An Introduction to TLA$^+$

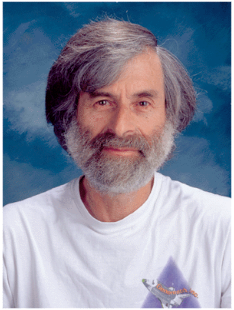
Stephan Merz

https://members.loria.fr/Stephan.Merz/

Inria Nancy – Grand Est & LORIA
Nancy, France

TLA$^+$ Community Meeting @ ETAPS
Paris, April 2023

Leslie Lamport http://www.lamport.org/



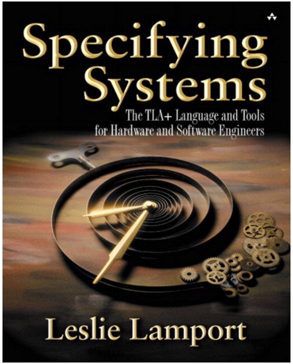
PhD 1972 (Brandeis University), Mathematics

- Mitre Corporation, 1962–65
- Marlboro College, 1965–69
- Massachusets Computer Associates, 1970–77
- SRI International, 1977–85
- Digital Equipment Corporation/Compaq, 1985–2001
- Microsoft Research, since 2001

Pioneer of distributed algorithms   Turing Award 2013

- Natl. Acad. of Engineering, Natl. Acad. of Sciences, American Acad. of Arts and Sciences
- PODC Influential Paper, ACM SIGOPS Hall of Fame (3x), J.C. Laprie Award (2x), LICS Award, John v. Neumann medal, E.W. Dijkstra Prize, NEC C&C Prize ...
- honorary doctorates: Rennes, Kiel, Lausanne, Lugano, Nancy, Brandeis

TLA$^+$ specification language



- describe and verify distributed and concurrent systems
- based on mathematical set theory plus temporal logic TLA
- TLA$^+$ Video Course
- book: Addison-Wesley, 2003 (free download for personal use)
- IDEs: TLA$^+$ Toolbox, Visual Studio Code Extension

Some other publications

- Y. Yu, P. Manolios, L. Lamport: *Model checking TLA$^+$ Specifications*. CHARME 1999, LNCS 1703.
- D. Cousineau et al.: *TLA$^+$ Proofs*. Formal Methods (FM 2012), LNCS 7436.
- I. Konnov et al.: *TLA$^+$ Model Checking Made Symbolic*. OOPSLA 2019.
- S. Merz: *The Specification Language TLA$^+$*. Logics of Specification Languages, Springer 2008.

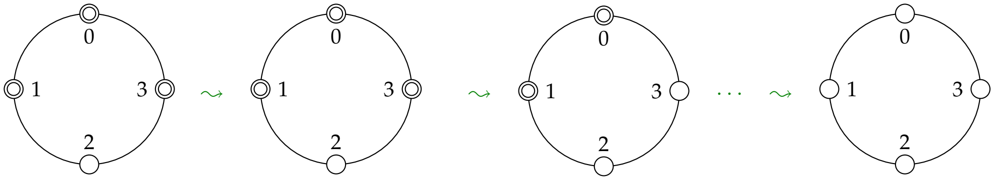
Objective of this presentation

- Explain basic concepts of TLA$^+$
- TLA$^+$ as a specification language
- Tool support for verification: model checking, proof, refinement
- Running example: distributed termination detection

Please interrupt for questions

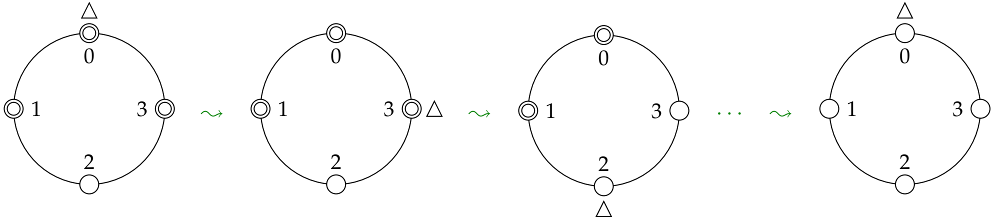
Outline

Distributed Termination Detection



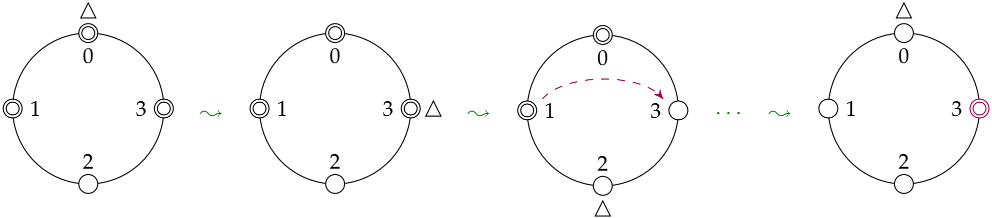
- Nodes perform some computation
  - a node can be active (double circle) or inactive (simple circle)
  - "master node" 0 wishes to detect when all nodes are inactive
- Relevant transitions
  - active node finishes its computation and terminates
  - master node detects termination

Distributed Termination Detection



- Nodes perform some computation
  - a node can be active (double circle) or inactive (simple circle)
  - "master node" 0 wishes to detect when all nodes are inactive
- Relevant transitions
  - active node finishes its computation and terminates
  - master node detects termination

Distributed Termination Detection



- Nodes perform some computation
  - a node can be active (double circle) or inactive (simple circle)
  - "master node" 0 wishes to detect when all nodes are inactive
- Relevant transitions
  - active node finishes its computation and terminates
  - master node detects termination
  - active node sends a message to some node in the network
  - node receives a message, waking up if inactive

Abstract Transition System for Describing the Problem

- State representation
  - activation status per node
  - number of pending messages
  - termination detected?

$$
\begin{aligned}
&\mathit{TypeOK} \triangleq \\
&\quad \land\ \mathit{active} \in [\mathit{Nodes} \rightarrow \text{BOOLEAN}] \\
&\quad \land\ \mathit{pending} \in [\mathit{Nodes} \rightarrow \mathit{Nat}] \\
&\quad \land\ \mathit{termDetect} \in \text{BOOLEAN} \\
&\mathit{terminated} \triangleq \forall n \in \mathit{Nodes} : \neg \mathit{active}[n] \land \mathit{pending}[n] = 0
\end{aligned}
$$

Abstract Transition System for Describing the Problem

- State representation
  - activation status per node
  - number of pending messages
  - termination detected?

$$\begin{array}{l} \mathit{TypeOK} \triangleq \\ \quad \land\ \mathit{active} \in [\mathit{Nodes} \rightarrow \text{BOOLEAN}] \\ \quad \land\ \mathit{pending} \in [\mathit{Nodes} \rightarrow \mathit{Nat}] \\ \quad \land\ \mathit{termDetect} \in \text{BOOLEAN} \\ \mathit{terminated} \triangleq \forall n \in \mathit{Nodes} : \neg \mathit{active}[n] \land \mathit{pending}[n] = 0 \end{array}$$

- Transitions
  - termination of a node
  - sending and receiving of messages
  - termination detection

$$\begin{array}{l} \mathit{RcvMsg}(i) \triangleq \\ \quad \land\ \mathit{pending}[i] > 0 \\ \quad \land\ \mathit{active}' = [\mathit{active}\ \text{EXCEPT}\ ![i] = \text{TRUE}] \\ \quad \land\ \mathit{pending}' = [\mathit{pending}\ \text{EXCEPT}\ ![i] = @ - 1] \\ \quad \land\ \text{UNCHANGED}\ \mathit{termDetect} \end{array}$$

Abstract Transition System for Describing the Problem

- State representation
  - activation status per node
  - number of pending messages
  - termination detected?

$$
\begin{array}{l}
TypeOK \triangleq \\
\quad \land\ active \in [Nodes \rightarrow \text{BOOLEAN}] \\
\quad \land\ pending \in [Nodes \rightarrow Nat] \\
\quad \land\ termDetect \in \text{BOOLEAN} \\
terminated \triangleq \forall n \in Nodes : \neg active[n] \land pending[n] = 0
\end{array}
$$

- Transitions
  - termination of a node
  - sending and receiving of messages
  - termination detection

$$
\begin{array}{l}
RcvMsg(i) \triangleq \\
\quad \land\ pending[i] > 0 \\
\quad \land\ active' = [active \text{ EXCEPT } ![i] = \text{TRUE}] \\
\quad \land\ pending' = [pending \text{ EXCEPT } ![i] = @ - 1] \\
\quad \land\ \text{UNCHANGED } termDetect
\end{array}
$$

- Overall specification

$$Spec \triangleq Init \land \Box[Next]_{vars} \land \text{WF}_{vars}(DetectTermination)$$

Outline

Expressing Correctness Properties

1. Safety properties: "nothing bad ever happens"
   - type correctness

     $Spec \Rightarrow \Box TypeOK$

     TypeOK is true throughout any execution of *Spec*

Expressing Correctness Properties

1. Safety properties: "nothing bad ever happens"

   - type correctness

     $Spec \Rightarrow \Box TypeOK$

     TypeOK is true throughout any execution of *Spec*

   - safety of detection

     $Spec \Rightarrow \Box(termDetected \Rightarrow terminated)$

     formally again expressed as an invariant

Expressing Correctness Properties

1. Safety properties: “nothing bad ever happens”

   - type correctness
     $Spec \Rightarrow \Box TypeOK$
     TypeOK is true throughout any execution of *Spec*

   - safety of detection
     $Spec \Rightarrow \Box(termDetected \Rightarrow terminated)$
     formally again expressed as an invariant

   - quiescence of the system
     $Spec \Rightarrow \Box(terminated \Rightarrow \Box terminated)$

Expressing Correctness Properties

1. Safety properties: "nothing bad ever happens"
   - type correctness $Spec \Rightarrow \Box TypeOK$

     TypeOK is true throughout any execution of *Spec*
   - safety of detection $Spec \Rightarrow \Box(termDetected \Rightarrow terminated)$

     formally again expressed as an invariant
   - quiescence of the system $Spec \Rightarrow \Box(terminated \Rightarrow \Box terminated)$
2. Liveness properties: "something good happens eventually"
   - eventual detection of termination $Spec \Rightarrow \Box(terminated \Rightarrow \Diamond termDetected)$

     note: the system isn't guaranteed to terminate

Explicit-State Model Checking Using TLC

- Create a model: finite instance of TLA$^+$ specification defined as a configuration
  - instantiate constant parameters, bound potentially infinite variable values

fix constants	$N = 4$
add state constraint	$\forall n \in Nodes : pending[n] \leq 3$

  - indicate formulas representing system specification and properties to be verified
  - TLC reports 4,097 distinct states (262,145 for $N = 6$)

Explicit-State Model Checking Using TLC

- Create a model: finite instance of TLA+ specification defined as a configuration
  - instantiate constant parameters, bound potentially infinite variable values

fix constants	$N = 4$
add state constraint	$\forall n \in Nodes : pending[n] \leq 3$

  - indicate formulas representing system specification and properties to be verified
  - TLC reports 4,097 distinct states (262,145 for $N = 6$)

- Exploit the automation of TLC for gaining confidence in the specification
  - check putative (non-)properties and make changes to specification
  - e.g., allow inactive node to send messages

Using TLAPS to Prove Safety Properties

- TLAPS: proof assistant for verifying TLA$^+$ specifications
  - proof effort is independent of the size of the instance
  - relies on user interaction to guide verification
  - uses automatic back-ends for discharging proof obligations

Using TLAPS to Prove Safety Properties

- TLAPS: proof assistant for verifying TLA$^+$ specifications
  - proof effort is independent of the size of the instance
  - relies on user interaction to guide verification
  - uses automatic back-ends for discharging proof obligations

- TLAPS proof of type correctness

THEOREM $TypeCorrect \triangleq Spec \Rightarrow \Box TypeOK$
$\langle 1 \rangle 1.\ Init \Rightarrow TypeOK$
$\langle 1 \rangle 2.\ TypeOK \land [Next]_{vars} \Rightarrow TypeOK'$
$\langle 1 \rangle 3.$ QED BY $\langle 1 \rangle 1, \langle 1 \rangle 2, PTL$ DEF $Spec$

  - hierarchical proof language represents proof tree
  - assertion follows from steps $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$ by temporal logic
  - prove non-temporal steps by expanding definitions and/or hierarchical subproofs

Proof of Main Safety Property

- Safety of termination detection is inductive relative to *TypeOK*

$Safe \triangleq termDetect \Rightarrow terminated$

THEOREM $Safety \triangleq Spec \Rightarrow \Box Safe$

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

$\langle 1 \rangle 2.\ TypeOK \land Safe \land [Next]_{vars} \Rightarrow Safe'$

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

  - use previously established theorem of type correctness
  - proofs of steps $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$ are similar as before

Proof of Main Safety Property

- Safety of termination detection is inductive relative to *TypeOK*

$Safe \triangleq termDetect \Rightarrow terminated$
THEOREM $Safety \triangleq Spec \Rightarrow \Box Safe$
$\langle 1 \rangle 1.\ Init \Rightarrow Safe$
$\langle 1 \rangle 2.\ TypeOK \land Safe \land [Next]_{vars} \Rightarrow Safe'$
$\langle 1 \rangle 3.$ QED BY $\langle 1 \rangle 1, \langle 1 \rangle 2, \langle 1 \rangle 3, TypeCorrect, PTL$ DEF $Spec$

  - use previously established theorem of type correctness
  - proofs of steps $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$ are similar as before

- Proof of quiescence is similar
  - proofs of safety properties require minimal temporal logic
  - automation of TLA$^+$ set theory is main concern

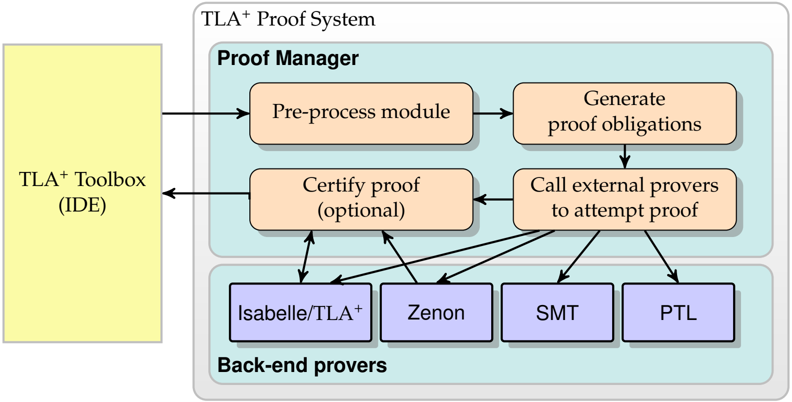
Proof of Main Safety Property

- Safety of termination detection is inductive relative to *TypeOK*

$Safe \triangleq termDetect \Rightarrow terminated$
THEOREM $Safety \triangleq Spec \Rightarrow \Box Safe$
$\langle 1 \rangle 1.\ Init \Rightarrow Safe$
$\langle 1 \rangle 2.\ TypeOK \land Safe \land [Next]_{vars} \Rightarrow Safe'$
$\langle 1 \rangle 3.$ QED BY $\langle 1 \rangle 1, \langle 1 \rangle 2, \langle 1 \rangle 3, TypeCorrect, PTL$ DEF $Spec$

  - use previously established theorem of type correctness
  - proofs of steps $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$ are similar as before
- Proof of quiescence is similar
  - proofs of safety properties require minimal temporal logic
  - automation of TLA^+ set theory is main concern
- Liveness proofs require establishing enabledness predicate
  - supported in development version of TLAPS

TLAPS Architecture

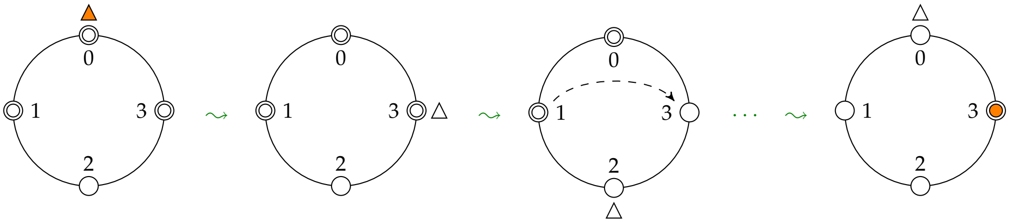


- Isabelle/TLA$^+$: faithful encoding of TLA$^+$ in Isabelle's meta-logic
- PTL: decision procedure for propositional temporal logic

Outline

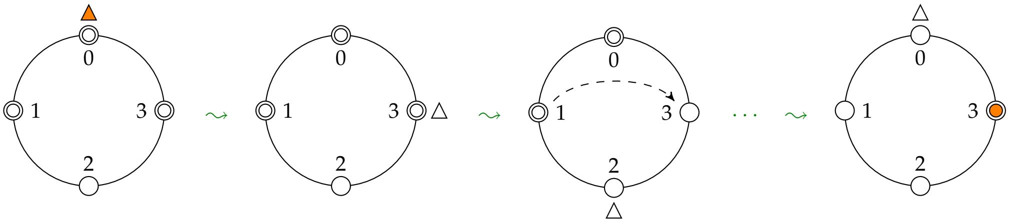
Overall Idea of Safra's algorithm (EWD 998, 1986)

- Token circulating on the ring



Overall Idea of Safra's algorithm (EWD 998, 1986)

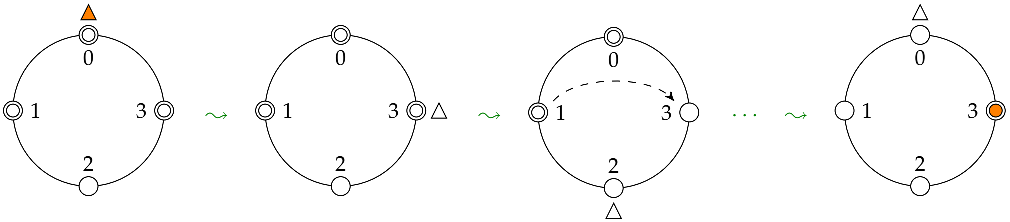
- Token circulating on the ring



  - nodes remember difference between numbers of messages sent and received
  - token accumulates sum of differences
  - receiving node becomes "stained", passing token collects "stain"

Overall Idea of Safra's algorithm (EWD 998, 1986)

- Token circulating on the ring



  - nodes remember difference between numbers of messages sent and received
  - token accumulates sum of differences
  - receiving node becomes "stained", passing token collects "stain"

- Condition for detecting termination
  - sum of counters at master node and token is zero
  - master node is inactive and clean, and it holds a clean token

Analyzing the Algorithm Using TLC

- Similar correctness properties as for the abstract state machine
  - type correctness, safety, quiescence, liveness
- Explicit-state model checking with TLC

# nodes	small bounds		modest bounds	
	# states	time	# states	time
3	0.23 M	0:00:09	1.5 M	0:00:21
4	18.7 M	0:03:54	248 M	0:33:53
5	1150 M	2:05:00	–	–

bounds on counters:

- small: all counters ≤ 2
- modest: nodes ≤ 3, token ≤ 9

used 32 cores for 5 nodes

Analyzing the Algorithm Using TLC

- Similar correctness properties as for the abstract state machine
  - type correctness, safety, quiescence, liveness
- Explicit-state model checking with TLC

# nodes	small bounds		modest bounds	
	# states	time	# states	time
3	0.23 M	0:00:09	1.5 M	0:00:21
4	18.7 M	0:03:54	248 M	0:33:53
5	1150 M	2:05:00	–	–

bounds on counters:

- small: all counters ≤ 2
- modest: nodes ≤ 3, token ≤ 9

used 32 cores for 5 nodes

- Does this give you enough confidence?
  - model checking suffers from state space explosion
  - one modification was incorrect for $N = 4$, but correct for $N = 3$
  - TLC supports random exploration, finds seeded bugs in majority of runs

An Inductive Invariant for Safra's Algorithm

- Inductive invariant adapted from Dijkstra

$$
\begin{array}{lcl}
Sum(f,S) & \triangleq & FoldFunctionOnSet(+,0,f,S) \\
Inv & \triangleq & \land\ Sum(pending, Node) = Sum(counter, Node) \\
 & & \land \lor \land\ \forall i \in token.pos + 1 \,..\, N - 1 : active[i] = \text{FALSE} \\
 & & \qquad \land\ token.q = Sum(counter, (token.pos + 1) \,..\, (N - 1)) \\
 & & \quad \lor\ Sum(counter, 0 \,..\, token.pos) + token.q > 0 \\
 & & \quad \lor\ \exists i \in 0 \,..\, token.pos : color[i] = \text{"orange"} \\
 & & \quad \lor\ token.color = \text{"orange"}
\end{array}
$$

An Inductive Invariant for Safra's Algorithm

- Inductive invariant adapted from Dijkstra

$$
\begin{array}{lcl}
Sum(f, S) & \triangleq & FoldFunctionOnSet(+, 0, f, S) \\
Inv & \triangleq & \land\ Sum(pending, Node) = Sum(counter, Node) \\
 & & \land\ \lor\ \land\ \forall i \in token.pos + 1 \,..\, N - 1 : active[i] = \text{FALSE} \\
 & & \qquad\ \ \land\ token.q = Sum(counter, (token.pos + 1) \,..\, (N - 1)) \\
 & & \quad\ \lor\ Sum(counter, 0 \,..\, token.pos) + token.q > 0 \\
 & & \quad\ \lor\ \exists i \in 0 \,..\, token.pos : color[i] = \text{"orange"} \\
 & & \quad\ \lor\ token.color = \text{"orange"}
\end{array}
$$

- Verification using TLAPS
  - first prove type correctness invariant
  - prove $Spec \Rightarrow \Box Inv$, based on type correctness
  - also prove that *Inv* implies main safety property
  - proofs require auxiliary facts about *Sum*

Correctness by Refinement

- Specifications and properties are TLA$^+$ formulas
  - THEOREM $Spec \Rightarrow Prop$ — every run of *Spec* satisfies property *Prop*

Correctness by Refinement

- Specifications and properties are TLA$^+$ formulas
  - THEOREM $Spec \Rightarrow Prop$ — every run of *Spec* satisfies property *Prop*
  - THEOREM $Impl \Rightarrow Spec$ — every run of *Impl* corresponds to a run of *Spec*

Correctness by Refinement

- Specifications and properties are TLA$^+$ formulas
  - THEOREM $\mathit{Spec} \Rightarrow \mathit{Prop}$ — every run of *Spec* satisfies property *Prop*
  - THEOREM $\mathit{Impl} \Rightarrow \mathit{Spec}$ — every run of *Impl* corresponds to a run of *Spec*
  - stuttering invariance of TLA$^+$ formulas is important here: allow for low-level steps in *Impl* that are invisible to *Spec*

Correctness by Refinement

- Specifications and properties are TLA$^+$ formulas
  - THEOREM $Spec \Rightarrow Prop$ — every run of *Spec* satisfies property *Prop*
  - THEOREM $Impl \Rightarrow Spec$ — every run of *Impl* corresponds to a run of *Spec*
  - stuttering invariance of TLA$^+$ formulas is important here: allow for low-level steps in *Impl* that are invisible to *Spec*
- Use existing tools for verifying refinement

$TD \triangleq$ INSTANCE *TerminationDetection*   THEOREM $Spec \Rightarrow TD!Spec$

Correctness by Refinement

- Specifications and properties are TLA$^+$ formulas
  - THEOREM $Spec \Rightarrow Prop$ — every run of *Spec* satisfies property *Prop*
  - THEOREM $Impl \Rightarrow Spec$ — every run of *Impl* corresponds to a run of *Spec*
  - stuttering invariance of TLA$^+$ formulas is important here: allow for low-level steps in *Impl* that are invisible to *Spec*
- Use existing tools for verifying refinement

$TD \triangleq$ INSTANCE *TerminationDetection* THEOREM $Spec \Rightarrow TD!Spec$

  - TLC verifies refinement just like it checks correctness properties
  - refinement proof checked by TLAPS, based on previous inductive invariant

Outline

Summing Up

- TLA^+: mathematical language for specifying systems
  - highly expressive and flexible language encourages abstract descriptions
  - state machine specifications represent system behavior
  - no distinction between systems and properties
  - refinement (and composition) reflected in logic
- Tool support
  - IDEs: TLA^+ Toolbox / VS Code Extension
  - TLC: push-button verification, support for random exploration
  - Apalache: bounded symbolic model checking (see separate tutorial)
  - TLAPS: interactive proof platform, automatic proof back-ends
  - PlusCal: front-end for generating TLA^+ from "pseudo code" language
- More information

  `http://lamport.azurewebsites.net/tla/tla.html` Google discussion group