Spec_C^{sub}$.~~

We have to show:

~~Any behavior satisfying $Spec_V$ satisfies $Spec_C^{sub}$~~ .

THEOREM $Spec_V \;\Rightarrow\; Spec_C^{sub}$

We have to show:

~~Any behavior satisfying $Spec_V$ satisfies $Spec_C^{sub}$~~.

THEOREM $Spec_V \Rightarrow Spec_C^{sub}$

Let's now see how that theorem is written in TLA$^+$.

THEOREM $Spec_V \Rightarrow Spec_C^{sub}$

THEOREM $Spec_V \Rightarrow Spec_C^{sub}$

We will write the theorem in module $Voting$

THEOREM $Spec_V \Rightarrow Spec_C^{sub}$

We will write the theorem in module $Voting$, so this is just $Spec$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

We will write the theorem in module $Voting$, so this is just $Spec$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

We will write the theorem in module $Voting$, so this is just $Spec$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

To write $Spec_C^{sub}$, module $Voting$ must import the definition of $Spec$ from module $Consensus$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

INSTANCE $Consensus$

This imports all the definitions from $Consensus$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

INSTANCE $Consensus$

This imports all the definitions from $Consensus$.

But to prevent name clashes, the names of the imported definitions must be changed.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

This imports all the definitions from $Consensus$ with their names prefixed by $C!$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

This imports all the definitions from $Consensus$ with their names prefixed by $C!$.

For example $Next$ is imported as $C!Next$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

– TLA^+ primitive operators

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

– TLA$^+$ primitive operators such as $\land \quad \subseteq \quad \Box$

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

– TLA$^+$ primitive operators such as $\land \quad \subseteq \quad \Box$

They are meaningful in module $Voting$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

– TLA$^+$ primitive operators

– Defined operators

THEOREM $Spec \Rightarrow Spec^{sub}_{C}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

- TLA$^+$ primitive operators
- Defined operators such as $Next$ Inv

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

– TLA$^+$ primitive operators

– Defined operators such as $Next$ Inv $IsFiniteSet$ $\leq$

THEOREM $Spec \Rightarrow Spec_{C}^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

- TLA^{+} primitive operators
- Defined operators such as $Next$ Inv $IsFiniteSet$ $\leq$

imported with EXTENDS

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

- TLA$^+$ primitive operators
- Defined operators such as $Next$ Inv $IsFiniteSet$ $\leq$
  They are imported (renamed) into $Voting$ with their definitions.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

- TLA$^+$ primitive operators
- Defined operators
- The declared symbols $Value$ and $chosen$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

– TLA$^+$ primitive operators

– Defined operators

– The declared symbols $Value$ and $chosen$.
  Those symbols have meanings in module $Voting$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

– TLA$^+$ primitive operators

– Defined operators

– The declared symbols $Value$ and $chosen$.

Those symbols have meanings in module $Voting$.

But how do we know those meanings are related to their meanings in module $Consensus$?

THEOREM $Spec \Rightarrow Spec_{C}^{sub}$

$C \triangleq$ INSTANCE $Consensus$

The following kinds of symbols appear in module $Consensus$:

– TLA^{+} primitive operators

– Defined operators

– The declared symbols $Value$ and $chosen$.

Those symbols have meanings in module $Voting$.

But how do we know those meanings are related to their meanings in module $Consensus$?

We have to say what expressions must be substituted for them.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

WITH $Value \leftarrow \quad , \; chosen \leftarrow$

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$

WITH $Value \leftarrow Value,\ chosen \leftarrow$

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$
WITH $Value \leftarrow Value, \; chosen \leftarrow chosen$

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$
WITH $\boxed{Value} \leftarrow Value$, $\boxed{chosen} \leftarrow chosen$

These are the declared symbols of module $Consensus$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$
WITH $Value \leftarrow \boxed{Value}$, $chosen \leftarrow \boxed{chosen}$

These are the declared symbols of module $Consensus$.

The are expressions of module $Voting$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$
WITH $Value \leftarrow Value,\ chosen \leftarrow chosen$

This symbol is defined in $Voting$

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$
WITH $Value \leftarrow Value,\ chosen \leftarrow chosen$

This symbol is defined in $Voting$ by

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

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$
WITH $Value \leftarrow Value,\ chosen \leftarrow \boxed{chosen}$

This symbol is defined in $Voting$ by

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

We can replace it

THEOREM $Spec \Rightarrow Spec_{C}^{sub}$

$C \triangleq$ INSTANCE $Consensus$
WITH $Value \leftarrow Value,\ chosen \leftarrow \boxed{chosen}$

This symbol is defined in *Voting* by

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

We can replace it by this

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$C \triangleq$ INSTANCE $Consensus$
WITH $Value \leftarrow Value,\ chosen \leftarrow \boxed{chosen}$

This symbol is defined in $Voting$ by

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

We can replace it by this, since the two are equivalent in module $Voting$.

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$$C \triangleq \text{INSTANCE } Consensus$$
$$\qquad \text{WITH } Value \leftarrow Value,\ chosen \leftarrow chosen$$

THEOREM $Spec \Rightarrow Spec_C^{sub}$

$$C \triangleq \text{INSTANCE } Consensus \\ \quad \text{WITH } Value \leftarrow Value,\ chosen \leftarrow chosen$$

So we can now write $Spec_C^{sub}$

THEOREM $Spec \Rightarrow C!Spec$

$$C \triangleq \text{INSTANCE } Consensus$$
$$\quad \text{WITH } Value \leftarrow Value,\ chosen \leftarrow chosen$$

So we can now write $Spec_C^{sub}$ as $C!Spec$.

THEOREM $Spec \Rightarrow C!Spec$

$$C \triangleq \text{INSTANCE } Consensus \\ \quad \text{WITH } Value \leftarrow Value,\ chosen \leftarrow chosen$$

$$C \triangleq \text{INSTANCE } Consensus$$
$$\text{WITH } Value \leftarrow Value,\ chosen \leftarrow chosen$$

$$\text{THEOREM } Spec \Rightarrow C!Spec$$

$$C \triangleq \text{INSTANCE } Consensus$$
$$\qquad\qquad \text{WITH } Value \leftarrow Value, \; chosen \leftarrow chosen$$

$$\text{THEOREM} \quad Spec \;\Rightarrow\; C!Spec$$

Of course, it has to go after the INSTANCE statement.

$$
\begin{array}{l}
C \;\triangleq\; \textsf{INSTANCE}\ \mathit{Consensus} \\
\qquad\qquad\quad \textsf{WITH}\ \ \mathit{Value} \leftarrow \mathit{Value},\ \mathit{chosen} \leftarrow \mathit{chosen}
\end{array}
$$

We can make one final simplification.

$$C \triangleq \text{INSTANCE } Consensus$$
$$\text{WITH } Value \leftarrow Value,\ \boxed{chosen} \leftarrow \boxed{chosen}$$

We can make one final simplification.

When we substitute the same symbol for a symbol

$$
\begin{aligned}
C \;\triangleq\; & \textsf{INSTANCE}\ \mathit{Consensus} \\
& \qquad \textsf{WITH}\ \ \mathit{Value} \leftarrow \mathit{Value}
\end{aligned}
$$

We can make one final simplification.

When we substitute the same symbol for a symbol we can omit that WITH clause.

$C \triangleq$ INSTANCE $Consensus$
WITH

We can make one final simplification.

When we substitute the same symbol for a symbol we can omit that WITH clause.

$C \triangleq$ INSTANCE $Consensus$

We can make one final simplification.

When we substitute the same symbol for a symbol we can omit that WITH clause.

$$C \triangleq \text{INSTANCE } Consensus$$

$$\text{THEOREM } \; Spec \Rightarrow C!Spec$$

$C \triangleq$ INSTANCE $Consensus$

THEOREM $Spec \Rightarrow C!Spec$

This theorem asserts that algorithm $Voting$ implements the $Consensus$ spec under its definition of $consensus$.

THEOREM $Spec \Rightarrow C!Spec$

THEOREM $Spec \Rightarrow C!Spec$

The model checker can check this theorem.

THEOREM $Spec \;\Rightarrow\; C!Spec$

The model checker can check this theorem.

We can also prove it

THEOREM $Spec \Rightarrow C!Spec$

The model checker can check this theorem.

We can also prove it, using a few simple proof rules.

THEOREM $Spec \Rightarrow C!Spec$

The model checker can check this theorem.

We can also prove it, using a few simple proof rules.

The proof uses an invariant maintained by the algorithm that explains why it is correct.

THEOREM $Spec \Rightarrow C!Spec$

The model checker can check this theorem.

We can also prove it, using a few simple proof rules.

The proof uses an invariant maintained by the algorithm that explains why it is correct.

The invariant is defined in the $Voting$ module.

Safety and Liveness

Safety and Liveness

I said this was a behavior satisfying the $Consensus$ spec

$$\left[chosen = \{\,\}\right] \longrightarrow \left[chosen = \{42\}\right]$$

Safety and Liveness

I said this was a behavior satisfying the $Consensus$ spec

$$\left[chosen = \{\,\}\right] \longrightarrow \left[chosen = \{42\}\right]$$

because the spec allowed no step from this state.

Safety and Liveness

I said this was a behavior satisfying the $Consensus$ spec

$$\left[chosen = \{\,\}\right] \longrightarrow \left[chosen = \{42\}\right] \longrightarrow \left[chosen = \{42\}\right] \longrightarrow \left[chosen = \{42\}\right] \longrightarrow \ldots$$

because the spec allowed no step from this state.

That was wrong because it allows these steps.

Safety and Liveness

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

The hour clock and everything else doesn't stop
just because the $Consensus$ "algorithm" stops.

Safety and Liveness

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

The hour clock and everything else doesn't stop just because the $Consensus$ "algorithm" stops.

Termination of a system execution is represented by a behavior ending in all stuttering steps.

Safety and Liveness

$$\left[chosen = \{\,\}\right] \longrightarrow \left[chosen = \{42\}\right] \longrightarrow \left[chosen = \{42\}\right] \longrightarrow \left[chosen = \{42\}\right] \longrightarrow \ldots$$

The hour clock and everything else doesn't stop
just because the $Consensus$ "algorithm" stops.

Termination of a system execution is represented by
a behavior ending in all stuttering steps.

This makes the math simpler.

The spec $Init \land \Box[Next]_{chosen}$ of module $Consensus$ allows these behaviors:

The spec $Init \wedge \Box[Next]_{chosen}$ of module $Consensus$ allows these behaviors:

$$\big[chosen = \{\,\}\big] \rightarrow \big[chosen = \{42\}\big] \rightarrow \big[chosen = \{42\}\big] \rightarrow \big[chosen = \{42\}\big] \rightarrow \ldots$$

The spec $Init \land \Box[Next]_{chosen}$ of module $Consensus$ allows these behaviors:

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \ldots$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \ldots$$

The spec $Init \land \Box[Next]_{chosen}$ of module $Consensus$ allows these behaviors:

$$[chosen = \{\,\}] \rightarrow [chosen = \{42\}] \rightarrow [chosen = \{42\}] \rightarrow [chosen = \{42\}] \rightarrow \ldots$$

$$[chosen = \{\,\}] \rightarrow [chosen = \{\,\}] \rightarrow [chosen = \{42\}] \rightarrow [chosen = \{42\}] \rightarrow \ldots$$

$$[chosen = \{\,\}] \rightarrow [chosen = \{\,\}] \rightarrow [chosen = \{\,\}] \rightarrow [chosen = \{42\}] \rightarrow \ldots$$

The spec $Init \land \Box[Next]_{chosen}$ of module $Consensus$ allows these behaviors:

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \ldots$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \ldots$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \ldots$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \ldots$$

The spec $Init \wedge \Box[Next]_{chosen}$ of module $Consensus$ allows these behaviors:

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

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

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

$$[chosen = \{\}] \rightarrow [chosen = \{\}] \rightarrow [chosen = \{\}] \rightarrow [chosen = \{\}] \rightarrow \cdots$$

$$\vdots$$

The spec $Init \land \Box[Next]_{chosen}$ of module $Consensus$ allows these behaviors:

$$\big[chosen = \{\,\}\big] \rightarrow \big[chosen = \{42\}\big] \rightarrow \big[chosen = \{42\}\big] \rightarrow \big[chosen = \{42\}\big] \rightarrow \cdots$$

$$\big[chosen = \{\,\}\big] \rightarrow \big[chosen = \{\,\}\big] \rightarrow \big[chosen = \{42\}\big] \rightarrow \big[chosen = \{42\}\big] \rightarrow \cdots$$

$$\big[chosen = \{\,\}\big] \rightarrow \big[chosen = \{\,\}\big] \rightarrow \big[chosen = \{\,\}\big] \rightarrow \big[chosen = \{42\}\big] \rightarrow \cdots$$

$$\big[chosen = \{\,\}\big] \rightarrow \big[chosen = \{\,\}\big] \rightarrow \big[chosen = \{\,\}\big] \rightarrow \big[chosen = \{\,\}\big] \rightarrow \cdots$$

$$\vdots$$

Behaviors that take an arbitrary number of stuttering steps before a value is chosen.

The spec $Init \wedge \Box[Next]_{chosen}$ of module $Consensus$ allows these behaviors:

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \cdots$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \cdots$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \cdots$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \cdots$$

$$\vdots$$

It also allows a behavior containing only stuttering steps

The spec $Init \land \Box[Next]_{chosen}$ of module $Consensus$ allows these behaviors:

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \cdots$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \cdots$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{42\}\right] \rightarrow \cdots$$

$$\left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \left[chosen = \{\,\}\right] \rightarrow \cdots$$

$$\vdots$$

It also allows a behavior containing only stuttering steps

describing an execution that terminates without choosing a value.

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

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

This formula says what steps are allowed to occur.

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

This formula says what steps are allowed to occur.

It doesn’t say what steps must occur.

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

An assertion of what is allowed to happen is called a *safety* property.

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

An assertion of what is allowed to happen is called a *safety* property.

An assertion of what must happen is called a *liveness* property.

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

An assertion of what is allowed to happen is called a *safety* property.

An assertion of what must happen is called a *liveness* property.

To specify that a value must be chosen,

$Init \land \Box[Next]_{chosen} \;\; \land \;\; \ldots$

An assertion of what is allowed to happen is called a *safety* property.

An assertion of what must happen is called a *liveness* property.

To specify that a value must be chosen, we'd have to conjoin a liveness property to the spec.

$Init \land \Box[Next]_{chosen} \;\; \land \;\; \ldots$

An assertion of what is allowed to happen is called a *safety* property.

An assertion of what must happen is called a *liveness* property.

To specify that a value must be chosen, we'd have to conjoin a liveness property to the spec.

That's easy to do

$Init \land \Box[Next]_{chosen} \;\; \land \;\; \ldots$

An assertion of what is allowed to happen is called a *safety* property.

An assertion of what must happen is called a *liveness* property.

To specify that a value must be chosen, we'd have to conjoin a liveness property to the spec.

That's easy to do (TLA$^+$ is great for specifying liveness)

$Init \land \Box[Next]_{chosen} \;\; \land \;\; \ldots$

An assertion of what is allowed to happen is called a *safety* property.

An assertion of what must happen is called a *liveness* property.

To specify that a value must be chosen, we'd have to conjoin a liveness property to the spec.

That's easy to do, but I won't do it.

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

An assertion of what is allowed to happen is called a *safety* property.

An assertion of what must happen is called a *liveness* property.

To specify that a value must be chosen, we'd have to conjoin a liveness property to the spec.

That's easy to do, but I won't do it.

Adding the requirement that a value must be chosen produces a spec that we can't implement.

The FLP Theorem

The FLP Theorem

Impossibility of Distributed Consensus with One Faulty Process

The FLP Theorem

Impossibility of Distributed Consensus with One Faulty Process

Michael J. **F**ischer, Nancy A. **L**ynch, and Michael S. **P**aterson

The FLP Theorem

Impossibility of Distributed Consensus with One Faulty Process

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson

Journal of the ACM, 1985

The FLP Theorem

Impossibility of Distributed Consensus with One Faulty Process

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson

Journal of the ACM, 1985

It’s impossible to implement consensus with the requirement
that a value is eventually chosen

The FLP Theorem

Impossibility of Distributed Consensus with One Faulty Process

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson

Journal of the ACM, 1985

It's impossible to implement consensus with the requirement that a value is eventually chosen in a distributed system in which any single process may fail by stopping.

The FLP Theorem

Impossibility of Distributed Consensus with One Faulty Process

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson

Journal of the ACM, 1985

It's impossible to implement consensus with the requirement that a value is eventually chosen in a distributed system in which any single process may fail by stopping.

The FLP Theorem

Impossibility of Distributed Consensus with One Faulty Process

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson

Journal of the ACM, 1985

It's impossible to implement consensus with the requirement that a value is eventually chosen in a distributed system in which any single process may fail by stopping.

The Paxos consensus algorithm is useful because

The FLP Theorem

Impossibility of Distributed Consensus with One Faulty Process

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson

Journal of the ACM, 1985

It's impossible to implement consensus with the requirement that a value is eventually chosen in a distributed system in which any single process may fail by stopping.

The Paxos consensus algorithm is useful because

- It never chooses more than one value no matter how many processes stop.

The FLP Theorem

Impossibility of Distributed Consensus with One Faulty Process

Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson

Journal of the ACM, 1985

It's impossible to implement consensus with the requirement that a value is eventually chosen in a distributed system in which any single process may fail by stopping.

The Paxos consensus algorithm is useful because

– It never chooses more than one value no matter how many processes stop.

– It has a very high probability of choosing a value if not too many processes stop.