An Introduction to the TLA$^+$ Language and its Tools

Stephan Merz

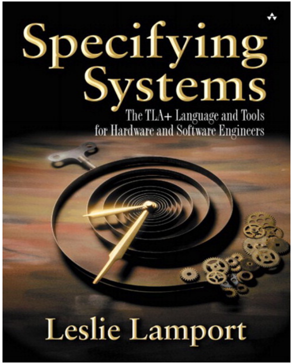
https://members.loria.fr/Stephan.Merz/

Inria Nancy – Grand Est & LORIA
Nancy, France

Loria
Laboratoire lorrain de recherche
en informatique et ses applications

35th International Symposium on Distributed Computing
Freiburg / online, October 2021

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)
- Hillel Wayne: Practical TLA$^+$, `https://learntla.com/` (focuses on PlusCal algorithm language)
- tools: TLC and Apalache model checkers, TLA$^+$ Proof System available from the TLA$^+$ Toolbox and VS Code Extension

Objective of this presentation

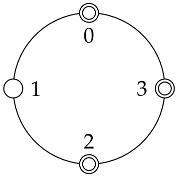
- Introduce basic concepts of TLA$^+$
- Model systems in TLA$^+$
- Tool support for verification: model checking and proof
- Presentation by example: distributed termination detection

Please interrupt for questions

Part I

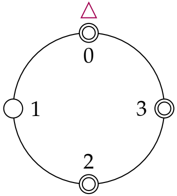
Modeling a Distributed Algorithm in TLA$^+$

Distributed Termination Detection



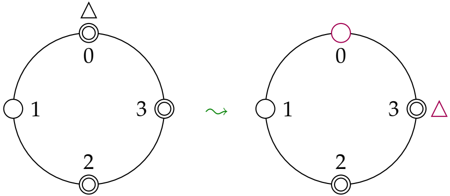
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive (simple circle)
  - “master node” 0 wishes to detect when all nodes are inactive

Distributed Termination Detection



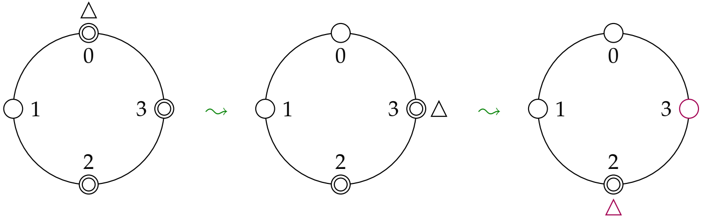
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive (simple circle)
  - "master node" 0 wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially, the master node holds the token

Distributed Termination Detection



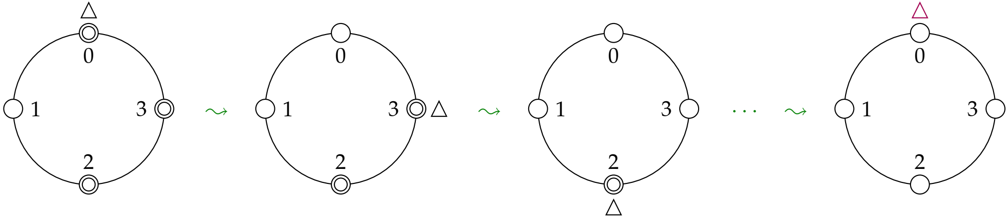
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive (simple circle)
  - "master node" 0 wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially, the master node holds the token
  - when the node holding the token has terminated, the token moves on

Distributed Termination Detection



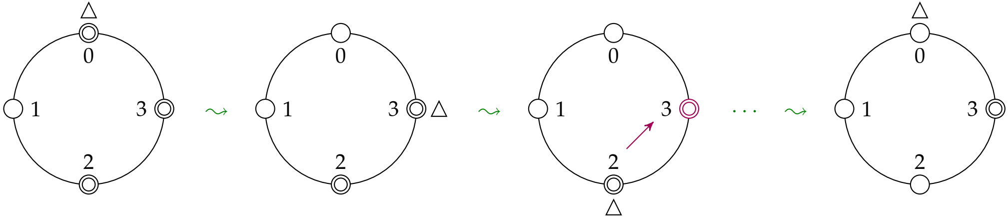
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive (simple circle)
  - "master node" 0 wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially, the master node holds the token
  - when the node holding the token has terminated, the token moves on

Distributed Termination Detection



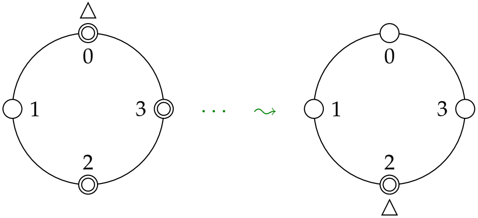
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive (simple circle)
  - "master node" 0 wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially, the master node holds the token
  - when the node holding the token has terminated, the token moves on
  - termination detected when token returns to (inactive) master node

Distributed Termination Detection



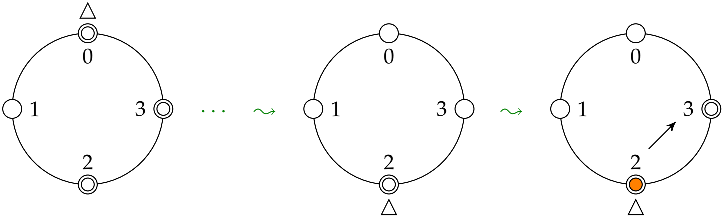
- Nodes arranged on a ring perform some computation
  - nodes can be active (double circle) or inactive (simple circle)
  - "master node" 0 wishes to detect when all nodes are inactive
- Token-based algorithm
  - initially, the master node holds the token
  - when the node holding the token has terminated, the token moves on
  - termination detected when token returns to (inactive) master node
- Complication: nodes may send messages, activating receiver

Dijkstra's Algorithm (EWD 840, 1983)



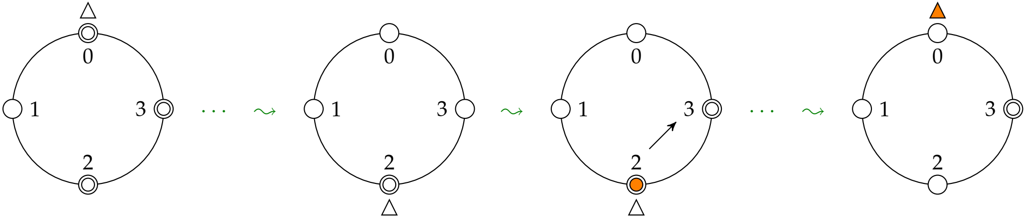
- Nodes and token colored orange or white
  - master node initiates probe by sending white token

Dijkstra's Algorithm (EWD 840, 1983)



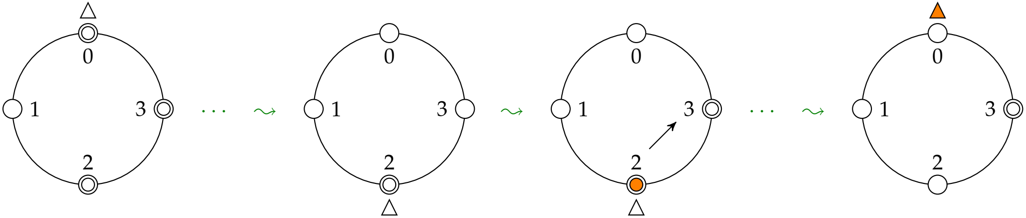
- Nodes and token colored orange or white
  - master node initiates probe by sending white token
  - node becomes black when sending a message to a higher-numbered node

Dijkstra's Algorithm (EWD 840, 1983)



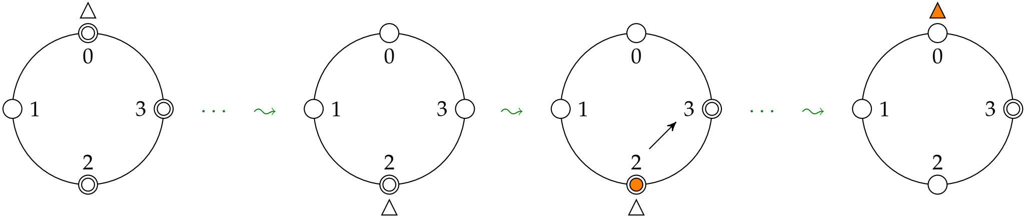
- Nodes and token colored orange or white
  - master node initiates probe by sending white token
  - node becomes black when sending a message to a higher-numbered node
  - when passing the token, an orange node transfers the orange color to the token

Dijkstra's Algorithm (EWD 840, 1983)



- Nodes and token colored orange or white
  - master node initiates probe by sending white token
  - node becomes black when sending a message to a higher-numbered node
  - when passing the token, an orange node transfers the orange color to the token
- Master node detects termination when it is inactive, white, and holds white token

Dijkstra's Algorithm (EWD 840, 1983)



- Nodes and token colored orange or white
  - master node initiates probe by sending white token
  - node becomes black when sending a message to a higher-numbered node
  - when passing the token, an orange node transfers the orange color to the token
- Master node detects termination when it is inactive, white, and holds white token
- Safety: termination detected only if all nodes are inactive
- Liveness: when all nodes inactive, termination will eventually be detected

Model-Based System Specifications in TLA$^+$

1. Describe the system configurations
   - state variables represent the state of the system
   - TLA$^+$ encourages abstractions in terms of sets, functions, tuples etc.
   - data model: classical (untyped) mathematical set theory

Model-Based System Specifications in TLA$^+$

1. Describe the system configurations

   - state variables represent the state of the system
   - TLA$^+$ encourages abstractions in terms of sets, functions, tuples etc.
   - data model: classical (untyped) mathematical set theory

2. State machine specification

initial condition *Init*	characterizes possible initial states	$x = 0 \land y \in Nat$
actions	describe effect of transitions	$x' = x + y \land y' = y$
next-state relation *Next*	disjunction of individual actions	
overall specification	models all system executions	$Init \land \Box[Next]_v$

Model-Based System Specifications in TLA$^+$

1. Describe the system configurations
   - state variables represent the state of the system
   - TLA$^+$ encourages abstractions in terms of sets, functions, tuples etc.
   - data model: classical (untyped) mathematical set theory

2. State machine specification

initial condition *Init*	characterizes possible initial states	$x = 0 \land y \in Nat$
actions	describe effect of transitions	$x' = x + y \land y' = y$
next-state relation *Next*	disjunction of individual actions	
overall specification	models all system executions	$Init \land \Box[Next]_v$

Specifications and properties expressed in mathematical logic

TLA$^+$ Specification of EWD 840: System Configurations

MODULE *EWD840*

$\text{EXTENDS } \mathit{Naturals}$

$\text{CONSTANT } N$

$\text{ASSUME } \mathit{NAssumption} \triangleq N \in \mathit{Nat} \setminus \{0\}$

$$\mathit{Nodes} \triangleq 0 \,..\, N-1$$
$$\mathit{Color} \triangleq \{\text{"white"}, \text{"orange"}\}$$

$\text{VARIABLES } \mathit{active}, \mathit{color}, \mathit{tpos}, \mathit{tcolor}$

$$\mathit{TypeOK} \triangleq \begin{array}{l} \land\ \mathit{active} \in [\mathit{Nodes} \to \text{BOOLEAN}] \land \mathit{color} \in [\mathit{Nodes} \to \mathit{Color}] \\ \land\ \mathit{tpos} \in \mathit{Nodes} \land \mathit{tcolor} \in \mathit{Color} \end{array}$$

- Declaration of constants and variables
- Definition of operators
  - sets *Nodes* and *Color*
  - *TypeOK* documents expected values of variables
  - *active* and *color* are arrays, i.e. functions

TLA$^+$ Specification of EWD 840: Initiation and System Transitions

$$Init \triangleq \land\ active \in [Nodes \rightarrow \text{BOOLEAN}] \land color \in [Nodes \rightarrow Color]$$
$$\land\ tpos \in Nodes \land tcolor = \text{“orange”}$$

- Initial condition: any “type-correct” values; token is initially orange

TLA$^+$ Specification of EWD 840: Initiation and System Transitions

$$
\begin{array}{l}
Init \triangleq \land\ active \in [Nodes \to \text{BOOLEAN}] \land color \in [Nodes \to Color] \\
\qquad\quad \land\ tpos \in Nodes \land tcolor = \text{“orange”} \\
InitiateProbe \triangleq \\
\qquad \land\ tpos = 0 \land (tcolor = \text{“orange”} \lor color[0] = \text{“orange”}) \\
\qquad \land\ tpos' = N - 1 \land tcolor' = \text{“white”} \\
\qquad \land\ color' = [color \text{ EXCEPT } ![0] = \text{“white”}] \\
\qquad \land\ active' = active \\
PassToken(i) \triangleq \\
\qquad \land\ tpos = i \land (\neg active[i] \lor color[i] = \text{“orange”} \lor tcolor = \text{“orange”}) \\
\qquad \land\ tpos' = i - 1 \\
\qquad \land\ tcolor' = \text{IF } color[i] = \text{“orange” THEN “orange” ELSE } tcolor \\
\qquad \land\ color' = [color \text{ EXCEPT } ![i] = \text{“white”}] \\
\qquad \land\ active' = active \\
System \triangleq InitiateProbe \lor \exists i \in Nodes \setminus \{0\} : PassToken(i)
\end{array}
$$

- Initial condition: any “type-correct” values; token is initially orange
- System transitions: token passing

TLA$^+$ Specification of EWD 840: Environment Transitions

$$
\begin{aligned}
&Terminate(i) \triangleq \\
&\quad \land\ active[i] \land active' = [active\ \text{EXCEPT}\ ![i] = \text{FALSE}] \\
&\quad \land\ \text{UNCHANGED}\ \langle color, tpos, tcolor \rangle \\
&SendMsg(i) \triangleq \\
&\quad \land\ active[i] \\
&\quad \land\ \exists j \in Nodes \setminus \{i\} : \land\ active' = [active\ \text{EXCEPT}\ ![j] = \text{TRUE}] \\
&\qquad\qquad\qquad\qquad\quad \land\ color' = [color\ \text{EXCEPT}\ ![i] = \text{IF}\ j > i\ \text{THEN}\ \text{“orange”}\ \text{ELSE}\ @] \\
&\quad \land\ \text{UNCHANGED}\ \langle tpos, tcolor \rangle \\
&Env \triangleq \exists i \in Nodes : Terminate(i) \lor SendMsg(i)
\end{aligned}
$$

- Definition of actions not controlled by the algorithm

TLA$^+$ Specification of EWD 840: Environment Transitions

$$
\begin{array}{l}
\mathit{Terminate}(i) \triangleq \\
\quad \land\ \mathit{active}[i] \land \mathit{active}' = [\mathit{active}\ \text{EXCEPT}\ ![i] = \text{FALSE}] \\
\quad \land\ \text{UNCHANGED}\ \langle \mathit{color}, \mathit{tpos}, \mathit{tcolor} \rangle \\
\mathit{SendMsg}(i) \triangleq \\
\quad \land\ \mathit{active}[i] \\
\quad \land\ \exists j \in \mathit{Nodes} \setminus \{i\} : \land\ \mathit{active}' = [\mathit{active}\ \text{EXCEPT}\ ![j] = \text{TRUE}] \\
\qquad\qquad\qquad\qquad\quad\ \land\ \mathit{color}' = [\mathit{color}\ \text{EXCEPT}\ ![i] = \text{IF}\ j > i\ \text{THEN}\ \text{“orange”}\ \text{ELSE}\ @] \\
\quad \land\ \text{UNCHANGED}\ \langle \mathit{tpos}, \mathit{tcolor} \rangle \\
\mathit{Env} \triangleq \exists i \in \mathit{Nodes} : \mathit{Terminate}(i) \lor \mathit{SendMsg}(i) \\
\mathit{Next} \triangleq \mathit{System} \lor \mathit{Env} \\
\mathit{vars} \triangleq \langle \mathit{tpos}, \mathit{tcolor}, \mathit{active}, \mathit{color} \rangle \\
\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{vars}}
\end{array}
$$

- Definition of actions not controlled by the algorithm
- Possible executions: initial condition, interleaving of transitions

Part II

Verification By Model Checking

Formulation of Safety Properties in TLA^+

1. Check type correctness
   - invariant of the specification: THEOREM $Spec \Rightarrow \Box TypeOK$
   - *TypeOK* is true throughout any execution of *Spec*

Formulation of Safety Properties in TLA$^+$

1. Check type correctness
   - invariant of the specification: $\text{THEOREM } Spec \Rightarrow \Box TypeOK$
   - *TypeOK* is true throughout any execution of *Spec*

2. All nodes are inactive when master node detects termination
   - master claims termination when it is white and inactive and holds a white token

$$
\begin{array}{l}
terminated \triangleq \forall i \in Nodes : \neg active[i] \\
terminationDetected \triangleq tpos = 0 \land tcolor = \text{“white”} \land color[0] = \text{“white”} \land \neg active[0] \\
TerminationDetection \triangleq terminationDetected \Rightarrow terminated \\
\text{THEOREM } Spec \Rightarrow \Box TerminationDetection
\end{array}
$$

   - formally again expressed as an invariant

Model Checking Using TLC

- Create a model: finite instance of TLA$^+$ specification
  - instantiate constant parameters by concrete values

    for example, create instance for $N = 5$
  - indicate operator corresponding to system specification

    heuristically set to *Spec* when that operator is defined in the module
  - indicate invariants to verify

    formulas *TypeOK* and *TerminationDetection*
  - TLC checks that these properties hold for this model
- TLC integrated into TLA$^+$ Toolbox (Eclipse GUI)

Model Checking Using TLC

- Create a model: finite instance of TLA+ specification
  - instantiate constant parameters by concrete values

    for example, create instance for $N = 5$
  - indicate operator corresponding to system specification

    heuristically set to *Spec* when that operator is defined in the module
  - indicate invariants to verify

    formulas *TypeOK* and *TerminationDetection*
  - TLC checks that these properties hold for this model
- TLC integrated into TLA+ Toolbox (Eclipse GUI)
- Exploit the automation of TLC for validating the specification
  - check both properties you believe to be true and false
  - gain confidence in your model, remove modeling errors

Checking Liveness of the Algorithm

- When all nodes are inactive, termination will be detected

$$Liveness \triangleq terminated \leadsto terminationDetected$$

$$\text{THEOREM } Spec \Rightarrow Liveness$$

Checking Liveness of the Algorithm

- When all nodes are inactive, termination will be detected

$$Liveness \triangleq terminated \leadsto terminationDetected$$

$$\text{THEOREM } Spec \Rightarrow Liveness$$

  - TLC produces a counter-example that ends in infinite stuttering
  - $\Box[Next]_{vars}$ allows for steps that do not change *vars*
  - we'll soon understand why TLA$^+$ specifications allow for stuttering

Checking Liveness of the Algorithm

- When all nodes are inactive, termination will be detected

$$Liveness \triangleq terminated \leadsto terminationDetected$$

$$\text{THEOREM } Spec \Rightarrow Liveness$$

  - TLC produces a counter-example that ends in infinite stuttering
  - $\Box[Next]_{vars}$ allows for steps that do not change *vars*
  - we'll soon understand why TLA$^+$ specifications allow for stuttering

- Fairness conditions rule out infinite stuttering
  - assert that an action will be taken, provided it is "often" enabled
  - abstractly represent assumptions about the "speed" of components

Checking Liveness of the Algorithm

- When all nodes are inactive, termination will be detected

$Liveness \triangleq terminated \leadsto terminationDetected$

THEOREM $Spec \Rightarrow Liveness$

  - TLC produces a counter-example that ends in infinite stuttering
  - $\Box[Next]_{vars}$ allows for steps that do not change *vars*
  - we'll soon understand why TLA$^+$ specifications allow for stuttering

- Fairness conditions rule out infinite stuttering
  - assert that an action will be taken, provided it is "often" enabled
  - abstractly represent assumptions about the "speed" of components

- Determining the right fairness conditions can be tricky

Fairness Conditions in TLA$^+$

- Two standard concepts of fairness
  - weak fairness — disallow behaviors where action is persistently enabled but never taken
  - strong fairness — disallow behaviors where action is repeatedly enabled but never taken

Fairness Conditions in TLA$^+$

- Two standard concepts of fairness
  - weak fairness disallow behaviors where action is persistently enabled but never taken
  - strong fairness disallow behaviors where action is repeatedly enabled but never taken
- Representation in temporal logic

$$\mathrm{WF}(A) \triangleq \Box((\Box \mathrm{ENABLED}\ A) \Rightarrow \Diamond A) \qquad \mathrm{SF}(A) \triangleq \Box((\Box\Diamond \mathrm{ENABLED}\ A) \Rightarrow \Diamond A)$$

Fairness Conditions in TLA$^+$

- Two standard concepts of fairness
  - weak fairness — disallow behaviors where action is persistently enabled but never taken
  - strong fairness — disallow behaviors where action is repeatedly enabled but never taken

- Representation in temporal logic

$$\mathrm{WF}_v(A) \triangleq \Box((\Box \mathrm{ENABLED}\ \langle A \rangle_v) \Rightarrow \Diamond \langle A \rangle_v) \qquad \mathrm{SF}_v(A) \triangleq \Box((\Box\Diamond \mathrm{ENABLED}\ \langle A \rangle_v) \Rightarrow \Diamond \langle A \rangle_v)$$

  - TLA$^+$ asserts fairness for non-stuttering actions

Fairness Conditions in TLA$^+$

- Two standard concepts of fairness
  - weak fairness — disallow behaviors where action is persistently enabled but never taken
  - strong fairness — disallow behaviors where action is repeatedly enabled but never taken
- Representation in temporal logic

$$\mathrm{WF}_v(A) \triangleq \Box((\Box\,\mathrm{ENABLED}\,\langle A\rangle_v) \Rightarrow \Diamond\langle A\rangle_v) \qquad \mathrm{SF}_v(A) \triangleq \Box((\Box\Diamond\,\mathrm{ENABLED}\,\langle A\rangle_v) \Rightarrow \Diamond\langle A\rangle_v)$$

  - TLA$^+$ asserts fairness for non-stuttering actions
- Fairness hypotheses for EWD 840
  - require fairness for the token-passing ("system") actions

$$\mathit{Spec} \triangleq \mathit{Init} \land \Box[\mathit{Next}]_{\mathit{vars}} \land \mathrm{WF}_{\mathit{vars}}(\mathit{System})$$

Another Way of Specifying the Problem

- We have modeled a concrete algorithm and verified its properties
  - checked that EWD 840 ensures expected safety and liveness properties
  - can we characterize the problem that the algorithm solves?

Another Way of Specifying the Problem

- We have modeled a concrete algorithm and verified its properties
  - checked that EWD 840 ensures expected safety and liveness properties
  - can we characterize the problem that the algorithm solves?
- Write a TLA$^+$ specification $\mathcal{TD}$ of termination detection
  - define the problem as an abstract state machine
  - then show that EWD 840 is an implementation of that problem

Another Way of Specifying the Problem

- We have modeled a concrete algorithm and verified its properties
  - checked that EWD 840 ensures expected safety and liveness properties
  - can we characterize the problem that the algorithm solves?
- Write a TLA$^+$ specification $\mathcal{TD}$ of termination detection
  - define the problem as an abstract state machine
  - then show that EWD 840 is an implementation of that problem
- TLA$^+$ doesn't distinguish between specifications and properties
  - implementation: every execution is allowed by the high-level state machine

    THEOREM $Spec \Rightarrow \mathcal{TD}!Spec$

  - insensitivity to stuttering is essential here:
    EWD 840 acts on variables (such as the token) that play no role in $\mathcal{TD}$

Specifying termination detection

MODULE *SyncTerminationDetection*

VARIABLES *active*, *terminationDetected*

$terminated \triangleq \forall n \in Nodes : \neg active[n]$

$Init \triangleq active \in [Nodes \rightarrow \text{BOOLEAN}] \land terminationDetected \in \{\text{FALSE}, terminated\}$

Specifying termination detection

MODULE *SyncTerminationDetection*

$$
\begin{array}{l}
\text{VARIABLES } active, terminationDetected \\
terminated \triangleq \forall n \in Nodes : \neg active[n] \\
Init \triangleq active \in [Nodes \to \text{BOOLEAN}] \land terminationDetected \in \{\text{FALSE}, terminated\} \\
Terminate(i) \triangleq \land active[i] \land active' = [active \text{ EXCEPT } ![i] = \text{FALSE}] \\
\qquad\qquad\qquad \land terminationDetected' \in \{terminationDetected, terminated'\} \\
Wakeup(i,j) \triangleq active[i] \land active' = [active \text{ EXCEPT } ![j] = \text{TRUE}] \land \text{UNCHANGED } terminationDetected \\
DetectTermination \triangleq terminated \land terminationDetected' = \text{TRUE} \land \text{UNCHANGED } active \\
Next \triangleq (\exists i \in Nodes : Terminate(i)) \lor (\exists i,j \in Nodes : Wakeup(i,j)) \lor DetectTermination \\
vars \triangleq \langle active, terminationDetected \rangle \\
Spec \triangleq Init \land \Box[Next]_{vars} \land \text{WF}_{vars}(DetectTermination)
\end{array}
$$

Specifying termination detection

MODULE *SyncTerminationDetection*

$$\begin{array}{l}
\text{VARIABLES } active, terminationDetected \\
terminated \triangleq \forall n \in Nodes : \neg active[n] \\
Init \triangleq active \in [Nodes \rightarrow \text{BOOLEAN}] \land terminationDetected \in \{\text{FALSE}, terminated\} \\
Terminate(i) \triangleq \land active[i] \land active' = [active \text{ EXCEPT } ![i] = \text{FALSE}] \\
\qquad\qquad\qquad \land terminationDetected' \in \{terminationDetected, terminated'\} \\
Wakeup(i,j) \triangleq active[i] \land active' = [active \text{ EXCEPT } ![j] = \text{TRUE}] \land \text{UNCHANGED } terminationDetected \\
DetectTermination \triangleq terminated \land terminationDetected' = \text{TRUE} \land \text{UNCHANGED } active \\
Next \triangleq (\exists i \in Nodes : Terminate(i)) \lor (\exists i,j \in Nodes : Wakeup(i,j)) \lor DetectTermination \\
vars \triangleq \langle active, terminationDetected \rangle \\
Spec \triangleq Init \land \Box[Next]_{vars} \land \text{WF}_{vars}(DetectTermination)
\end{array}$$

- Same overall structure as algorithm specification, can verify properties

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

$Spec \Rightarrow (terminated \leadsto terminationDetected)$

Checking Refinement

- Within module *EWD840*, create an instance of *SyncTerminationDetection*

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

  - parameters instantiated by the operators of the same name in *EWD840*
  - an instance may also substitute expressions for parameters ("refinement mapping")
  - formula *Spec* refers to the specification of module *EWD840*

- Refinement can be verified in the same way as other properties

Termination Detection When Communication is Asynchronous

- EWD840 assumes that messages between nodes are delivered instantaneously
- What if message delivery is asynchronous?

Termination Detection When Communication is Asynchronous

- EWD840 assumes that messages between nodes are delivered instantaneously
- What if message delivery is asynchronous?
  - token may go around the ring twice while the message is in transit
  - master node may declare termination and receiver later becomes active
  - easy exercise: adapt the specification and detect the problem using model checking

Termination Detection When Communication is Asynchronous

- EWD840 assumes that messages between nodes are delivered instantaneously
- What if message delivery is asynchronous?
  - token may go around the ring twice while the message is in transit
  - master node may declare termination and receiver later becomes active
  - easy exercise: adapt the specification and detect the problem using model checking
- Adaptation suggested by Shmuel Safra, published as EWD998 (1987)
  - each node stores difference δ_i between the number of messages sent and received locally
  - the token sums up the differences δ_i of nodes it passes as $tkn.q$
  - master detects termination when it's inactive and $\delta_0 + tkn.q = 0$ (plus color conditions)

Part III

Deductive Verification Using the TLA$^+$ Proof System

Using TLAPS to Prove Safety Properties

- TLAPS: proof assistant for verifying TLA$^+$ specifications
  - ⊕ verification is independent of the size of the model / state space
  - ⊖ interactive proof checker: user must guide the proof

Using TLAPS to Prove Safety Properties

- TLAPS: proof assistant for verifying TLA$^+$ specifications
  - ⊕ verification is independent of the size of the model / state space
  - ⊖ interactive proof checker: user must guide the proof

- Proving a simple invariant in TLAPS (for arbitrary N)

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

  - hierarchical proof language represents proof tree
  - individual 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

Simple Proofs

- Prove that *Init* implies *TypeOK*

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

  - relevant definitions and facts must be cited explicitly
  - this helps manage the size of the search space for proof tools

Simple Proofs

- Prove that *Init* implies *TypeOK*

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

  - relevant definitions and facts must be cited explicitly
  - this helps manage the size of the search space for proof tools

- Attempt similar proof for step $\langle 1 \rangle 2$

$\langle 1 \rangle 2.\ TypeOK \land [Next]_{vars} \Rightarrow TypeOK'$
BY $NAssumption$ DEFS $TypeOK, Next, vars, InitiateProbe, \ldots$

  - decompose proof into smaller steps when brute force fails

Hierarchical Proofs

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

$\quad \langle 2 \rangle$ SUFFICES ASSUME $\mathit{TypeOK}, [\mathit{Next}]_{\mathit{vars}}$

$\qquad\qquad\qquad$ PROVE TypeOK'

$\qquad$ OBVIOUS

$\quad \langle 2 \rangle$ USE $\mathit{NAssumption}$ DEF TypeOK

$\quad \langle 2 \rangle 1.$ CASE $\mathit{InitiateProbe}$

$\qquad$ BY $\langle 2 \rangle 1$ DEF $\mathit{InitiateProbe}$

$\quad \langle 2 \rangle 2.$ ASSUME NEW $i \in \mathit{Node} \setminus \{0\}, \mathit{PassToken}(i)$

$\qquad\quad$ PROVE TypeOK'

$\qquad$ BY $\langle 2 \rangle 2$ DEF $\mathit{PassToken}$

$\quad$ ... similarly for the remaining actions ...

$\quad \langle 2 \rangle$ QED BY $\langle 2 \rangle 1, \langle 2 \rangle 2, \ldots$ DEF Next

- SUFFICES steps represent backward chaining
- Toolbox IDE helps with hierarchical decomposition

Proof of Main Safety Property

- *TerminationDetection* is not preserved by the next-state relation
  - we need an inductive invariant that provides information about all reachable states

$Inv \triangleq \lor \forall i \in Nodes : i > tpos \Rightarrow \neg active[i]$	all nodes behind the token are inactive
$\lor \exists j \in 0 \,..\, tpos : color[j] = \text{"orange"}$	some node ahead is dirty
$\lor tcolor = \text{"orange"}$	the token is dirty

Proof of Main Safety Property

- *TerminationDetection* is not preserved by the next-state relation
  - we need an inductive invariant that provides information about all reachable states

$Inv \triangleq \lor \forall i \in Nodes : i > tpos \Rightarrow \neg active[i]$	all nodes behind the token are inactive
$\lor \exists j \in 0 \,..\, tpos : color[j] = \text{"orange"}$	some node ahead is dirty
$\lor\ tcolor = \text{"orange"}$	the token is dirty

  - use TLC to check that *Inv* is inductive (relative to *TypeOK*)

$$TypeOK \land Inv \land \Box[Next]_{vars} \Rightarrow \Box Inv$$

Proof of Main Safety Property

- *TerminationDetection* is not preserved by the next-state relation
  - we need an inductive invariant that provides information about all reachable states

$$\begin{aligned} Inv \triangleq\ & \lor \forall i \in Nodes : i > tpos \Rightarrow \neg active[i] && \text{all nodes behind the token are inactive} \\ & \lor \exists j \in 0 \,..\, tpos : color[j] = \text{“orange”} && \text{some node ahead is dirty} \\ & \lor tcolor = \text{“orange”} && \text{the token is dirty} \end{aligned}$$

  - use TLC to check that *Inv* is inductive (relative to *TypeOK*)

$$TypeOK \land Inv \land \Box[Next]_{vars} \Rightarrow \Box Inv$$

- Proof of the theorem

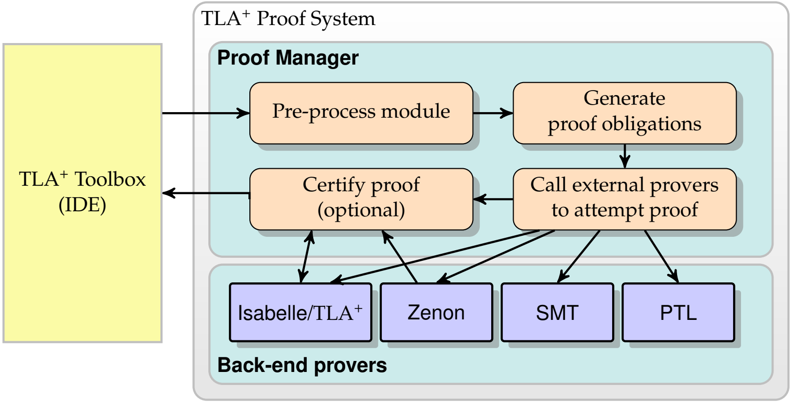
$\langle 1 \rangle 1.\ Init \Rightarrow Inv$

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

$\langle 1 \rangle 3.\ Inv \Rightarrow TerminationDetection$

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

TLAPS Architecture



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

Proving Liveness of EWD 840

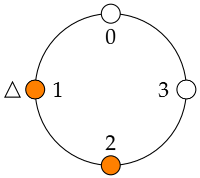
$$Liveness \triangleq terminated \leadsto terminationDetected$$

$$\text{THEOREM } Spec \Rightarrow Liveness$$

Proving Liveness of EWD 840

$Liveness \triangleq terminated \leadsto terminationDetected$

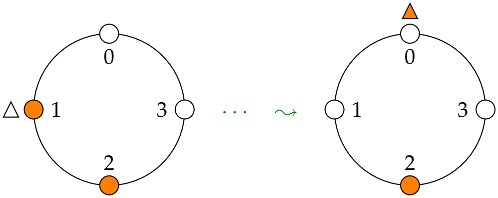
THEOREM $Spec \Rightarrow Liveness$



Proving Liveness of EWD 840

$Liveness \triangleq terminated \leadsto terminationDetected$

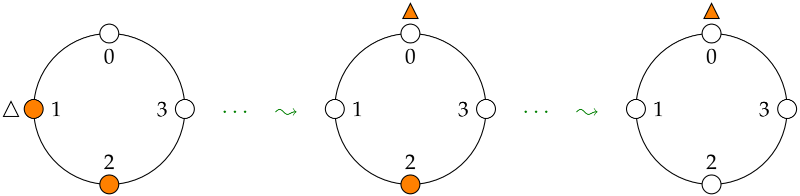
THEOREM $Spec \Rightarrow Liveness$



Proving Liveness of EWD 840

$Liveness \triangleq terminated \leadsto terminationDetected$

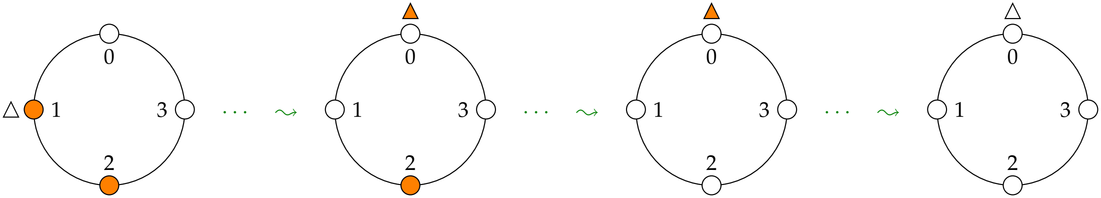
THEOREM $Spec \Rightarrow Liveness$



Proving Liveness of EWD 840

$Liveness \triangleq terminated \leadsto terminationDetected$

THEOREM $Spec \Rightarrow Liveness$

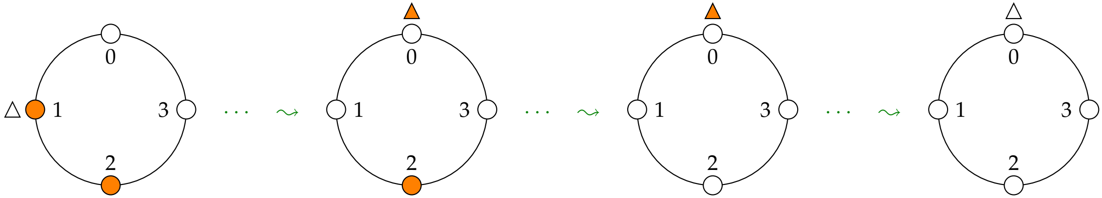


Three rounds of the token may be required between termination and detection

Proving Liveness of EWD 840

$Liveness \triangleq terminated \leadsto terminationDetected$

THEOREM $Spec \Rightarrow Liveness$



Three rounds of the token may be required between termination and detection

- Proof by contradiction: assume that termination is never detected

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

THEOREM $BSpec \Rightarrow Liveness$

Reasoning about ENABLED

- Liveness relies on the fairness hypothesis $\mathrm{WF}_{vars}(\mathit{System})$
  - remember: $\mathrm{WF}_v(A) \triangleq \Box((\Box\text{ENABLED}\ \langle A\rangle_v) \Rightarrow \Diamond\langle A\rangle_v)$
  - reasoning about fairness requires reasoning about ENABLED
  - prove $\text{ENABLED}\ \langle A\rangle_v \equiv P$ for some state predicate P

Reasoning about ENABLED

- Liveness relies on the fairness hypothesis $\mathrm{WF}_{vars}(System)$
  - remember: $\mathrm{WF}_v(A) \triangleq \Box((\Box \text{ENABLED } \langle A \rangle_v) \Rightarrow \Diamond \langle A \rangle_v)$
  - reasoning about fairness requires reasoning about ENABLED
  - prove $\text{ENABLED } \langle A \rangle_v \equiv P$ for some state predicate P
- $\text{ENABLED } \langle A \rangle_v \triangleq \exists v' : A \land v' \neq v$
  - expansion of ENABLED introduces quantifiers: problematic for automatic backends
  - non-stuttering conjunct adds extra complications
  - better: rely on specific rules for simplifying ENABLED

Computing ENABLED $\langle System \rangle_{vars}$

1. Observe that all *System* steps are non-stuttering

   LEMMA $TypeOK \Rightarrow (\langle System \rangle_{vars} \equiv System)$

   - established using standard action-level reasoning

Computing ENABLED $\langle System \rangle_{vars}$

1. Observe that all *System* steps are non-stuttering

LEMMA $TypeOK \Rightarrow (\langle System \rangle_{vars} \equiv System)$

- established using standard action-level reasoning

2. Use monotonicity of ENABLED

COROLLARY $TypeOK \Rightarrow (\text{ENABLED } \langle System \rangle_{vars} \equiv \text{ENABLED } System)$

- embodied in proof directive ENABLEDrules

Computing ENABLED $\langle System \rangle_{vars}$

1. Observe that all *System* steps are non-stuttering

   LEMMA $TypeOK \Rightarrow (\langle System \rangle_{vars} \equiv System)$

   - established using standard action-level reasoning

2. Use monotonicity of ENABLED

   COROLLARY $TypeOK \Rightarrow (\text{ENABLED } \langle System \rangle_{vars} \equiv \text{ENABLED } System)$

   - embodied in proof directive ENABLEDrules

3. ENABLEDrewrites pushes ENABLED across logical operators

   $\text{ENABLED } (A \lor B) \equiv (\text{ENABLED } A) \lor (\text{ENABLED } B)$
   $\text{ENABLED } (A \land B) \equiv (\text{ENABLED } A) \land (\text{ENABLED } B)$ if A and B have disjoint primed variables
   $\text{ENABLED } (x' = t) \equiv \text{TRUE}$ if t has no primed variables
   $\text{ENABLED } P \equiv P$ if P is a state predicate *etc.*

Theorem on the Enabledness Condition

$$SystemEnabled \triangleq \lor\ tpos = 0 \land (tcolor = \text{"orange"} \lor color[0] = \text{"orange"})$$
$$\lor\ \exists\, i \in Nodes \setminus \{0\} : tpos = i \land (\neg active[i] \lor tcolor = \text{"orange"} \lor color[i] = \text{"orange"})$$

THEOREM ASSUME $TypeOK$
PROVE ENABLED $\langle System \rangle_{vars} \equiv SystemEnabled$

$\langle 1 \rangle 1.\ System \equiv \langle System \rangle_{vars}$
BY DEF $TypeOK$, $System$, $vars$, $InitiateProbe$, $PassToken$, $Nodes$

$\langle 1 \rangle 2.$ (ENABLED $System$) $\equiv$ ENABLED $\langle System \rangle_{vars}$
BY $\langle 1 \rangle 1$, ENABLED*rules*

$\langle 1 \rangle 3.$ QED
BY $\langle 1 \rangle 2$, ENABLED*rewrites* DEF $System$, $InitiateProbe$, $PassToken$, $SystemEnabled$

Liveness Proof: First Round

Prove that token will return to node 0

$$\text{LEMMA } Round1 \triangleq BSpec \Rightarrow (terminated \leadsto (terminated \land tpos = 0))$$

Liveness Proof: First Round

Prove that token will return to node 0

$$\text{LEMMA } Round1 \triangleq BSpec \Rightarrow (terminated \leadsto (terminated \land tpos = 0))$$

- show by induction that token will travel from any node i to node 0

$$P(i) \triangleq terminated \land i \in Nodes \land tpos = i$$
$$R(i) \triangleq BSpec \Rightarrow (P(i) \leadsto P(0))$$

Liveness Proof: First Round

Prove that token will return to node 0

$$\text{LEMMA } \mathit{Round1} \triangleq \mathit{BSpec} \Rightarrow (\mathit{terminated} \leadsto (\mathit{terminated} \land \mathit{tpos} = 0))$$

- show by induction that token will travel from any node i to node 0

$$P(i) \triangleq \mathit{terminated} \land i \in \mathit{Nodes} \land \mathit{tpos} = i$$
$$R(i) \triangleq \mathit{BSpec} \Rightarrow (P(i) \leadsto P(0))$$

- $R(0)$ holds trivially

Liveness Proof: First Round

Prove that token will return to node 0

$$\text{LEMMA } Round1 \triangleq BSpec \Rightarrow (terminated \leadsto (terminated \land tpos = 0))$$

- show by induction that token will travel from any node i to node 0

$$P(i) \triangleq terminated \land i \in Nodes \land tpos = i$$
$$R(i) \triangleq BSpec \Rightarrow (P(i) \leadsto P(0))$$

- $R(0)$ holds trivially
- for any $i \in Nodes$, prove that $R(i) \Rightarrow R(i+1)$

$\langle 3 \rangle 1.\ TypeOK \land P(i+1) \land [Next]_{vars} \Rightarrow P(i+1)' \lor P(i)'$
$\langle 3 \rangle 2.\ TypeOK \land P(i+1) \land \langle System \rangle_{vars} \Rightarrow P(i)'$
$\langle 3 \rangle 3.\ TypeOK \land P(i+1) \Rightarrow \text{ENABLED } \langle System \rangle_{vars}$
$\langle 3 \rangle 4.\ BSpec \Rightarrow (P(i+1) \leadsto P(i))$ BY $\langle 3 \rangle 1$, $\langle 3 \rangle 2$, $\langle 3 \rangle 3$, PTL DEF $BSpec$
$\langle 3 \rangle$. QED BY $\langle 3 \rangle 4$, PTL

Finishing the Liveness Proof

- Prove two similar lemmas about the two remaining rounds

$$allWhite \triangleq \forall i \in Nodes : color[i] = \text{"white"}$$

$$\text{LEMMA } Round2 \triangleq BSpec \Rightarrow (terminated \land tpos = 0 \leadsto (terminated \land tpos = 0 \land allWhite))$$

$$\text{LEMMA } Round3 \triangleq BSpec \Rightarrow (terminated \land tpos = 0 \land allWhite \leadsto (terminated \land tpos = 0 \land allWhite \land tcolor = \text{"white"}))$$

  - proofs of these lemmas obtained by copy and paste from that of *Round*1
  - clearly: $terminated \land tpos = 0 \land allWhite \land tcolor = \text{"white"} \Rightarrow terminationDetected$

Finishing the Liveness Proof

- Prove two similar lemmas about the two remaining rounds

$$allWhite \triangleq \forall i \in Nodes : color[i] = \text{"white"}$$

$$\text{LEMMA } Round2 \triangleq BSpec \Rightarrow (terminated \land tpos = 0 \leadsto (terminated \land tpos = 0 \land allWhite))$$

$$\text{LEMMA } Round3 \triangleq BSpec \Rightarrow (terminated \land tpos = 0 \land allWhite \leadsto (terminated \land tpos = 0 \land allWhite \land tcolor = \text{"white"}))$$

  - proofs of these lemmas obtained by copy and paste from that of $Round1$
  - clearly: $terminated \land tpos = 0 \land allWhite \land tcolor = \text{"white"} \Rightarrow terminationDetected$

- Putting everything together

$$\text{THEOREM } Spec \Rightarrow Liveness$$

$$\text{BY } TypeCorrect, Round1, Round2, Round3, \text{ PTL DEF } Spec, BSpec, Liveness$$

Part IV

Conclusion

Summing Up

- TLA$^+$: formal language for specifying systems based on mathematics
  - highly expressive and flexible language favors abstraction
  - state machines for representing system behavior
  - no distinction between systems and properties
  - refinement (and composition) reflected in logic
- Support tools
  - TLA$^+$ Toolbox: editor, syntax/semantic analysis, pretty printer
  - TLC: explicit-state model checker, checkpointing, parallelization
  - TLAPS: interactive proof platform, automatic proof back-ends
  - Apalache: bounded model checking based on SMT encoding
  - PlusCal: front-end for generating TLA$^+$ from "pseudo code" language
- More information

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