

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 is

[slide 4]

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.

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.

```
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.

```
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 = 3.785411784 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.

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.

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.

They needed exacty 4 gallons.

There were no markings on the jugs.

They needed exacty 4 gallons.

[slide 12]

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.

[slide 13]





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

[slide 15]

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 Hollywood movie, and I can't affort big jugs.

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



Our heroes start with both jugs empty.





Our heroes start with both jugs empty.



Our heroes start with both jugs empty.



Our heroes start with both jugs empty.



Our heroes start with both jugs empty.



Our heroes start with both jugs empty.



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]















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.



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.


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



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



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



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



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]



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 42]



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 43]



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 44]



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 45]

























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

And so on.



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



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

Let's write it more formally.

[slide 59]

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.



Initially, both jugs have 0 gallons of water.



Initially, both jugs have 0 gallons of water.

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

[slide 62]



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.

[slide 63]





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.

[slide 65]



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.

[slide 66]

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

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.

[slide 67]

What did we learn from this behavior?

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

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

[slide 68]

What did we learn from this behavior?

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

1. What the variables are.

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

First, what the variables are.

[slide 69]

What did we learn from this behavior?

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

1. What the variables are.

2. What constitutes a step.

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. $\begin{bmatrix}small: & 0\\big & : & 0\end{bmatrix} \rightarrow \begin{bmatrix}small: & 3\\big & : & 3\end{bmatrix} \rightarrow \begin{bmatrix}small: & 3\\big & : & 3\end{bmatrix} \rightarrow \begin{bmatrix}small: & 1\\big & : & 5\end{bmatrix} \rightarrow \begin{bmatrix}small: & 1\\big & : & 5\end{bmatrix} \rightarrow \vdots$ 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.

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

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.



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?

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

1. What the variables are.

2. What constitutes a step.

Filling a jug is a single step.

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

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

[slide 76]

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

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

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.

[slide 78]



We can now start writing the actual TLA+ specification.

[slide 79]



[slide 80]



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

[slide 81]

MODULE DieHard —
VARIABLES small, big
Declares the variables.

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

and the VARIABLES statement declares our two variables.

[slide 82]



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.

[slide 83]

MODULE DieHard EXTENDS Integers VARIABLES small, big
TLA ⁺ has no type declarations. Type correctness means variables have
Sensible values.

EXTENDS Integers
VARIABLES small, big
We define a formula that asserts type correctness.

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

EXTENDS Integers MODULE DieHard	
VARIABLES small, big	
We define a formula that asserts type correctness. Helps to understand spec.	

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

It helps a reader to understand the spec.

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.

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.

[slide 87]

EXTENDS Integers
VARIABLES small, big
$TypeOK \stackrel{\Delta}{=}$
We define a formula that asserts type correctness.
Helps to understand spec.
TLC can check that it's always true.



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



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.

[slide 90]



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.



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.

[slide 92]



The Initial-State Formula

Init \triangleq

The initial-state formula.

As usual, let's name it Init.

[slide 94]

The Initial-State Formula

$$\begin{array}{rcl} Init \ \triangleq \ \land big &= 0 \\ \land small = 0 \end{array}$$

The initial-state formula.

As usual, let's name it Init.

It asserts that both jugs are empty.

[slide 95]

THE NEXT-STATE FORMULA

The next-state formula.

[slide 96]

The Next-State Formula The next-state formula describes all permitted steps.

The next-state formula describes all permitted steps.

The next-state formula describes all permitted steps.

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

The next-state formula describes all permitted steps.

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

The next-state formula 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.

[slide 99]

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 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.

[slide 101]

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 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.
- 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.

[slide 103]



[slide 104]



First we allow steps that fill a jug.

There are two jugs, so we have two possible kinds of steps.

[slide 105]



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.

[slide 106]



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.

[slide 107]



Similarly for steps that empty a jug.

[slide 108]


[slide 109]



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

[slide 110]



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.

[slide 111]



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.

[slide 112]

 $Next \stackrel{\Delta}{=} \lor FillSmall$

- \lor FillBig
- $\lor EmptySmall$
- $\lor EmptyBig$
- \lor SmallToBig
- \lor BigToSmall



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

Next $\stackrel{\Delta}{=}$ \vee *FillSmall* FillBig EmptySmall $\vee EmptyBig$ \lor SmallToBig \lor BigToSmall

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*.



Most people first learning TLA⁺ would write this definition.



Most people would write this definition.

We now define *FillSmall*.

Most people first learning TLA⁺ would write this definition.

```
FillSmall \stackrel{\Delta}{=} 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.

 $FillSmall \stackrel{\Delta}{=} small' = 3$

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.

[slide 119]

$FillSmall \triangleq small' = 3$
If you didn't figure it out, you're thinking of this as setting <i>small</i> 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 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.

[slide 121]



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 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.

[slide 123]



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

$$FillSmall \triangleq \boxed{small' = 3}$$
$$\begin{bmatrix} small : 0 \\ big : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small : 3 \\ big : 3 \end{bmatrix}$$
$$\begin{bmatrix} small : 0 \\ big : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small : 3 \\ big : \sqrt{7} \end{bmatrix}$$
$$\begin{bmatrix} small : 0 \\ big : 3 \end{bmatrix} \rightarrow \begin{bmatrix} small : 3 \\ big : *abc \\ big : *abc \end{bmatrix}$$

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.



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



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.

[slide 127]



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

[slide 128]



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.

[slide 129]



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



The definition of *FillBig* is similar.

[slide 131]

POURING BETWEEN JUGS

Pouring from one jug into another.

[slide 132]





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.



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



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



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



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



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



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



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



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.

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.

[slide 144]
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.

[slide 145]

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.

[slide 146]

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.

[slide 147]

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.

[slide 148]

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.

[slide 149]







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

[slide 152]



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.

[slide 153]



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

Case 2

[slide 154]



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.

[slide 155]



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*.

[slide 156]



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

$$\begin{array}{rcl} SmallToBig \ \triangleq \ \mbox{if} \ big + small \leq 5 \\ & \mbox{THEN} \ \land big' \ = big + small \\ & \ \land small' = 0 \\ & \mbox{ELSE} \ \land big' \ = ? \\ & \ \land small' = ? \end{array}$$

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.

[slide 158]



We'll now use TLC to save our heroes.

[slide 159]



Open the Toolbox.

[slide 160]



Open the Toolbox.

And then open a new spec named *DieHard*.

[slide 161]



Remember you click on

[slide 162]



Remember you click on File. Then on

[slide 163]



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

	r TLA+ Toolbox File Window Help	
😭 New	LA+ Specification	- 🗆 X
Creat Enter	s a new TLA+ specification a complete file name like c:\jones\specs\foo.tla or click on	Browse.
Root-r Specifi	nodule file:	Browse
	Import existing models	
?		Finish Cancel

	=0				
	TT ILA+ Io	oolbox			
	File Windo	ow Help			
r					
New 1	'LA+ Speci	ification			
Create Enter a	s a new TLA+ a complete file	specification e name like c:\jones	syecs/foo.tla or click	c on Browse.	
Root-n	nodule file:				Browse
Specifi	cation name:]
		Import existing	models		
?				Finish	Cancel

TLA+ Toolbox	
File Window Help	
	- 🗆 X
New TLA+ Specification	
😭 Open root file	×
\leftarrow \rightarrow \checkmark \uparrow \checkmark \checkmark \checkmark \checkmark \circlearrowright	Search specs $ ho$
Organize 🔻 New folder	III 🕶 🔟 😲
Intel ^ Name ^	Date modified Type
lamport SimpleProgram.toolbox	3/24/2017 6:25 PM File folder
MININT SimpleProgram.tla	3/24/2017 2:49 PM TLA File
MSOTraceLite	
PerfLogs	
Peri64	
Program Files 👻 <	>
File name: 🗸 🗸	TLA+ files ∨
	Open Cancel

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

TLA+ Toolbox		
File Window Help		
	— D	×
New TLA+ Specification		
📸 Open root file		×
\leftarrow \rightarrow \checkmark \uparrow \square \ll talks \Rightarrow mooc \Rightarrow video \Rightarrow specs \checkmark \heartsuit	Search specs	م
Organize 🔻 New folder		• 🔳 🕐
Intel ^ Name	Date modified	Туре
lamport SimpleProgram.toolbox	3/24/2017 6:25 PM	File folder
MININT SimpleProgram.tla	3/24/2017 2:49 PM	TLA File
MSOTraceLite		
PerfLogs		
Perl		
Perl64		
- Program Files 🗸 <		>
File name: DieHard 🗸 🗸	TLA+ files	~
	0	Connact

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

[slide 169]

😭 TLA+	Toolbox		
File Win	dow Help		
Ē			×
New TLA+ Spe	cification		
😭 Open root file			×
$\leftarrow \rightarrow \cdot \uparrow$ – « talk	s → mooc → video → specs 🛛 🗸 Ö	Search specs	م
Organize 🔻 New folder	r		• 🔳 🕐
intel ^	Name	Date modified	Туре
📙 lamport	SimpleProgram.toolbox	3/24/2017 6:25 PM	File folder
	SimpleProgram.tla	3/24/2017 2:49 PM	TLA File
MSOTraceLite			
PerfLogs			
Perl			
Perl64			
📙 Program Files 💙	<		>
File na	me: DieHard 🗸	TLA+ files	~

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*

[slide 170]

		_	
T ILA+ I	oolbox		
File Wind	ow Help		
E C			
New TLA+ Spec	ification		
🚯 Root file name de	oes not exist. A new file will be created.		
Root-module file:	C:\Users\lamport\DieHard.tla		Browse
Specification name:	DieHard		
	Import existing models		
?		Finish	Cancel

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]

oolbox	-	
ow Help		
ification		
pes not exist. A new file will be created.		
C:\Users\lamport\DieHard.tla		Browse
DieHard		
Import existing models		
		0.1
	oolbox ow Help ification pes not exist. A new file will be created. C:\Users\lamport\DieHard.tla DieHard ☑ Import existing models	oolbox ow Help ification ses not exist. A new file will be created. C:\Users\lamport\DieHard.tla DieHard ☑ Import existing models

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

[slide 172]

1	 	MODULE	DieHard	 	
3					
4 ==	 			 	

An empty spec named diehard.

[slide 173]

1 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.

[slide 174]



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.

[slide 175]

```
MODULE DieHard -----
 2 EXTENDS Integers
 4 VARIABLES small, big
 6 TypeOK == / small \in 0..3
 7
          /\ big \in 0..5
 9 Init == / big = 0
       / \ small = 0
12 FillSmall == /\ small' = 3
              /  big' = big
14
15 FillBig == / big' = 5 at them yet.
            /\ small' = small
16
17
18 EmptySmall == / small' = 0
19
              / big' = big
21 EmptyBig == /  big' = 0
             /\ small' = small
```

```
MODULE DieHard
  EXTENDS Integers
  VARIABLES small, big
6 TypeOK == / small \in 0..3
            /\ big \in 0..5
9 Init == / big = 0
         / small = 0
12 FillSmall == /\ small' = 3
               /  big' = big
14
15 FillBig == / big' = 5
             / \ small' = small
16
18 EmptySmall == /  small' = 0
19
                / big' = big
21 EmptyBig == / big' = 0
              /\ small' = small
```

The module contains the complete definitions of SmallToBig and BigToSmall.

And here's what you should see.

The module contains the complete definitions of *SmallToBig* and Big ToSmall

```
MODULE DieHard
 EXTENDS Integers
  VARIABLES small, big
                                  The module contains the
6 TypeOK == / small \in 0..3
                                  complete definitions of
           /\ big \in 0..5
                                  SmallToBig and BigToSmall.
9 Init == / big = 0
         / \text{small} = 0
12 FillSmall == /\ small' = 3
                                  But don't look
              /  big' = big
14
                                  at them yet.
15 FillBig == / big' = 5
            / \ small' = small
16
18 EmptySmall == /\ small' = 0
19
               / big' = big
21 EmptyBig == / big' = 0
             /\ small' = small
```

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
 2 EXTENDS Integers
  VARIABLES small, big
                                  Save the module.
 6 TypeOK == / small \in 0..3
            /\ big \in 0..5
 9 Init == / big = 0
        / small = 0
12 FillSmall == /\ small' = 3
               /  big' = big
14
15 FillBig == / big' = 5
16
            / small' = small
18 EmptySmall == /  small' = 0
19
                / big' = big
21 EmptyBig == / big' = 0
              /\ small' = small
```

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

[slide 179]

```
MODULE DieHard
  EXTENDS Integers
  VARIABLES small, big
                                  Save the module.
 6 TypeOK == / small \in 0..3
                                  (Type Ctl+S.)
            /\ big \in 0..5
 9 Init == / big = 0
          / \ small = 0
12 FillSmall == /\ small' = 3
               /  big' = big
14
15 FillBig == / big' = 5
16
             /\ small' = small
18 EmptySmall == /  small' = 0
19
                / big' = big
21 EmptyBig == / big' = 0
              /\ small' = small
```

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.

[slide 180]




Clicking on the TLC Model Checker menu



Clicking on the TLC Model Checker menu

Selecting NewModel

[slide 183]

	To run TI C, we create a model.	
📸 New model	×	
Please input the name of the model to cr Model_1	ate	
	OK Cancel	

Clicking on the TLC Model Checker menu

Selecting NewModel

Entering a model name and

[slide 184]

To run	TLC,	we	create	aı	nod	el.

ancel

Clicking on the TLC Model Checker menu

Selecting NewModel

Entering a model name and

Clicking OK.

[slide 185]

Model Overview		
) 🔐 🗏 🛱 🖘		
Model description		
What is the behavior spec?	What is the model?	
	Specify the values of declared constants.	
 Initial predicate and next-state relation 		
Init Init		
Next Next		
 Temporal formula 		
^	Advanced parts of the model: Advitional definitions	
	State constraints, Action constraints,	
O No Behavior Spec	How to run?	
What to check?		
_		
✓ Deadlock		
Invariants		
D Proposition		

This raises the Model Overview page

Model Overview		
) 🔐 🖩 🛱 👳		
Model description		
What is the behavior spec?	What is the model?	
-	Specify the values of declared constants.	
Initial predicate and next-state relation	n	
Init Init	Initial and next-s	tate formulas
Next: Next	milliar and next e	
 Temporal formula 		
	Advanced parts of the model: Additional definition State constraints, Action constraints,	L.
O No Behavior Spec	How to run?	
What to check?		
Deadlock		
Deadlock Invariants		

This raises the Model Overview page

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

	r a si
Model Overview Advanced Options Model Chee	cking Results
Model Overview	TLO
ola 🛛 🖉 🖉 🖉	ILC.
Model description	
What is the behavior spec?	What is the model?
	Specify the values of declared constants.
Initial predicate and next-state relation	
Net Next	
	Advanced parts of the model: Additional definitions,
~	State constraints, Action constraints,
O No Behavior Spec	How to run?
What to check?	
Deadlock	
Invariants	
Properties	

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.

Image: Image	3 (0) = (0) - (0)							
Start ime Thu Apr 20 11:5448 PDT 2017 Last checkpoint time: Thu Apr 20 11:5448 PDT 2017 Last checkpoint time: Thu Apr 20 11:5448 PDT 2017 Current status: Not running Errons detected: Not arrsis Fingerprint collision probability: calculated: 7.06-17; observed: 3.7E-18 States States Diameter States progress (click column header for graph) Coverage at 201 Time Diameter States Found Distinct States Queue Size 2017-04-20 11:5449 9 97 16 0 Diefand Diefand Diefand Diefand Diefand Diefand Diefand Diefand Diefand Diefand	/ 🞯 🔲 🖬 🕬 🎼	8						
Start inne: Thu Apr 20 11:5448 PDT 2017 End inne: Thu Apr 20 11:5448 PDT 2017 Last checkpoint inne: Extended point inne: Current status: Not running Errors didexted: Rearrant Fingerprint collision probability: calculated: 7.0E-17, observed: 3.7E-18 Statistics Statespee progress (click column header for graph) Time Diameter States Found Dicitint States Queue Size 2017-04-20 11:54-49 9 7 16 DieHand DieHand DieHand DieHand DieHand DieHand DieHand DieHand	- General							
End time: Thu Apr 20 11:54-89 PDT 2017 Last checkpoint development of the second development of	Start time:		Thu Apr	20 11:54:48 PDT 20	117			
Last checkpoint time: Not numing: Current datus: Not numing: Errors detacted: Not numing: Coverage at 201 Diameter States Found Distinct States Queue Size Distinct Distinct States Queue Size Distinct Distinct DieHard <td c<="" td=""><td>End time:</td><td></td><td>Thu Apr</td><td>20 11:54:49 PDT 20</td><td>17</td><td></td><th></th></td>	<td>End time:</td> <td></td> <td>Thu Apr</td> <td>20 11:54:49 PDT 20</td> <td>17</td> <td></td> <th></th>	End time:		Thu Apr	20 11:54:49 PDT 20	17		
Current status: Not running Errors detected: Noe remost Fingerprint collision probability: calculated: 7.0E-17, observed: 3.7E-18 Statistics Statistics Statistics Coverage at 201 Time Diameter States Found Dicitint States 2017-04-20 11:54-49 9 97 16 Dieland Dieland Dieland Dieland Dieland Dieland Dieland Dieland Dieland Dieland Dieland	Last checkpoint time:							
Error detected: No.errors Fingerprint collision probability: calculated: 7.06-17, observed: 3.7E-18 Statistics: Statistics: State space progress (click column header for graph): Coverage at 201 1mme Diameter States Coverage at 201 2017-04-20 11:54-49 9 97 16 Dieland Dieland Dieland Dieland Dieland Dieland Dieland Dieland Dieland Dieland Dieland	Current status:		Not runn	ing				
Fingerprint collision probability: calculated: 7.0F-17, observed: 3.7E-18 Statistics Statistics State spece progress (click column header for graph) Coverage at 201 Time Diameter States Found Distint States Queue Size OliveHard 2017-04-20 11:54-49 9 97 16 0 DieHard DieHard DieHard DieHard DieHard DieHard DieHard DieHard	Errors detected:			No errors				
E Statistics State space progress (click column header for graph) Time Diameter States Found Distinct States Queue Size 2017-04-20 11:56-49 9 9 97 16 0 Distind Disting distin	Fingerprint collision p	robability:	calculated	l: 7.0E-17, observe	ed: 3.7E-18			
State space progress (click column header for graph) State space progress (click column header for graph) Timme Diameter States Found Distinct States 2017-04-20 11:54-49 9 97 16 0 Head Diversard D	Statistics							
State space progress (click column header for graph) Coverage at 201 Time Diameter States Found Distinct States Queue Size 2017-04-2011:54-49 9 97 16 DiefHard DieHard DieHard DieHard DieHard DieHard DieHard DieHard DieHard								
Time Diameter States Found Distinct States Queue Size Module 2017-04-20 11:5649 9 97 16 0 DieHard DieHard 0 0 DieHard DieHard DieHard 0 0 DieHard DieHard DieHard 0 0 DieHard DieHard	State space progress (lick column	header for gra	ph)		Coverage at 201		
2017-04-20 11:54:49 9 97 16 0 Divisind Divisind Divisind Divisind Divisind Divisind Divisind Divisind Divisind Divisind Divisind	Time	Diameter	States Found	Distinct States	Queue Size	Module		
DieHand DieHand DieHand DieHand DieHand	2017-04-20 11:54:49	9	97	16	0	DieHard		
DieHard DieHard DieHard DieHard						DieHard		
DieHard DieHard						DieHard		
DieHard						DieHard		
						DieHard		
	Expression:					Value		
Expression: Value						^		

TLC quickly finishes, displaying the

General					
Start time:		Thu Apr	20 11:54:48 PDT 20	017	
End time:		Thu Apr	20 11:54:49 PDT 20	017	
Last checkpoint time:					
Current status:		Not runn	ing		
Errors detected:			No errors		
Fingerprint collision p	robability:	calculated	l: 7.0E-17, observe	ed: 3.7E-18	
- Statistics					
Charles	Calculation of the second		-		Courses at 20
state space progress (c	lick column	i neader for gra	pn)		Coverage at 20
Time	Diameter	States Found	Distinct States	Queue Size	Module
2017-04-20 11:54:49	9	97	16	0	DieHard
Evaluate Constant E	kpression				
Expression:					Valu
					^

TLC quickly finishes, displaying the Model Checking Results page

) 🔐 🔳 🖨 🖙 🕼						TLC reports no errors
General						
Start time:		Thu Apr	20 11:54:48 PDT 20	17		
End time:		Thu Apr	20 11:54:49 PDT 20	17		
Last checkpoint time:						
Current status:		Not runn	ing			
Errors detected:			No errors			
Fingerprint collision p	robability:	calculated	: 7.0E-17, observe	ed: 3.7E-18		
Statistics						
State space progress (o	lick column	header for gra	ph)		Coverage at 201	
Time	Diameter	States Found	Distinct States	Queue Size	Module	
2017-04-20 11:54:49	9	97	16	0	DieHard	
					DieHard	
					DieHard	
					DieHard	
					DieHard	
Evaluate Constant F	enression					
Evaluate constant L	Apression -				Value	
					<u>A</u>	

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

Model Checki	iy Resu	iits				TLC reports po orrors
) 🖑 🗏 🛱 🖙 🕼	1					TLC reports no errors.
General						
Start time:		Thu Apr	20 11:54:48 PDT 20	17		
ind time:		Thu Apr	20 11:54:49 PDT 20	17		This means it could
Last checkpoint time:						This means it could
Current status:		Not runn	ing			rup the ener
Errors detected:			No errors			run me spec.
ingerprint collision p	robability:	calculated	l: 7.0E-17, observe	ed: 3.7E-18		
Statistics						
tate space progress (o	lick column	header for gra	ph)		Coverage at 201	
Time	Diameter	States Found	Distinct States	Queue Size	Module	
2017-04-20 11:54:49	9	97	16	0	DieHard	
					DieHard	
					DieHard	
					DieHard	
Evaluate Constant E	xpression					
expression:					Value	
					^	

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
General						
Start time:		Thu Apr	20 11:54:48 PDT 20	17		reachable states.
End time:		Thu Apr	20 11:54:49 PDT 20	17		
Last checkpoint time:						
Current status:		Not runn	iing			
Errors detected:			No errors			
Fingerprint collision p	robability:	calculated	l: 7.0E-17, observe	ed: 3.7E-18		
Canalistics						
Statistics						
State space progress (o	lick column	header for gra	ph)		Coverage at 201	
Time	Diameter	States Found	Distinct States	Queue Size	Module	
2017-04-20 11:54:49	9	97	16	0	DieHard	
					DieHard	
					DieHard	
					DieHard	
					Dierlard	
Evaluate Constant E	xpression					
Expression:					Value	
					^	

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

[slide 193]

Hodel Checki	ng Resu	ilts				
) 🔐 🗏 🛱 😔 [1					TLC found 16
General						
Start time:		Thu Apr.	20 11:54:48 PDT 20	17		reachable states.
End time:		Thu Apr	20 11:54:49 PDT 20	17		
Last checkpoint time:						
Current status:		Not runn	ing			(States occurring in some
Errors detected:			No errors			· · · · · · · · · · · · · · · · · · ·
Fingerprint collision p	robability:	calculated	1: 7.0E-17, observe	ed: 3.7E-18		behavior allowed by spec.
Statistics						2 1
State space progress (lick column	ı header for gra	ph)		Coverage at 201	
	Diameter	States Found	Distinct States	Queue Size	Module	
Time		97	16	0	DieHard	
Time 2017-04-20 11:54:49	9				DieHard	
Time 2017-04-20 11:54:49	9				Dicitatu	
Time 2017-04-20 11:54:49	9				DieHard	
Time 2017-04-20 11:54:49	9				DieHard DieHard DieHard	
Time 2017-04-20 11:54:49	9				DieHard DieHard DieHard DieHard	
Time 2017-04-20 11:54:49 Evaluate Constant E	9 xpression				DieHard DieHard DieHard	
Time 2017-04-20 11:54:49 Evaluate Constant E Expression:	9 xpression				DieHard DieHard DieHard DieHard	

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.

[slide 194]

Model Overview	Advanced Options	Model Checking Results
----------------	------------------	------------------------

Model Checking Results

0 🔐 🔳 🛱 🖙 🗎

Let's check	type	correctness
-------------	------	-------------

General
Start time:
End time:

Start time:	Thu Apr 20 11:54:48 PDT 2017
End time:	Thu Apr 20 11:54:49 PDT 2017
Last checkpoint time:	
Current status:	Not running
Errors detected:	No errors
Fingerprint collision probability:	calculated: 7.0E-17, observed: 3.7E-18

1	ingerprint
-	Statistics

Time	Diameter	States Found	Distinct States	Queue Size	Module	
2017-04-20 11:54:49	9	97	16	0	DieHard	
					DieHard	
					DieHard	
					DieHard	
					DieHard	
xpression:						2
					^	

Let's now check type correctness - which means

Model Checking Results

0 🔐 🔳 🖴 🕞

		I.	ra	c	en	G	-	
--	--	----	----	---	----	---	---	--

Statistics

Start time:	Thu Apr 20 11:54:48 PDT 2017
End time:	Thu Apr 20 11:54:49 PDT 2017
Last checkpoint time:	
Current status:	Not running
Errors detected:	No errors
Fingerprint collision probability:	calculated: 7.0E-17, observed: 3.7E-18

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

Time	Diameter	States Found	Distinct States	Queue Size	Module
2017-04-20 11:54:49	9	97	16	0	DieHard
					DieHard
Evaluate Constant E	xpression				
Evaluate Constant Expression:	xpression				
Evaluate Constant E xpression:	xpression				0
Evaluate Constant E xpression:	xpression				^
Evaluate Constant E: xpression:	xpression				^
Evaluate Constant E xpression:	xpression				

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

[slide 196]

Model Checki	ng Resu	lits				Lat's shack type correction
) 💞 🗏 🛱 💬 🎼						Let's check type correctnes
General						
Start time: End time: Last checkpoint time: Current status: Errors detected: Fingerprint collision p	robability:	Thu Apr Thu Apr Not runn calculated	20 11:54:48 PDT 20 20 11:54:49 PDT 20 ing <u>No errors</u> : 7.0E-17, observe	017 017 ed: 3.7E-18		satisfies TypeOK).
Statistics						
State space progress (o	lick column	header for gra	oh)		Coverage at 20	
Time 2017-04-20 11:54:49	Diameter 9	States Found 97	Distinct States	Queue Size 0	Module DieHard DieHard DieHard DieHard DieHard	
Evaluate Constant E	xpression				Valu	

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.

	Thu Apr	20 11:54:48 PDT 20	017	
	Thu Apr	20 11:54:49 PDT 20	017	
	Not runn	ing		
		No errors		
robability:	calculated	l: 7.0E-17, observe	ed: 3.7E-18	
lick columr	header for gra	ph)		Coverage at 201
Diameter	States Found	Distinct States	Queue Size	Module
9	97	16	0	DieHard
				a ter fara
xpression				
				Valu
				^
	obability: lick column Diameter 9 cpression	Thu Apr Thu Apr Not runn obability: calculated lick column header for gra Diameter 9 97 spression	Thu Apr 20 11:54:48 PDT 21 Thu Apr 20 11:54:48 PDT 21 Not running Not errors calculated: 7.06-17, observi- lick column Header for graph graph 97 16 16	Thu Apr 20 11:54:48 POT 2017 Thu Apr 20 11:54:48 POT 2017 Not running <u>Not running</u> Ideamost calculated: 7.05-17, observed: 3.75-18 Ilck column header for graph/ Diameter States Found Distinct States 9 97 16 0 States Pot 20 10 States Po

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.

[slide 198]

🖶 Model Overview	
D 🔐 🗏 🛱 🐖	
Model description	
The Million is the background of the second	D Milestie Alexander
What is the behavior spec:	Specify the values of declared constants.
Initial predicate and next-state relation	
Init Init	
Next Next	
 Temporal formula 	
Ô	Advanced parts of the model: <u>Additional definitions</u> , <u>State constraints</u> , <u>Action constraints</u> ,
O No Behavior Spec	How to run?
What to check?	
_	
Deadlock	
Invariants	
Properties	

Model Overview		
) 🔐 🗏 🛱 👳		
Model description		
What is the behavior spec?	What is the model?	
	Specify the values of declared constants.	A formula true in every
Init Init		
Next Next		reachable state is called
Temporal formula		
~ ~	Advanced parts of the model: <u>Additional definitions.</u> <u>State constraints.</u> <u>Action constraints.</u>	an <i>invariant</i> .
No Behavior Spec	How to run?	
What to check?		
Deadlock		
Invariants		
m Bernetter		

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

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

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.

What to check?		
✓ Deadlock		
 Invariants Formulas true in every 	reachable state.	
	Add Edit Remove	
• Properties		

Enter TypeOK. And click on Finish.

[slide 202]

What to check?	
Deadlock	
Formulas true in every reachable state.	
Add	
Remove	
Properties	

Enter TypeOK. And click on Finish.

[slide 203]

	- 🗆 X	
Formulas true in every reachable state.		
	^	
0	Finith Cancel	
U	- Concer	

Enter TypeOK. And click on Finish.

[slide 204]

11 Contraction of the second sec	– 🗆 X	
Invariants Formulas true in every reachable state.		
Typeor	^	
	~	
?	Finish Cancel	

Enter TypeOK. And click on Finish.

[slide 205]

11 Contraction of the second sec	– 🗆 X	
Invariants Formulas true in every reachable state.		
Typeor	^	
	~	
?	Finish Cancel	

Enter TypeOK. And click on Finish.

[slide 206]

Invariants		
Formulas true in every reachable state.		
TypeoK	×) ~	
0	Finish Cancel	

Enter TypeOK. And click on Finish.

[slide 207]

Deadlock Invariants Formulas true in every reachable stat TypeOK Add	
Invariants Formulas true in every reachable stat TypeOK Add	
TypeOK Add	ste.
Edit Remove	
Properties	

Enter TypeOK. And click on Finish.

[slide 208]



Enter TypeOK. And click on Finish.

And run TLC on the model again.

[slide 209]

	Ĩ					
# = = = = !!]					
eneral						
irt time:		Thu Apr	20 15:59:31 PDT 20	17		
d time:		Thu Apr	20 15:59:32 PDT 20	17		
t checkpoint time:						
rrent status:		Not runn	ing			
ors detected:			No errors			
gerprint collision p	robability:	calculated	l: 7.0E-17, observe	d: 3.7E-18		
Indiction						
and a construction of the second s						
te space progress (o	lick column	header for gra	ph)		Coverage at 201	
ime	Diameter	States Found	Distinct States	Queue Size	Module	
017-04-20 15:59:32	9	97	16	0	DieHard	
					DieHard	
					DieHard	
					DieHard	
					DieHard	
valuate Constant E	xpression					
pression:					Value	
					^	

The Model Checking Results page

)					
General						
Start time:		Thu Apr 2	20 15:59:31 PDT 20	017		
End time:		Thu Apr 2	20 15:59:32 PDT 20	017		
Last checkpoint time:						
Current status:		Not runn	ing			
Errors detected:			No errors			
Fingerprint collision p	obability:	calculated	: 7.0E-17, observe	ed: 3.7E-18		
Statistics State space progress (o	lick columr	header for grap	ph)		Coverage at 201	TLC reports no errors.
Time	Diameter	States Found	Distinct States	Queue Size	Module	
2017-04-20 15:59:32	9	97	16	0	DieHard	
					DieHard	
					DieHard	
					DieHard	
					DieHard	
Evaluate Constant E	cpression					
Evaluate Constant E	cpression				Value	
Evaluate Constant E Expression:	cpression (Value	
Evaluate Constant E Expression:	cpression				Value	

The Model Checking Results page

again shows that TLC found no errors.

Saving Our Heroes

Now we're ready to save our heroes.

[slide 212]

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.

[slide 213]

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.

[slide 214]

```
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.



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*).


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.

[slide 217]

What to check?	
🗹 Deadlock	
 Invariants Formulas true in every res 	achable state.
☑ ТуреОК	Add Edit Remove
Properties	

What to check?		
✓ Deadlock		
 Invariants Formulas true in every reachable 	state.	
TypeOK Ac	id it ove	
Properties		

We add

[slide 219]

	– o x	
Formulas true in every reachable state.		
	^	
	,	
(?)	Finish Cancel	

We add another invariant

[slide 220]

📽 Invariants	- D X	
Formulas true in every reachable state.		
big /= 4	×	
0	Finish Cancel	

We add another invariant big not equal to 4

😭	- C X		
Formulas true in every reachable state.			
big /= 4	^		
	~		
0	Finish Cancel	-	

We add another invariant big not equal to 4

What to check?			
Deadlock Invariants Formulas true in even 	reachable state.		
 ✓ TypeOK ✓ big /= 4 	Add Edit Remove		
Properties			

We add another invariant big not equal to 4



We add another invariant big not equal to 4

And we run TLC.

[slide 224]

) 🔐 🗏 🛤 🚱	1					
General						
Start time:		Fri Apr 21	11:30:52 PDT 2017			
End time:		Fri Apr 21	11:30:52 PDT 2017			
Last checkpoint time:						
Current status:		Not runni	ng			
Errors detected:		1	Error			
Fingerprint collision p	robability:					
State space progress (e	lick column	header for gra	ph) Distinct States	Queue Size	Coverage at 201 Module	
2017-04-21 11-30-52	7	73	14	Queue Size	DieHard	
2017 04 21 11:50:52		15			DieHard	
					DieHard	l de la construcción de la constru
					DieHard	
					DieHard	
Evaluate Constant E	xpression					
Expression:	÷				Value	
					^	



And the Toolbox opens this error window.



And the Toolbox opens this error window.

Which tells us that the invariant was violated

[slide 227]



And the Toolbox opens this error window.

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



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

[slide 229]

Model_1	C	<u>0</u>
Invariant big /= 4 is violat	ed.	\$
Error-Trace Exploration		
Error-Trace		A behavior ending in state
Name ✓ kinial predicate> big small ✓ Action line 15, col 12 to li ● big small ✓ Action line 30, col 15 to li ● big small ✓ Action line 30, col 15 to li ● big small ✓ Action line 30, col 15 to li ● big small ✓ Action line 30, col 15 to li ● big small ✓ Action line 15, col 15 to li ● big small ✓ < Action line 30, col 15 to li ● big small ✓ Action line 30, col 15 to li ● big small	Value	$\begin{bmatrix} big & : 4 \\ small : 3 \end{bmatrix}$ with the invariant false.

Invariant big /= 4 is violat	ed. ^	The complete behavior:
<	>	The complete behavior.
Error-Trace Exploration		
Error-Trace		
Name	Value	
✓ ▲ <initial predicate=""></initial>	State (num = 1)	
🖬 big	0	
small	0	
✓ ▲ <action 12="" 15,="" col="" li<="" line="" p="" to=""></action>	State (num = 2)	
big	5	
small	0	
🗸 🔺 <action 15="" 30,="" col="" li<="" line="" td="" to=""><td>State (num = 3)</td><td></td></action>	State (num = 3)	
big	2	
small	3	
🗸 🔺 <action 15="" 18,="" col="" li<="" line="" td="" to=""><td>State (num = 4)</td><td></td></action>	State (num = 4)	
big	2	
small	0	
✓ ▲ <action 15="" 30,="" col="" li<="" line="" p="" to=""></action>	State (num = 5)	
🖬 big	0	
small	2	
✓ ▲ <action 12="" 15,="" col="" li<="" line="" p="" to=""></action>	State (num = 6)	
big	5	
small	2	
✓ ▲ <action 15="" 30,="" col="" li<="" line="" p="" to=""></action>	State (num = 7)	
big	4	
small	3	

It shows this complete behavior

[slide 231]

Invariant big /= 4 is viol	ated.	The complete behavior:
<		
Error-Trace		
		big : 0
Name	Value	small · 0
✓ ▲ <initial predicate=""></initial>	State (num = 1)	
■ big	0	
■ small	0	
Action line 15, col 12 to	o li State (num = 2)	
Big	5	
small	0	
Action line 30, col 15 to	o li State (num = 3)	
Big	2	
small	5	
Action line 18, col 15 to	o II State (num = 4)	
a big	0	
 small cAction line 30 col 15 tr 	o li State (num - 5)	
big	0	
small	2	
× A section line 15 col 12 to	o li State (num = 6)	
B big	5	
=	2	
✓ ▲ <action 15="" 30,="" col="" line="" p="" to<=""></action>	o li State (num = 7)	
B big	4	
a coall	2	

It shows this complete behavior

[slide 232]

Model_1		
Invariant big /= 4 is violated.		The complete behavior:
Error-Trace Exploration		
Error-Trace		
Name	Value	$\begin{vmatrix} big & : 0 \\ amall & 0 \end{vmatrix} \rightarrow \begin{vmatrix} big & : 5 \\ amall & 0 \end{vmatrix}$
A <initial predicate=""></initial>	State (num = 1)	sman. O sman. O
■ big	0	
small	0	
Action line 15, col 1	(num = 2)	
a big	3	
strian	5 to li State (num = 2)	
 Accion nine 50, con n bio 	2	
small	3	
✓ ▲ <action 18,="" 1<="" col="" line="" p=""></action>	15 to li State (num = 4)	
big	2	
small	0	
✓ ▲ <action 1<="" 30,="" col="" line="" p=""></action>	15 to li State (num = 5)	
🔳 big	0	
small	2	
Action line 15, col 1	12 to li State (num = 6)	
big	5	
small	2	
✓ ▲ <action 1<="" 30,="" col="" line="" p=""></action>	15 to li State (num = 7)	
B big	4	

It shows this complete behavior

[slide 233]

Invariant big (= 4 is vio	lated	•
invariant big /= 4 is vio	lated.	The complete behavior:
Error-Trace Exploration		
Error-Trace		$\begin{bmatrix}bia & 0\end{bmatrix}$ $\begin{bmatrix}bia & 5\end{bmatrix}$ $\begin{bmatrix}bia & 1\end{bmatrix}$
Name	Value	$ a = 0 \rightarrow a $
✓ ▲ <initial predicate=""></initial>	State (num = 1)	smail: 0 smail: 0 smail: 0
🖬 big	0	
small	0	
✓ ▲ <action 12<="" 15,="" col="" line="" p=""></action>	to li State (num = 2)	
iii big	5	
small	0	
✓ ▲ <action 15<="" 30,="" col="" line="" p=""></action>	to li State (num = 3)	
B big	2	
small	5	
 Action line 18, col 13 	con state (num = 4)	
a org	0	
A sAction line 30 col 15	to li State (num = 5)	
B big	0	
small	2	
✓ ▲ <action 12<="" 15,="" col="" line="" p=""></action>	to li State (num = 6)	
■ big	5	
small	2	
✓ ▲ <action 15<="" 30,="" col="" line="" p=""></action>	to li State (num = 7)	
■ big	4	
small	3	

It shows this complete behavior

[slide 234]



It shows this complete behavior

[slide 235]



It shows this complete behavior

[slide 236]



It shows this complete behavior

[slide 237]



It shows this complete behavior

[slide 238]



It shows this complete behavior gallons of water in the big jug.

From this behavior, our heroes should be able to see how to get 4



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.



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

[slide 241]



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

[slide 242]



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

[slide 243]



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.

[slide 244]



Formulas *SmallToBig* and *BigToSmall*.

[slide 245]

SmallToBig ==

BigToSmall ==

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

[slide 246]

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

Let's start with *SmallToBig*.

[slide 247]

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



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

[slide 250]

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.

[slide 251]

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



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.

 $A \wedge B = B \wedge A$

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

[slide 255]

 $A \wedge B = B \wedge A$

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.

[slide 256]

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.

[slide 257]

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

.

WARNING!

Here's a warning about writing specs.

[slide 260]



The equality operator is also commutative.

.

[slide 261]



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

.

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

.

1



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]



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 265]



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 266]



And the TLAPS proof system treats them exactly the same.

[slide 267]



And the TLAPS proof system treats them exactly the same.

But TLC handles only this one.

[slide 268]



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.

[slide 269]

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.

[slide 271]

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.

[slide 272]

For now, just follow this simple rule:

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

 $v' = \dots$ and $v' \in \dots$

For now, just follow this simple rule:

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

[slide 274]

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



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.

[slide 275]

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



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.

[slide 276]



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

[slide 277]

Your definitions are probably not exactly the same as mine.

But they may still be correct.

Your definitions are probably not exactly the same as mine.

But they may still be correct.

[slide 279]

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.

[slide 280]

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.

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.





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



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



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



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

Now save the spec.

[slide 287]



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.

[slide 288]


And it puts this error mark in the module editor.

[slide 289]



And it puts this error mark in the module editor.

Clicking here in the error window

[slide 290]



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.

[slide 291]

Another common error found by parsing:

Here's another common error found by the parser.

[slide 292]

Another common error found by parsing:

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.

[slide 293]

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.

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

[slide 295]

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.)

For example, change this five to quote five.

[slide 296]

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.)

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

Running TLC now produces

[slide 297]

TLC "Execution Errors"					
🍫 TLC Errors 🕱 🗖					
Model_1					
<pre>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</pre>					
The error occurred when TLC was evaluating the nested					
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

[slide 298]

TLC "Execution Errors"					
🎨 TLC Errors 🛛					
Model_1					
<pre>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 :</pre>					
The error occurred when TLC was evaluating the nested					
expressions at the following positions:					
U. Line 27, Column 24 to line 27, Column 53 in DieMard					
2. Line 28. column 33 to line 28. column 51 in DieMard					
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]



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**.

[slide 300]



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

[slide 301]

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

First

[slide 302]

Comment out my definitions

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

First comment out my definitions by

[slide 303]

Comment out my definitions

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

First comment out my definitions by adding these comment delimiters.

[slide 304]

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

First comment out my definitions by adding these comment delimiters.

And add your own definitions.

[slide 305]



Save your definitions

[slide 306]

Save your definitions and correct any errors the parser finds.

Run TLC.

Save your definitions and correct any errors the parser finds.

Run TLC.

[slide 308]

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.

[slide 309]

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.

[slide 310]

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 $\mathit{big} \neq 4$.

[slide 311]

To be sure, go here			
	□ Invariants Formulas true in every reachable state.		
	✓ TypeOK ✓ big /= 4	Add	
		Remove	

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



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

Uncheck this box.

[slide 313]

To be sure, go here				
	 Invariants Formulas true in every reachable state. 			
	✓ TypeOK Add ☐ big /= 4 Edit Remove			
Uncł	neck 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.

[slide 314]

To be sure, go here					
	 Invariants Formulas true in every reachable state. 				
	✓ TypeOK Add big /= 4 Edit Remove				
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.

[slide 315]



If TLC finds no error

[slide 316]

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.

[slide 317]

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.

[slide 318]

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]



[slide 320]