# Tool Support for TLA$^+$: TLC, Apalache, and TLAPS

Stephan Merz

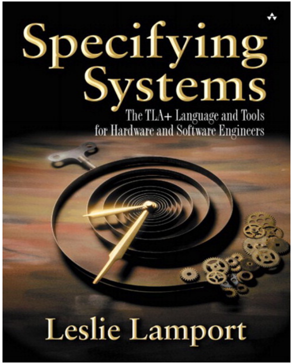
(joint work with Igor Konnov and Markus Kuppe)

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

Inria Nancy – Grand Est & LORIA
Nancy, France

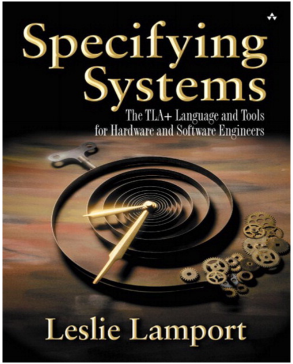
IFIP WG 2.2 meeting 2022, Münster, Germany

# TLA$^+$ specification language




- describe and verify distributed and concurrent systems
- based on mathematical set theory and temporal logic TLA
- TLA$^+$ Video Course
- documentation available from TLA$^+$ home page

# TLA$^+$ specification language



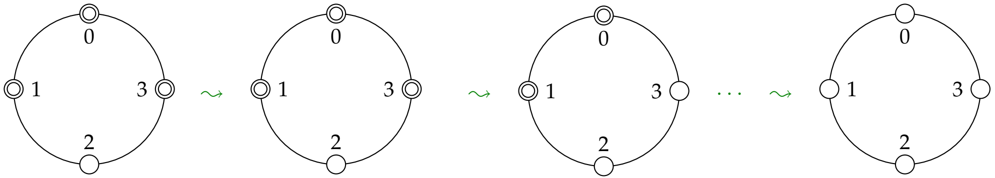

- describe and verify distributed and concurrent systems
- based on mathematical set theory and temporal logic TLA
- TLA$^+$ Video Course
- documentation available from TLA$^+$ home page
- support tools
  - TLC — explicit-state model checking
  - Apalache — bounded (symbolic) model checking
  - TLAPS — interactive proof assistant
  - PlusCal — algorithmic language, front-end translator
  - IDEs — TLA$^+$ Toolbox, VS Code extension

# Objective of this presentation

- Present three main verification tools for TLA$^+$
  - verify (safety and liveness) properties of specifications
  - check that a specification refines another one
- Expose complementary strengths and weaknesses
  - push-button verification vs. human interaction
  - coverage and confidence provided
- Suggest a workflow for analyzing TLA$^+$ specifications
- Presentation by example: distributed termination detection

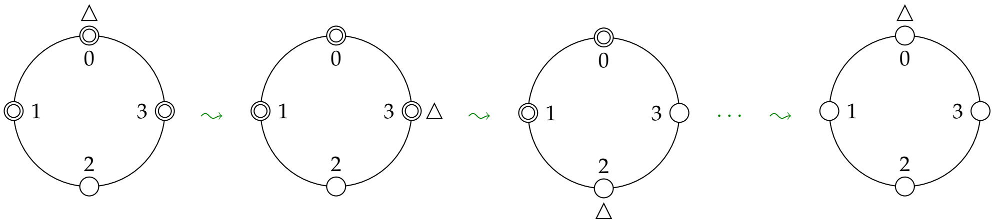
# Outline

# Distributed Termination Detection



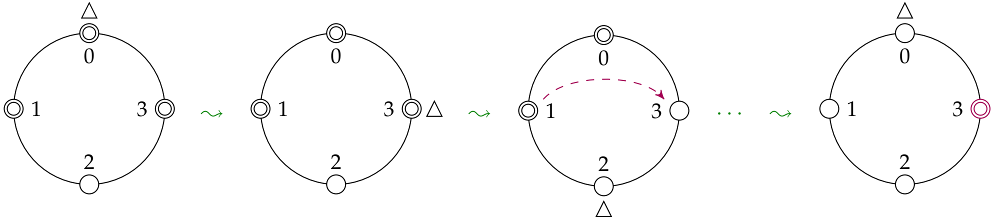

- Nodes perform some computation
  - a node can be active (double circle) or inactive (simple circle)
  - “master node” 0 wishes to detect when all nodes are inactive
- Relevant transitions
  - active node finishes its computation and terminates
  - master node announces termination

# 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 announces 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 announces termination
  - active node sends a message to some node in the network
  - node receives a message, waking up if inactive

# TLA$^+$ Specification: State Representation

MODULE *TerminationDetection*

EXTENDS *Naturals*
CONSTANT $N$
ASSUME $NAssumption \triangleq N \in Nat \setminus \{0\}$
$Nodes \triangleq 0 \,..\, N-1$
VARIABLES $active, pending, termDetect$
$TypeOK \triangleq active \in [Nodes \rightarrow \text{BOOLEAN}] \land pending \in [Nodes \rightarrow Nat] \land termDetect \in \text{BOOLEAN}$
$vars \triangleq \langle active, pending, termDetect \rangle$
$terminated \triangleq \forall n \in Node : \neg active[n] \land pending[n] = 0$

- Declaration of constants and variables
- Definition of operators
  - *TypeOK* documents expected values of variables: *active* and *color* are arrays (functions)
  - *terminated* describes configurations in which the systems is globally inactive

# TLA$^+$ Specification: Abstract State Machine

$$
\begin{aligned}
\mathit{Init} \triangleq\ & \land\ \mathit{active} \in [\mathit{Nodes} \rightarrow \text{BOOLEAN}] \\
& \land\ \mathit{pending} = [n \in \mathit{Nodes} \mapsto 0] \\
& \land\ \mathit{termDetect} \in \{\text{FALSE}, \mathit{terminated}\}
\end{aligned}
$$

- initial condition: arbitrary activation status, no pending messages

# TLA$^+$ Specification: Abstract State Machine

$$
\begin{aligned}
Init \triangleq\ & \land\ active \in [Nodes \rightarrow \text{BOOLEAN}] \\
& \land\ pending = [n \in Nodes \mapsto 0] \\
& \land\ termDetect \in \{\text{FALSE}, terminated\}
\end{aligned}
$$

$$
\begin{aligned}
& Terminate(i) \triangleq \\
& \quad \land\ active[i] \\
& \quad \land\ active' = [active\ \text{EXCEPT}\ ![i] = \text{FALSE}] \\
& \quad \land\ pending' = pending \\
& \quad \land\ termDetect' \in \{termDetect, terminated\}
\end{aligned}
$$

$$
\begin{aligned}
& DetectTermination \triangleq \\
& \quad \land\ terminated \\
& \quad \land\ termDetect' = \text{TRUE} \\
& \quad \land\ \text{UNCHANGED}\ \langle active, pending \rangle
\end{aligned}
$$

- initial condition: arbitrary activation status, no pending messages
- state transitions: local termination, termination detection,

# TLA$^+$ Specification: Abstract State Machine

$$
\begin{array}{l}
Init \triangleq \land\ active \in [Nodes \rightarrow \text{BOOLEAN}] \\
\qquad\quad \land\ pending = [n \in Nodes \mapsto 0] \\
\qquad\quad \land\ termDetect \in \{\text{FALSE}, terminated\}
\end{array}
$$

$$
\begin{array}{l}
Terminate(i) \triangleq \\
\quad \land\ active[i] \\
\quad \land\ active' = [active \text{ EXCEPT } ![i] = \text{FALSE}] \\
\quad \land\ pending' = pending \\
\quad \land\ termDetect' \in \{termDetect, terminated\}
\end{array}
$$

$$
\begin{array}{l}
DetectTermination \triangleq \\
\quad \land\ terminated \\
\quad \land\ termDetect' = \text{TRUE} \\
\quad \land\ \text{UNCHANGED } \langle active, pending \rangle
\end{array}
$$

$$
\begin{array}{l}
SendMsg(i,j) \triangleq \\
\quad \land\ active[i] \\
\quad \land\ pending' = [pending \text{ EXCEPT } ![j] = @ + 1] \\
\quad \land\ \text{UNCHANGED } \langle active, termDetect \rangle
\end{array}
$$

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

- initial condition: arbitrary activation status, no pending messages
- state transitions: local termination, termination detection, sending/receiving of messages

# TLA$^+$ Specification: Abstract State Machine

$$
\begin{array}{l}
Init \triangleq \land\ active \in [Nodes \to \text{BOOLEAN}] \\
\qquad\quad \land\ pending = [n \in Nodes \mapsto 0] \\
\qquad\quad \land\ termDetect \in \{\text{FALSE}, terminated\}
\end{array}
$$

$$
\begin{array}{l}
Terminate(i) \triangleq \\
\quad \land\ active[i] \\
\quad \land\ active' = [active \text{ EXCEPT } ![i] = \text{FALSE}] \\
\quad \land\ pending' = pending \\
\quad \land\ termDetect' \in \{termDetect, terminated\}
\end{array}
$$

$$
\begin{array}{l}
DetectTermination \triangleq \\
\quad \land\ terminated \\
\quad \land\ termDetect' = \text{TRUE} \\
\quad \land\ \text{UNCHANGED } \langle active, pending \rangle
\end{array}
$$

$$
\begin{array}{l}
SendMsg(i,j) \triangleq \\
\quad \land\ active[i] \\
\quad \land\ pending' = [pending \text{ EXCEPT } ![j] = @ + 1] \\
\quad \land\ \text{UNCHANGED } \langle active, termDetect \rangle
\end{array}
$$

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

$$
\begin{array}{l}
Next \triangleq \lor\ \exists i \in Node : Terminate(i) \lor RcvMsg(i) \\
\qquad\quad \lor\ \exists i,j \in Node : SendMsg(i,j) \\
\qquad\quad \lor\ DetectTermination
\end{array}
$$

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

- initial condition: arbitrary activation status, no pending messages
- state transitions: local termination, termination detection, sending/receiving of messages

# Outline

# Expressing Correctness Properties

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

     *TypeOK* is true throughout any execution of *Spec*

# Expressing Correctness Properties

1. Safety properties: "nothing bad ever happens"

   - type correctness $\quad Spec \Rightarrow \Box TypeOK$

     *TypeOK* is true throughout any execution of *Spec*

   - safety of detection $\quad Spec \Rightarrow \Box(termDetect \Rightarrow terminated)$

     formally again expressed as an invariant

# Expressing Correctness Properties

1. Safety properties: “nothing bad ever happens”

   - type correctness $\quad Spec \Rightarrow \Box TypeOK$

     *TypeOK* is true throughout any execution of *Spec*

   - safety of detection $\quad Spec \Rightarrow \Box(termDetect \Rightarrow terminated)$

     formally again expressed as an invariant

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

# 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 $Spec \Rightarrow \Box(termDetect \Rightarrow terminated)$

     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 $Spec \Rightarrow \Box(terminated \Rightarrow \Diamond termDetect)$

     note: the system isn't guaranteed to terminate

# Explicit-State Model Checking Using TLC

- Create a model: finite instance of a TLA+ specification
  - instantiate constant parameters and bound potentially large variable values

    for example, create instance for $N = 4$

    add state constraint $\forall n \in Nodes : pending[n] \leq 3$
  - indicate operator corresponding to system specification and properties to verify
  - TLC reports 4,097 distinct states (262,145 for $N = 6$)
- TLC integrated into TLA+ Toolbox and Visual Studio Code Extension

# Explicit-State Model Checking Using TLC

- Create a model: finite instance of a TLA$^+$ specification
  - instantiate constant parameters and bound potentially large variable values

    for example, create instance for $N = 4$

    add state constraint $\forall n \in Nodes : pending[n] \leq 3$
  - indicate operator corresponding to system specification and properties to verify
  - TLC reports 4,097 distinct states (262,145 for $N = 6$)
- TLC integrated into TLA$^+$ Toolbox and Visual Studio Code Extension
- Exploit the automation of TLC for gaining confidence in the specification
  - check putative (non-)properties and make changes to specification
  - e.g., remove guard $active[i]$ from definition of $SendMsg(i, j)$

# Bounded Model Checking Using Apalache

- Apalache: symbolic (SMT-based) model checker
  - check safety properties for finite executions of $k$ transitions
  - relies on constraint solving rather than state enumeration
  - requires type annotations for constant and variable parameters
  - must fix $N$, no bound on the number of pending messages

```
CONSTANT
  \* @type: Int;
  N
VARIABLES
  \* @type: Int -> Bool;
  active,
  \* @type: Int -> Int;
  pending,
  \* @type: Bool;
  termDetect
```

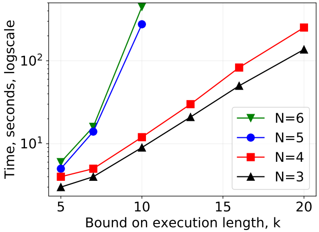
# Bounded Model Checking Using Apalache

CONSTANT
  * @type: Int;
  *N*
VARIABLES
  * @type: Int → Bool;
  *active*,
  * @type: Int → Int;
  *pending*,
  * @type: Bool;
  *termDetect*

- Apalache: symbolic (SMT-based) model checker
  - check safety properties for finite executions of $k$ transitions
  - relies on constraint solving rather than state enumeration
  - requires type annotations for constant and variable parameters
  - must fix $N$, no bound on the number of pending messages

- Performance when increasing $N$ and $k$

  checking both invariants:
  - type correctness
  - safety of termination detection




- Apalache is particularly sensitive to the number of transitions

# Apalache for Checking Inductive Invariants

- $TypeOK \land (termDetect \Rightarrow terminated)$ is an inductive invariant
  - implied by the initial condition
  - preserved by every step allowed by the transition relation

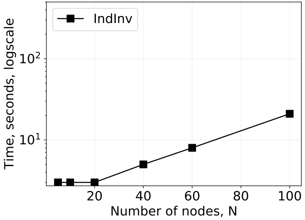
# Apalache for Checking Inductive Invariants

- $TypeOK \land (termDetect \Rightarrow terminated)$ is an inductive invariant
  - implied by the initial condition
  - preserved by every step allowed by the transition relation
- Apalache is well suited for verifying inductive invariants
  - check $Init \Rightarrow IndInv$ and $IndInv \land [Next]_{vars} \Rightarrow IndInv'$ through Apalache queries for executions of length 0 and 1
  - verify quiescence property by checking $IndInv \land [Next]_{vars} \Rightarrow (terminated \Rightarrow terminated')$

# Apalache for Checking Inductive Invariants

- $\mathit{TypeOK} \land (\mathit{termDetect} \Rightarrow \mathit{terminated})$ is an inductive invariant
  - implied by the initial condition
  - preserved by every step allowed by the transition relation

- Apalache is well suited for verifying inductive invariants
  - check $\mathit{Init} \Rightarrow \mathit{IndInv}$ and $\mathit{IndInv} \land [\mathit{Next}]_{\mathit{vars}} \Rightarrow \mathit{IndInv}'$ through Apalache queries for executions of length 0 and 1
  - verify quiescence property by checking $\mathit{IndInv} \land [\mathit{Next}]_{\mathit{vars}} \Rightarrow (\mathit{terminated} \Rightarrow \mathit{terminated}')$



# Using TLAPS to Prove Correctness 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-end provers for discharging proof obligations

# Using TLAPS to Prove Correctness 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-end provers for discharging proof obligations

- TLAPS proof of type correctness

$$\textsc{theorem}\ \mathit{TypeCorrect} \triangleq \mathit{Spec} \Rightarrow \Box \mathit{TypeOK}$$
$$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{TypeOK}$$
$$\langle 1 \rangle 2.\ \mathit{TypeOK} \land [\mathit{Next}]_{\mathit{vars}} \Rightarrow \mathit{TypeOK}'$$
$$\langle 1 \rangle 3.\ \textsc{qed} \quad \textsc{by}\ \langle 1 \rangle 1, \langle 1 \rangle 2, \mathit{PTL}\ \textsc{def}\ \mathit{Spec}$$

  - hierarchical proof language represents proof tree
  - steps can be proved in any order: usually start with QED step
  - invariant follows from steps $\langle 1 \rangle 1$ and $\langle 1 \rangle 2$ by temporal logic

# Proving non-temporal facts

- Brute force: cite relevant facts, expand definitions

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{TypeOK}$
  BY $\mathit{NAssumption}$ DEFS $\mathit{Init}, \mathit{TypeOK}, \mathit{Node}, \mathit{terminated}$

# Proving non-temporal facts

- Brute force: cite relevant facts, expand definitions

$\langle 1 \rangle 1.\ \mathit{Init} \Rightarrow \mathit{TypeOK}$
  BY $\mathit{NAssumption}$ DEFS $\mathit{Init}, \mathit{TypeOK}, \mathit{Node}, \mathit{terminated}$

- Hierarchical proofs when brute force fails

$\langle 1 \rangle 2.\ \mathit{TypeOK} \land [\mathit{Next}]_{vars} \Rightarrow \mathit{TypeOK}'$
  $\langle 2 \rangle$ SUFFICES ASSUME $\mathit{TypeOK}, [\mathit{Next}]_{vars}$ PROVE $\mathit{TypeOK}'$
    OBVIOUS
  $\langle 2 \rangle$ USE $\mathit{NAssumption}$ DEF $\mathit{Node},\ \mathit{TypeOK}$
  $\langle 2 \rangle 1.$ CASE $\mathit{DetectTermination}$
    BY $\langle 2 \rangle 1$ DEF $\mathit{DetectTermination}$
  $\langle 2 \rangle 2.$ ASSUME NEW $i \in \mathit{Node},\ \mathit{Terminate}(i)$ PROVE $\mathit{TypeOK}'$
    BY $\langle 2 \rangle 2$ DEF $\mathit{Terminate},\ \mathit{terminated}$
  ... similarly for the remaining actions ...
  $\langle 2 \rangle$ QED BY $\langle 2 \rangle 1, \langle 2 \rangle 2, \ldots$ DEF $\mathit{Next}$

# Proving non-temporal facts

- Brute force: cite relevant facts, expand definitions

$\langle 1 \rangle 1.\ Init \Rightarrow TypeOK$
BY $NAssumption$ DEFS $Init, TypeOK, Node, terminated$

- Hierarchical proofs when brute force fails

$\langle 1 \rangle 2.\ TypeOK \land [Next]_{vars} \Rightarrow TypeOK'$
$\langle 2 \rangle$ SUFFICES ASSUME $TypeOK, [Next]_{vars}$ PROVE $TypeOK'$
OBVIOUS
$\langle 2 \rangle$ USE $NAssumption$ DEF $Node,\ TypeOK$
$\langle 2 \rangle 1.$ CASE $DetectTermination$
BY $\langle 2 \rangle 1$ DEF $DetectTermination$
$\langle 2 \rangle 2.$ ASSUME NEW $i \in Node,\ Terminate(i)$ PROVE $TypeOK'$
BY $\langle 2 \rangle 2$ DEF $Terminate,\ terminated$
... similarly for the remaining actions ...
$\langle 2 \rangle$ QED BY $\langle 2 \rangle 1, \langle 2 \rangle 2, \ldots$ DEF $Next$

Toolbox IDE assists with decomposition

# Proofs of Remaining Safety Properties

$$Safe \triangleq termDetect \Rightarrow terminated$$

- Apalache suggested that *Safe* is inductive relative to *TypeOK*

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

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

# Proofs of Remaining Safety Properties

$$Safe \triangleq termDetect \Rightarrow terminated$$

- Apalache suggested that *Safe* is inductive relative to *TypeOK*

  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 $\langle 1 \rangle 1$, $\langle 1 \rangle 2$, *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 essentially no temporal logic
  - automation of TLA$^+$ set theory is the main concern

# Liveness Proof

- Liveness properties require fairness hypotheses
  - reasoning about fairness requires establishing enabledness of action

$$\text{LEMMA } \mathit{EnabledDT} \triangleq \text{ASSUME } \mathit{TypeOK}$$
$$\text{PROVE } (\text{ENABLED } \langle \mathit{DetectTermination} \rangle_{\mathit{vars}}) \equiv \mathit{terminated} \land \neg \mathit{termDetect}$$

  - TLAPS provides specific backends for reasoning about ENABLED

# Liveness Proof

- Liveness properties require fairness hypotheses
  - reasoning about fairness requires establishing enabledness of action

$$\text{LEMMA } \mathit{EnabledDT} \triangleq \text{ ASSUME } \mathit{TypeOK} \\ \text{PROVE } (\text{ENABLED } \langle \mathit{DetectTermination} \rangle_{vars}) \equiv \mathit{terminated} \land \neg \mathit{termDetect}$$

  - TLAPS provides specific backends for reasoning about ENABLED

- Now prove liveness theorem

$$\text{THEOREM } \mathit{Liveness} \triangleq \mathit{Spec} \Rightarrow \Box(\mathit{terminated} \Rightarrow \Diamond \mathit{termDetect})$$

$\langle 1 \rangle$ DEFINE $P \triangleq \mathit{terminated} \land \neg \mathit{termDetect}$

$\langle 1 \rangle 1.$ $\mathit{TypeOK} \land P \land [\mathit{Next}]_{vars} \Rightarrow P' \lor \mathit{termDetect}'$

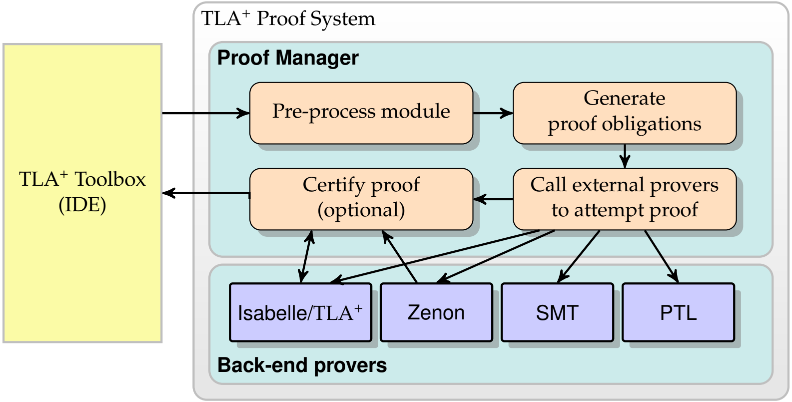
$\langle 1 \rangle 2.$ $\mathit{TypeOK} \land P \land \langle \mathit{DetectTermination} \rangle_{vars} \Rightarrow \mathit{termDetect}'$

$\langle 1 \rangle 3.$ $\mathit{TypeOK} \land P \Rightarrow \text{ENABLED } \langle \mathit{DetectTermination} \rangle_{vars}$

$\langle 1 \rangle 4.$ QED BY $\langle 1 \rangle 1, \langle 1 \rangle 2, \langle 1 \rangle 3, \mathit{TypeCorrect}, \mathit{PTL}$ DEF $\mathit{Spec}$

  - again handled by action-level reasoning and propositional temporal logic

# TLAPS Architecture

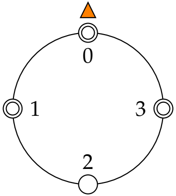



- Isabelle/TLA$^+$: faithful encoding of TLA$^+$ in Isabelle's meta-logic
- PTL: decision procedure for propositional temporal logic

# Outline

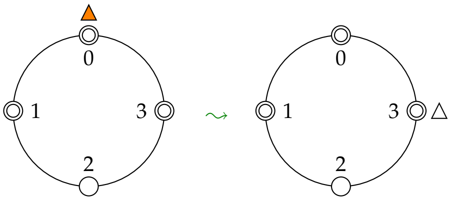
# Overall Idea of Safra's algorithm (EWD 998, 1986)

- Token circulating on the ring



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

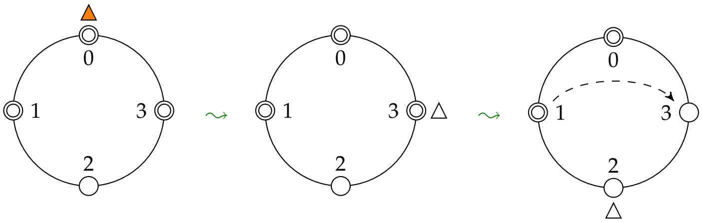
- Token circulating on the ring




  - 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)

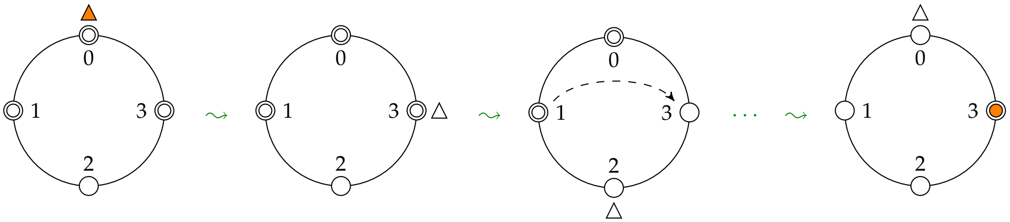
- Token circulating on the ring




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

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

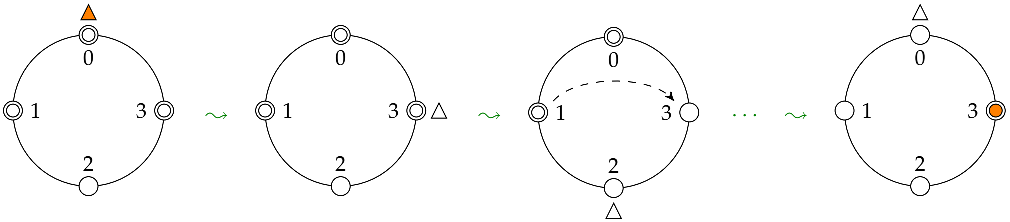
- Token circulating on the ring




  - 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

# Verification in TLA$^+$ (1)

- Similar correctness properties as for the abstract state machine
  - type correctness, safety, liveness, quiescence

# Verification in TLA+ (1)

- Similar correctness properties as for the abstract state machine
  - type correctness, safety, liveness, quiescence
- Explicit model checking using TLC
  - fix number of nodes, assume bounds on counter values

| values of bounds | # states | time |
|---|---|---|
| 3 nodes, node counters $\leq 3$ | 1.3 million | 42 sec |
| 4 nodes, node counters $\leq 3$ | 219 million | 50 min |

# Verification in TLA+ (1)

- Similar correctness properties as for the abstract state machine
  - type correctness, safety, liveness, quiescence
- Explicit model checking using TLC
  - fix number of nodes, assume bounds on counter values

| values of bounds | # states | time |
|---|---|---|
| 3 nodes, node counters $\leq 3$ | 1.3 million | 42 sec |
| 4 nodes, node counters $\leq 3$ | 219 million | 50 min |

- Is this enough for gaining confidence?
  - model checking for 5 or 6 nodes looks infeasible
  - experiment: error found for $N = 4$, but not $N = 3$
  - TLC supports random exploration, finds seeded bugs in majority of runs

# Verification in TLA$^+$ (2)

- Checking inductive invariants
  - type correctness
  - invariant provided by Dijkstra (EWD 998), inductive relative to type correctness

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

# Verification in TLA$^+$ (2)

- Checking inductive invariants
  - type correctness
  - invariant provided by Dijkstra (EWD 998), inductive relative to type correctness

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

- Verification with TLA$^+$ tools
  - Apalache confirms that $TypeOK \land Inv$ is inductive
  - TLAPS proves $Spec \Rightarrow \Box Inv$ for arbitrary $N$, modulo lemmas on $Sum$

# Correctness by Refinement

- Specifications and properties are $\text{TLA}^+$ formulas
  - THEOREM $\mathit{Spec} \Rightarrow \mathit{Prop}$ 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 $Spec \Rightarrow Prop$ — 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

# 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
- 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
- Use existing tools for verifying refinement

$$TD \triangleq \text{INSTANCE } \mathit{TerminationDetection} \qquad \text{THEOREM } Spec \Rightarrow TD!Spec$$

  - TLC checks refinement relation just as it verifies correctness properties
  - Apalache verifies safety part of refinement by checking implications

$$Init \Rightarrow TD!Init \qquad TypeOK \land Inv \land [Next]_{vars} \Rightarrow [TD!Next]_{TD!vars}$$

# Proving Refinement Using TLAPS

- Safety part of refinement
  - rely on previous proofs of type correctness and inductive invariant
  - proving initialization and step simulation is then straightforward

# Proving Refinement Using TLAPS

- Safety part of refinement
  - rely on previous proofs of type correctness and inductive invariant
  - proving initialization and step simulation is then straightforward

- Proof of liveness: set up proof by contradiction

$$BSpec \triangleq \Box TypeOK \land \Box Inv \land \Box\neg termDetect \land \Box[Next]_{vars} \land \mathrm{WF}_{vars}(System)$$

  - 3 rounds of the token on the ring may be necessary
  - (i) bring the token back to node 0, (ii) all nodes are white, (iii) token is also white
  - prove corresponding lemmas, e.g. $BSpec \Rightarrow (terminated \leadsto (terminated \land token.p = 0))$
  - conclude that action *TD.DetectTermination* cannot be always enabled
  - effort: 245 lines of proof, less than one person-day

# Outline

# Summing Up

- Complementary strengths and weaknesses of TLA$^+$ tools
  - TLC essentially push-button, random exploration finds trivial bugs
  - Apalache: bounded model checking, particularly for verifying inductive invariants
  - TLAPS: highest confidence for proving properties of arbitrary instances

# Summing Up

- Complementary strengths and weaknesses of TLA$^+$ tools
  - TLC essentially push-button, random exploration finds trivial bugs
  - Apalache: bounded model checking, particularly for verifying inductive invariants
  - TLAPS: highest confidence for proving properties of arbitrary instances
- Tools share the same input language, modulo restrictions
  - TLC and Apalache require finite models, Apalache doesn't handle general recursion
  - TLAPS does not yet support recursive operators and relies on theorem libraries

# Summing Up

- Complementary strengths and weaknesses of TLA+ tools
  - TLC essentially push-button, random exploration finds trivial bugs
  - Apalache: bounded model checking, particularly for verifying inductive invariants
  - TLAPS: highest confidence for proving properties of arbitrary instances
- Tools share the same input language, modulo restrictions
  - TLC and Apalache require finite models, Apalache doesn't handle general recursion
  - TLAPS does not yet support recursive operators and relies on theorem libraries
- Ongoing and future work
  - TLC: parallelize liveness checking, visualize counter-examples
  - Apalache: explore alternative SMT encodings, adapt algorithms such as IC3
  - TLAPS: better support for higher-order reasoning and for liveness proofs
  - IDEs: help with joint use of the tools, e.g. model checking from a proof step