TLA$^+$ Video Course – Lecture 4

Leslie Lamport

DIE HARD

This video should be viewed in conjunction with a Web page.
To find that page, search the Web for *TLA+ Video Course* .

The TLA$^+$ Video Course
Lecture 4
Die Hard

In this video, you'll learn how TLC can save your life. . . if you ever find yourself in the middle of a Hollywood action movie.

This will require you to learn some more about TLA+, TLC, and the Toolbox — which could turn out to be useful even outside of Hollywood.

[slide 2]

THE DIE HARD PROBLEM

Die Hard 3

Die Hard 3 is

Die Hard 3

A 1995 action film.

Die Hard 3 is a 1995 action film starring Bruce Willis and Samuel L. Jackson as the heroes.

Die Hard 3

A 1995 action film.

The heroes had to put exactly 4 gallons of water in a jug.

Die Hard 3 is a 1995 action film starring Bruce Willis and Samuel L. Jackson as the heroes.

To disarm a bomb, they had to put exactly 4 gallons of water in a jug.

Die Hard 3

A 1995 action film.

The heroes had to put exactly 4 gallons of water in a jug.

1 U.S. gallon $= \frac{1}{64}$ hogshead

Die Hard 3 is a 1995 action film starring Bruce Willis and Samuel L. Jackson as the heroes.

To disarm a bomb, they had to put exactly 4 gallons of water in a jug.

Die Hard 3

A 1995 action film.

The heroes had to put exactly 4 gallons of water in a jug.

$$1 \text{ U.S. gallon} = \tfrac{1}{64} \text{ hogshead} = 3.785411784 \text{ liters}$$

Die Hard 3 is a 1995 action film starring Bruce Willis and Samuel L. Jackson as the heroes.

To disarm a bomb, they had to put exactly 4 gallons of water in a jug.

Die Hard 3

A 1995 action film.

The heroes had to put exactly 4 gallons of water in a jug.

They had a 3 gallon jug, a 5 gallon jug, and a water faucet.

They were given a 3 gallon jug, a 5 gallon jug, and a water faucet.

Die Hard 3

A 1995 action film.

The heroes had to put exactly 4 gallons of water in a jug.

They had a 3 gallon jug, a 5 gallon jug, and a water faucet.

Search the Web for: *Die Hard Jugs Problem YouTube.*

They were given a 3 gallon jug, a 5 gallon jug, and a water faucet.

You can watch the relevant scene by searching the Web for *Die Hard Jugs Problem YouTube.*

There were no markings on the jugs.

There were no markings on the jugs.

There were no markings on the jugs.

They needed exacty 4 gallons.

There were no markings on the jugs.

They needed exacty 4 gallons.

There were no markings on the jugs.

They needed exacty 4 gallons.

Not 3.99 or 4.01.

There were no markings on the jugs.

They needed exacty 4 gallons.

Not 3.99 or 4.01 gallons.

GETTING STARTED

Getting Started on a Spec

When we want to write a spec, what should we do first?

Getting Started on a Spec

The best way:

Write a single correct behavior.

When we want to write a spec, what should we do first?

I recommend writing the start of a single correct behavior.

Getting Started on a Spec

The best way:

Write a single correct behavior.

Informally.

When we want to write a spec, what should we do first?

I recommend writing the start of a single correct behavior.

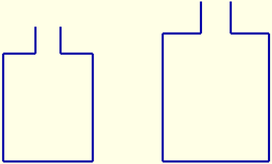
Informally at first.

This isn’t a big budget movie.

This isn’t a big budget Hollywood movie, and I can’t affort big jugs.

This isn’t a big budget movie.

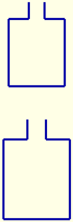
I’ll use these cartoon jugs:



This isn’t a big budget Hollywood movie, and I can’t affort big jugs.

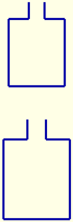
So instead, I’ll illustrate the spec with these cartoon jugs.

They start with both jugs empty.



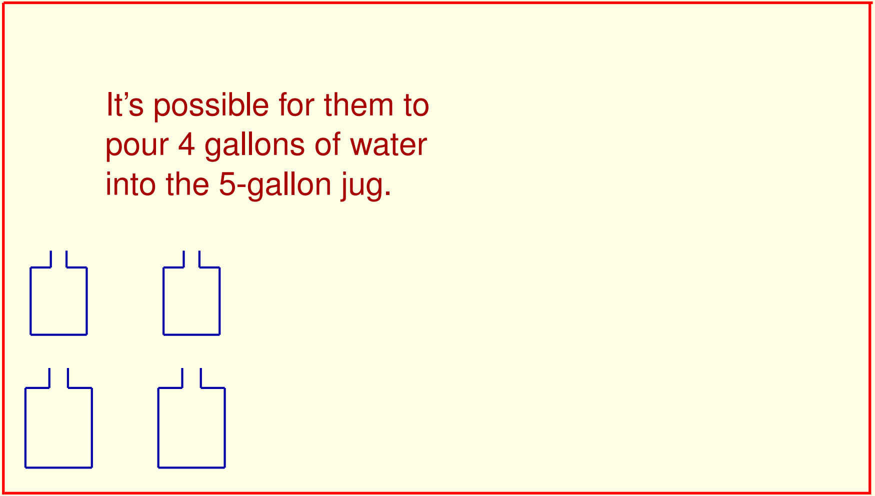
Our heroes start with both jugs empty.

It's possible for them to pour 4 gallons of water into the 5-gallon jug.



Our heroes start with both jugs empty.

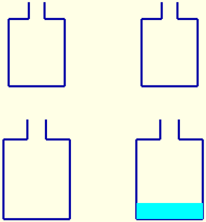
It's possible for them to solve their problem by simply pouring 4 gallons of water into the 5-gallon jug.



Our heroes start with both jugs empty.

It’s possible for them to solve their problem by simply pouring 4 gallons of water into the 5-gallon jug.

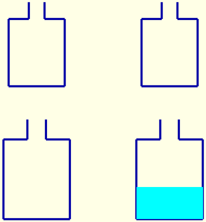
It’s possible for them to pour 4 gallons of water into the 5-gallon jug.



Our heroes start with both jugs empty.

It’s possible for them to solve their problem by simply pouring 4 gallons of water into the 5-gallon jug.

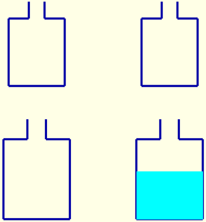
It’s possible for them to
pour 4 gallons of water
into the 5-gallon jug.



Our heroes start with both jugs empty.

It’s possible for them to solve their problem by simply pouring 4 gallons of water into the 5-gallon jug.

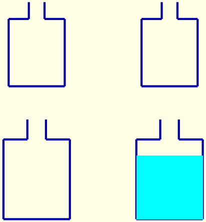
It’s possible for them to
pour 4 gallons of water
into the 5-gallon jug.



Our heroes start with both jugs empty.

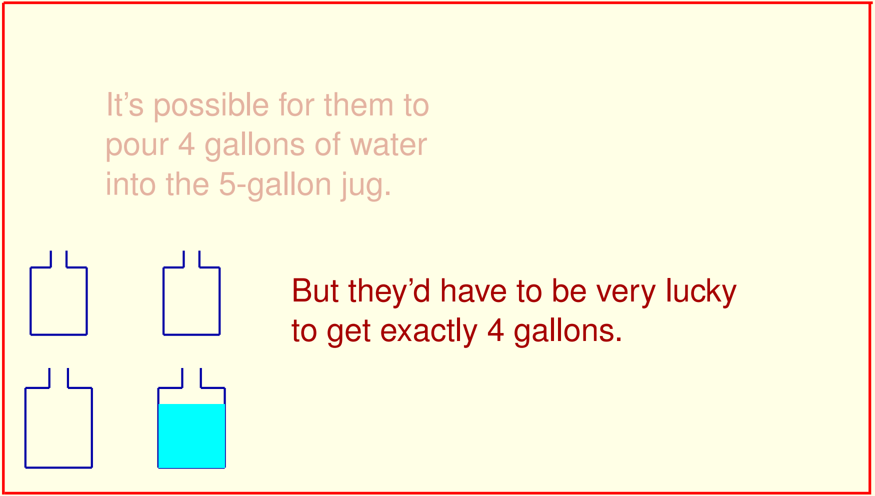
It’s possible for them to solve their problem by simply pouring 4 gallons of water into the 5-gallon jug.

It’s possible for them to
pour 4 gallons of water
into the 5-gallon jug.



Our heroes start with both jugs empty.

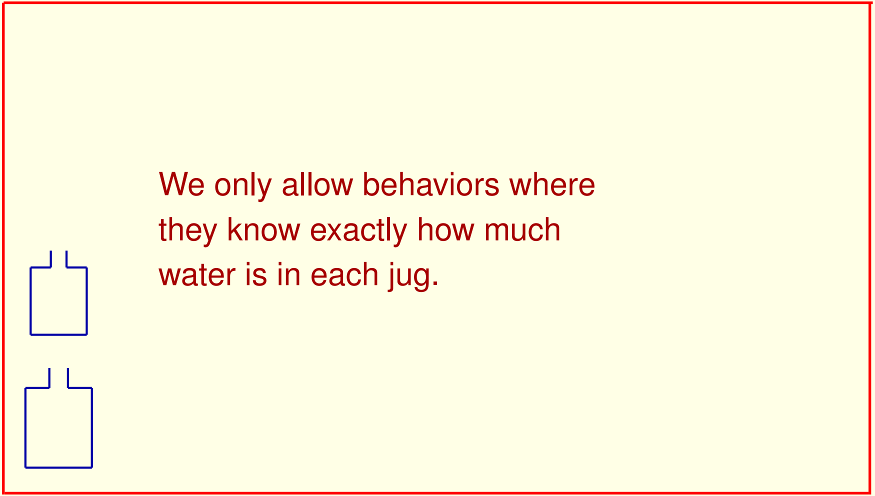
It’s possible for them to solve their problem by simply pouring 4 gallons of water into the 5-gallon jug.



Our heroes start with both jugs empty.

It's possible for them to solve their problem by simply pouring 4 gallons of water into the 5-gallon jug.

But they'd have to be very lucky to get exactly 4 gallons.



Our heroes start with both jugs empty.

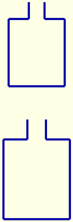
It's possible for them to solve their problem by simply pouring 4 gallons of water into the 5-gallon jug.

But they'd have to be very lucky to get exactly 4 gallons.

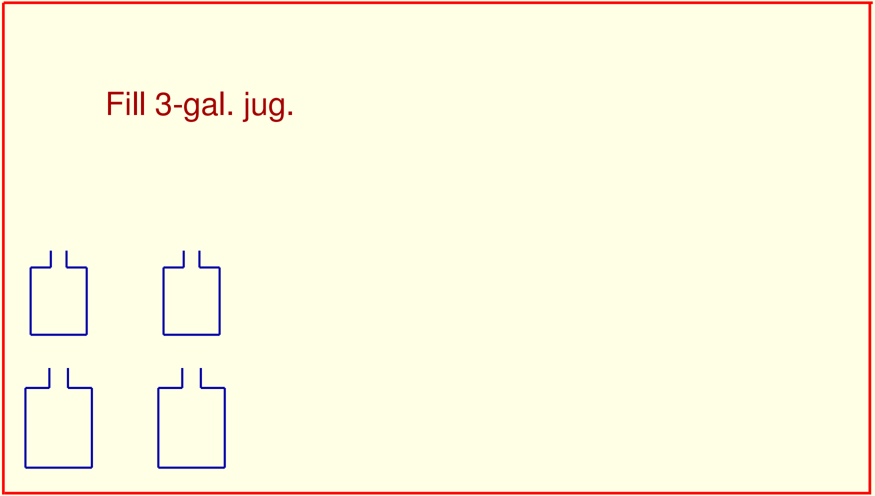
So we only allow behaviors in which they always know exactly how much water is in each jug.

[slide 28]

Initially, both jugs empty.



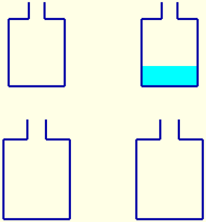
Again, they start with both jugs empty.



Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

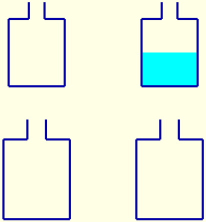
Fill 3-gal. jug.



Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

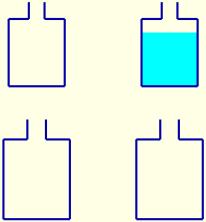
Fill 3-gal. jug.



Again, they start with both jugs empty.

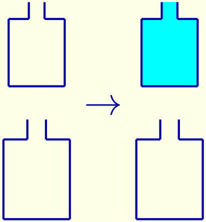
The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Fill 3-gal. jug.



Again, they start with both jugs empty.

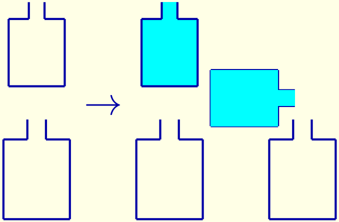
The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.



Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Empty 3-gal. jug into 5-gal. jug.

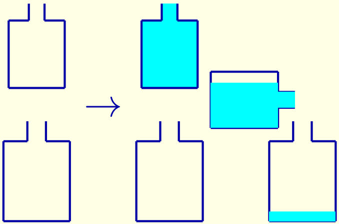


Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

Empty 3-gal. jug into 5-gal. jug.

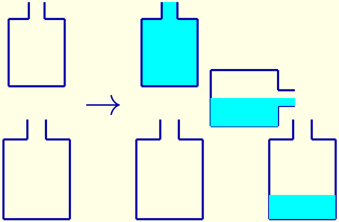


Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

Empty 3-gal. jug into 5-gal. jug.

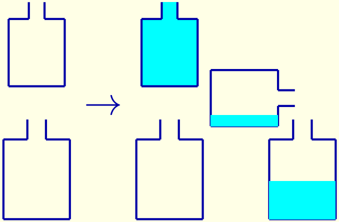


Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

Empty 3-gal. jug into 5-gal. jug.

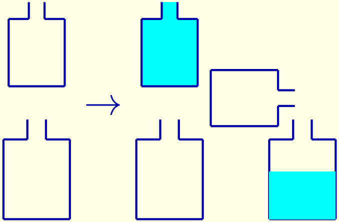


Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

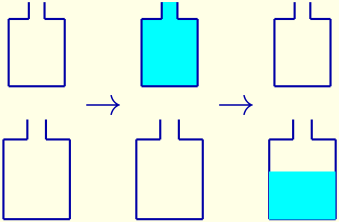
Empty 3-gal. jug into 5-gal. jug.



Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

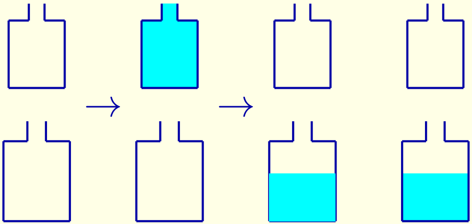


Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

Fill 3-gal. jug.



Again, they start with both jugs empty.

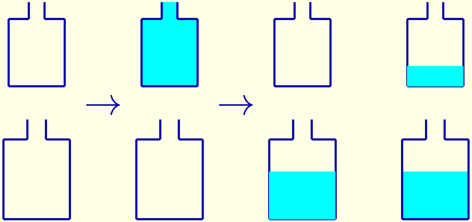
The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

Now they fill the 3 gallon jug.

[slide 41]

Fill 3-gal. jug.



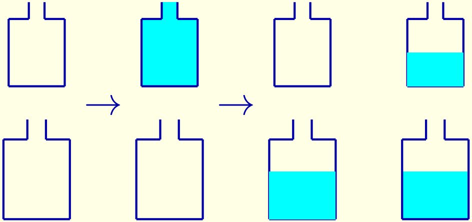
Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

Now they fill the 3 gallon jug.

Fill 3-gal. jug.



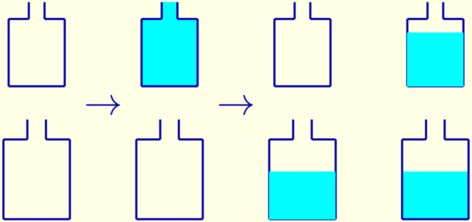
Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

Now they fill the 3 gallon jug.

Fill 3-gal. jug.

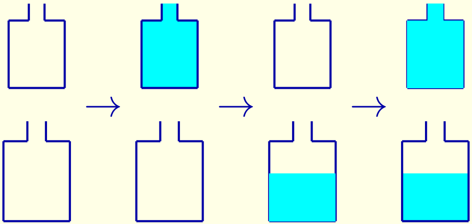


Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

Now they fill the 3 gallon jug.



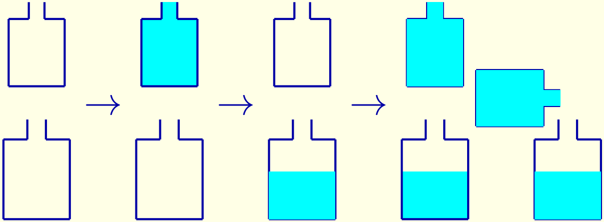
Again, they start with both jugs empty.

The only thing they can do now is fill a jug. Suppose they fill the 3 gallon jug.

Next, they empty the water from the 3 gallon jug into the 5 gallon jug.

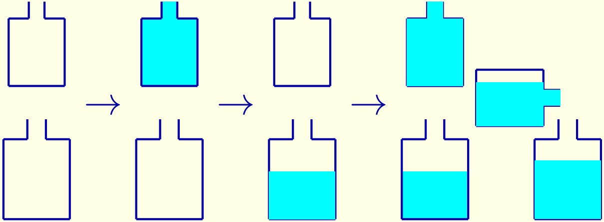
Now they fill the 3 gallon jug.

Fill 5-gal. jug from 3-gal. jug.



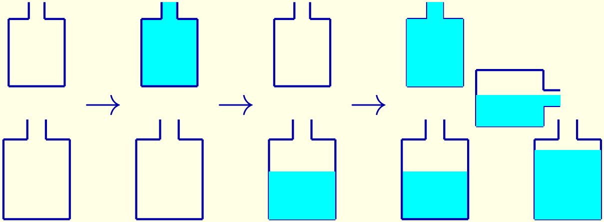
Now they fill the 5 gallon jug from the 3 gallon jug.

Fill 5-gal. jug from 3-gal. jug.



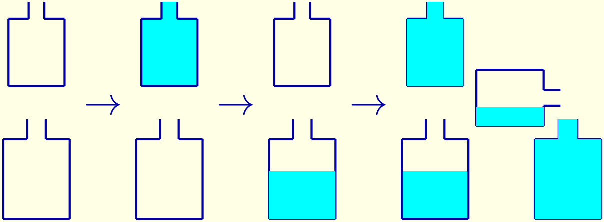
Now they fill the 5 gallon jug from the 3 gallon jug.

Fill 5-gal. jug from 3-gal. jug.

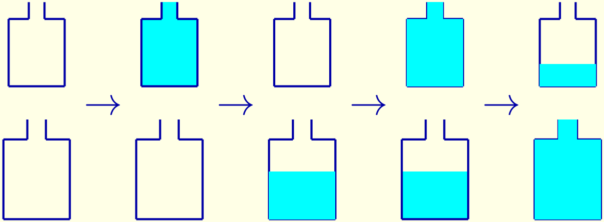


Now they fill the 5 gallon jug from the 3 gallon jug.

Fill 5-gal. jug from 3-gal. jug.

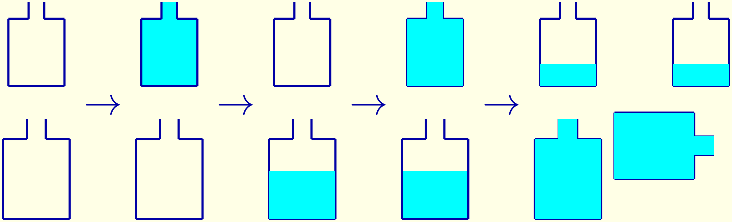


Now they fill the 5 gallon jug from the 3 gallon jug.



Now they fill the 5 gallon jug from the 3 gallon jug.

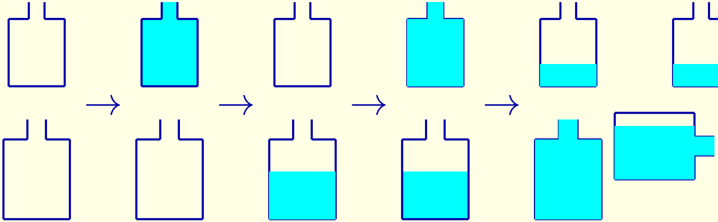
Empty 5-gal. jug.



Now they fill the 5 gallon jug from the 3 gallon jug.

They then empty the 5-gallon jug onto the ground.

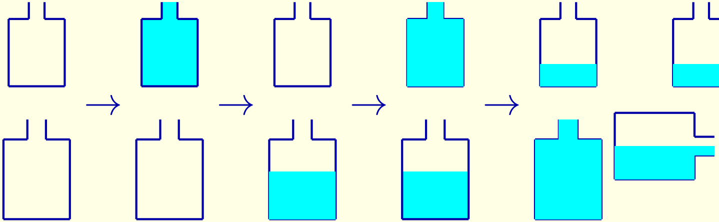
Empty 5-gal. jug.



Now they fill the 5 gallon jug from the 3 gallon jug.

They then empty the 5-gallon jug onto the ground.

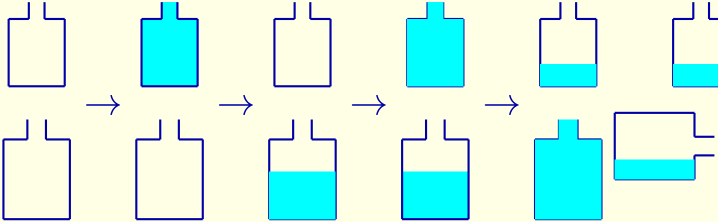
Empty 5-gal. jug.



Now they fill the 5 gallon jug from the 3 gallon jug.

They then empty the 5-gallon jug onto the ground.

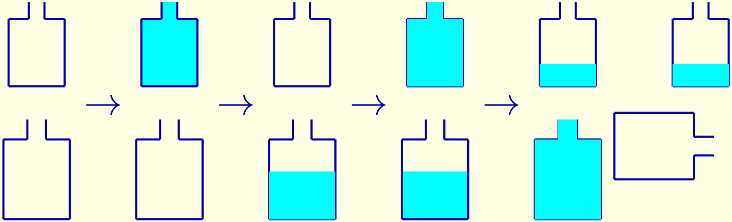
Empty 5-gal. jug.



Now they fill the 5 gallon jug from the 3 gallon jug.

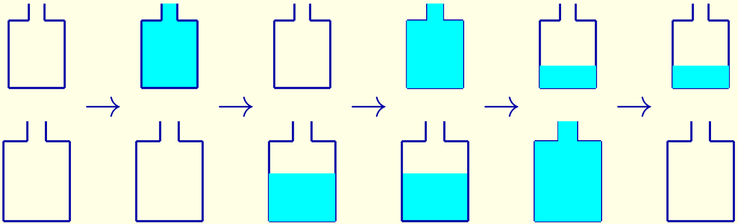
They then empty the 5-gallon jug onto the ground.

Empty 5-gal. jug.



Now they fill the 5 gallon jug from the 3 gallon jug.

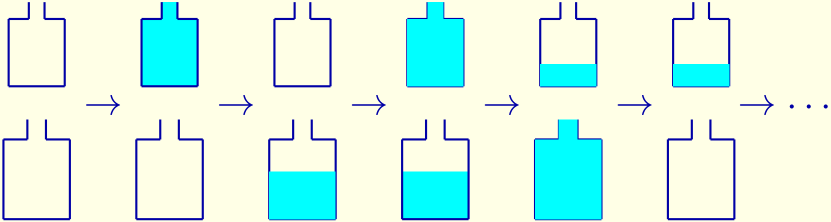
They then empty the 5-gallon jug onto the ground.



Now they fill the 5 gallon jug from the 3 gallon jug.

They then empty the 5-gallon jug onto the ground.

And so on.

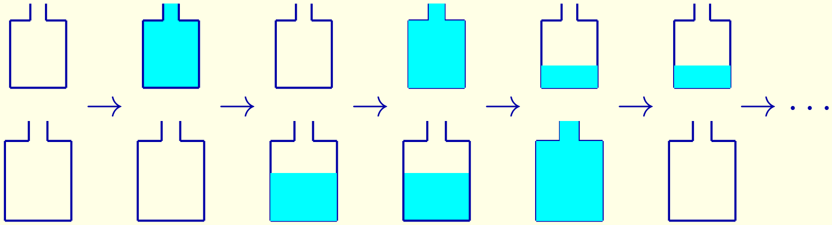


Now they fill the 5 gallon jug from the 3 gallon jug.

They then empty the 5-gallon jug onto the ground.

And so on.

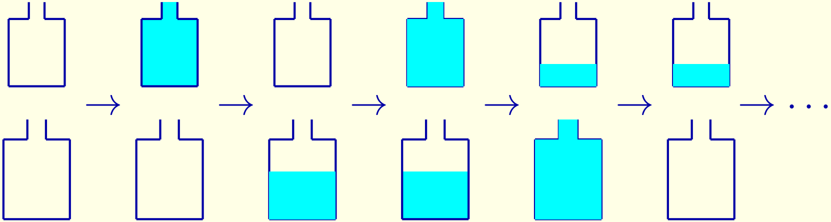
This is just one possible behavior.



This is just one of many possible ways a behavior can begin.

This is just one possible behavior.

Let's write it more formally.



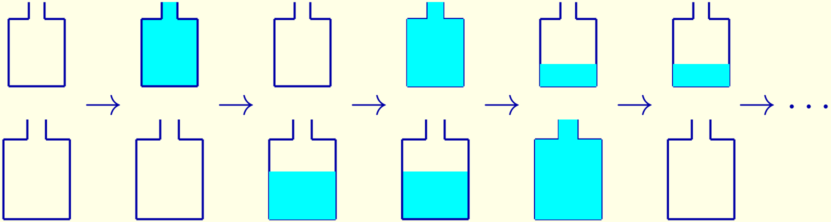
This is just one of many possible ways a behavior can begin.

Let's write it more formally.

This is just one possible behavior.

Let's write it more formally.

Let values of $small$ and big represent number of gallons in each jug.

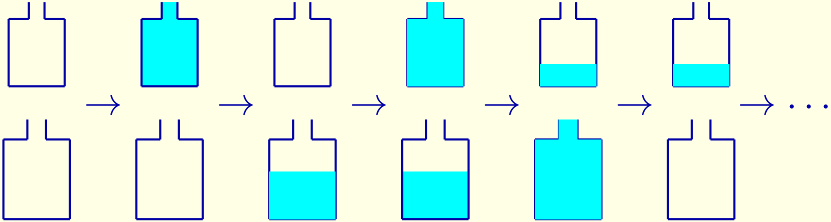


This is just one of many possible ways a behavior can begin.

Let's write it more formally.

Let the values of the variables $small$ and big represent the number of gallons of water in each jug.

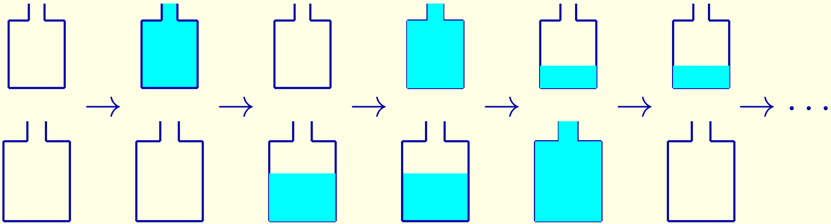
$$\begin{bmatrix} small : 0 \\ big \quad : 0 \end{bmatrix}$$



Initially, both jugs have 0 gallons of water.

Fill small jug.

$\begin{bmatrix} small : 0 \\ big \quad : 0 \end{bmatrix}$ $\begin{bmatrix} small : 3 \\ big \quad : 0 \end{bmatrix}$

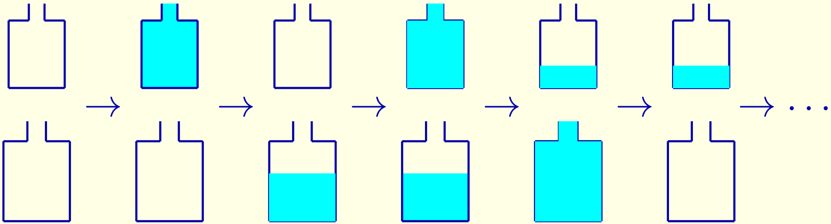


Initially, both jugs have 0 gallons of water.

Filling the small jug puts 3 gallons of water in it.

Empty small jug
into big jug.

$\begin{bmatrix} small: 0 \\ big\ \ \ \ : 0 \end{bmatrix}$ $\begin{bmatrix} small: 3 \\ big\ \ \ \ : 0 \end{bmatrix}$ $\begin{bmatrix} small: 0 \\ big\ \ \ \ : 3 \end{bmatrix}$



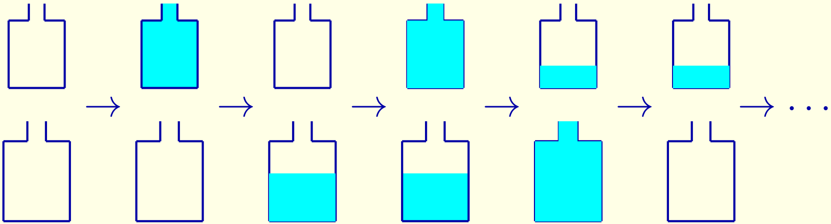
Initially, both jugs have 0 gallons of water.

Filling the small jug puts 3 gallons of water in it.

Those 3 gallons are transferred from the small jug to the big jug.

Fill small jug.

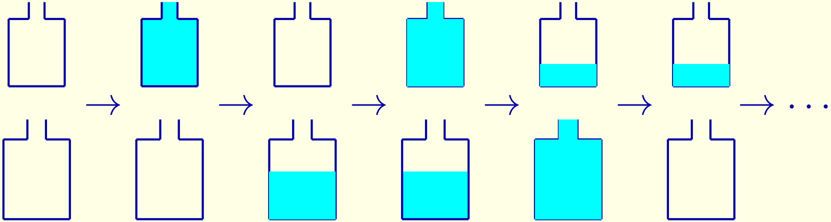
$$\begin{bmatrix} small: 0 \\ big\ \ \ : 0 \end{bmatrix} \quad \begin{bmatrix} small: 3 \\ big\ \ \ : 0 \end{bmatrix} \quad \begin{bmatrix} small: 0 \\ big\ \ \ : 3 \end{bmatrix} \quad \begin{bmatrix} small: 3 \\ big\ \ \ : 3 \end{bmatrix}$$



3 gallons are then added to the small jug.

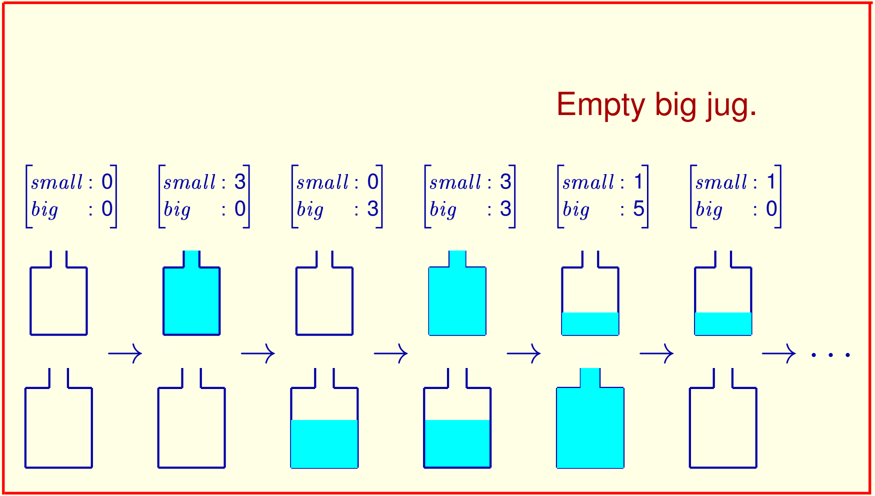
Fill big jug
from small jug.

$$\begin{bmatrix} small: 0 \\ big\ \ : 0 \end{bmatrix} \quad \begin{bmatrix} small: 3 \\ big\ \ : 0 \end{bmatrix} \quad \begin{bmatrix} small: 0 \\ big\ \ : 3 \end{bmatrix} \quad \begin{bmatrix} small: 3 \\ big\ \ : 3 \end{bmatrix} \quad \begin{bmatrix} small: 1 \\ big\ \ : 5 \end{bmatrix}$$



3 gallons are then added to the small jug.

The big jug is then filled from the small jug, putting 5 gallons in the big jug and leaving 1 gallon in the small jug.



3 gallons are then added to the small jug.

The big jug is then filled from the small jug, putting 5 gallons in the big jug and leaving 1 gallon in the small jug.

The big jug is then emptied, leaving 0 gallons in it.

$$\begin{bmatrix} small: 0 \\ big \quad : 0 \end{bmatrix} \to \begin{bmatrix} small: 3 \\ big \quad : 0 \end{bmatrix} \to \begin{bmatrix} small: 0 \\ big \quad : 3 \end{bmatrix} \to \begin{bmatrix} small: 3 \\ big \quad : 3 \end{bmatrix} \to \begin{bmatrix} small: 1 \\ big \quad : 5 \end{bmatrix} \to \begin{bmatrix} small: 1 \\ big \quad : 0 \end{bmatrix} \to \cdots$$

3 gallons are then added to the small jug.

The big jug is then filled from the small jug, putting 5 gallons in the big jug and leaving 1 gallon in the small jug.

The big jug is then emptied, leaving 0 gallons in it.

What did we learn from this behavior?

$$\begin{bmatrix} small: 0 \\ big \quad : 0 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big \quad : 0 \end{bmatrix} \rightarrow \begin{bmatrix} small: 0 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big \quad : 5 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big \quad : 0 \end{bmatrix} \rightarrow \cdots$$

What did we learn by writing this behavior?

We learned two things.

What did we learn from this behavior?

$$\begin{bmatrix} small: 0 \\ big \quad : 0 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big \quad : 0 \end{bmatrix} \rightarrow \begin{bmatrix} small: 0 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big \quad : 5 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big \quad : 0 \end{bmatrix} \rightarrow \cdots$$

1. What the variables are.

What did we learn by writing this behavior?
We learned two things.

First, what the variables are.

What did we learn from this behavior?

$$\begin{bmatrix} small: 0 \\ big \quad : 0 \end{bmatrix} \to \begin{bmatrix} small: 3 \\ big \quad : 0 \end{bmatrix} \to \begin{bmatrix} small: 0 \\ big \quad : 3 \end{bmatrix} \to \begin{bmatrix} small: 3 \\ big \quad : 3 \end{bmatrix} \to \begin{bmatrix} small: 1 \\ big \quad : 5 \end{bmatrix} \to \begin{bmatrix} small: 1 \\ big \quad : 0 \end{bmatrix} \to \cdots$$

1. What the variables are.
2. What constitutes a step.

What did we learn by writing this behavior?
We learned two things.

First, what the variables are.

And second, what constitutes a step. For example. . .

Filling a jug is a single step.

$$\begin{bmatrix} small: 0 \\ big\ \ \ \ : 0 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big\ \ \ \ : 0 \end{bmatrix} \rightarrow \begin{bmatrix} small: 0 \\ big\ \ \ \ : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big\ \ \ \ : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big\ \ \ \ : 5 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big\ \ \ \ : 0 \end{bmatrix} \rightarrow \cdots$$

1. What the variables are.

2. What constitutes a step.

Filling a jug is a single step.

Filling a jug is a single step.

No intermediate states.

$$\begin{bmatrix} small: 0 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 2 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big \quad : 3 \end{bmatrix}$$

1. What the variables are.
2. What constitutes a step.

Filling a jug is a single step.

There's no intermediate partially-filled state or states.

Filling a jug is a single step.

No intermediate states.

$$\begin{bmatrix} small : 0 \\ big : 3 \end{bmatrix} \rightarrow \cancel{\begin{bmatrix} small : 1 \\ big : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small : 2 \\ big : 3 \end{bmatrix}} \rightarrow \begin{bmatrix} small : 3 \\ big : 3 \end{bmatrix}$$

1. What the variables are.

2. What constitutes a step.

Filling a jug is a single step.

There's no intermediate partially-filled state or states.

What did we learn from this behavior?

$$\begin{bmatrix} small: 0 \\ big \ \ \ : 0 \end{bmatrix} \to \begin{bmatrix} small: 3 \\ big \ \ \ : 0 \end{bmatrix} \to \begin{bmatrix} small: 0 \\ big \ \ \ : 3 \end{bmatrix} \to \begin{bmatrix} small: 3 \\ big \ \ \ : 3 \end{bmatrix} \to \begin{bmatrix} small: 1 \\ big \ \ \ : 5 \end{bmatrix} \to \begin{bmatrix} small: 1 \\ big \ \ \ : 0 \end{bmatrix} \to \cdots$$

1. What the variables are.
2. What constitutes a step.

Filling a jug is a single step.

There's no intermediate partially-filled state or states.

Simplest abstraction of real jugs and water

$$\begin{bmatrix} small: 0 \\ big\ \ \ \ : 0 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big\ \ \ \ : 0 \end{bmatrix} \rightarrow \begin{bmatrix} small: 0 \\ big\ \ \ \ : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big\ \ \ \ : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big\ \ \ \ : 5 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big\ \ \ \ : 0 \end{bmatrix} \rightarrow \cdots$$

Filling a jug is a single step.

There's no intermediate partially-filled state or states.

This is the simplest abstraction of the behavior of real jugs and water

Simplest abstraction of real jugs and water

$$\begin{bmatrix} small: 0 \\ big\ \ \ \ : 0 \end{bmatrix} \to \begin{bmatrix} small: 3 \\ big\ \ \ \ : 0 \end{bmatrix} \to \begin{bmatrix} small: 0 \\ big\ \ \ \ : 3 \end{bmatrix} \to \begin{bmatrix} small: 3 \\ big\ \ \ \ : 3 \end{bmatrix} \to \begin{bmatrix} small: 1 \\ big\ \ \ \ : 5 \end{bmatrix} \to \begin{bmatrix} small: 1 \\ big\ \ \ \ : 0 \end{bmatrix} \to \cdots$$

for this problem.

Filling a jug is a single step.

There's no intermediate partially-filled state or states.

This is the simplest abstraction of the behavior of real jugs and water for the particular problem faced by our heroes.

$$\begin{bmatrix} small: 0 \\ big \quad : 0 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big \quad : 0 \end{bmatrix} \rightarrow \begin{bmatrix} small: 0 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big \quad : 5 \end{bmatrix} \rightarrow \begin{bmatrix} small: 1 \\ big \quad : 0 \end{bmatrix} \rightarrow \cdots$$

Real specifications are written to eliminate some kinds of errors.

Real specifications are written for a purpose.
Usually to eliminate some particular kinds of errors.

$$\begin{bmatrix} small : 0 \\ big \quad : 0 \end{bmatrix} \to \begin{bmatrix} small : 3 \\ big \quad : 0 \end{bmatrix} \to \begin{bmatrix} small : 0 \\ big \quad : 3 \end{bmatrix} \to \begin{bmatrix} small : 3 \\ big \quad : 3 \end{bmatrix} \to \begin{bmatrix} small : 1 \\ big \quad : 5 \end{bmatrix} \to \begin{bmatrix} small : 1 \\ big \quad : 0 \end{bmatrix} \to \cdots$$

Real specifications are written to eliminate some kinds of errors.

Like getting blown up.

Real specifications are written for a purpose.
Usually to eliminate some particular kinds of errors.

For example, to avoid getting blown up.

THE SPECIFICATION

We can now start writing the actual TLA+ specification.

MODULE $DieHard$

Beginning of module.

The spec is in a module called $DieHard$.

MODULE *DieHard*

EXTENDS *Integers*

Imports operators of arithmetic.

The spec is in a module called $DieHard$.

As in our $Simple\ Program$ example, the EXTENDS statement imports operators of arithmetic

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

Declares the variables.

The spec is in a module called *DieHard*.

As in our *Simple Program* example, the EXTENDS statement imports operators of arithmetic

and the VARIABLES statement declares our two variables.

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

TLA$^+$ has no type declarations.

The spec is in a module called *DieHard*.

As in our *Simple Program* example, the EXTENDS statement imports operators of arithmetic
and the VARIABLES statement declares our two variables.

In TLA+ we don't write type declarations.

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

TLA$^+$ has no type declarations.

Type correctness means variables have sensible values.

Type correctness means that all the variables have sensible values.

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

We define a formula that asserts type correctness.

Type correctness means that all the variables have sensible values.

It's a good idea to define a formula that asserts type correctness.

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

We define a formula that asserts type correctness.

Helps to understand spec.

Type correctness means that all the variables have sensible values.

It's a good idea to define a formula that asserts type correctness.

It helps a reader to understand the spec.

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

We define a formula that asserts type correctness.

Helps to understand spec.

TLC can check that it's always true.

Type correctness means that all the variables have sensible values.

It's a good idea to define a formula that asserts type correctness.

It helps a reader to understand the spec.

And TLC can type-check the spec by checking that this formula is always true.

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

$TypeOK \triangleq$

We define a formula that asserts type correctness.

Helps to understand spec.

TLC can check that it's always true.

I like to call this formula $TypeOK$.

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

$TypeOK \triangleq \land small \in 0 \,..\, 3$

I like to call this formula $TypeOK$.

It asserts that the value of $small$ is an integer from 0 through 3.

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

$$TypeOK \triangleq \begin{array}{l} \land\ small \in 0 \,..\, 3 \\ \land\ big \;\;\;\; \in 0 \,..\, 5 \end{array}$$

I like to call this formula $TypeOK$.

It asserts that the value of $small$ is an integer from 0 through 3.
and the value of big is an integer from 0 through 5.

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

$$TypeOK \triangleq \begin{array}{l} \land\ small \in 0\,..\,3 \\ \land\ big \in 0\,..\,5 \end{array}$$

This definition is not part of the spec.

I like to call this formula $TypeOK$.

It asserts that the value of $small$ is an integer from 0 through 3.
and the value of big is an integer from 0 through 5.

This definition is not part of the spec.

MODULE *DieHard*

EXTENDS *Integers*

VARIABLES *small*, *big*

$$TypeOK \triangleq \begin{array}{l} \land\ small \in 0\,..\,3 \\ \land\ big \ \ \ \in 0\,..\,5 \end{array}$$

This definition is not part of the spec.

Removing it doesn't change anything.

I like to call this formula $TypeOK$.

It asserts that the value of $small$ is an integer from 0 through 3.
and the value of big is an integer from 0 through 5.

This definition is not part of the spec.

Removing it doesn't change anything.

The Initial-State Formula

The initial-state formula.

The Initial-State Formula

$$Init \triangleq$$

The initial-state formula.

As usual, let's name it $Init$.

The Initial-State Formula

$$
\begin{aligned}
Init \;\triangleq\;\; & \land\; big \;\;\; = 0 \\
& \land\; small = 0
\end{aligned}
$$

The initial-state formula.

As usual, let's name it $Init$.

It asserts that both jugs are empty.

THE NEXT-STATE FORMULA

The next-state formula.

The Next-State Formula

The next-state formula describes all permitted steps.

The next-state formula describes all permitted steps.

The Next-State Formula

The next-state formula describes all permitted steps.

It's usually written as $F_1 \lor F_2 \lor \ldots \lor F_n$

The next-state formula describes all permitted steps.

It's usually written as F1 or F2 or (and so on) ,

The Next-State Formula

The next-state formula describes all permitted steps.

It's usually written as $F_1 \lor F_2 \lor \ldots \lor F_n$,
where each F_i allows a different kind of step.

The next-state formula describes all permitted steps.

It's usually written as F1 or F2 or (and so on) ,

where each formula F allows a different kind of step.

The Next-State Formula

The next-state formula describes all permitted steps.

It's usually written as $F_1 \lor F_2 \lor \ldots \lor F_n$,
where each F_i allows a different kind of step.

The behavior we wrote has 3 kinds of steps:

The behavior we just wrote has 3 different kinds of steps:

The Next-State Formula

The next-state formula describes all permitted steps.

It's usually written as $F_1 \lor F_2 \lor \ldots \lor F_n$,
where each F_i allows a different kind of step.

The behavior we wrote has 3 kinds of steps:

– Fill a jug.

The behavior we just wrote has 3 different kinds of steps:

Steps that fill a jug.

The Next-State Formula

The next-state formula describes all permitted steps.

It's usually written as $F_1 \lor F_2 \lor \ldots \lor F_n$,
where each F_i allows a different kind of step.

The behavior we wrote has 3 kinds of steps:

- Fill a jug.
- Empty a jug.

The behavior we just wrote has 3 different kinds of steps:

Steps that fill a jug.

Steps that empty a jug.

The Next-State Formula

The next-state formula describes all permitted steps.

It's usually written as $F_1 \vee F_2 \vee \ldots \vee F_n$,
where each F_i allows a different kind of step.

The behavior we wrote has 3 kinds of steps:

- Fill a jug.
- Empty a jug.
- Pour from one jug into the other.

The behavior we just wrote has 3 different kinds of steps:

Steps that fill a jug.

Steps that empty a jug.

And steps that pour from one jug into the other.

$$Next \triangleq$$

As usual, we call the next-state formula $Next$.

$Next \triangleq$ – Fill a jug.

As usual, we call the next-state formula $Next$.

First we allow steps that fill a jug.
There are two jugs, so we have two possible kinds of steps.

$$Next \triangleq \lor \boxed{FillSmall} \quad \text{Fill small jug.}$$
$$\qquad\qquad \lor FillBig$$

As usual, we call the next-state formula $Next$.

First we allow steps that fill a jug.
There are two jugs, so we have two possible kinds of steps.

Steps that fill the small jug.

$$Next \triangleq \lor\ FillSmall$$
$$\qquad\qquad \lor\ \boxed{FillBig}$$

Fill big jug.

As usual, we call the next-state formula $Next$.

First we allow steps that fill a jug.
There are two jugs, so we have two possible kinds of steps.

Steps that fill the small jug. And steps that fill the big jug.

$$Next \triangleq \lor FillSmall$$
$$\lor FillBig$$

– Empty a jug.

Similarly for steps that empty a jug.

$$
\begin{aligned}
Next \triangleq\ & \lor\ FillSmall \\
& \lor\ FillBig \\
& \lor\ EmptySmall \\
& \lor\ EmptyBig
\end{aligned}
$$

– Empty a jug.

Similarly for steps that empty a jug.

$$
\begin{aligned}
Next \triangleq\ & \lor\ FillSmall \\
& \lor\ FillBig \\
& \lor\ EmptySmall \\
& \lor\ EmptyBig
\end{aligned}
$$

– Pour from one jug
into the other.

Similarly for steps that empty a jug.

And there are two kinds of steps that pour from one jug to the other.

$$
\begin{aligned}
Next \;\triangleq\;\; & \lor\; FillSmall \\
& \lor\; FillBig \\
& \lor\; EmptySmall \\
& \lor\; EmptyBig \\
& \lor\; \boxed{SmallToBig} \quad \text{From small jug to big jug.} \\
& \lor\; BigToSmall
\end{aligned}
$$

Similarly for steps that empty a jug.

And there are two kinds of steps that pour from one jug to the other.

Steps that pour from the small jug to the big jug.

$$
\begin{aligned}
Next \triangleq\ & \lor\ FillSmall \\
& \lor\ FillBig \\
& \lor\ EmptySmall \\
& \lor\ EmptyBig \\
& \lor\ SmallToBig \\
& \lor\ \boxed{BigToSmall} \quad \text{From big jug to small jug.}
\end{aligned}
$$

Similarly for steps that empty a jug.

And there are two kinds of steps that pour from one jug to the other.

Steps that pour from the small jug to the big jug.

And steps that pour from the big jug to the small jug.

$$
\begin{aligned}
Next \;\triangleq\;\; & \lor\ FillSmall \\
& \lor\ FillBig \\
& \lor\ EmptySmall \\
& \lor\ EmptyBig \\
& \lor\ SmallToBig \\
& \lor\ BigToSmall
\end{aligned}
$$

$$Next \triangleq \lor FillSmall$$
$$\lor FillBig$$
$$\lor EmptySmall$$
$$\lor EmptyBig$$
$$\lor SmallToBig$$
$$\lor BigToSmall$$

Names must be defined before they are used.

In TLA+, names must be defined before they're used.

$$
\begin{aligned}
Next \triangleq\ & \lor \boxed{\begin{array}{l} FillSmall \\ FillBig \\ EmptySmall \\ EmptyBig \\ SmallToBig \\ BigToSmall \end{array}}
\end{aligned}
$$

Names must be defined before they are used.

The definitions of these names must precede this definition of $Next$.

In TLA+, names must be defined before they're used.

The definitions of $FillSmall$, $FillBig$, etc. must precede this definition of $Next$.

$FillSmall \triangleq$

We now define $FillSmall$.

Most people first learning TLA+ would write this definition.

$$FillSmall \triangleq small' = 3$$

Most people would write this definition.

We now define $FillSmall$.

Most people first learning TLA+ would write this definition.

$$FillSmall \triangleq small' = 3$$

Most people would write this definition.

Stop the video now and figure out
why it's wrong.

We now define $FillSmall$.

Most people first learning TLA$^+$ would write this definition.

Stop the video now and figure out why it's wrong.

$FillSmall \triangleq \boxed{small' = 3}$

If you didn't figure it out, you're thinking of this as setting $small$ to 3.

We now define $FillSmall$.

Most people first learning TLA+ would write this definition.

Stop the video now and figure out why it's wrong.

If you didn't figure it out, it means that you're thinking of this as an assignment statement that sets $small$ to 3. It's not.

$FillSmall \triangleq \boxed{small' = 3}$

If you didn't figure it out, you're thinking of this as setting $small$ to 3.

It's a formula that's true for some steps and false for others.

It's a formula that's true for some steps and false for others.

$FillSmall \triangleq \boxed{small' = 3}$

If you didn't figure it out, you're thinking of this as setting $small$ to 3.

It's a formula that's true for some steps and false for others.

It's true for any step in which $small = 3$ in the second state.

It's a formula that's true for some steps and false for others.

It's true for any step in which the value of $small$ in the second state is 3.

$$FillSmall \triangleq \boxed{small' = 3}$$

It's a formula that's true for some steps and false for others.

It's true for any step in which the value of $small$ in the second state is 3.

$$FillSmall \triangleq \boxed{small' = 3}$$

$$\begin{bmatrix} small : 0 \\ big \ \ \ \ : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small : 3 \\ big \ \ \ \ : 3 \end{bmatrix}$$

It's a formula that's true for some steps and false for others.

It's true for any step in which the value of $small$ in the second state is 3.

It's true for this step that appeared in the behavior we constructed.

$$FillSmall \triangleq \boxed{small' = 3}$$

$$\begin{bmatrix} small : 0 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small : 3 \\ big \quad : 3 \end{bmatrix}$$

$$\begin{bmatrix} small : 0 \\ big \quad : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small : 3 \\ big \quad : \sqrt{7} \end{bmatrix}$$

It's also true for this step in which big equals the square root of 7 in the second state.

$$FillSmall \triangleq \boxed{small' = 3}$$

$$\begin{bmatrix} small: 0 \\ big \;\;\;: 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big \;\;\;: 3 \end{bmatrix}$$

$$\begin{bmatrix} small: 0 \\ big \;\;\;: 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big \;\;\;: \sqrt{7} \end{bmatrix}$$

$$\begin{bmatrix} small: 0 \\ big \;\;\;: 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big \;\;\;: \text{“}abc\text{”} \end{bmatrix}$$

It's also true for this step in which big equals the square root of 7 in the second state.

And it's also true for this step in which big equals the string abc in the second state.

$$FillSmall \triangleq \boxed{small' = 3}$$

$$\begin{bmatrix} small: 0 \\ big\ \ : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big\ \ : 3 \end{bmatrix}$$

$$\boxed{\begin{matrix} \begin{bmatrix} small: 0 \\ big\ \ : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big\ \ : \sqrt{7} \end{bmatrix} \\ \begin{bmatrix} small: 0 \\ big\ \ : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small: 3 \\ big\ \ : \text{“}abc\text{”} \end{bmatrix} \end{matrix}}$$

Should be false for these.

Of course, these two steps shouldn't be allowed, so $FillSmall$ should equal false for them.

$$FillSmall \triangleq \begin{array}{l} \land\ small' = 3 \\ \land\ big' \quad = big \end{array}$$

This is the correct definition.

Of course, these two steps shouldn't be allowed, so $FillSmall$ should equal false for them.

And the correct definition should require the value of big to be unchanged.

$$FillSmall \triangleq \begin{array}{l} \land\ small' = 3 \\ \boxed{\land\ big' \quad = big} \end{array}$$

Most people think this shouldn't be needed.

When they first see TLA+, most computer engineers and computer scientists think that this part of the formula shouldn't be needed.

And that you shouldn't have to say what's left unchanged.

My years of experience writing specifications and a couple of thousand years of mathematics say

$$FillSmall \triangleq \land small' = 3$$
$$\boxed{\land big' = big}$$

Most people think this shouldn't be needed.

That's a bad idea!

When they first see TLA+, most computer engineers and computer scientists think that this part of the formula shouldn't be needed.

And that you shouldn't have to say what's left unchanged.

My years of experience writing specifications and a couple of thousand years of mathematics say that's a bad idea.

$$FillSmall \triangleq \land small' = 3 \quad \boxed{\land big' = big}$$

Most people think this shouldn't be needed.

That's a bad idea! It's not math.

It would leave the simple, elegant realm of mathematics — and enter the more complicated world of programming languages.

$$
\begin{aligned}
FillSmall \;\triangleq\;& \land\; small' = 3 \\
& \land\; big' \;\;\;\; = big
\end{aligned}
$$

$$
\boxed{\begin{aligned}
FillBig \;\triangleq\;& \land\; big' \;\;\;\; = 5 \\
& \land\; small' = small
\end{aligned}}
$$

The definition of $FillBig$ is similar.

The definition of $FillBig$ is similar.

POURING BETWEEN JUGS

Pouring from one jug into another.

$$SmallToBig \triangleq$$

We now define $SmallToBig$.

$SmallToBig \triangleq$

There are two cases:

We now define $SmallToBig$.

In the behavior we constructed, we saw that there are two cases:

$SmallToBig \triangleq$

There are two cases:

1. There is room in big for the water in $small$.

We now define $SmallToBig$.

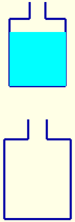
In the behavior we constructed, we saw that there are two cases:

In case 1, there **is** room in the big jug for all the water in the $small$ jug.

$SmallToBig \triangleq$

There are two cases:

1. There is room in big for the water in $small$.



We now define $SmallToBig$.

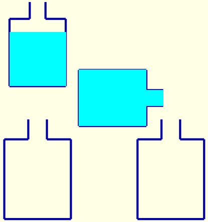
In the behavior we constructed, we saw that there are two cases:

In case 1, there **is** room in the big jug for all the water in the $small$ jug.
Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There **is** room in big for the water in $small$.



We now define $SmallToBig$.

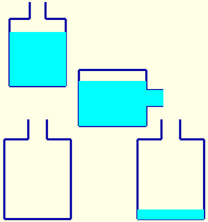
In the behavior we constructed, we saw that there are two cases:

In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There **is** room in big for the water in $small$.



We now define $SmallToBig$.

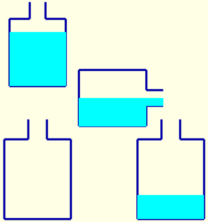
In the behavior we constructed, we saw that there are two cases:

In case 1, there **is** room in the big jug for all the water in the $small$ jug.
Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There **is** room in big for the water in $small$.



We now define $SmallToBig$.

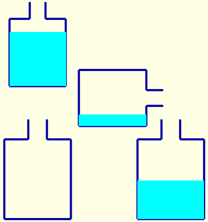
In the behavior we constructed, we saw that there are two cases:

In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There **is** room in big for the water in $small$.



We now define $SmallToBig$.

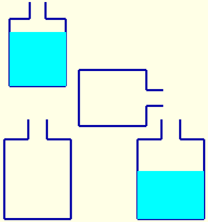
In the behavior we constructed, we saw that there are two cases:

In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There is room in big for the water in $small$.



We now define $SmallToBig$.

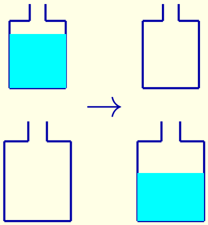
In the behavior we constructed, we saw that there are two cases:

In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There **is** room in big for the water in $small$.



We now define $SmallToBig$.

In the behavior we constructed, we saw that there are two cases:

In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There is room in big for the water in $small$.
2. There isn't room in big for the water in $small$.

We now define $SmallToBig$.

In the behavior we constructed, we saw that there are two cases:

In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

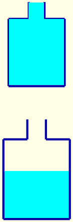
In case 2, there **isn't** room in the big jug for all the water in the $small$ jug.

$SmallToBig \triangleq$

There are two cases:

1. There is room in big for the water in $small$.

2. There isn't room in big for the water in $small$.



We now define $SmallToBig$.

In the behavior we constructed, we saw that there are two cases:

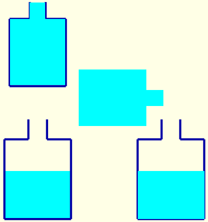
In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

In case 2, there **isn't** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There **is** room in big for the water in $small$.
2. There **isn't** room in big for the water in $small$.



We now define $SmallToBig$.

In the behavior we constructed, we saw that there are two cases:

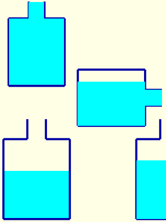
In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

In case 2, there **isn't** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There is room in big for the water in $small$.
2. There isn't room in big for the water in $small$.



We now define $SmallToBig$.

In the behavior we constructed, we saw that there are two cases:

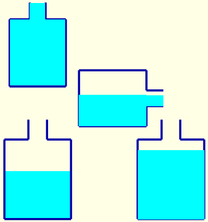
In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

In case 2, there **isn't** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There is room in big for the water in $small$.
2. There isn't room in big for the water in $small$.



We now define $SmallToBig$.

In the behavior we constructed, we saw that there are two cases:

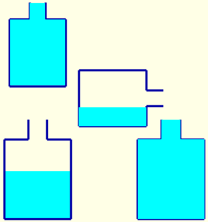
In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

In case 2, there **isn't** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There **is** room in big for the water in $small$.
2. There **isn't** room in big for the water in $small$.



We now define $SmallToBig$.

In the behavior we constructed, we saw that there are two cases:

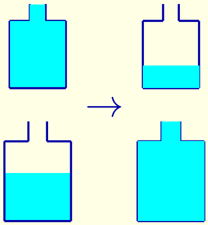
In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

In case 2, there **isn't** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There is room in big for the water in $small$.
2. There isn't room in big for the water in $small$.



We now define $SmallToBig$.

In the behavior we constructed, we saw that there are two cases:

In case 1, there **is** room in the big jug for all the water in the $small$ jug. Here was that case.

In case 2, there **isn't** room in the big jug for all the water in the $small$ jug. Here was that case.

$SmallToBig \triangleq$

There are two cases:

1. There is room in big for the water in $small$.
2. There isn't room in big for the water in $small$.

$$SmallToBig \triangleq \text{IF } big + small \leq 5$$

THEN 1. There is room: empty $small$.

ELSE 2. There isn't room: fill big.

There are two cases:

1. There is room in big for the water in $small$.
2. There isn't room in big for the water in $small$.

Which case it is depends on the total amount of water in the two jugs.

$$SmallToBig \triangleq \text{IF } big + small \leq 5$$
$$\text{THEN } \land\, big' = big + small$$

ELSE 2. There isn't room: fill big.

Put water from $small$ into big

Which case it is depends on the total amount of water in the two jugs.

In case 1, we put all the water from the small jug into the big jug,

$$
\begin{array}{lll}
SmallToBig \triangleq & \text{IF} & big + small \leq 5 \\
& \quad\text{THEN} & \land\ big' \quad = big + small \\
& & \land\ small' = 0 \\
& \quad\text{ELSE} & \text{2. There isn't room: fill } big\text{.}
\end{array}
$$

Put water from $small$ into big, emptying $small$.

Which case it is depends on the total amount of water in the two jugs.

In case 1, we put all the water from the small jug into the big jug, which empties the small jug.

$$
\begin{array}{rl}
SmallToBig \triangleq & \text{IF } big + small \leq 5 \\
& \quad \text{THEN } \land\, big' \quad = big + small \\
& \qquad\quad\;\; \land\, small' = 0 \\
& \quad \text{ELSE } \;\, \land\, big' \quad = \\
& \qquad\quad\;\; \land\, small' =
\end{array}
$$

Which case it is depends on the total amount of water in the two jugs.

In case 1, we put all the water from the small jug into the big jug, which empties the small jug.

Case 2

$$
\begin{array}{lll}
SmallToBig \triangleq \text{IF } big + small \leq 5 & & \\
\quad \text{THEN} & \land\ big' & = big + small \\
 & \land\ small' & = 0 \\
\quad \text{ELSE} & \land\ big' & = \,? \\
 & \land\ small' & = \,?
\end{array}
$$

Problem: Complete the definition of $SmallToBig$

Which case it is depends on the total amount of water in the two jugs.

In case 1, we put all the water from the small jug into the big jug, which empties the small jug.

Case 2 is left as a problem.

$$
\begin{array}{llll}
SmallToBig \triangleq & \text{IF } big + small \leq 5 & & \\
& \quad \text{THEN} & \land\ big' & = big + small \\
& & \land\ small' & = 0 \\
& \quad \text{ELSE} & \land\ big' & = \ ? \\
& & \land\ small' & = \ ?
\end{array}
$$

Problem: Complete the definition of $SmallToBig$ and write the definition of $BigToSmall$.

Which case it is depends on the total amount of water in the two jugs.

In case 1, we put all the water from the small jug into the big jug, which empties the small jug.

Case 2 is left as a problem. As is writing the definition of $BigToSmall$.

$$
\begin{array}{lll}
SmallToBig \triangleq & \text{IF } big + small \leq 5 & \\
& \text{THEN } \land\; big' & = big + small \\
& \quad\quad\;\; \land\; small' & = 0 \\
& \text{ELSE } \;\land\; big' & = \;? \\
& \quad\quad\;\; \land\; small' & = \;?
\end{array}
$$

Problem: Complete the definition of $SmallToBig$ and write the definition of $BigToSmall$.

Stop the video and solve it now.

Stop the video and solve it now, writing down your solution.

$$
\begin{array}{lll}
SmallToBig \triangleq & \text{IF } big + small \leq 5 & \\
& \quad \text{THEN } \land\ big' = big + small \\
& \quad\quad\quad\quad\ \ \land\ small' = 0 \\
& \quad \text{ELSE } \ \land\ big' = ? \\
& \quad\quad\quad\quad\ \ \land\ small' = ?
\end{array}
$$

Problem: Complete the definition of $SmallToBig$ and write the definition of $BigToSmall$.

Stop the video and solve it now.

You'll check your solution later.

Stop the video and solve it now, writing down your solution.

You'll check your solution later.

SAVING OUR HEROES

We'll now use TLC to save our heroes.

Open the Toolbox.

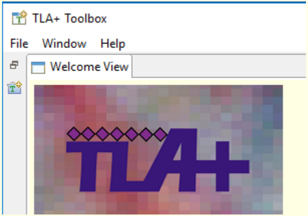
Open the Toolbox.

Open the Toolbox.

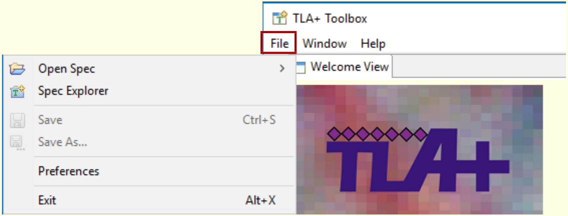
Open a new spec named $DieHard$.

Open the Toolbox.

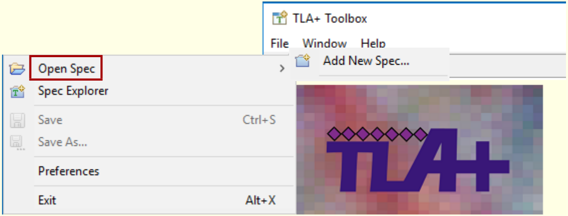
And then open a new spec named $DieHard$.



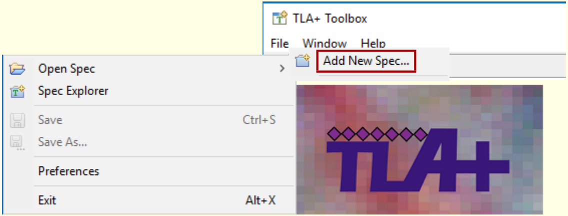
Remember you click on



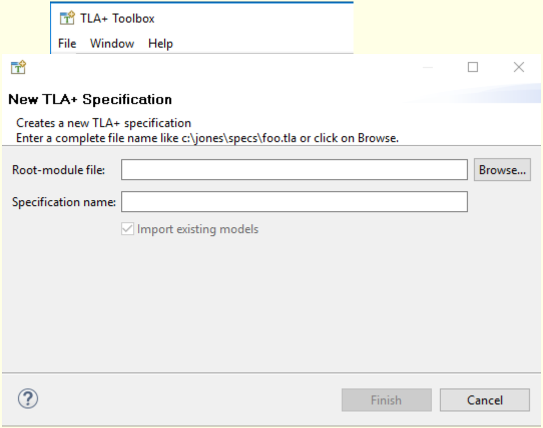
Remember you click on $File$. Then on



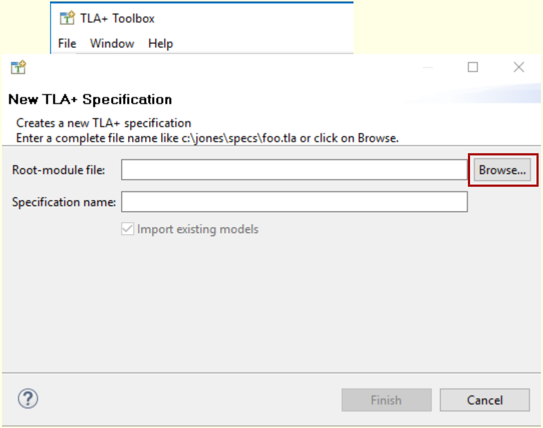
Remember you click on *File*. Then on *Open Spec*. Then on



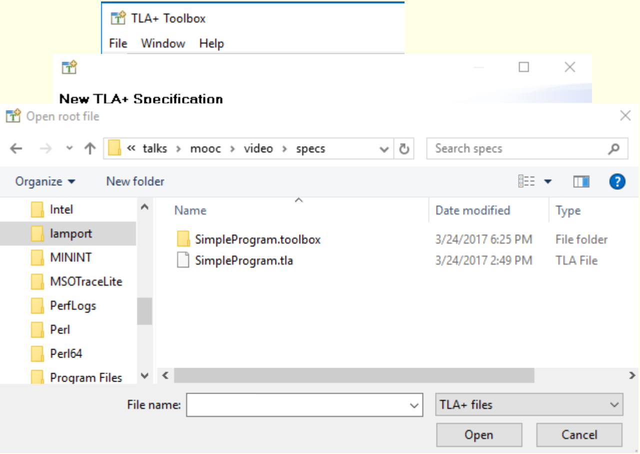
Remember you click on *File*. Then on *Open Spec*. Then on *Add New Spec*, which opens



Remember you click on *File*. Then on *Open Spec*. Then on *Add New Spec*, which opens This window. Then click on

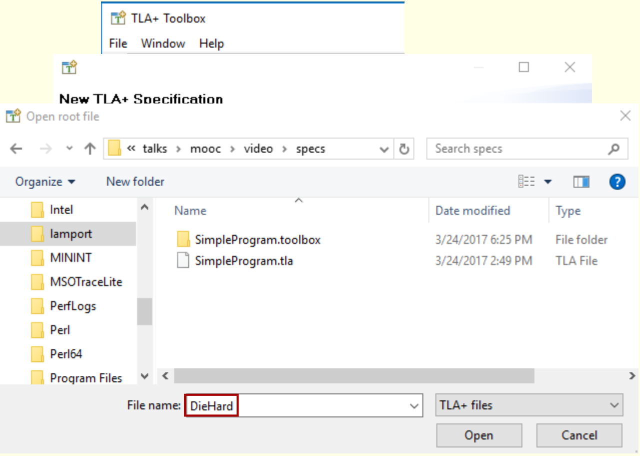


Remember you click on *File*. Then on *Open Spec*. Then on *Add New Spec*, which opens This window. Then click on *Browse* which raises

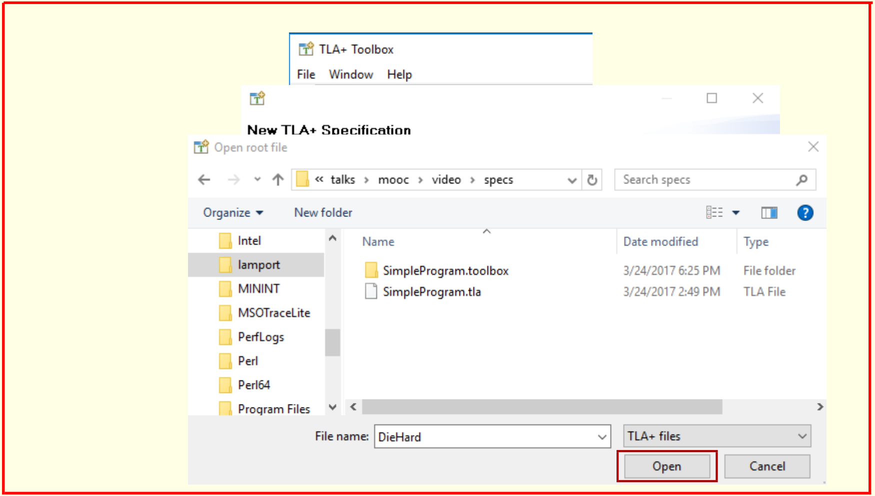


Remember you click on *File*. Then on *Open Spec*. Then on *Add New Spec*, which opens This window. Then click on *Browse* which raises

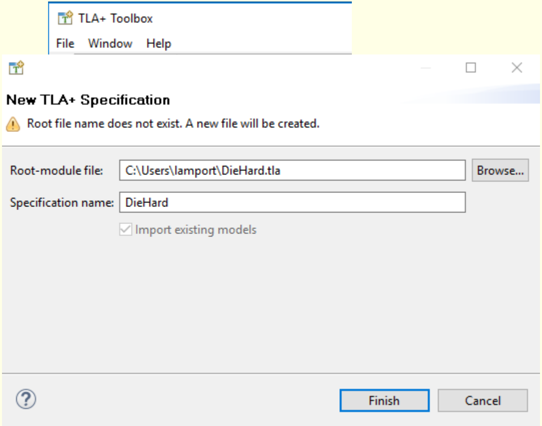
a file browser window—probably on the folder in which you put the *SimpleProgram* spec. Select any folder and enter



Remember you click on $File$. Then on $Open\ Spec$. Then on $Add\ New\ Spec$, which opens This window. Then click on $Browse$ which raises a file browser window—probably on the folder in which you put the $SimpleProgram$ spec. Select any folder and enter the file name $DieHard$. Then click on

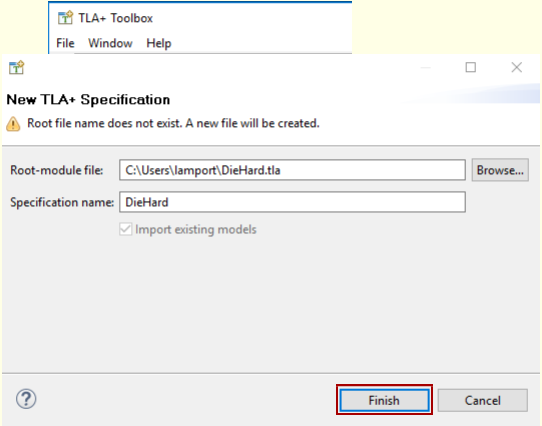


Remember you click on *File*. Then on *Open Spec*. Then on *Add New Spec*, which opens This window. Then click on *Browse* which raises
a file browser window—probably on the folder in which you put the *SimpleProgram* spec. Select any folder and enter the file name *DieHard*. Then click on *Open*



Remember you click on *File*. Then on *Open Spec*. Then on *Add New Spec*, which opens This window. Then click on *Browse* which raises
a file browser window—probably on the folder in which you put the *SimpleProgram* spec. Select any folder and enter the file name *DieHard*.
Then click on *Open* and then on

[slide 171]



Remember you click on *File*. Then on *Open Spec*. Then on *Add New Spec*, which opens This window. Then click on *Browse* which raises
a file browser window—probably on the folder in which you put the *SimpleProgram* spec. Select any folder and enter the file name *DieHard*. Then click on *Open* and then on *Finish* which opens

```
---------------------------- MODULE DieHard ----------------------------


========================================================================
```

An empty spec named diehard.

```
------------------------------ MODULE DieHard ------------------------------


=============================================================================
```

Stop the video and copy the body of the spec.

An empty spec named diehard.

Stop the video now and copy the body of the specification that we just wrote.

```
----------------------------- MODULE DieHard -----------------------------


==========================================================================
```

Stop the video and copy the body of the spec.

Paste it here.

An empty spec named diehard.

Stop the video now and copy the body of the specification that we just wrote.

and then paste the text in the module here.

```
------------------------------- MODULE DieHard -------------------------------
EXTENDS Integers

VARIABLES small, big

TypeOK == /\ small \in 0..3
          /\ big   \in 0..5

Init == /\ big   = 0
        /\ small = 0

FillSmall == /\ small' = 3
             /\ big'   = big

FillBig == /\ big'   = 5
           /\ small' = small

EmptySmall == /\ small' = 0
              /\ big'   = big

EmptyBig == /\ big'   = 0
            /\ small' = small
```

And here’s what you should see.

```
------------------------------ MODULE DieHard ------------------------------
EXTENDS Integers

VARIABLES small, big

TypeOK == /\ small \in 0..3
          /\ big   \in 0..5

Init == /\ big   = 0
        /\ small = 0

FillSmall == /\ small' = 3
             /\ big'   = big

FillBig == /\ big'   = 5
           /\ small' = small

EmptySmall == /\ small' = 0
              /\ big'   = big

EmptyBig == /\ big'   = 0
            /\ small' = small
```

The module contains the complete definitions of $SmallToBig$ and $BigToSmall$.

But don't look at them yet.

And here's what you should see.

The module contains the complete definitions of $SmallToBig$ and $BigToSmall$

```
------------------------------ MODULE DieHard ------------------------------
EXTENDS Integers

VARIABLES small, big

TypeOK == /\ small \in 0..3
          /\ big   \in 0..5

Init == /\ big   = 0
        /\ small = 0

FillSmall == /\ small' = 3
             /\ big'   = big

FillBig == /\ big'   = 5
           /\ small' = small

EmptySmall == /\ small' = 0
              /\ big'   = big

EmptyBig == /\ big'   = 0
            /\ small' = small
```

The module contains the complete definitions of *SmallToBig* and *BigToSmall*.

But don't look at them yet.

And here's what you should see.

The module contains the complete definitions of *SmallToBig* and *BigToSmall* But don't look at them until after we've saved our heroes.

```
------------------------------ MODULE DieHard ------------------------------
EXTENDS Integers

VARIABLES small, big

TypeOK == /\ small \in 0..3
          /\ big   \in 0..5

Init == /\ big   = 0
        /\ small = 0

FillSmall == /\ small' = 3
             /\ big'   = big

FillBig == /\ big'   = 5
           /\ small' = small

EmptySmall == /\ small' = 0
              /\ big'   = big

EmptyBig == /\ big'   = 0
            /\ small' = small
```

Save the module.

(Type Ctl+S.)

And here's what you should see.

The module contains the complete definitions of $SmallToBig$ and $BigToSmall$ But don't look at them until after we've saved our heroes.

And first, you have to save the module – which you can do

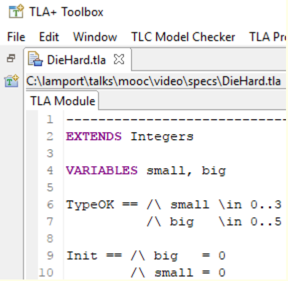
```
------------------------------ MODULE DieHard ------------------------------
EXTENDS Integers

VARIABLES small, big

TypeOK == /\ small \in 0..3
          /\ big   \in 0..5

Init == /\ big   = 0
        /\ small = 0

FillSmall == /\ small' = 3
             /\ big'   = big

FillBig == /\ big'   = 5
           /\ small' = small

EmptySmall == /\ small' = 0
              /\ big'   = big

EmptyBig == /\ big'   = 0
            /\ small' = small
```

Save the module.
(Type `Ctl+S`.)

And here's what you should see.

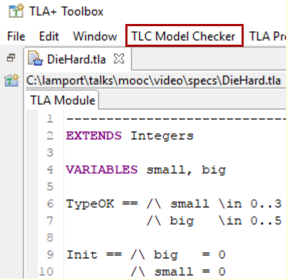
The module contains the complete definitions of $SmallToBig$ and $BigToSmall$ But don't look at them until after we've saved our heroes.

And first, you have to save the module – which you can do by typing Control S.



To run TLC, we create a model.

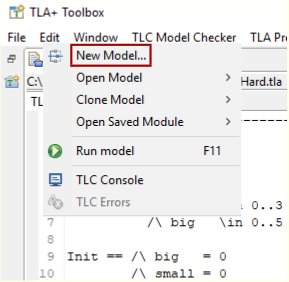
To run TLC, we create a model by



To run TLC, we create a model.

To run TLC, we create a model by

Clicking on the TLC Model Checker menu



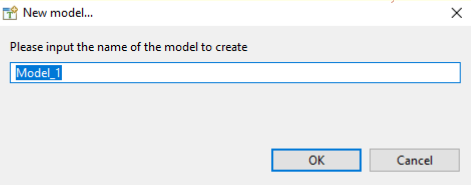
To run TLC, we create a model.

To run TLC, we create a model by

Clicking on the TLC Model Checker menu

Selecting $NewModel$

To run TLC, we create a model.



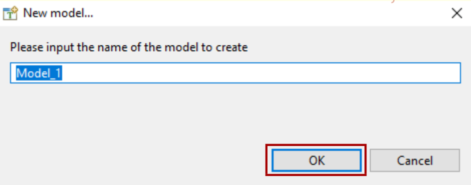
To run TLC, we create a model by

Clicking on the TLC Model Checker menu

Selecting $NewModel$

Entering a model name and

To run TLC, we create a model.



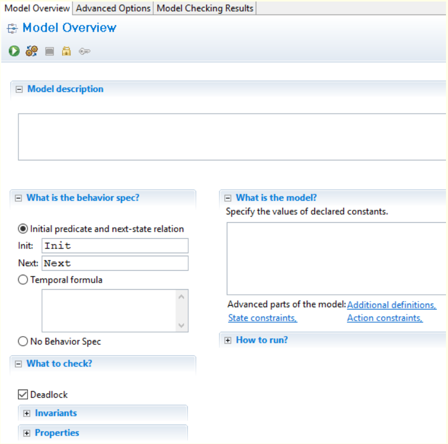
To run TLC, we create a model by

Clicking on the TLC Model Checker menu

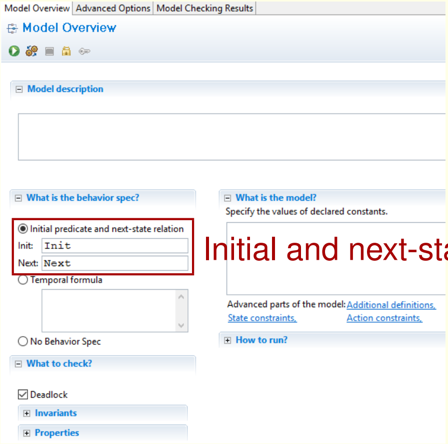
Selecting $NewModel$

Entering a model name and

Clicking OK.



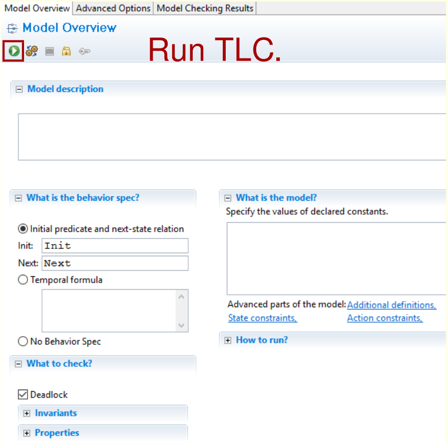
This raises the Model Overview page



Initial and next-state formulas

This raises the Model Overview page

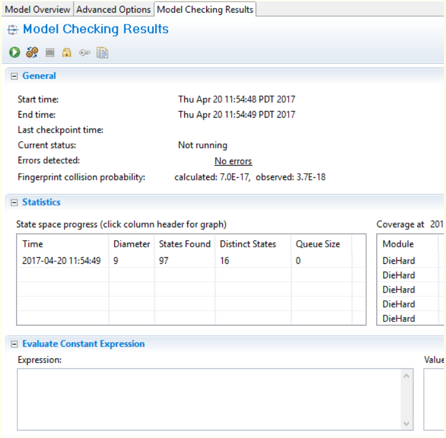
Where the Toolbox has filled in the initial formula and the next-state formula.



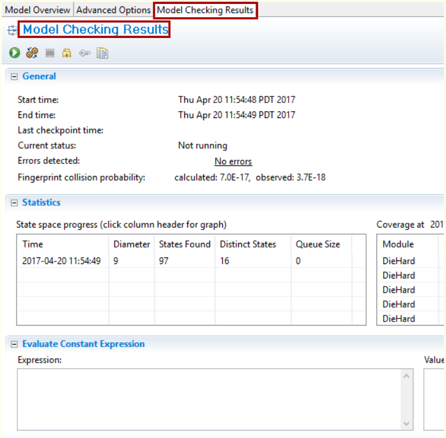
This raises the Model Overview page

Where the Toolbox has filled in the initial formula and the next-state formula.

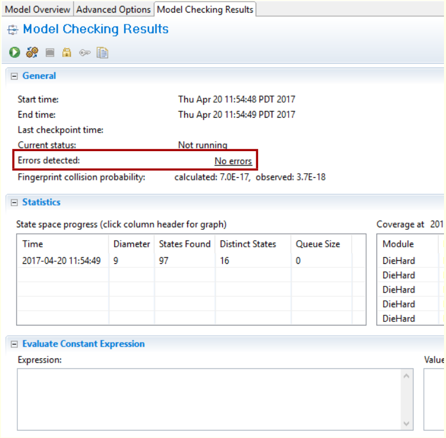
Let’s now run TLC by clicking on this button.



TLC quickly finishes, displaying the

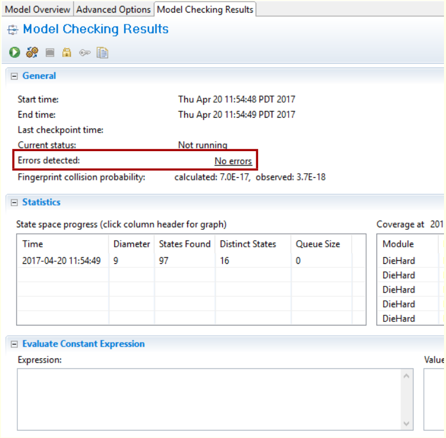


TLC quickly finishes, displaying the *Model Checking Results* page



TLC reports no errors.

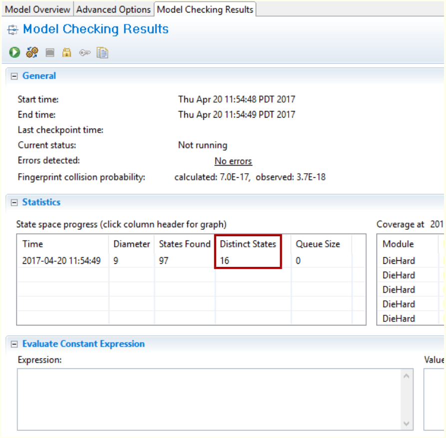
TLC quickly finishes, displaying the *Model Checking Results* page which reports that it found no errors. We didn't ask TLC to check anything, so



TLC reports no errors.

This means it could run the spec.

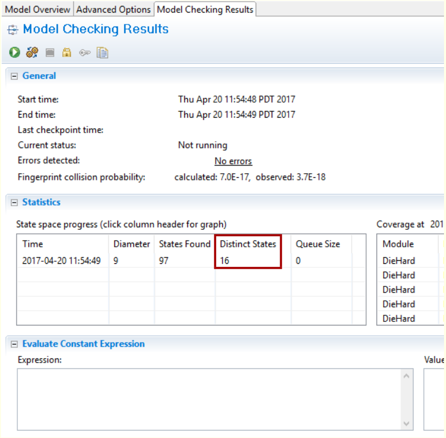
TLC quickly finishes, displaying the *Model Checking Results* page which reports that it found no errors. We didn't ask TLC to check anything, so this just means that the spec is one that it could execute.



TLC found 16
reachable states.

TLC quickly finishes, displaying the *Model Checking Results* page which reports that it found no errors. We didn't ask TLC to check anything, so this just means that the spec is one that it could execute.

TLC also reports that it found 16 reachable states

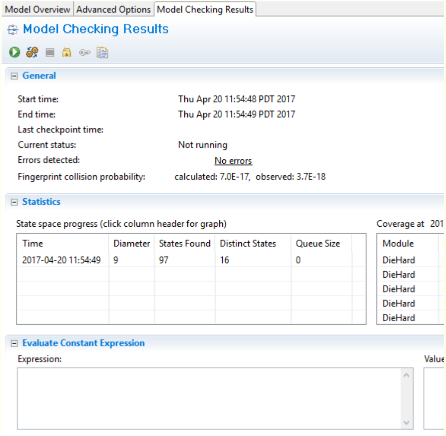


TLC found 16 reachable states.

(States occurring in some behavior allowed by spec.)

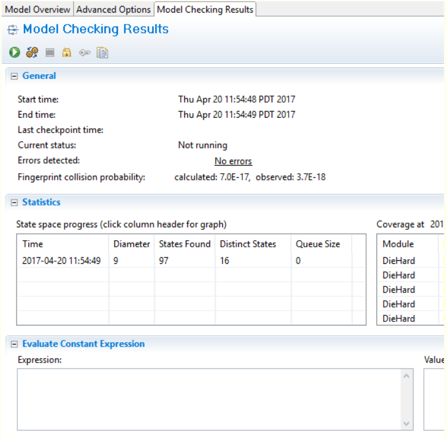
TLC quickly finishes, displaying the *Model Checking Results* page which reports that it found no errors. We didn't ask TLC to check anything, so this just means that the spec is one that it could execute.

TLC also reports that it found 16 reachable states which are states that occur in some behavior allowed by the spec.



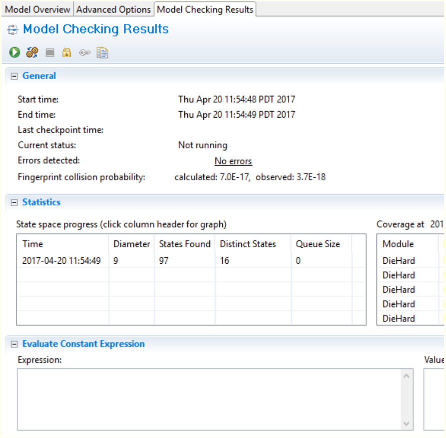
Let’s check type correctness

Let’s now check type correctness – which means



Let’s check type correctness (every reachable state satisfies $TypeOK$).

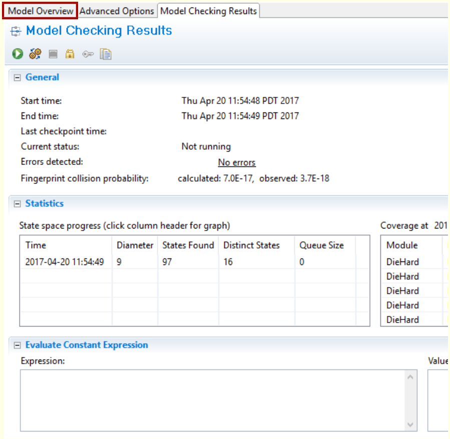
Let’s now check type correctness – which means that every reachable state satisfies formula $TypeOK$.



Let's check type correctness (every reachable state satisfies $TypeOK$).

Let's now check type correctness – which means that every reachable state satisfies formula $TypeOK$.

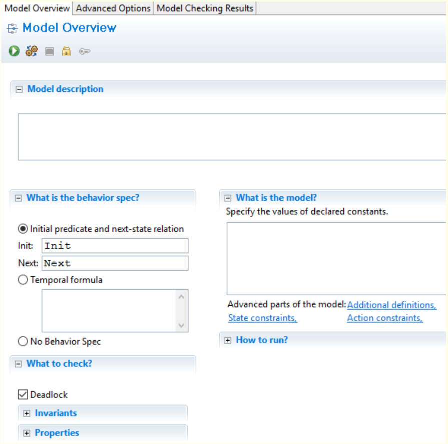
Remember that this formula asserts that each variable has a reasonable value.

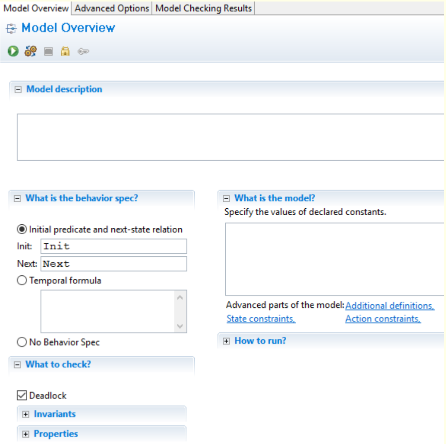


Let's now check type correctness – which means that every reachable state satisfies formula $TypeOK$.

Remember that this formula asserts that each variable has a reasonable value.

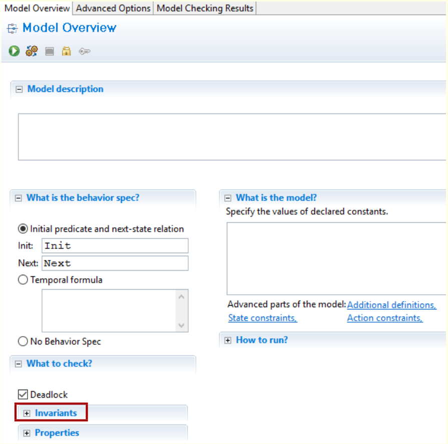
To do this, we must go back to the *Model Overview* page.


Model Overview
Advanced Options
Model Checking Results
Model Overview
Model description
What is the behavior spec?
Initial predicate and next-state relation
Init: Init
Next: Next
Temporal formula
No Behavior Spec
What to check?
Deadlock
Invariants
Properties
What is the model?
Specify the values of declared constants.
Advanced parts of the model: Additional definitions,
State constraints,
Action constraints,
How to run?

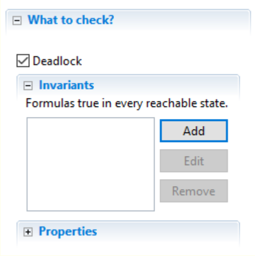


A formula true in every reachable state is called an *invariant*.

A formula that is true in every reachable state is called an *invariant*. To have TLC check an invariant

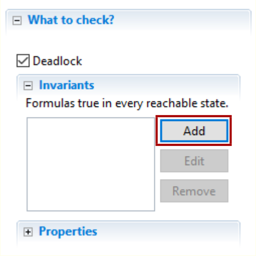


A formula that is true in every reachable state is called an *invariant*. To have TLC check an invariant Open the *Invariants* section of the model overview page.



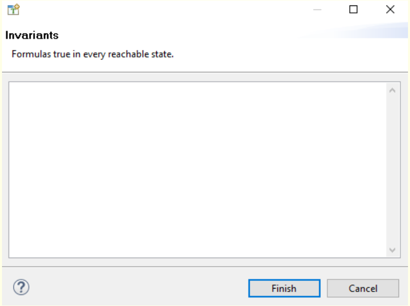
Click on *Add*.

Enter *TypeOK*. And click on *Finish*.



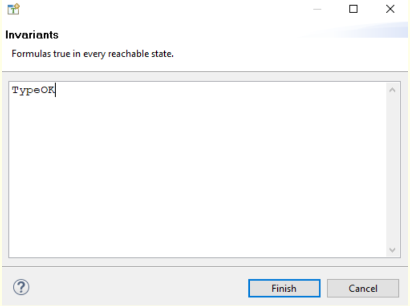
Click on *Add*.

Enter *TypeOK*. And click on *Finish*.



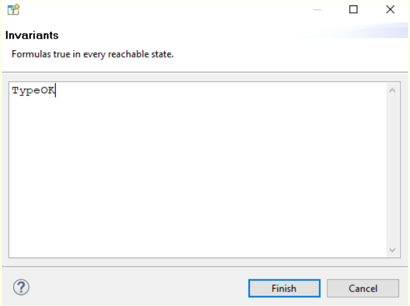
Click on *Add*.

Enter *TypeOK*. And click on *Finish*.



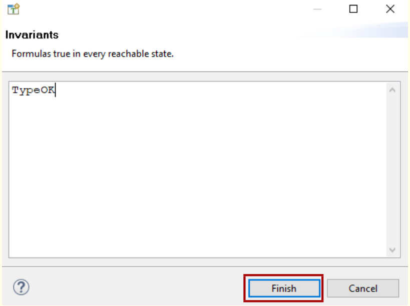
Click on *Add*.

Enter *TypeOK*. And click on *Finish*.



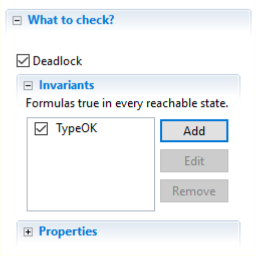
Click on *Add*.

Enter *TypeOK*. And click on *Finish*.



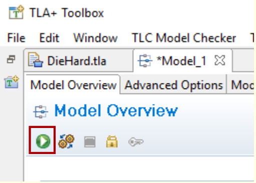
Click on *Add*.

Enter *TypeOK*. And click on *Finish*.



Click on *Add*.

Enter *TypeOK*. And click on *Finish*.

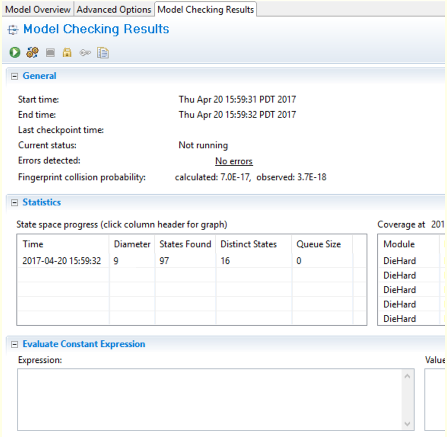


Run TLC.

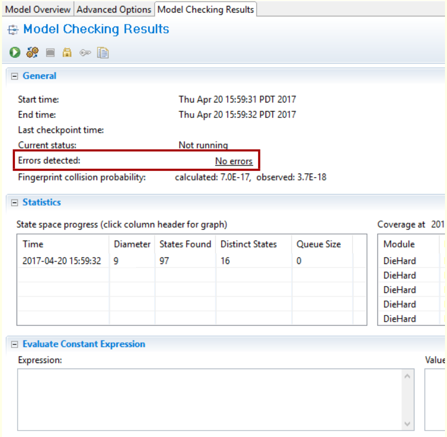
Click on *Add*.

Enter *TypeOK*. And click on *Finish*.

And run TLC on the model again.



The *Model Checking Results* page



TLC reports no errors.

The *Model Checking Results* page

again shows that TLC found no errors.

Saving Our Heroes

Now we're ready to save our heroes.

Saving Our Heroes

The four gallons must be in the big jug.

Now we're ready to save our heroes.

The four gallons of water our heroes need must be in the big jug.

Saving Our Heroes

The four gallons must be in the big jug.

We let TLC check if $big \neq 4$ is an invariant.

Now we're ready to save our heroes.

The four gallons of water our heroes need must be in the big jug.

We let TLC check if big *not equal to* 4 is an invariant.

Saving Our Heroes

The four gallons must be in the big jug.

We let TLC check if $big \neq 4$ is an invariant.

If it isn't, TLC will show us a behavior ending in a state with $big \neq 4$ false.

If it isn't, TLC will show us a behavior ending in a state with $big \neq 4$ false – a behavior that tells our heroes what they have to do to put 4 gallons in the big jug.

Saving Our Heroes

The four gallons must be in the big jug.

We let TLC check if $big \neq 4$ is an invariant.

$\neq$ is written in ASCII as $/=$ or $\#$

If it isn't, TLC will show us a behavior ending in a state with $big \neq 4$ false – a behavior that tells our heroes what they have to do to put 4 gallons in the big jug.

In TLA+, *not equal* is written in ASCII as either forward slash equal-sign or sharp (also called *pound sign*).

Saving Our Heroes

The four gallons must be in the big jug.

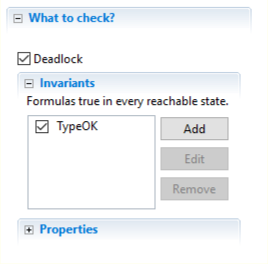
We let TLC check if $big \neq 4$ is an invariant.

We add this invariant to the model.

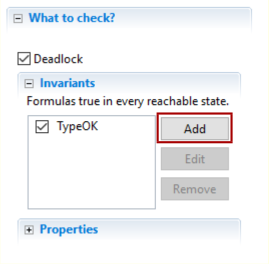
If it isn't, TLC will show us a behavior ending in a state with $big \neq 4$ false – a behavior that tells our heroes what they have to do to put 4 gallons in the big jug.

In TLA+, *not equal* is written in ASCII as either forward slash equal-sign or sharp (also called *pound sign*).

We now add this invariant to the model.

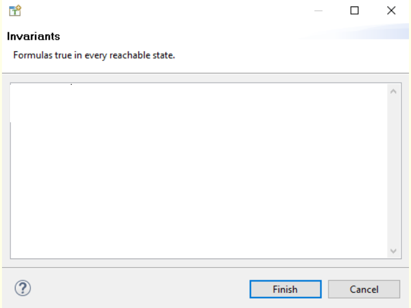


In the Invariants section of the model overview page



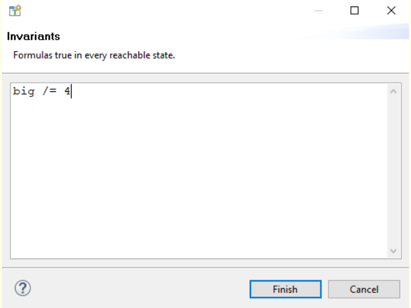
In the Invariants section of the model overview page

We add



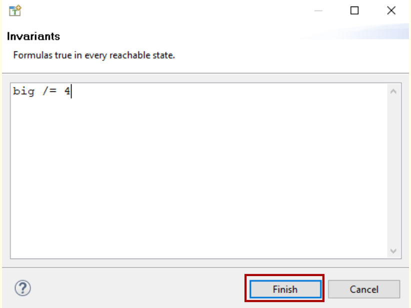
In the Invariants section of the model overview page

We add another invariant



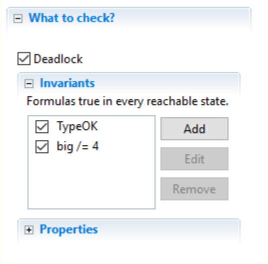
In the Invariants section of the model overview page

We add another invariant **big not equal to 4**



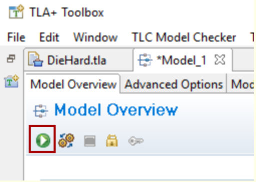
In the Invariants section of the model overview page

We add another invariant big not equal to 4



In the Invariants section of the model overview page

We add another invariant big not equal to 4

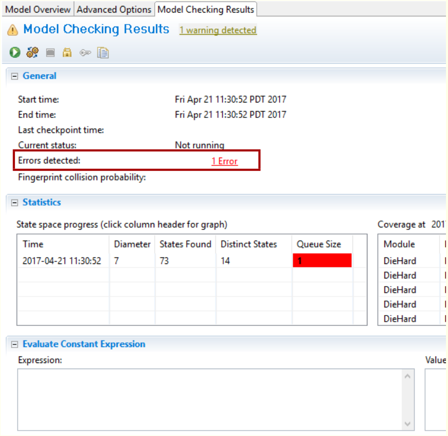


Run TLC.

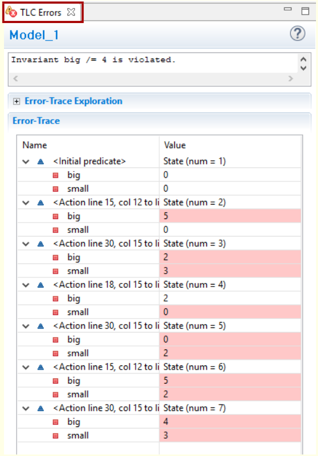
In the Invariants section of the model overview page

We add another invariant big not equal to 4

And we run TLC.

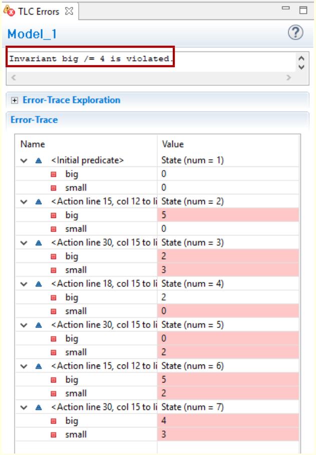


This time TLC reports an error.



This time TLC reports an error.

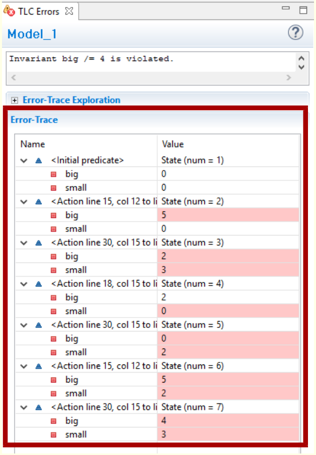
And the Toolbox opens this error window.



This time TLC reports an error.

And the Toolbox opens this error window.

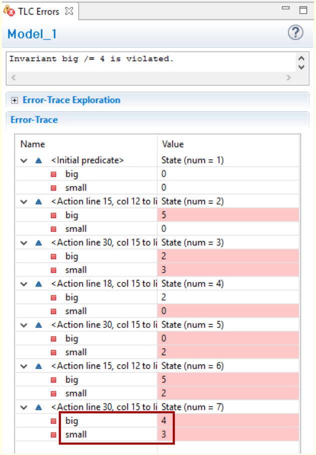
Which tells us that the invariant was violated



This time TLC reports an error.

And the Toolbox opens this error window.

Which tells us that the invariant was violated And displays this error trace.



A behavior ending in state

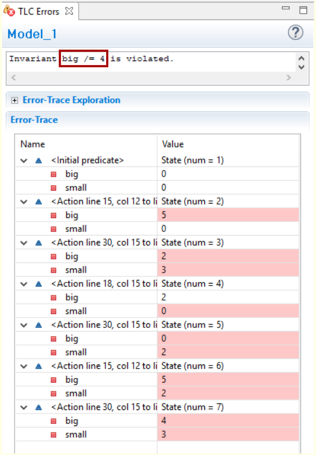
$$\left[\begin{array}{ll} big & : 4 \\ small & : 3 \end{array}\right]$$

This time TLC reports an error.

And the Toolbox opens this error window.

Which tells us that the invariant was violated And displays this error trace.

The error trace is a behavior satisfying the spec ending in this state

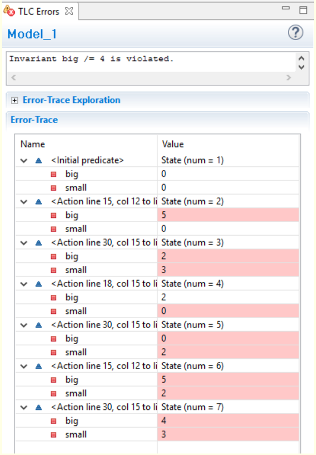


A behavior ending in state

$$\begin{bmatrix} big & : 4 \\ small & : 3 \end{bmatrix}$$

with the invariant false.

a state in which the invariant equals false.

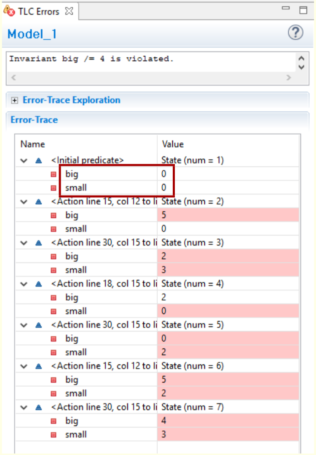


The complete behavior:

a state in which the invariant equals false.

It shows this complete behavior

[slide 231]

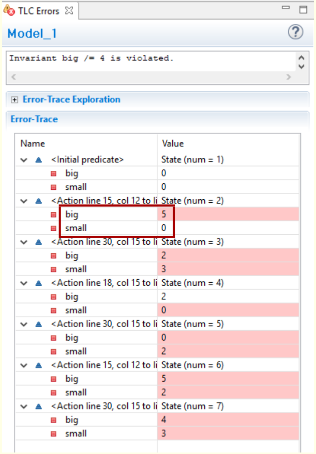


The complete behavior:

$$\begin{bmatrix} big & : 0 \\ small & : 0 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior

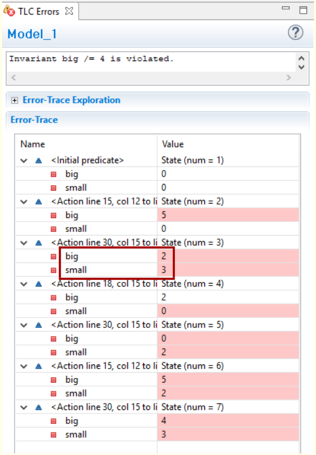


The complete behavior:

$$\begin{bmatrix} big & : 0 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 0 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior



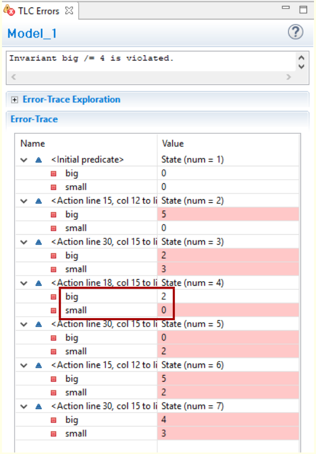
The complete behavior:

$$\begin{bmatrix} big & : 0 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 2 \\ small & : 3 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior

[slide 234]



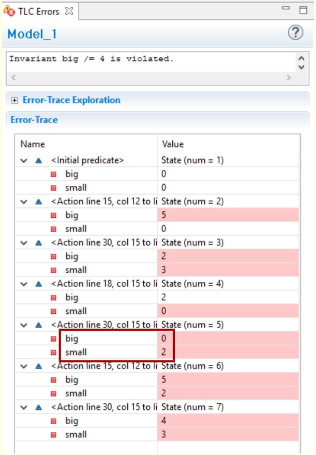
The complete behavior:

$$\begin{bmatrix} big & : 0 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 2 \\ small & : 3 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} big & : 2 \\ small & : 0 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior



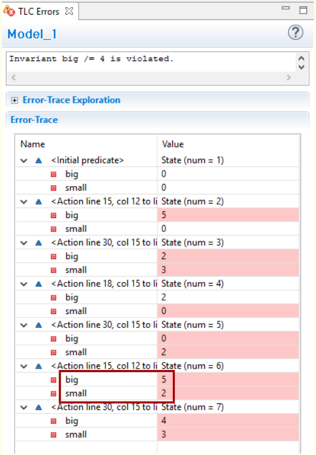
The complete behavior:

$$\begin{bmatrix} big & : 0 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 2 \\ small & : 3 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} big & : 2 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 0 \\ small & : 2 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior



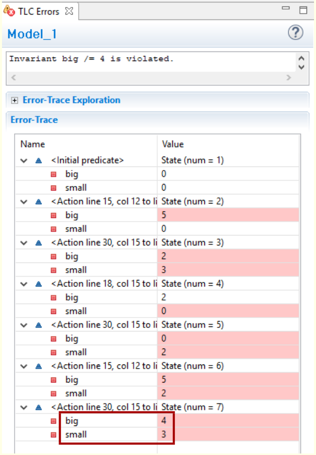
The complete behavior:

$$\begin{bmatrix} big & : 0 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 2 \\ small & : 3 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} big & : 2 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 0 \\ small & : 2 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 2 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior



The complete behavior:

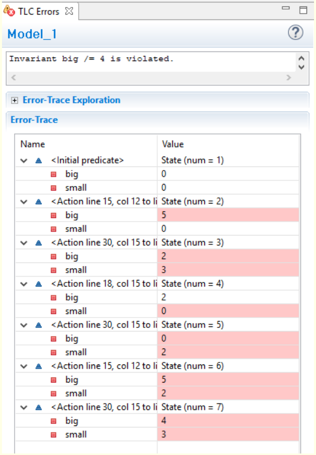
$$\begin{bmatrix} big & : 0 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 2 \\ small & : 3 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} big & : 2 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 0 \\ small & : 2 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 2 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} big & : 4 \\ small & : 3 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior



Name	Value
<Initial predicate>	State (num = 1)
big	0
small	0
<Action line 15, col 12 to li	State (num = 2)
big	5
small	0
<Action line 30, col 15 to li	State (num = 3)
big	2
small	3
<Action line 18, col 15 to li	State (num = 4)
big	2
small	0
<Action line 30, col 15 to li	State (num = 5)
big	0
small	2
<Action line 15, col 12 to li	State (num = 6)
big	5
small	2
<Action line 30, col 15 to li	State (num = 7)
big	4
small	3

$$\begin{bmatrix} big & : 0 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 2 \\ small & : 3 \end{bmatrix}$$

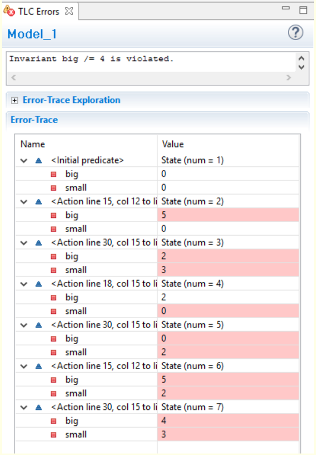
$$\rightarrow \begin{bmatrix} big & : 2 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 0 \\ small & : 2 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 2 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} big & : 4 \\ small & : 3 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior

From this behavior, our heroes should be able to see how to get 4 gallons of water in the big jug.



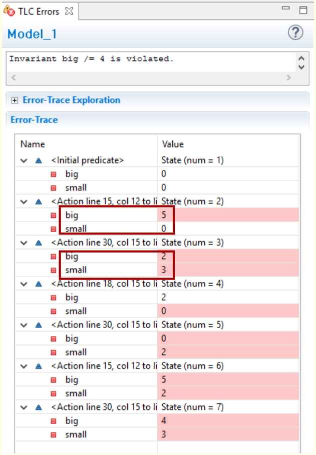
$$\begin{bmatrix} big & : 0 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 2 \\ small & : 3 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} big & : 2 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 0 \\ small & : 2 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 5 \\ small & : 2 \end{bmatrix}$$

$$\rightarrow \begin{bmatrix} big & : 4 \\ small & : 3 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior From this behavior, our heroes should be able to see how to get 4 gallons of water in the big jug. But they might not be the brightest bulbs on the block, and they may need help figuring out how to get from one state to the next. The Toolbox provides that help.

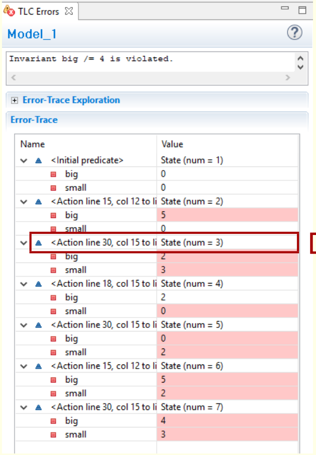


$$\begin{bmatrix} big & : 5 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 2 \\ small & : 3 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior From this behavior, our heroes should be able to see how to get 4 gallons of water in the big jug. But they might not be the brightest bulbs on the block, and they may need help figuring out how to get from one state to the next. The Toolbox provides that help.

To see why this step is allowed by the spec



Double-click here.

$$\begin{bmatrix} big & : 5 \\ small & : 0 \end{bmatrix} \rightarrow \begin{bmatrix} big & : 2 \\ small & : 3 \end{bmatrix}$$

a state in which the invariant equals false.

It shows this complete behavior From this behavior, our heroes should be able to see how to get 4 gallons of water in the big jug. But they might not be the brightest bulbs on the block, and they may need help figuring out how to get from one state to the next. The Toolbox provides that help.

To see why this step is allowed by the spec Double-click here to find the part of the next-state formula that allows this step

```
                 ELSE /\ big'   = 5
                      /\ small' = small - (5 - big)

BigToSmall == IF big + small =< 3
                 THEN /\ big'   = 0
                      /\ small' = big + small
                 ELSE /\ big'   = small - (3 - big)
                      /\ small' = 3

Next == \/ FillSmall
        \/ FillBig
```

a state in which the invariant equals false.

It shows this complete behavior From this behavior, our heroes should be able to see how to get 4 gallons of water in the big jug. But they might not be the brightest bulbs on the block, and they may need help figuring out how to get from one state to the next. The Toolbox provides that help.

To see why this step is allowed by the spec Double-click here to find the part of the next-state formula that allows this step And even Hollywood actors should be able to figure out

```
                    ELSE /\ big'   = 5
                         /\ small' = small - (5 - big)

BigToSmall == IF big + small =< 3
                 THEN /\ big'   = 0
                      /\ small' = big + small
                 ELSE /\ big'   = small - (3 - big)
                      /\ small' = 3

Next == \/ FillSmall
        \/ FillBig
```

a state in which the invariant equals false.

It shows this complete behavior From this behavior, our heroes should be able to see how to get 4 gallons of water in the big jug. But they might not be the brightest bulbs on the block, and they may need help figuring out how to get from one state to the next. The Toolbox provides that help.

To see why this step is allowed by the spec Double-click here to find the part of the next-state formula that allows this step And even Hollywood actors should be able to figure out that they have to pour the big jug into the small jug.

$SmallToBig$ **AND** $BigToSmall$

Formulas $SmallToBig$ and $BigToSmall$.

```
SmallToBig ==


BigToSmall ==
```

Now that we've saved our heroes, let's take a look at the definitions of $SmallToBig$ and $BigToSmall$.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

Now that we've saved our heroes, let's take a look at the definitions of $SmallToBig$ and $BigToSmall$.

Let's start with $SmallToBig$.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

$\leq$ is typed as **=<**

Notice that *less than or equal* is represented in ASCII as equal-sign less-than.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

Notice that *less than or equal* is represented in ASCII as equal-sign less-than.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

big is filled.

Notice that *less than or equal* is represented in ASCII as equal-sign less-than.

Remember that this is the case in which the big jug is filled from the small one.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

amount poured

Notice that *less than or equal* is represented in ASCII as equal-sign less-than.

Remember that this is the case in which the big jug is filled from the small one.

The amount poured into the big jug is removed from the small jug.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

Someone who hasn't seen TLA+ before would think this is wrong because this value of big

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

Someone who hasn't seen TLA+ before would think this is wrong because this value of big Is set to 5 here.

```
SmallToBig == IF big + small =< 5
                THEN /\ big'   = big + small
                     /\ small' = 0
                ELSE /\ big'   = 5
                     /\ small' = small - (5 - big)
```

Someone who hasn't seen TLA+ before would think this is wrong because this value of big Is set to 5 here.

That's because she thinks of this as two assignment statements.
But you know that it's actually a formula that specifies allowed steps.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

$$A \wedge B \; = \; B \wedge A$$

Someone who hasn't seen TLA+ before would think this is wrong because this value of big Is set to 5 here.

That's because she thinks of this as two assignment statements. But you know that it's actually a formula that specifies allowed steps. And that *and* is commutative, so

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

$$A \wedge B \;=\; B \wedge A$$

Someone who hasn't seen TLA+ before would think this is wrong because this value of big Is set to 5 here.

That's because she thinks of this as two assignment statements. But you know that it's actually a formula that specifies allowed steps. And that *and* is commutative, so

Changing the order of the two sub-formulas makes no difference.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

Someone who hasn't seen TLA+ before would think this is wrong because this value of big Is set to 5 here.

That's because she thinks of this as two assignment statements.
But you know that it's actually a formula that specifies allowed steps.
And that *and* is commutative, so
Changing the order of the two sub-formulas makes no difference.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5

BigToSmall == IF big + small =< 3
               THEN /\ big'   = 0
                    /\ small' = big + small
               ELSE /\ big'   = small - (3 - big)
                    /\ small' = 3
```

You can look at the definition of $BigToSmall$ in the module by yourself later.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

WARNING !

Here’s a warning about writing specs.

.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

$=$ is also commutative

Here's a warning about writing specs.

The equality operator is also commutative.

.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

$=$ is also commutative

SO `small' = 0`

Here's a warning about writing specs.

The equality operator is also commutative. so *small prime equals 0*

.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ 0 = small'
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

= is also commutative

so `small' = 0` is equivalent to `0 = small'` .

Here's a warning about writing specs.

The equality operator is also commutative. so *small prime equals 0* is completely equivalent to *0 equals small prime*.

.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ 0 = small'
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

These two specs are equivalent.

Here's a warning about writing specs.

The equality operator is also commutative. so *small prime equals 0* is completely equivalent to *0 equals small prime*.

These two specs are equivalent. .

[slide 264]

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

These two specs are equivalent.

Here’s a warning about writing specs.

The equality operator is also commutative. so *small prime equals 0* is completely equivalent to *0 equals small prime*.

These two specs are equivalent. .

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ 0 = small'
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

These two specs are equivalent.

Here's a warning about writing specs.

The equality operator is also commutative. so *small prime equals 0* is completely equivalent to *0 equals small prime*.

These two specs are equivalent. .

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ 0 = small'
               ELSE /\ small' = small - (5 - big)
                    /\ big'   = 5
```

These two specs are equivalent.

The TLAPS proof system treats them the same.

And the TLAPS proof system treats them exactly the same.

```
SmallToBig == IF big + small =< 5
                THEN /\ big'   = big + small
                     /\ small' = 0
                ELSE /\ small' = small - (5 - big)
                     /\ big'   = 5
```

These two specs are equivalent.

But TLC handles only this one.

And the TLAPS proof system treats them exactly the same.

But TLC handles only this one.

```
SmallToBig == IF big + small =< 5
                THEN /\ big'   = big + small
                     /\ 0 = small'
                ELSE /\ small' = small - (5 - big)
                     /\ big'   = 5
```

These two specs are equivalent.

It reports an error on this one.

And the TLAPS proof system treats them exactly the same.

But TLC handles only this one.

It reports an error if you run it on this one.

There are many ways to write a correct specification.

There are many ways to write a correct specification.

There are many ways to write a correct specification.

TLC can almost always handle the ones most engineers naturally write.

There are many ways to write a correct specification.

TLC can almost always handle the ones most engineers naturally write.

There are many ways to write a correct specification.

TLC can almost always handle the ones most engineers naturally write.

Later, you’ll learn what specs TLC can handle.

There are many ways to write a correct specification.

TLC can almost always handle the ones most engineers naturally write.

Later, you’ll learn what specs TLC can handle.

For now, follow this simple rule:

For now, just follow this simple rule:

For now, follow this simple rule:

Use a primed variable v' only in one of these two kinds of formulas:

$$v' = \ldots \qquad \text{and} \qquad v' \in \ldots$$

For now, just follow this simple rule:

Use a primed variable v-prime only in one of these two kinds of formulas

For now, follow this simple rule:

Use a primed variable v' only in one of these two kinds of formulas:

$$v' = \boxed{\ldots} \qquad \text{and} \qquad v' \in \boxed{\ldots}$$

no primed variables

For now, just follow this simple rule:

Use a primed variable v-prime only in one of these two kinds of formulas where dot-dot-dot is an expression not containing primes.

For now, follow this simple rule:

Use a primed variable v' only in one of these two kinds of formulas:

$$v' = \boxed{\ldots} \qquad \text{and} \qquad v' \in \boxed{\ldots}$$

no primed variables

We'll relax this rule later.

For now, just follow this simple rule:

Use a primed variable v-prime only in one of these two kinds of formulas where dot-dot-dot is an expression not containing primes.

We'll relax this rule later.

CHECKING YOUR DEFINITIONS

Let's now check your definitions of $SmallToBig$ and $BigToSmall$.

Your definitions of $SmallToBig$ and $BigToSmall$ are probably not exactly like mine.

Your definitions are probably not exactly the same as mine.

Your definitions of $SmallToBig$ and $BigToSmall$ are probably not exactly like mine.

But they may still be correct.

Your definitions are probably not exactly the same as mine.

But they may still be correct.

Your definitions of $SmallToBig$ and $BigToSmall$ are probably not exactly like mine.

But they may still be correct.

Math provides many ways of writing the same formula.

Your definitions are probably not exactly the same as mine.

But they may still be correct.

Math provides many ways of writing the same formula.

Your definitions of $SmallToBig$ and $BigToSmall$
are probably not exactly like mine.

But they may still be correct.

Math provides many ways of writing the same formula.

Let's check your definitions.

Let's check your definitions.

Your definitions of $SmallToBig$ and $BigToSmall$
are probably not exactly like mine.

But they may still be correct.

Math provides many ways of writing the same formula.

Let's check your definitions.

But first, let's see how we find errors.

Let's check your definitions.

But first, let's see how we find errors.

Parsing Errors

When writing a spec, our first mistakes are found by the parser.

Parsing Errors

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

Modify the spec.

When writing a spec, our first mistakes are found by the parser.

For example, in the Toolbox, modify the definition of $SmallToBig$ by deleting

Parsing Errors

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

Modify the spec.

When writing a spec, our first mistakes are found by the parser.

For example, in the Toolbox, modify the definition of *SmallToBig* by deleting this plus sign.

Parsing Errors

```
SmallToBig == IF big small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

When writing a spec, our first mistakes are found by the parser.

For example, in the Toolbox, modify the definition of $SmallToBig$ by deleting this plus sign.

Parsing Errors

```
SmallToBig == IF big small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

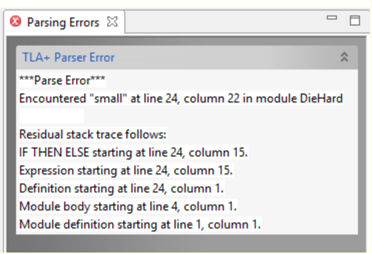
Save the spec.

When writing a spec, our first mistakes are found by the parser.

For example, in the Toolbox, modify the definition of $SmallToBig$ by deleting this plus sign.

Now save the spec.

Parsing Errors



When writing a spec, our first mistakes are found by the parser.

For example, in the Toolbox, modify the definition of $SmallToBig$ by deleting this plus sign.

Now save the spec.

The Toolbox runs the parser, which raises this error window.

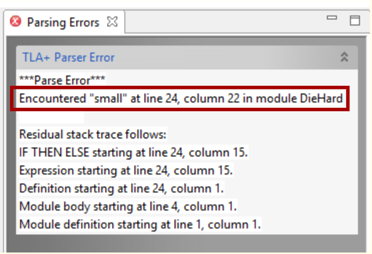
Parsing Errors

```
SmallToBig == IF big small =< 5
                THEN /\ big'   = big + small
                     /\ small' = 0
                ELSE /\ big'   = 5
                     /\ small' = small - (5 - big)
```

And it puts this error mark in the module editor.

Parsing Errors

Click here.



And it puts this error mark in the module editor.

Clicking here in the error window

Parsing Errors

```
SmallToBig == IF big small =< 5
                 THEN /\ big'   = big + small
                      /\ small' = 0
                 ELSE /\ big'   = 5
                      /\ small' = small - (5 - big)
```

And it puts this error mark in the module editor.

Clicking here in the error window Highlights this part of the module and jumps to it.

Another common error found by parsing:

```
SmallToBig == IF big + smell =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

Here's another common error found by the parser.

Another common error found by parsing:

```
SmallToBig == IF big + smell =< 5
                 THEN /\ big'   = big + small
                      /\ small' = 0
                 ELSE /\ big'   = 5
                      /\ small' = small - (5 - big)
```

An identifier not defined or declared.

Here's another common error found by the parser.

An identifier not yet defined or declared. This is usually a typo.

TLC “Execution Errors”

Errors TLC finds trying to “execute” the spec.

After there are no more parsing errors, TLC can often find errors while trying to *quote* execute the spec. (We’ll see in a later video how TLC does that.)

TLC "Execution Errors"

Errors TLC finds trying to "execute" the spec.

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)
```

After there are no more parsing errors, TLC can often find errors while trying to *quote* execute the spec. (We'll see in a later video how TLC does that.)

For example, change

TLC "Execution Errors"

Errors TLC finds trying to "execute" the spec.

```
SmallToBig == IF big + small =< 5
                THEN /\ big'   = big + small
                     /\ small' = 0
                ELSE /\ big'   = 5
                     /\ small' = small - (5 - big)
```

After there are no more parsing errors, TLC can often find errors while trying to *quote* execute the spec. (We'll see in a later video how TLC does that.)

For example, change this five to quote five.

TLC "Execution Errors"

Errors TLC finds trying to "execute" the spec.

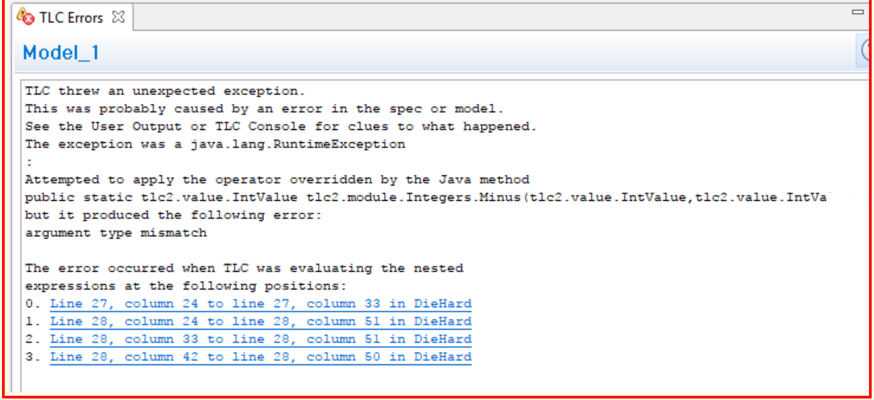
```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - ("5" - big)
```

After there are no more parsing errors, TLC can often find errors while trying to *quote* execute the spec. (We'll see in a later video how TLC does that.)

For example, change this five to quote five. And save the spec.

Running TLC now produces

TLC “Execution Errors”



this error. You can read the complete error report later if you’re curious.

For now, just

TLC “Execution Errors”

TLC Errors

Model_1

```
TLC threw an unexpected exception.
This was probably caused by an error in the spec or model.
See the User Output or TLC Console for clues to what happened.
The exception was a java.lang.RuntimeException
:
Attempted to apply the operator overridden by the Java method
public static tlc2.value.IntValue tlc2.module.Integers.Minus(tlc2.value.IntValue,tlc2.value.IntVa
but it produced the following error:
argument type mismatch

The error occurred when TLC was evaluating the nested
expressions at the following positions:
0. Line 27, column 24 to line 27, column 33 in DieHard
1. Line 28, column 24 to line 28, column 51 in DieHard
2. Line 28, column 33 to line 28, column 51 in DieHard
3. Line 28, column 42 to line 28, column 50 in DieHard
```

this error. You can read the complete error report later if you’re curious.

For now, just click **here**, which selects and goes to

[slide 299]

TLC “Execution Errors”

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - ("5" - big)
```

this error. You can read the complete error report later if you’re curious.

For now, just click **here**, which selects and goes to this part of the module.

Checking Your Definitions

Now, check your definitions of $SmallToBig$ and $BigToSmall$.

Checking Your Definitions

```
SmallToBig == IF big + small =< 5
               THEN /\ big'   = big + small
                    /\ small' = 0
               ELSE /\ big'   = 5
                    /\ small' = small - (5 - big)

BigToSmall == IF big + small =< 3
               THEN /\ big'   = 0
                    /\ small' = big + small
               ELSE /\ big'   = small - (3 - big)
                    /\ small' = 3
```

Now, check your definitions of $SmallToBig$ and $BigToSmall$.

First

Checking Your Definitions

```
(*
SmallToBig == IF big + small =< 5
                THEN /\ big'   = big + small
                     /\ small' = 0
                ELSE /\ big'   = 5
                     /\ small' = small - (5 - big)

BigToSmall == IF big + small =< 3
                THEN /\ big'   = 0
                     /\ small' = big + small
                ELSE /\ big'   = small - (3 - big)
                     /\ small' = 3
*)
```

Comment out my definitions

Now, check your definitions of $SmallToBig$ and $BigToSmall$.

First comment out my definitions by

Checking Your Definitions

```
(*
SmallToBig == IF big + small =< 5
                THEN /\ big'   = big + small
                     /\ small' = 0
                ELSE /\ big'   = 5
                     /\ small' = small - (5 - big)

BigToSmall == IF big + small =< 3
                THEN /\ big'   = 0
                     /\ small' = big + small
                ELSE /\ big'   = small - (3 - big)
                     /\ small' = 3
*)
```

Comment out my definitions

Now, check your definitions of $SmallToBig$ and $BigToSmall$.

First comment out my definitions by adding these comment delimiters.

Checking Your Definitions

```
(*
SmallToBig == IF big + small =< 5
                THEN /\ big'   = big + small
                     /\ small' = 0
                ELSE /\ big'   = 5
                     /\ small' = small - (5 - big)

BigToSmall == IF big + small =< 3
                THEN /\ big'   = 0
                     /\ small' = big + small
                ELSE /\ big'   = small - (3 - big)
                     /\ small' = 3
*)
```

Comment out my definitions and add your own.

Now, check your definitions of $SmallToBig$ and $BigToSmall$.

First comment out my definitions by adding these comment delimiters.

And add your own definitions.

Save your definitions

Save your definitions

Save your definitions and correct any errors the parser finds.

Save your definitions and correct any errors the parser finds.

Save your definitions and correct any errors the parser finds.

Run TLC.

Save your definitions and correct any errors the parser finds.

Run TLC.

Save your definitions and correct any errors the parser finds.

Run TLC.

Your definitions are probably correct if TLC:

– Finds no “execution” errors.

Save your definitions and correct any errors the parser finds.

Run TLC.

Your definitions are probably correct if TLC:

– Finds no “execution” errors.

Save your definitions and correct any errors the parser finds.

Run TLC.

Your definitions are probably correct if TLC:

– Finds no “execution” errors.

– Finds no violation of the invariant $TypeOK$.

Save your definitions and correct any errors the parser finds.

Run TLC.

Your definitions are probably correct if TLC:

– Finds no “execution” errors.

– Finds no violation of the invariant $TypeOK$.

Save your definitions and correct any errors the parser finds.

Run TLC.

Your definitions are probably correct if TLC:

– Finds no “execution” errors.

– Finds no violation of the invariant $TypeOK$.

– Finds a violation of the alleged invariant $big \neq 4$.

Save your definitions and correct any errors the parser finds.

Run TLC.

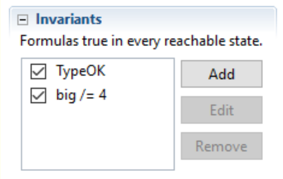
Your definitions are probably correct if TLC:

– Finds no “execution” errors.

– Finds no violation of the invariant $TypeOK$.

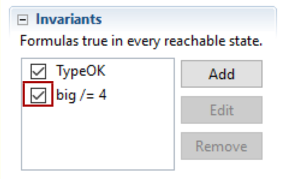
– And finds a violation of the alleged invariant $big \neq 4$.

To be sure, go here



To be sure, go to the *Invariants* section of the *Model Overview* page and

To be sure, go here

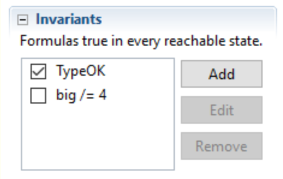


Uncheck this box

To be sure, go to the *Invariants* section of the *Model Overview* page and

Uncheck this box.

To be sure, go here



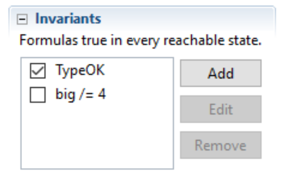
Uncheck this box so only $TypeOK$ will be tested.

To be sure, go to the *Invariants* section of the *Model Overview* page and

Uncheck this box.

So only the $TypeOK$ invariant will be tested by TLC.

To be sure, go here



Uncheck this box so only $TypeOK$ will be tested.

Run TLC again.

To be sure, go to the *Invariants* section of the *Model Overview* page and

Uncheck this box.

So only the $TypeOK$ invariant will be tested by TLC.

And run TLC again.

If TLC finds no error

If TLC finds no error

If TLC finds no error, try to find a different way to write the definitions.

If TLC finds no error try to find a different way to write the definitions.

If TLC finds no error, try to find a different way to write the definitions.

The best way to learn is by making mistakes.

If TLC finds no error, try to find a different way to write the definitions.

The best way to learn is by making mistakes.

Now that we've used TLC to save our heroes from certain death, it's time to leave the glamour of Hollywood for the more romantic subject of marriage and commitment. In the next lecture, we'll examine an algorithm that has been used for many years in weddings and database systems.

[slide 319]

End of Lecture 4

DIE HARD