# Predicting and Preventing Chaos with Formal Methods in TLA+

Thomas Gebert

July 29, 2022

# Who Am I?

Thomas Gebert

# Who Am I?

Thomas Gebert

- Software Engineer

# Who Am I?

Thomas Gebert

- Software Engineer
- Ph.D. Student at University of York.

# Who Am I?

Thomas Gebert

- Software Engineer
- Ph.D. Student at University of York.
- Cartoon Enthusiast

# What are Formal Methods?

# What are Formal Methods?

- Toolkits for explaining a program mathematically.

# What are Formal Methods?

- Toolkits for explaining a program mathematically.
- Ways in which to describe an algorithm without describing unnecessary details.

# Examples of Formal Method Systems.

Super Theoretical

# Examples of Formal Method Systems.

## Super Theoretical

- Isabelle/HOL
- Coq
- Agda

# Examples of Formal Method Systems.

## Super Theoretical

- Isabelle/HOL
- Coq
- Agda

## Engineer-Focused

# Examples of Formal Method Systems.

## Super Theoretical

- Isabelle/HOL
- Coq
- Agda

## Engineer-Focused

- FDR4/CSP-M

# Examples of Formal Method Systems.

## Super Theoretical

- Isabelle/HOL
- Coq
- Agda

## Engineer-Focused

- FDR4/CSP-M
- Alloy

# Examples of Formal Method Systems.

## Super Theoretical

- Isabelle/HOL
- Coq
- Agda

## Engineer-Focused

- FDR4/CSP-M
- Alloy
- TLA+

# Examples of Formal Method Systems.

## Super Theoretical

- Isabelle/HOL
- Coq
- Agda

## Engineer-Focused

- FDR4/CSP-M
- Alloy
- TLA+

## In Between

# Examples of Formal Method Systems.

## Super Theoretical

- Isabelle/HOL
- Coq
- Agda

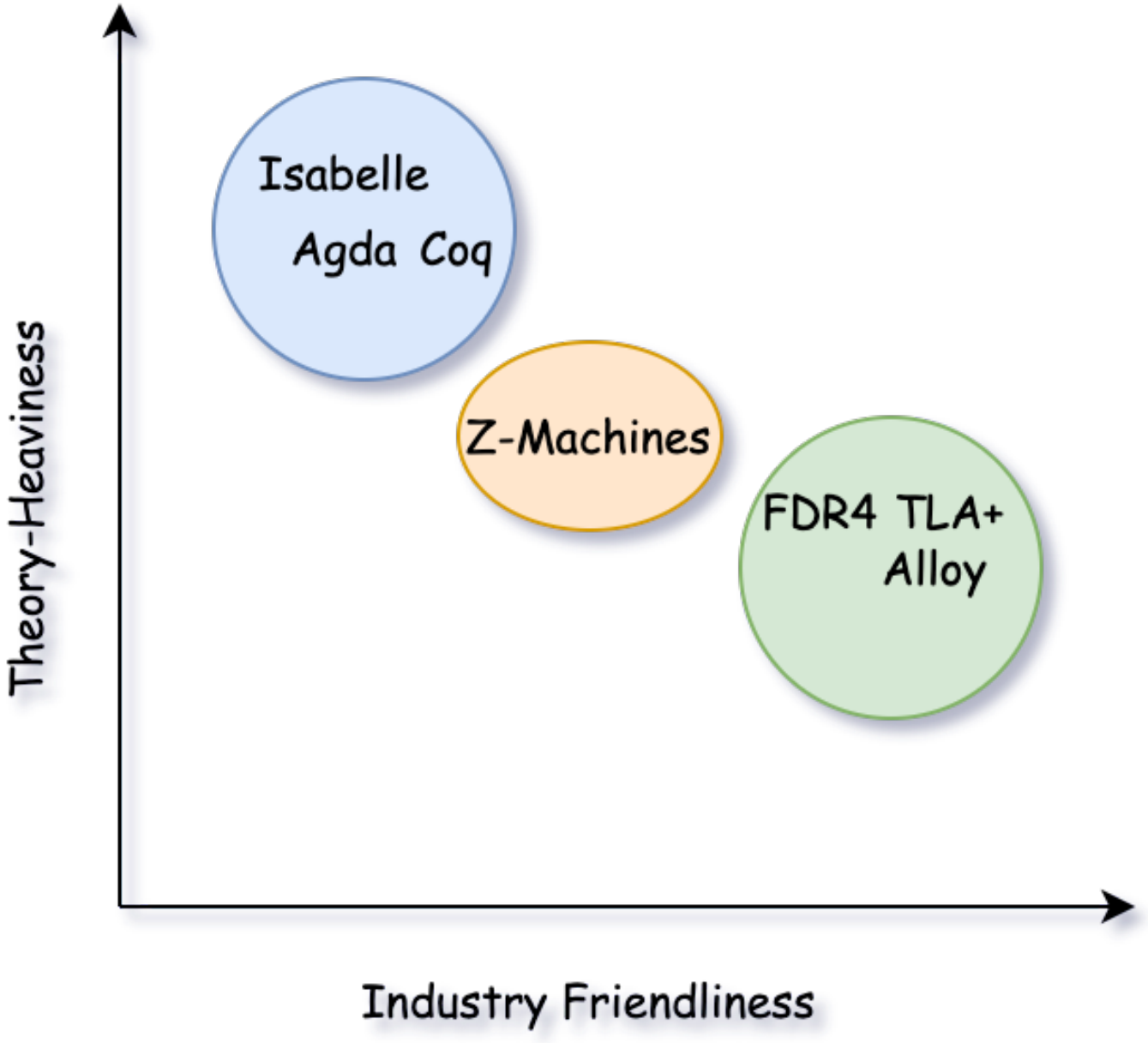
## Engineer-Focused

- FDR4/CSP-M
- Alloy
- TLA+

## In Between

- Z Machines

# Examples of Formal Method Systems.



# What do Formal Methods give you?

# What do Formal Methods give you?

- Bragging Rights.

# What do Formal Methods give you?

- Bragging Rights.
- A forced understanding as to what you're actually building.

# What do Formal Methods give you?

- Bragging Rights.
- A forced understanding as to what you're actually building.
- A model of the algorithm that lives "above" the code.

# What do Formal Methods give you?

- Bragging Rights.
- A forced understanding as to what you're actually building.
- A model of the algorithm that lives "above" the code.
- This allows you to avoid worrying about unimportant details of the *algorithm*.

# What do Formal Methods give you?

- Bragging Rights.
- A forced understanding as to what you're actually building.
- A model of the algorithm that lives "above" the code.
- This allows you to avoid worrying about unimportant details of the *algorithm*.
- "If You're Not Writing a Program, Don't use a Programming Language" – Lamport.

# What is TLA+?

# What is TLA+?

- "Temporal Logic of Actions."

# What is TLA+?

- "Temporal Logic of Actions."
- Formal specification language.

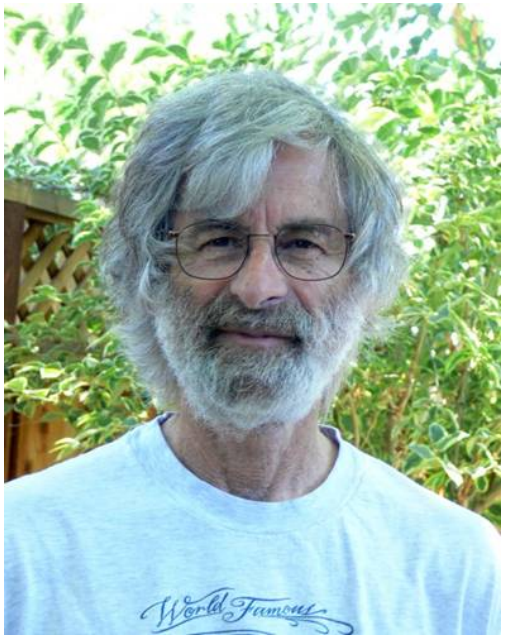
# What is TLA+?

- "Temporal Logic of Actions."
- Formal specification language.
- Uses a combination of set theory, state machines, and temporal logic to describe programs.

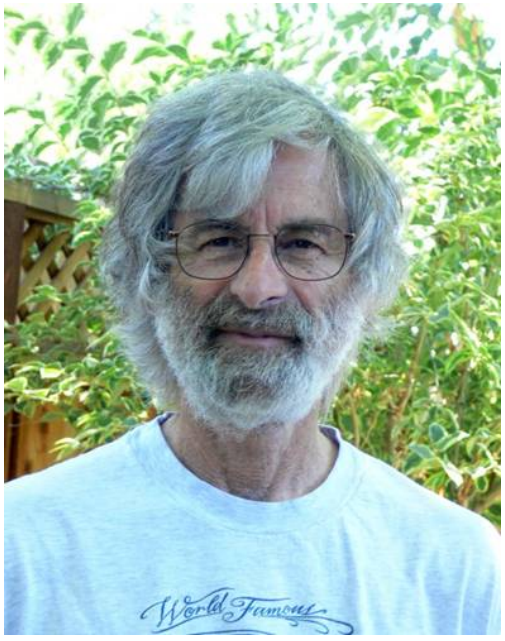
# What is TLA+?

- "Temporal Logic of Actions."
- Formal specification language.
- Uses a combination of set theory, state machines, and temporal logic to describe programs.
- Specifications can be model checked for correctness.

# Leslie Lamport.

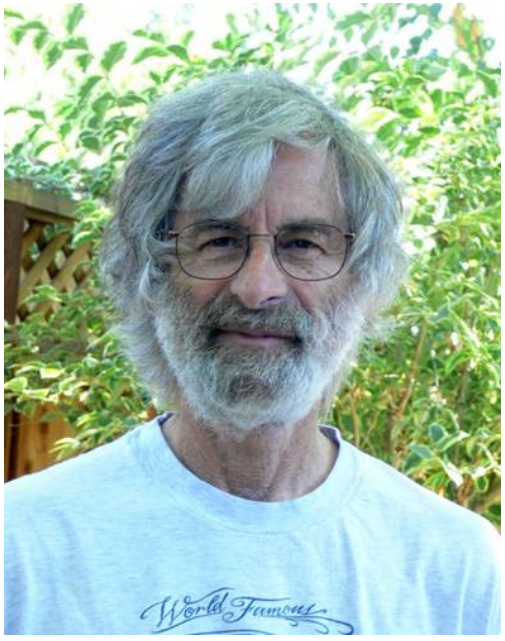


# Leslie Lamport.



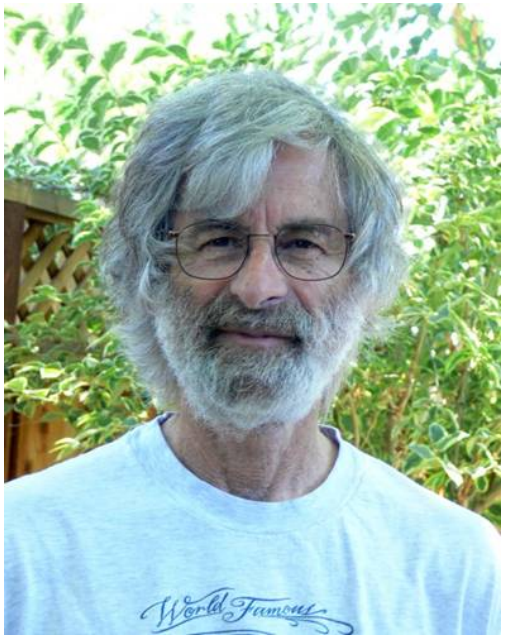

- Inventor of Paxos

# Leslie Lamport.



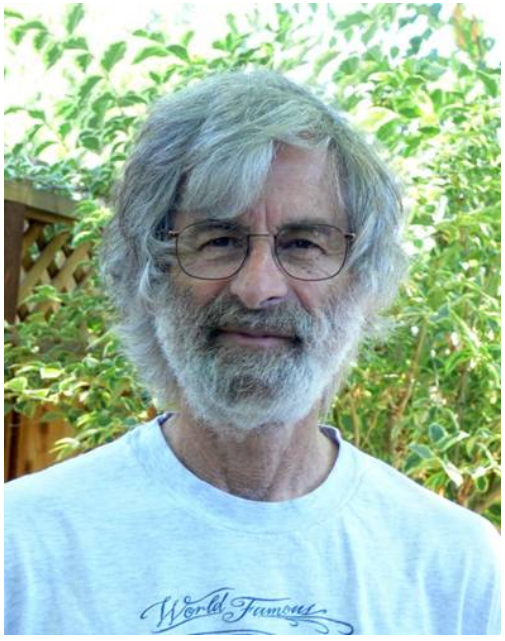

- Inventor of Paxos
- Inventor of Bakery Algorithm

# Leslie Lamport.



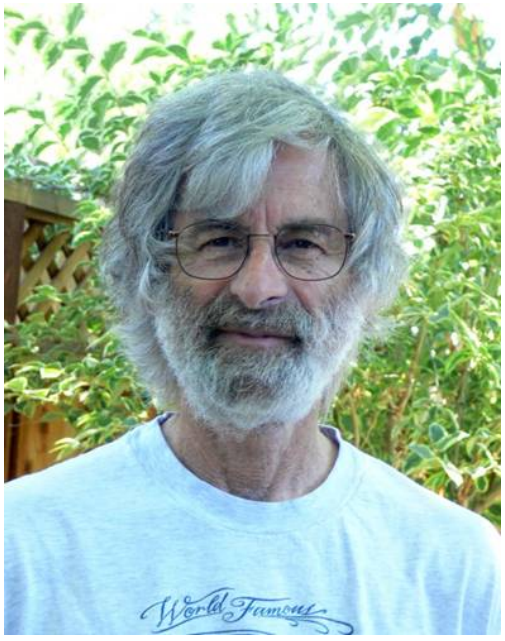

- Inventor of Paxos
- Inventor of Bakery Algorithm
- Inventor of Lamport Timestamps

# Leslie Lamport.




- Inventor of Paxos
- Inventor of Bakery Algorithm
- Inventor of Lamport Timestamps
- Inventor of LaTeX

# Leslie Lamport.




- Inventor of Paxos
- Inventor of Bakery Algorithm
- Inventor of Lamport Timestamps
- Inventor of LaTeX
- Inventor of TLA+

# Why TLA+ over other systems?

# Why TLA+ over other systems?

- TLA+ is focused specifically on *software engineering* problems.

# Why TLA+ over other systems?

- TLA+ is focused specifically on *software engineering* problems.
- Comparatively less mathematics is required to become useful.
  - Able to increase usage of more interesting mathematics as one becomes more comfortable.

# Why TLA+ over other systems?

- TLA+ is focused specifically on *software engineering* problems.
- Comparatively less mathematics is required to become useful.
  - Able to increase usage of more interesting mathematics as one becomes more comfortable.
- Can work at nearly any level of a computational system desired.

# What is TLA+ missing over over systems?

- The proof system is not as versatile or intuitive as something like Isabelle or Coq.

# What is TLA+ missing over over systems?

- The proof system is not as versatile or intuitive as something like Isabelle or Coq.
- Model checking is a fair bit slower than something like FDR4.

# What is TLA+ missing over over systems?

- The proof system is not as versatile or intuitive as something like Isabelle or Coq.
- Model checking is a fair bit slower than something like FDR4.
- There is much less academic literature in TLA+ than Isabelle or Coq.

# What is TLA+ missing over over systems?

- The proof system is not as versatile or intuitive as something like Isabelle or Coq.
- Model checking is a fair bit slower than something like FDR4.
- There is much less academic literature in TLA+ than Isabelle or Coq.
- No typing
  - Can be worked around with type invariants.

# What is TLA+ missing over over systems?

- The proof system is not as versatile or intuitive as something like Isabelle or Coq.
- Model checking is a fair bit slower than something like FDR4.
- There is much less academic literature in TLA+ than Isabelle or Coq.
- No typing
  - Can be worked around with type invariants.
- No code export from TLA+ specifications to “real” code

# What is TLA+ missing over over systems?

- The proof system is not as versatile or intuitive as something like Isabelle or Coq.
- Model checking is a fair bit slower than something like FDR4.
- There is much less academic literature in TLA+ than Isabelle or Coq.
- No typing
  - Can be worked around with type invariants.
- No code export from TLA+ specifications to “real” code
- . . . It’s imperative. . .

# Industrial Use of TLA+.

# Industrial Use of TLA+.

- Amazon

# Industrial Use of TLA+.

- Amazon
  *As of February 2014, we have used TLA+ on 10 large complex real-world systems. In every case TLA + has added significant value, either preventing subtle serious bugs from reaching production, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness.*

# Industrial Use of TLA+.

- Amazon
  *As of February 2014, we have used TLA+ on 10 large complex real-world systems. In every case TLA + has added significant value, either preventing subtle serious bugs from reaching production, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness.*
- Microsoft uses TLA+ for use for verifying consistency levels in CosmosDB

# Industrial Use of TLA+.

- Amazon
  *As of February 2014, we have used TLA+ on 10 large complex real-world systems. In every case TLA + has added significant value, either preventing subtle serious bugs from reaching production, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness.*
- Microsoft uses TLA+ for use for verifying consistency levels in CosmosDB
- MongoDB uses TLA+ for verifying replication.

# Large Projects

# Large Projects

OpenComRTOS

# Large Projects

## OpenComRTOS

- Real-time operating system, fully specified in TLA+ before being built.

# Large Projects

## OpenComRTOS

- Real-time operating system, fully specified in TLA+ before being built.

## Pastry

# Large Projects

## OpenComRTOS

- Real-time operating system, fully specified in TLA+ before being built.

## Pastry

- A distributed hash table algorithm, specified and proven with TLA+.

# TLA+ Platform.

# TLA+ Platform.

- TLA+ Language

# TLA+ Platform.

- TLA+ Language
- TLA+ Toolbox

# TLA+ Platform.

- TLA+ Language
- TLA+ Toolbox
- PlusCal

# TLA+ Platform.

- TLA+ Language
- TLA+ Toolbox
- PlusCal
- TLAPS

# TLA+ Platform.

- TLA+ Language
- TLA+ Toolbox
- PlusCal
- TLAPS
- TLC

# TLA+ Semantics - Conjunction and Disjunction

# TLA+ Semantics - Conjunction and Disjunction

- /\ and \/
- $\wedge$ and $\vee$
- `1 = 0 \/ 1 = 1`

# TLA+ Semantics - Conjunction and Disjunction

- /\ and \/
- $\wedge$ and $\vee$
- `1 = 0 \/ 1 = 1`

```
\/  /\ x = 1
    /\ y = 2
\/  /\ x = 3
    /\ y = 4
```

# TLA+ Semantics - Predicate Logic

# TLA+ Semantics - Predicate Logic

- Implies
  - `=>`
  - $\Rightarrow$
  - `MyVariable \in {"ok", "fine"} => IsStable`

# TLA+ Semantics - Predicate Logic

- Implies
  - `=>`
  - $\Rightarrow$
  - `MyVariable \in {"ok", "fine"} => IsStable`
- If and only if
  - `<=>`
  - $\Leftrightarrow$

# TLA+ Semantics - Sets

## Set Membership

- `1 \in {1, 2, 3}`
- $1 \in \{1, 2, 3\}$

## Union

- `{1,2,3} \union {3,4,5}`
- $\{1, 2, 3\} \cup \{3, 4, 5\}$

# TLA+ Semantics - Operators

# TLA+ Semantics - Operators

- Look like functions, closer to macros.

# TLA+ Semantics - Operators

- Look like functions, closer to macros.

```
MyOperator(x) = x + 1
```

# TLA+ Semantics - Functions and Tuples

# TLA+ Semantics - Functions and Tuples

## Tuples

# TLA+ Semantics - Functions and Tuples

## Tuples

- `<<1, 2, 3>>`

# TLA+ Semantics - Functions and Tuples

## Tuples

- `<<1, 2, 3>>`

## Functions

# TLA+ Semantics - Functions and Tuples

## Tuples

- `<<1, 2, 3>>`

## Functions

- Closer to a map than a “function” in most programming languages.

# TLA+ Semantics - Functions and Tuples

## Tuples

- `<<1, 2, 3>>`

## Functions

- Closer to a map than a "function" in most programming languages.
- Can be recursively defined:

# TLA+ Semantics - Functions and Tuples

## Tuples

- `<<1, 2, 3>>`

## Functions

- Closer to a map than a "function" in most programming languages.
- Can be recursively defined:

```
fact[x \in Int] == IF x <= 0 THEN 1 ELSE x * fact[x-1]
```

# TLA+ Semantics - Functions and Tuples

## Tuples

- `<<1, 2, 3>>`

## Functions

- Closer to a map than a "function" in most programming languages.
- Can be recursively defined:

```
fact[x \in Int] == IF x <= 0 THEN 1 ELSE x * fact[x-1]
```

- Can also be directly mapped:

# TLA+ Semantics - Functions and Tuples

## Tuples

- `<<1, 2, 3>>`

## Functions

- Closer to a map than a "function" in most programming languages.
- Can be recursively defined:

```
fact[x \in Int] == IF x <= 0 THEN 1 ELSE x * fact[x-1]
```

- Can also be directly mapped:

```
[x \in {"a", "b", "c" |-> 10 ]
```

- More or less JavaScript equivalent:

# TLA+ Semantics - Functions and Tuples

## Tuples

- `<<1, 2, 3>>`

## Functions

- Closer to a map than a "function" in most programming languages.
- Can be recursively defined:

```
fact[x \in Int] == IF x <= 0 THEN 1 ELSE x * fact[x-1]
```

- Can also be directly mapped:

```
[x \in {"a", "b", "c" |-> 10 ]
```

- More or less JavaScript equivalent:

```
{
  "a": 10,
  "b": 10,
  "c": 10
}
```

# TLA+ Semantics - Universal Quantifier

# TLA+ Semantics - Universal Quantifier

- `\A`
- $\forall$

# TLA+ Semantics - Universal Quantifier

- \A
- ∀

```
\A x \in {1,2,3} : Func[x] % 2 = 0
```

# TLA+ Semantics - Existential Quantifier

# TLA+ Semantics - Existential Quantifier

- `\E`
- $\exists$

# TLA+ Semantics - Existential Quantifier

- \E
- ∃

```
MySet == {1, 2, 3}

Next == \E x \in MySet:
           /\ z' = x + 1
```

# TLA+ Semantics - Existential Quantifier

- `\E`
- $\exists$

```
MySet == {1, 2, 3}

Next == \E x \in MySet:
           /\ z' = x + 1
```

The same as:

```
Next == \/ x = 1
           /\ z' = x + 1
        \/ x = 2
           /\ z' = x + 1
        \/ x = 3
           /\ z' = x + 1
```

# TLA+ Semantics - State

# TLA+ Semantics - State

- State is assigned via assertion.

# TLA+ Semantics - State

- State is assigned via assertion.
- Next state is “assigned” with “primed” variables.
  - $x' = 2$

# TLA+ Semantics - State

- State is assigned via assertion.
- Next state is "assigned" with "primed" variables.
  - $x' = 2$
- Any valid assertion is a valid "assignment".
  - $x' \in \{\text{“}ok\text{”}, \text{“}notok\text{”}\}$

# TLA+ Semantics - State

- State is assigned via assertion.
- Next state is "assigned" with "primed" variables.
  - $x' = 2$
- Any valid assertion is a valid "assignment".
  - $x' \in \{\text{“}ok\text{”}, \text{“}notok\text{”}\}$
- All variables must either by updated or labeled as `UNCHANGED`.

# Example: Tower of Hanoi.

# Example: Tower of Hanoi.

```
EXTENDS Integers, Sequences
VARIABLES tower1, tower2, tower3

NumRings == 1..10

Init == /\ tower1 = [x \in NumRings |-> [size |-> x] ]
        /\ tower2 = <<>>
        /\ tower3 = <<>>
```

# Example: Tower of Hanoi.

# Example: Tower of Hanoi.

```
Next == \/ /\ MoveRing(tower1, tower2)
           /\ UNCHANGED <<tower3>>
        \/ /\ MoveRing(tower1, tower3)a
           /\ UNCHANGED <<tower2>>
        \/ /\ MoveRing(tower2, tower1)
           /\ UNCHANGED <<tower3>>
        \/ /\ MoveRing(tower2, tower3)
           /\ UNCHANGED <<tower1>>
        \/ /\ MoveRing(tower3, tower1)
           /\ UNCHANGED <<tower2>>
        \/ /\ MoveRing(tower3, tower2)
           /\ UNCHANGED <<tower1>>
```

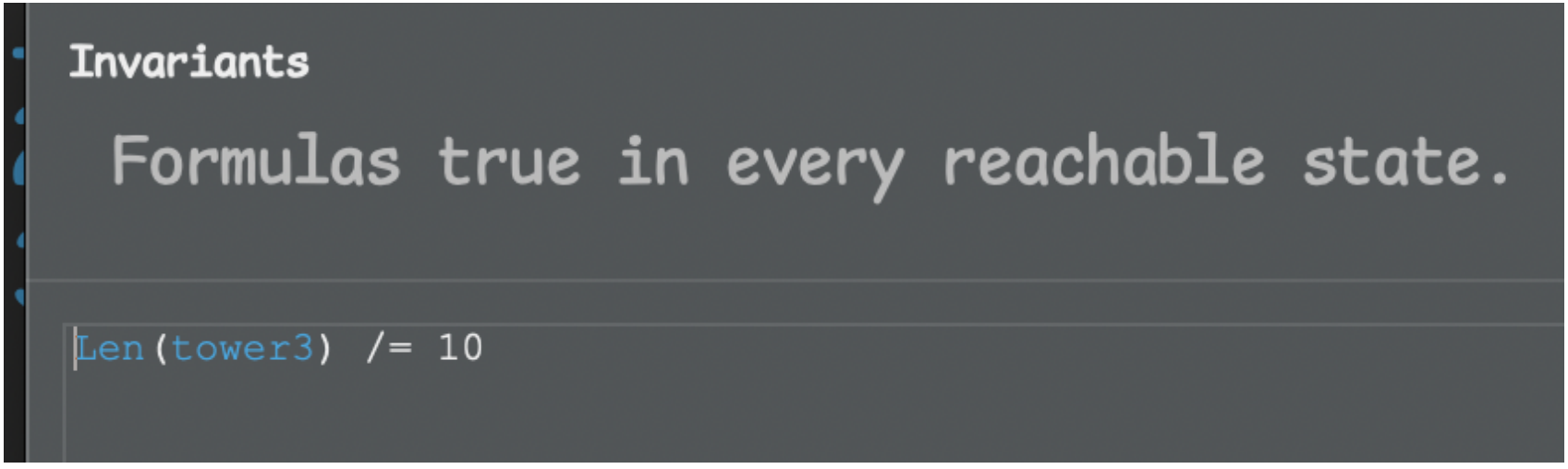
# Example: Tower of Hanoi.

# Example: Tower of Hanoi.

```
MoveRing(firstTower, secondTower) ==
 \/ /\ firstTower = <<>>
    /\ UNCHANGED <<firstTower, secondTower>>
 \/ /\ secondTower = <<>>
    /\ firstTower /= <<>>
    /\ secondTower' = <<Head(firstTower)>> \o secondTower
    /\ firstTower' = Tail(firstTower)
 \/ /\ secondTower /= <<>>
    /\ firstTower /= <<>>
    /\ Head(firstTower).size < Head(secondTower).size
    /\ firstTower' = Tail(firstTower)
    /\ secondTower' = <<Head(firstTower)>> \o secondTower
 \/ /\ secondTower /= <<>>
    /\ firstTower /= <<>>
    /\ Head(firstTower).size > Head(secondTower).size
          /\ UNCHANGED <<firstTower, secondTower>>
```

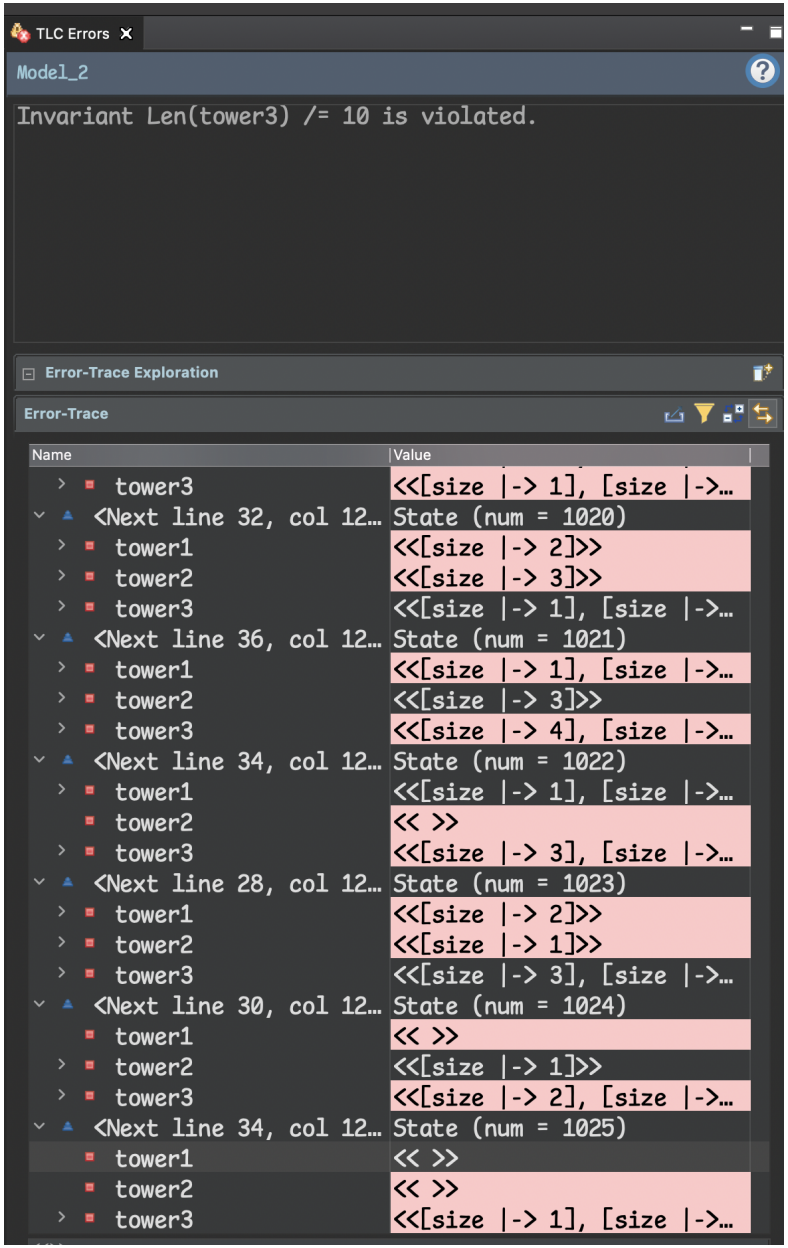
# Example: Tower of Hanoi - Checking

# Example: Tower of Hanoi - Checking



# Example: Tower of Hanoi - Checking

# Example: Tower of Hanoi - Checking



# Example: Race Condition

# Example: Race Condition

```
---- MODULE counter ----
EXTENDS Integers, Sequences

VARIABLES pc, counter, tmp
vars == <<pc, counter, tmp>>

Threads == 1..2

States == {"start", "inc", "done"}
```

# Example: Race Condition

```
Trans(thread, from, to) ==
  /\ pc[thread] = from
  /\ pc' = [pc EXCEPT ![thread] = to]

Init ==
  /\ pc = [t \in Threads |-> "start"]
  /\ counter = 0
  /\ tmp = [t \in Threads |-> 0]

GetCounter(t) ==
  /\ tmp' = [tmp EXCEPT ![t] = counter]
  /\ UNCHANGED counter

IncCounter(t) ==
  /\ counter' = tmp[t] + 1
  /\ UNCHANGED tmp
```

# Example: Race Condition

# Example: Race Condition

```
Next ==
  \/ \E t \in Threads:
    \/ /\ Trans(t, "start", "inc")
       /\ GetCounter(t)
    \/ /\ Trans(t, "inc", "done")
       /\ IncCounter(t)
```

# Example: Race Condition: Checking

# Example: Race Condition: Checking

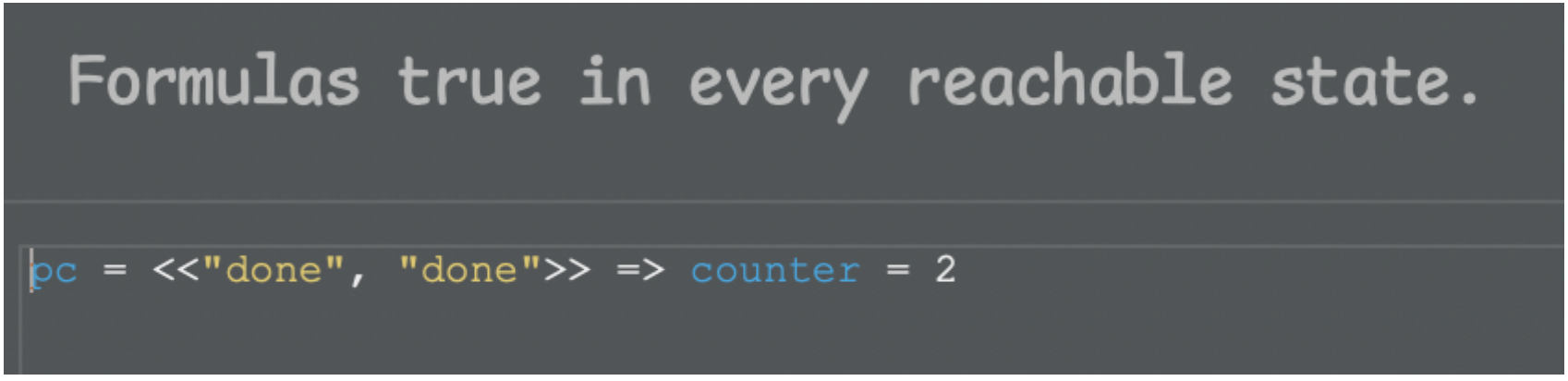
- We can set an invariant in the TLA+ Workbench.

# Example: Race Condition: Checking

- We can set an invariant in the TLA+ Workbench.
- Works fine if we set the number of `Threads` to `1`.

# Example: Race Condition: Checking

- We can set an invariant in the TLA+ Workbench.
- Works fine if we set the number of `Threads` to `1`.



# Example: Race Condition: Checking

# Example: Race Condition: Checking

| Name | Value |
| --- | --- |
| ▾ <Initial predicate> | State (num = 1) |
| counter | 0 |
| ▸ pc | <<"start", "start">> |
| ▸ tmp | <<0, 0>> |
| ▾ <Next line 31, col 8 to line 3… | State (num = 2) |
| counter | 0 |
| ▸ pc | <<"start", "inc">> |
| ▸ tmp | <<0, 0>> |
| ▾ <Next line 31, col 8 to line 3… | State (num = 3) |
| counter | 0 |
| ▸ pc | <<"inc", "inc">> |
| ▸ tmp | <<0, 0>> |
| ▾ <Next line 34, col 8 to line 3… | State (num = 4) |
| counter | 1 |
| ▸ pc | <<"done", "inc">> |
| ▸ tmp | <<0, 0>> |
| ▾ <Next line 34, col 8 to line 3… | State (num = 5) |
| counter | 1 |
| ▸ pc | <<"done", "done">> |
| ▸ tmp | <<0, 0>> |

# Example: Race Condition: Fix

# Example: Race Condition: Fix

```
EXTENDS Integers, Sequences

VARIABLES pc, counter, tmp, lock
vars == <<pc, counter, tmp, lock>>

Threads == 1..2

States == {"start", "inc", "done"}
```

# Example: Race Condition: Fix

# Example: Race Condition: Fix

```
Trans(thread, from, to) ==
  /\ pc[thread] = from
  /\ pc' = [pc EXCEPT ![thread] = to]

Init ==
  /\ pc = [t \in Threads |-> "start"]
  /\ counter = 0
  /\ tmp = <<0, 0>>
  /\ lock = 0

AcquireLock(t) ==
  /\ lock = 0
  /\ lock' = t

ReleaseLock(t) ==
  /\ lock = t
  /\ lock' = 0
```

# Example: Race Condition: Fix

# Example: Race Condition: Fix

```
GetCounter(t) ==
  /\ tmp' = [tmp EXCEPT ![t] = counter]
  /\ UNCHANGED counter

IncCounter(t) ==
  /\ counter' = tmp[t] + 1
  /\ UNCHANGED tmp
```

# Example: Race Condition: Fix

# Example: Race Condition: Fix

```
Next ==
  \/ \E t \in Threads:
    \/ /\ Trans(t, "start", "inc")
       /\ GetCounter(t)
       /\ AcquireLock(t)
    \/ /\ Trans(t, "inc", "done")
       /\ IncCounter(t)
       /\ ReleaseLock(t)
```

# More to Learn

# More to Learn

- There is a *lot* more to TLA+ if you want to dig into it.

# More to Learn

- There is a *lot* more to TLA+ if you want to dig into it.
- If any of this seems interesting to you, it's worth reading about.

# Further Reading

- Lamport's Videos
- Specifying Systems book
- TLA+ in Practice and Theory
- LearnTLA.com

# Final Words

# Final Words

- Writing correct code is very hard.

# Final Words

- Writing correct code is very hard.
- As engineers we should use every bit of tooling that we can get to make finding bugs easier.

# Final Words

- Writing correct code is very hard.
- As engineers we should use every bit of tooling that we can get to make finding bugs easier.
- Formal modeling can help you catch bugs in difficult projects *before writing any code*.

# Contact


- thomas@gebert.app
- gitlab.com/tombert