TLA$^+$: The Tools, The Language, The Application

Markus A. Kuppe
makuppe@microsoft.com

Microsoft

March, 2023

Table of Contents

Temporal Logic of Actions$^+$

TLA$^+$ is a *specification* language to model and verify reactive systems.

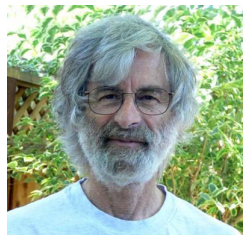


Figure: Leslie Lamport

Key observations $\rightsquigarrow$ TLA (note the absent +)

- Linear Temporal Logic [Pnueli, 1977]
  - Convenient to reason about & express fairness and liveness of reactive systems
  - Less practical to axiomatically specify the actual system
- Instead use programming language to spec system?
  - In TLA, we model in the logic (with temporal part pushed aside)
- => Extend and restrict LTL

TLA: Temporal Logic Actions

- Recall LTL only has state formulas
- TLA [Lamport, 1994] expressions state, action, or temporal formulae
  - Interpreted over a state, pair of states, or sequence of states (behaviors)
- TLA only two[1] temporal operators (let P be a state- and A an action-predicate):
  - $\Box P$: P holds in every step/state of a behavior B
  - $\Diamond P$: P holds in at least one step of B
  - $\Diamond\Box P$: P holds in the (infinite) suffix of B
  - $\Box\Diamond P$: P holds repeatedly in B

- $\Box [A]_v$: The system only ever takes A steps
- $\Diamond [A]_v$: A is true of one or more steps of a B
- $\Diamond\Box [A]_v$: B's suffix only A steps
- $\Box\Diamond [A]_v$: Repeatedly A steps

[1] $\sim \Box \sim P \iff \Diamond P$

TLA: Temporal Logic Actions

- Recall LTL only has state formulas
- TLA [Lamport, 1994] expressions state, action, or temporal formulae
  - Interpreted over a state, pair of states, or sequence of states (behaviors)
- TLA only two[1] temporal operators (let P be a state- and A an action-predicate):
  - $\Box P$: P holds in every step/state of a behavior B
  - $\Diamond P$: P holds in at least one step of B
  - $\Diamond\Box P$: P holds in the (infinite) suffix of B
  - $\Box\Diamond P$: P holds repeatedly in B

  - $\Box [A]_v$: The system only ever takes A steps
  - $\Diamond [A]_v$: A is true of one or more steps of a B
  - $\Diamond\Box [A]_v$: B's suffix only A steps
  - $\Box\Diamond [A]_v$: Repeatedly A steps

[1] $\sim \Box \sim P \iff \Diamond P$

TLA: Specs

MODULE *HourClock*

VARIABLE *hr*

$Init \triangleq hr = \ldots$ Defines initial states

$Next \triangleq hr' =$ Constrains the next state

$Spec \triangleq Init \wedge \Box[Next]_{hr} \wedge F$ Defines behaviors

TLA: Specs

MODULE *HourClock*

VARIABLE hr

$Init \triangleq hr = \ldots$ Defines initial states

$Next \triangleq hr' =$ Constrains the next state

$Spec \triangleq Init \wedge \Box[Next]_{hr} \wedge F$ Defines behaviors

---

$Safety \triangleq \Box(hr \in \ldots)$

$Liveness \triangleq \Box\Diamond(hr = Midnight)$

TLA: Specs

MODULE *HourClock*

VARIABLE *hr*

...

$Spec \triangleq Init \wedge \Box[Next]_{hr} \wedge F$

$Safety \triangleq \Box(hr \in \ldots)$

$Liveness \triangleq \Box\Diamond(hr = 12)$

THEOREM $Spec \implies Safety$

$\langle 1 \rangle 1.\ Init \implies IInv$

$\langle 1 \rangle 2.\ IInv \wedge [Next]_{hr} \implies IInv'$

$\langle 1 \rangle 3.\ IInv \implies Safety$

$\langle 1 \rangle 4.$ QED

THEOREM $Spec \implies Liveness$

TLA: Refinement

MODULE *MinuteClock*

VARIABLE hr, min

$Init \triangleq hr = \ldots \land min = \ldots$

$Next \triangleq hr' = \ldots \land min' = \ldots$

$Spec \triangleq Init \land \Box[Next]_{\langle hr, min \rangle} \land G$

TLA: Refinement

MODULE *MinuteClock*

VARIABLE hr, min

$Init \triangleq hr = \ldots \land min = \ldots$

$Next \triangleq hr' = \ldots \land min' = \ldots$

$Spec \triangleq Init \land \Box[Next]_{\langle hr, min \rangle} \land G$

THEOREM $Spec \implies HR!Spec$

$\langle 1 \rangle 1.\ Init \implies HR!Init$

$\langle 1 \rangle 2.\ Inv \land [Next]_{\langle \ldots \rangle} \implies [HR!Next]_{(HR!hr)}$

$\langle 1 \rangle 3. \Box Inv \land \Box[Next]_{\langle \ldots \rangle} \land G \implies F$

$\langle 1 \rangle 4.$ QED

- Implementation is (logical) implication

TLA: Invariance under stuttering

- Invariance of TF under stuttering
  - $\sigma_{HC} = \langle s_0, s_1, s_1, s_1, s_1, s_1, s_2, s_3, s_4, \ldots \rangle$
  - $\tau_{MC} = \langle s_0, s_{1_1}, s_{1_2}, s_{1_3}, s_{1_4}, s_2, s_3, s_4, \ldots \rangle$
- Restrictions to be invariant
  - No (LTL) next operator
  - $\Box A$ with A an action is not well-formed: $\Box [A]_v$
  - $[A]_v \iff A \vee v = v'$
  - $\langle A \rangle_v \iff A \wedge v \neq v'$ [2]

[2] Is $\Box \langle A \rangle_v$ a valid formula?

TLA$^+$

- Untyped[3]: Zermelo–Fraenkel set theory with choice
- LET/IN, CASE, ITE, Recursion, Modules

MODULE *MinuteClock*

EXTENDS *FiniteSets*, *Sequences*
CONSTANT N

VARIABLE hr, min

$$
\begin{array}{ll}
Init & \triangleq\ hr \in (1 \,..\, N) \land min \in (0 \,..\, 59) \\
Tick & \triangleq\ min = 59 \land min' = 0 \land \text{IF } hr \neq N \text{ THEN } hr + 1 \text{ ELSE } 1 \\
Tock & \triangleq\ min < 59 \land min' = min + 1 \land \text{UNCHANGED } hr \\
Spec & \triangleq\ Init \land \Box[Tick \lor Tock]_{\langle hr,\, min \rangle} \land G
\end{array}
$$

[3]Types are useful with programming languages.

PlusCal Algorithm Language

- PlusCal embeds and transpiles to, but less expressive than TLA$^+$ [Lamport, 2009]

```
--algorithm BlockingQueue{
  variable buffer = ⟨⟩ ; waitset = {} ;
  notify ≜ ∃ w ∈ waitSet : waitSet' = waitSet \ {w}
  wait(t) ≜ waitSet' = waitSet ∪ {t}
  process (p ∈ P){
      put: while (TRUE){
             if (isFull){
               wait(self) ;
             }else {
               notify() ;
               buffer := Append(buffer, self) ;
      lbl:     skip ;
             } ;
           } ;
  }
```

Table of Contents

TLA Proof System

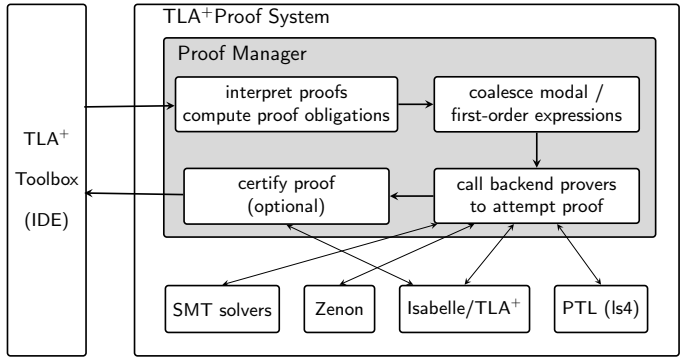


Figure: Architecture of TLAPS [from Merz, 2019]

- [Chaudhuri et al., 2008, 2010, Cousineau et al., 2012]
- `https://github.com/tlaplus/tlapm`

Apalache

- Symbolic model checker (z3[4]) for TLA$^+$ [Konnov et al., 2015]
- Handles a subclass of TLA$^+$ that is useful in practice
  - No Hiding (\EE & \AA), Composition, ...
  - No Recursion (but folds)
  - Requires types
  - No Liveness checking
    - `https://youtu.be/K777MY4Xugs` based on Biere et al. [2002]
- Safety checking is bounded model checking
- `https://github.com/informalsystems/apalache/`

[4] [de Moura and Bjørner, 2008]

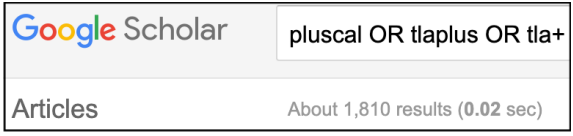
TLC

- Explicit-state model checker for TLA$^+$ [Yu et al., 1999]
- Handles (defined) a subclass of TLA$^+$ that is useful in practice
  - No Hiding (\EE & \AA), Composition, ...
- Safety checking corresponds to Breadth-First search over on-the-fly generated state graph
- Liveness checking corresponds to search for SCCs over behavior graph[5]
- `https://github.com/tlaplus/tlaplus`

[5] [Manna and Pnueli, 1995]

Table of Contents

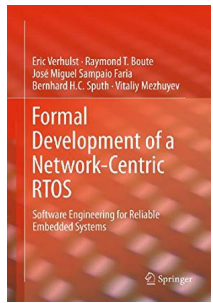
Applications of TLA+



Cleaner Architecture

Verhulst [2011], Head OpenComRTOS development group:

> *"The [TLA+] abstraction helped a lot in coming to a much cleaner architecture (we witnessed first hand the brain washing done by years of C programming). One of the results was that the code size is about 5-10x less than [in the previous version]"*



Amazon AWS

- DynamoDB: scalable high-performance "no SQL" data store with cross datacenter replication and strong consistency guarantees
- First informal proofs and excessive (fault-injecting) testing

- TLC[6] found subtle bug: shortest error trace 35 steps
- "*Using TLA$^+$ in place of traditional proof writing would thus likely have improved time to market, in addition to achieving greater confidence in the system's correctness.*" [Newcombe, 2014, Newcombe et al., 2015]

- Fast forward to 2022: "*The core replication protocol was specified using TLA+. When new features that affect the replication protocol are added, they are incorporated into the specification and model checked.*" [Elhemali et al., 2022]

[6] Distributed TLC: 10 nodes with 80+ cores and 230GB RAM.

Amazon AWS

- DynamoDB: scalable high-performance "no SQL" data store with cross datacenter replication and strong consistency guarantees
- First informal proofs and excessive (fault-injecting) testing

- TLC[6] found subtle bug: shortest error trace 35 steps
- "*Using TLA$^+$ in place of traditional proof writing would thus likely have* improved time to market*, in addition to achieving* greater confidence *in the system's correctness.*" [Newcombe, 2014, Newcombe et al., 2015]

- Fast forward to 2022: "*The core replication protocol was specified using TLA+. When new features that affect the replication protocol are added, they are incorporated into the specification and model checked.*" [Elhemali et al., 2022]

[6] Distributed TLC: 10 nodes with 80+ cores and 230GB RAM.

Amazon AWS

- DynamoDB: scalable high-performance "no SQL" data store with cross datacenter replication and strong consistency guarantees
- First informal proofs and excessive (fault-injecting) testing

- TLC[6] found subtle bug: shortest error trace 35 steps
- "*Using TLA$^+$ in place of traditional proof writing would thus likely have* improved time to market*, in addition to achieving* greater confidence *in the system's correctness.*" [Newcombe, 2014, Newcombe et al., 2015]

- Fast forward to 2022: "*The core replication protocol was specified using TLA+. When new features that affect the replication protocol are added, they are incorporated into the specification and model checked.*" [Elhemali et al., 2022]

[6] Distributed TLC: 10 nodes with 80+ cores and 230GB RAM.

High-Profile Cloud Incident

[Hackett et al., 2022]

- Affected thousands of Azure customers
- 28 days to detect and mitigate (rollback)
- Span multiple teams & services

- Root Cause:
  - `Old: GetResourcesDataProvider(India)`
  - `New: GetResourcesDataProvider(Europe)`
- Part of resilience layer & Vetted by senior engineers

- 3000+ word postmortem[7]

[7] ICM postmortem #521677 (Microsoft-internal)

High-Profile Cloud Incident

[Hackett et al., 2022]

- Affected thousands of Azure customers
- 28 days to detect and mitigate (rollback)
- Span multiple teams & services

- Root Cause:
  - `Old: GetResourcesDataProvider(India)`
  - `New: GetResourcesDataProvider(Europe)`
- Part of resilience layer & Vetted by senior engineers

- 3000+ word postmortem[7]

[7] ICM postmortem #521677 (Microsoft-internal)

High-Profile Cloud Incident

[Hackett et al., 2022]

- Affected thousands of Azure customers
- 28 days to detect and mitigate (rollback)
- Span multiple teams & services

- Root Cause:
  - `Old: GetResourcesDataProvider(India)`
  - `New: GetResourcesDataProvider(Europe)`
- Part of resilience layer & Vetted by senior engineers

- 3000+ word postmortem[7]

[7] ICM postmortem #521677 (Microsoft-internal)

High-Profile Cloud Incident

- Testing poor coverage because services only exhibit some patterns at scale

Figure: What to model at what level of detail?

High-Profile Cloud Incident

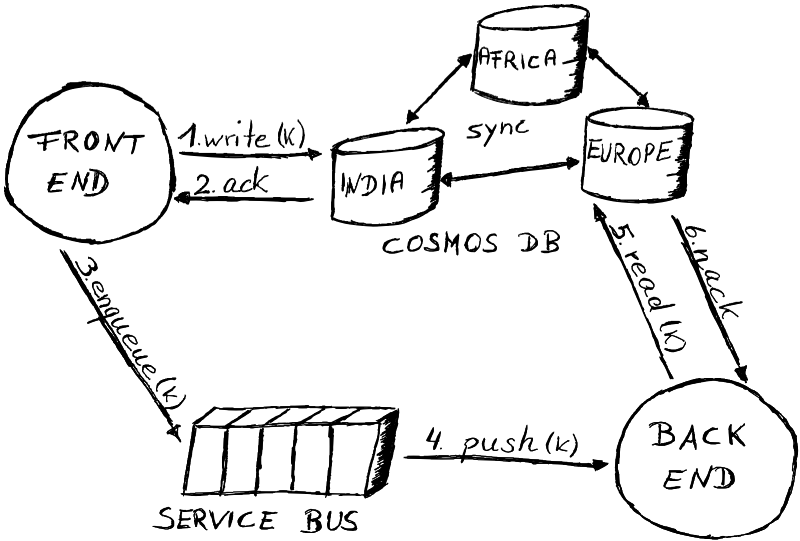


Figure: Architecture

High-Profile Cloud Incident

Demo

Database spec

- Reusable DB spec modeling all five consistency levels, message loss, and replica & region failures

- Found database documentation bugs
- Replicas and regions is the wrong mental model
  - TLA$^+$ specs linked as ultimate truth from the official Azure documentation

- Rollback does not fully address the issue => latent bug
- Redesign of the storage system

- Code (SQL) level: *MonkeyDB* by Biswas et al. [2021]

Database spec

- Reusable DB spec modeling all five consistency levels, message loss, and replica & region failures

- Found database documentation bugs
- Replicas and regions is the wrong mental model
  - TLA$^+$ specs linked as ultimate truth from the official Azure documentation

- Rollback does not fully address the issue => latent bug
- Redesign of the storage system

- Code (SQL) level: *MonkeyDB* by Biswas et al. [2021]

Database spec

- Reusable DB spec modeling all five consistency levels, message loss, and replica & region failures

- Found database documentation bugs
- Replicas and regions is the wrong mental model
  - TLA$^+$ specs linked as ultimate truth from the official Azure documentation

- Rollback does not fully address the issue => latent bug
- Redesign of the storage system

- Code (SQL) level: *MonkeyDB* by Biswas et al. [2021]

Database spec

- Reusable DB spec modeling all five consistency levels, message loss, and replica & region failures

- Found database documentation bugs
- Replicas and regions is the wrong mental model
  - TLA$^+$ specs linked as ultimate truth from the official Azure documentation

- Rollback does not fully address the issue => latent bug
- Redesign of the storage system

- Code (SQL) level: *MonkeyDB* by Biswas et al. [2021]

Confidential Consortium Framework (CCF)

- CCF: "Confidential Computer with Decentralized Trust" [Shamis et al., 2022]
- Based on raft with dynamic reconfiguration [Ongardie, 2016]
- TLA$^+$ spec: modify raft's def of commit to include signature scheme

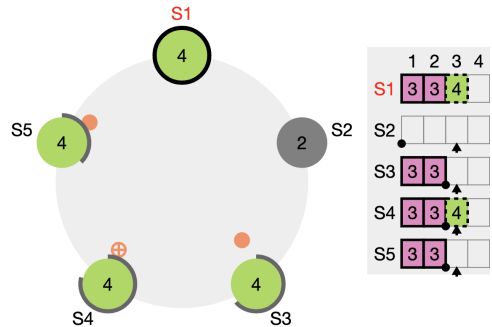


Figure: raft sketch (Raft Simulator)

Confidential Consortium Framework (CCF)

- CCF: "Confidential Computer with Decentralized Trust" [Shamis et al., 2022]
- Based on raft with dynamic reconfiguration [Ongardie, 2016]
- TLA$^+$ spec: modify raft's def of commit to include signature scheme

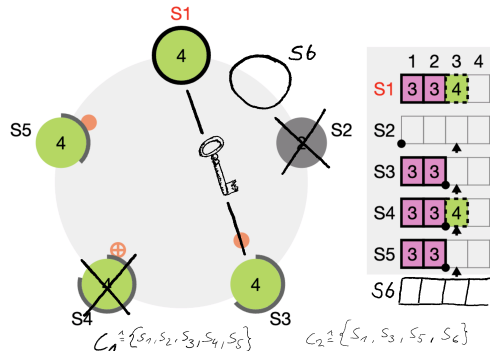


Figure: raft sketch (Raft Simulator)

CCF: Quorum across all configs vs. quorum in every configs[8]

- Safety violation (multiple leaders) if ordinary leader election coincides with reconfiguration, s.t. $abs\left(|c_1| - |c_2|\right) \geq 2$
  - Trace length 29 states
- First found by TLC (BFS) after 2d on a beefy machine

- Independently found with TLC (random exploration) in $\sim 10m$
  - Trace length 89 states
  - Reduce probability of failure actions
  - Start from randomly drawn subset of any reachable state

```
@@ -408,9 +406,8 @@ AppendEntries(i, j) ==
408  406  \* Candidate i transitions to leader.
409  407  BecomeLeader(i) ==
410  408      /\ state[i] = Candidate
411       -    /\ \* To become leader, a Quorum of _all_ known nodes must have voted for this server (across configurations)
412       -       LET relevantServers == GetServerSet(i)
413       -       IN (votesGranted[i] \cap relevantServers) \in CalculateQuorum(relevantServers)
     409  +    \* To become leader, the candidate must have received votes from a majority in each active configuration
     410  +    /\ \A k \in 1..Len(Configurations[i]) : votesGranted[i] \in CalculateQuorum(Configurations[i][k][2])
414  411      /\ state'      = [state EXCEPT ![i] = Leader]
415  412      /\ nextIndex'  = [nextIndex EXCEPT ![i] =
416  413                           [j \in PossibleServer |-> Len(log[i]) + 1]]
```

[8] Chamayou, Howard, and Maffre

CCF: Quorum across all configs vs. quorum in every configs[8]

- Safety violation (multiple leaders) if ordinary leader election coincides with reconfiguration, s.t. $abs\left(|c_1| - |c_2|\right) \geq 2$
  - Trace length 29 states
- First found by TLC (BFS) after 2d on a beefy machine
- Independently found with TLC (random exploration) in $\sim 10m$
  - Trace length 89 states
  - Reduce probability of failure actions
  - Start from randomly drawn subset of any reachable state

```
@@ -408,9 +406,8 @@ AppendEntries(i, j) ==
408 406  \* Candidate i transitions to leader.
409 407  BecomeLeader(i) ==
410 408      /\ state[i] = Candidate
411      -   /\  \* To become leader, a Quorum of _all_ known nodes must have voted for this server (across configurations)
412      -       LET relevantServers == GetServerSet(i)
413      -       IN (votesGranted[i] \cap relevantServers) \in CalculateQuorum(relevantServers)
     409 +   \* To become leader, the candidate must have received votes from a majority in each active configuration
     410 +   /\ \A k \in 1..Len(Configurations[i]) : votesGranted[i] \in CalculateQuorum(Configurations[i][k][2])
414 411      /\ state'      = [state EXCEPT ![i] = Leader]
415 412      /\ nextIndex'  = [nextIndex EXCEPT ![i] =
416 413                           [j \in PossibleServer |-> Len(log[i]) + 1]]
```

[8] Chamayou, Howard, and Maffre

CCF: Quorum across all configs vs. quorum in every configs[8]

- Safety violation (multiple leaders) if ordinary leader election coincides with reconfiguration, s.t. $abs\left(|c_1| - |c_2|\right) \geq 2$
  - Trace length 29 states
- First found by TLC (BFS) after 2d on a beefy machine
- Independently found with TLC (random exploration) in $\sim 10m$
  - Trace length 89 states
  - Reduce probability of failure actions
  - Start from randomly drawn subset of any reachable state

[8] Chamayou, Howard, and Maffre

Implementation of Specs

- Refinement all the way down
  - Validate low-level executions against spec [Joshi et al., 2003]
- Code generation
  - (PGo Beschastnikh et. al.)
- Model-driven verification
  - Replay high-level behaviors with implementation [Dorminey, 2019]
- Test-case generation
  - Davis et al. [2020]
- Translate TLA$^+$ properties to code-level properties

CCF: Trace Validation

MODULE *SystemLog*

EXTENDS *VectorClock*, ...

$ExecLog \triangleq$
  $CausalOrder(Deserialize(\text{“logfile.json”}))$

...

$Next \triangleq$
  $\lor \land ExecLog(level).e = \text{“BecomeLeader”}$
    $\land state' = ExecLog(level).state$
    $\land term' \in 1 \,..\, ExecLog(level).idx$ log holes
  $\lor \land ExecLog(level).e = \text{“Compact”}$
    $\land$ UNCHANGED *vars*

$CCF \triangleq$ INSTANCE *CCF* with $N \leftarrow \ldots$

THEOREM $Init \land \Box[Next]_{vars} \implies$
 $\exists\, votes : CCF!Init$
  $\land \Box[CCF!Next \lor CCF!BL \cdot CCF!TO]_{CCF!vars}$

- Spec-driven development & regression testing
- Example: EWD998

- Only safety!
- Embarrassingly parallel
- Spec coverage implies code coverage

- Spec as stimulus for the execution
- Conformance monitoring

CCF: Trace Validation

MODULE *SystemLog*

EXTENDS *VectorClock*, ...

ExecLog $\triangleq$
  CausalOrder(*Deserialize*("logfile.json"))

...

Next $\triangleq$
  $\lor \land$ *ExecLog*(*level*).*e* = "BecomeLeader"
    $\land$ *state*′ = *ExecLog*(*level*).*state*
    $\land$ *term*′ $\in 1 \,..\, ExecLog(level).idx$ log holes
  $\lor \land$ *ExecLog*(*level*).*e* = "Compact"
    $\land$ UNCHANGED *vars*

CCF $\triangleq$ INSTANCE *CCF* with *N* $\leftarrow$ ...

THEOREM *Init* $\land \Box[Next]_{vars} \implies$
  $\exists$ *votes* : *CCF*!*Init*
  $\land \Box[CCF!Next \lor CCF!BL \cdot CCF!TO]_{CCF!vars}$

- Spec-driven development & regression testing
- Example: EWD998

- Only safety!
- Embarrassingly parallel
- Spec coverage implies code coverage

- Spec as stimulus for the execution
- Conformance monitoring

CCF: Trace Validation

MODULE *SystemLog*

EXTENDS *VectorClock*, ...

ExecLog $\triangleq$
  CausalOrder(*Deserialize*("logfile.json"))

...

Next $\triangleq$
  $\lor \land$ *ExecLog*(*level*).*e* = "BecomeLeader"
    $\land$ *state*′ = *ExecLog*(*level*).*state*
    $\land$ *term*′ $\in 1 \,..\, ExecLog(level).idx$ log holes
  $\lor \land$ *ExecLog*(*level*).*e* = "Compact"
    $\land$ UNCHANGED *vars*

CCF $\triangleq$ INSTANCE *CCF* with *N* $\leftarrow$ ...

THEOREM *Init* $\land \Box[Next]_{vars} \implies$
  $\exists$ *votes* : *CCF*!*Init*
  $\land \Box[CCF!Next \lor CCF!BL \cdot CCF!TO]_{CCF!vars}$

- Spec-driven development & regression testing
- Example: EWD998

- Only safety!
- Embarrassingly parallel
- Spec coverage implies code coverage

- Spec as stimulus for the execution
- Conformance monitoring

Idea Trace Validation

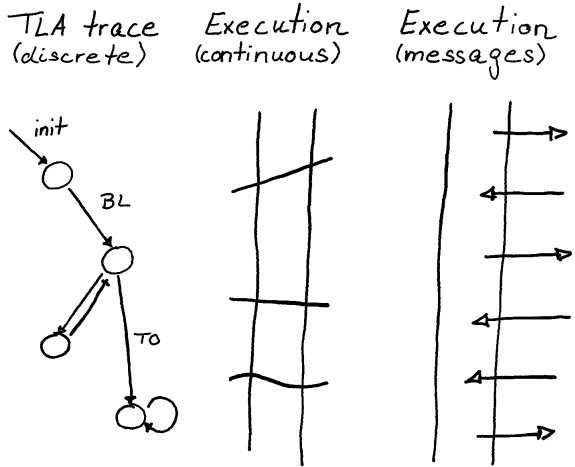


Figure: Conceptual idea TLA$^+$ trace validation [Pressler, 2018, 2020] previously explored by, e.g., Davis et al. [2020].

AI, FM & modeling?

- "*AI Advancements Mean Code Will Be Written in Natural English*" (Vice)
- "*In 2015, Michael Deardeuff of Amazon informed me that one sentence in [Paxos Made Simple] is ambiguous, and interpreting it the wrong way leads to an incorrect algorithm. Deardeuff found that a number of Paxos implementations on Github implemented this incorrect algorithm. Apparently, the implementors did not bother to read the precise description of the algorithm in [Part-Time Parliament]. I am not going to remove this ambiguity or reveal where it is. Prose is not the way to precisely describe algorithms. Do not try to implement the algorithm from this paper. Use [Lamport, 1998] instead.*"
  (`http://lamport.azurewebsites.net/pubs/pubs.html`)
- Generate program code from pseudocode/specs/math and validate & verify the AI-generated code against the high-level description

AI, FM & modeling?

- "*AI Advancements Mean Code Will Be Written in Natural English*" (Vice)
- "I*n 2015, Michael Deardeuff of Amazon informed me that one sentence in [Paxos Made Simple] is ambiguous, and interpreting it the wrong way leads to an incorrect algorithm. Deardeuff found that a number of Paxos implementations on Github implemented this incorrect algorithm. Apparently, the implementors did not bother to read the precise description of the algorithm in [Part-Time Parliament]. I am not going to remove this ambiguity or reveal where it is. Prose is not the way to precisely describe algorithms. Do not try to implement the algorithm from this paper. Use [Lamport, 1998] instead.*"
  (`http://lamport.azurewebsites.net/pubs/pubs.html`)
- Generate program code from pseudocode/specs/math and validate & verify the AI-generated code against the high-level description

AI, FM & modeling?

- "*AI Advancements Mean Code Will Be Written in Natural English*" (Vice)
- "I*n 2015, Michael Deardeuff of Amazon informed me that one sentence in [Paxos Made Simple] is ambiguous, and interpreting it the wrong way leads to an incorrect algorithm. Deardeuff found that a number of Paxos implementations on Github implemented this incorrect algorithm. Apparently, the implementors did not bother to read the precise description of the algorithm in [Part-Time Parliament]. I am not going to remove this ambiguity or reveal where it is. Prose is not the way to precisely describe algorithms. Do not try to implement the algorithm from this paper. Use [Lamport, 1998] instead.*"
  (`http://lamport.azurewebsites.net/pubs/pubs.html`)
- Generate program code from pseudocode/specs/math and validate & verify the AI-generated code against the high-level description

Q&A

Q&A

Bibliography I

Armin Biere, Cyrille Artho, and Viktor Schuppan. Liveness Checking as Safety Checking. In *In FMICS'02: Formal Methods for Industrial Critical Systems, Volume 66(2) of ENTCS*. Elsevier, 2002.

Ranadeep Biswas, Diptanshu Kakwani, Jyothi Vedurada, Constantin Enea, and Akash Lal. MonkeyDB: Effectively testing correctness under weak isolation levels. *Proceedings of the ACM on Programming Languages*, 5(OOPSLA):1–27, October 2021. ISSN 2475-1421. doi: 10.1145/3485546.

Kaustuv Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. Verifying Safety Properties With the TLA+ Proof System. *arXiv:1011.2560 [cs]*, 6173: 142–148, 2010. doi: 10.1007/978-3-642-14203-1_12.

Kaustuv C. Chaudhuri, Damien Doligez, Leslie Lamport, and Stephan Merz. A TLA+ Proof System. *arXiv:0811.1914 [cs]*, November 2008.

Bibliography II

E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. Symmetry Reductions in Model Checking. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Alan J. Hu, and Moshe Y. Vardi, editors, *Computer Aided Verification*, volume 1427, pages 147–158. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-64608-2 978-3-540-69339-0.

Denis Cousineau, Damien Doligez, Leslie Lamport, Stephan Merz, Daniel Ricketts, and Hernán Vanzetto. TLA+ Proofs. *arXiv:1208.5933 [cs]*, August 2012.

A. Jesse Jiryu Davis, Judah Schvimer, and Max Hirschhorn. eXtreme Modelling in Practice. *arXiv:2006.00915 [cs]*, May 2020. doi: 10.14778/3397230.3397233.

Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, C. R. Ramakrishnan, and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963, pages 337–340. Springer Berlin

Bibliography III

Heidelberg, Berlin, Heidelberg, 2008. ISBN 978-3-540-78799-0 978-3-540-78800-3. doi: 10.1007/978-3-540-78800-3_24.

Star Dorminey. Keyfabe: Checking C# against TLA+. 2019.

Mostafa Elhemali, Niall Gallagher, Nick Gordon, Joseph Idziorek, Richard Krog, Colin Lazier, Erben Mo, Akhilesh Mritunjai, Somasundaram Perianayagam, Tim Rath, Swami Sivasubramanian, James Christopher Sorenson III, Sroaj Sosothikul, Doug Terry, and Akshat Vig. Amazon DynamoDB: A scalable, predictably performant, and fully managed NoSQL database service. In *2022 USENIX Annual Technical Conference (USENIX ATC 22)*, pages 1037–1048, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-29-46.

A. Finn Hackett, Joshua Rowe, and Markus Alexander Kuppe. Understanding Inconsistency in Azure Cosmos DB with TLA+, October 2022.

Gerard J. Holzmann and Markus Alexander Kuppe. Scalability comparison of SPIN and TLC, 2017.

Bibliography IV

Rajeev Joshi, Leslie Lamport, John Matthews, Serdar Tasiran, Mark Tuttle, and Yuan Yu. Checking Cache-Coherence Protocols with TLA+. *Formal Methods in System Design*, 22(2):125–131, March 2003. ISSN 0925-9856, 1572-8102. doi: 10.1023/A:1022969405325.

Igor Konnov, Josef Widder, and Helmut Veith. Challenges in Model Checking of Fault-tolerant Designs in TLA+, 2015.

Leslie Lamport. The Temporal Logic of Actions. *ACM Transactions on Programming Languages and Systems*, 16(3):872–923, May 1994. ISSN 01640925. doi: 10.1145/177492.177726.

Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16 (2):133–169, May 1998. ISSN 0734-2071, 1557-7333. doi: 10.1145/279227.279229.

Leslie Lamport. The PlusCal Algorithm Language. In Martin Leucker and Carroll Morgan, editors, *Theoretical Aspects of Computing - ICTAC 2009*, volume 5684, pages 36–60. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-03465-7 978-3-642-03466-4.

Bibliography V

Leslie Lamport. Using TLC to Check Inductive Invariance, June 2018.

Zohar Manna and Amir Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, New York, 1995. ISBN 978-0-387-94459-3.

Stephan Merz. *Formal Specification and Verification*. Association for Computing Machinery, October 2019. ISBN 978-1-4503-7270-1. doi: $10.1145/3335772.3335780$.

Chris Newcombe. Why Amazon Chose TLA+. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Alfred Kobsa, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Demetri Terzopoulos, Doug Tygar, Gerhard Weikum, Yamine Ait Ameur, and Klaus-Dieter Schewe, editors, *Abstract State Machines, Alloy, B, TLA, VDM, and Z*, volume 8477, pages 25–39. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014. ISBN 978-3-662-43651-6 978-3-662-43652-3.

Bibliography VI

Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon Web Services Uses Formal Methods. *Communications of the ACM*, 58(4):66–73, March 2015. ISSN 00010782. doi: 10.1145/2699417.

Diego Ongardie. Raft consensus algorithm. https://github.com/ongardie/raft.tla, 2016.

Amir Pnueli. The temporal logic of programs. pages 46–57. IEEE, September 1977. doi: 10.1109/SFCS.1977.32.

Ron Pressler. Verifying Software Traces Against a Formal Specification with TLA+ and TLC, 2018.

Ron Pressler. Conjunction Capers: A TLA+ Truffle - Ron Pressler, September 2020.

Alex Shamis, Peter Pietzuch, Miguel Castro, Cédric Fournet, Edward Ashton, Amaury Chamayou, Sylvan Clebsch, Antoine Delignat-Lavaud, Matthew Kerner, Julien Maffre, Manuel Costa, and Mark Russinovich. IA-CCF: Individual Accountability for Permissioned Ledgers, March 2022.

Bibliography VII

Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972. ISSN 0097-5397, 1095-7111. doi: 10.1137/0201010.

Robert E. Tarjan. The concurrent algorithm, 2015.

Jaco van de Pol, Vincent Bloemen, and Alfons Laarman. Multi-core on-the-fly SCC decomposition. pages 1–12. ACM Press, 2016. ISBN 978-1-4503-4092-2. doi: 10.1145/2851141.2851161.

Eric Verhulst. *Formal Development of a Network-Centric RTOS: Software Engineering for Reliable Embedded Systems*. Springer, New York, 2011. ISBN 978-1-4419-9735-7.

Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model Checking TLA+ Specifications. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Laurence Pierre, and Thomas Kropf, editors, *Correct Hardware Design and Verification Methods*, volume 1703, pages 54–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. ISBN 978-3-540-66559-5 978-3-540-48153-9. doi: 10.1007/3-540-48153-2_6.

Table of Contents

Find Inductive Invariant Candidates

MODULE *Foo*

EXTENDS *Naturals*

VARIABLE x

$TypeOK \triangleq x \in \text{SUBSET } (1 \,..\, 500)$ or x ∈ Nat, ...
$H \triangleq \ldots$ "interesting part"
$Inv \triangleq TypeOK \wedge H$

$CheckInductiveSpec \triangleq Inv \wedge \Box[Next]_v$ Make Inv the initial predicate.

Find Inductive Invariant Candidates

MODULE *Foo*

EXTENDS *Integers*, *Randomization*

VARIABLE x

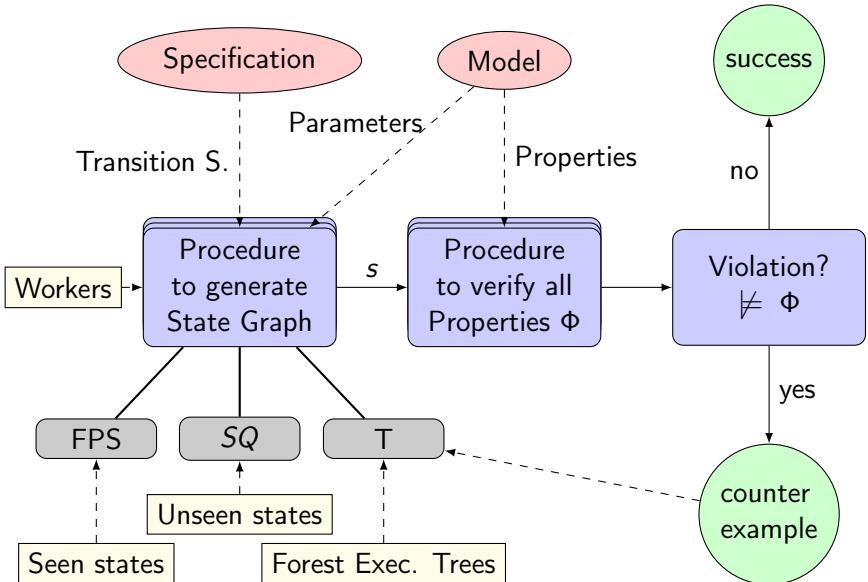
$TypeOK \triangleq x \in RandomSubset(4711, \text{SUBSET } (1 \,..\, 500))$
$H \triangleq \ldots$
$Inv \triangleq TypeOK \wedge H$

$CheckInductiveSpec \triangleq Inv \wedge \Box[\ldots]_{\ldots}$

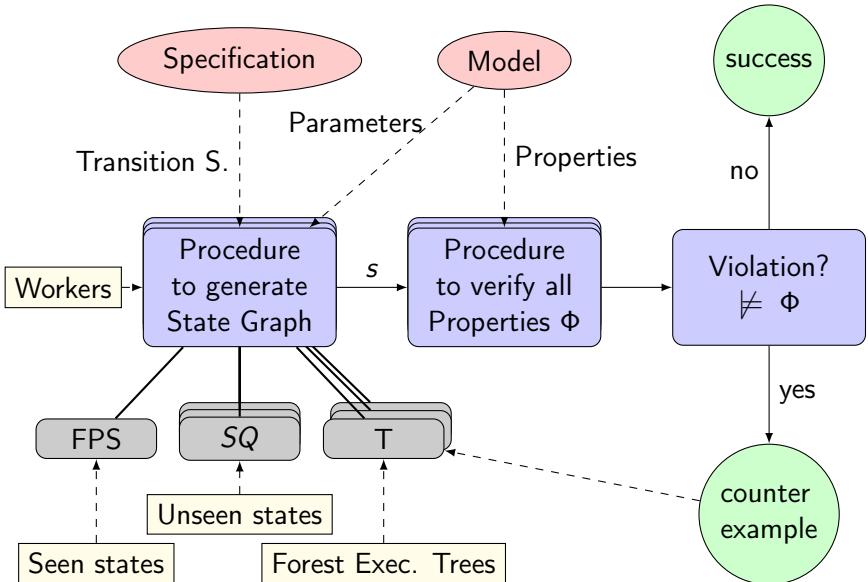
[Lamport, 2018]

Table of Contents

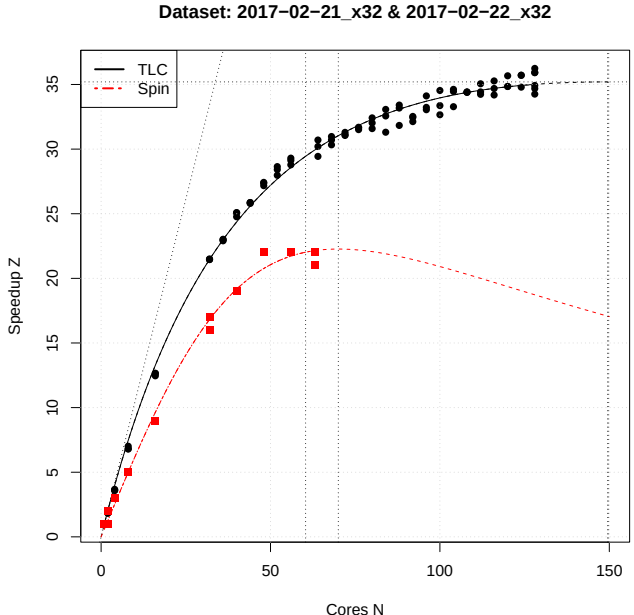
Schematic TLC



Schematic TLC



TLC: Vertical Scalability BFS

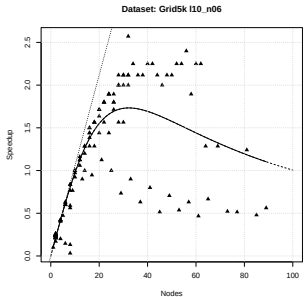


- TLC beats scalability of SPIN [Holzmann and Kuppe, 2017]

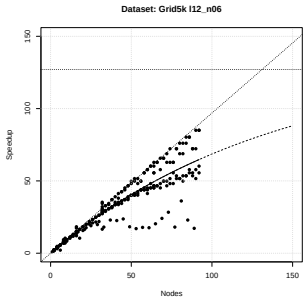
TLC: Horizontal Scalability BFS

- Executes TLC on network of machines
- Distributed Fingerprint Set (DHT)
  - Nearby memory faster than (local) disks
- Limitations
  - Master is bottleneck & SPOF
  - No liveness checking

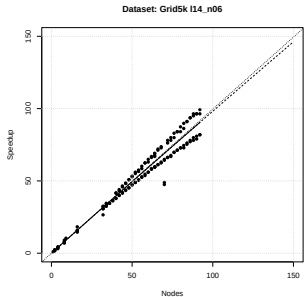
TLC: Horizontal Scalability BFS



(a) $Cost/State = 2^{10}$



(b) $Cost/State = 2^{12}$



(c) $Cost/State = 2^{14}$

Figure: Scalability distributed TLC

TLC: Liveness Checking

- Check Liveness: (Periodically) Find and check lassos for fulfilling cycles
- Strongly Connected Components with Tarjan [1972]
  - Approaches: [Tarjan, 2015, van de Pol et al., 2016]

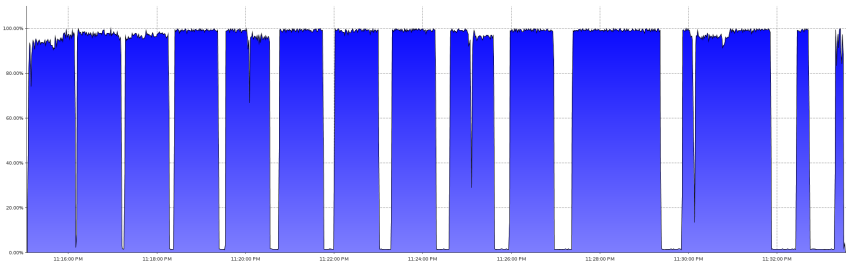


Figure: CPU usage with periodic liveness checking (32 core machine)

TLC: Performance Next-State

- Simple left-to-right evaluation (interpreter) of expressions
  - No intermediate language, no compiler
  - No partial evaluation
- $\Rightarrow$ Evaluation of next-state $> 100x$ slower compared to SPIN

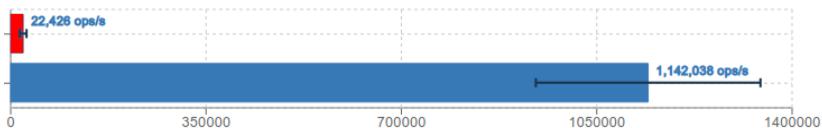


Figure: Throughput (ops/s) normal evaluation (red) vs. module overwrite (blue)

TLC: No Partial Evaluation

MODULE *Frob*

VARIABLES x, y

$Init \triangleq x = 0 \land y = 0$

$expensiveOp(n) \triangleq$ CHOOSE $e \in$ SUBSET $(1 \,..\, n)$: TRUE

$$NextOuch \triangleq \land x' \in 1 \,..\, 100 \\ \land y' = expensiveOp(23)$$

$$NextYeah \triangleq \land y' = expensiveOp(23) \\ \land x' \in 1 \,..\, 100$$

TLC: Symmetry Reduction

- Chooses a representative of equivalence classes (orbit) of states
- Constructive Orbit Problem - in general - is NP-hard [see Clarke et al., 1998]
- For each state enumerate $|vars| * |S|! * |T|!$ where S and T are two symmetry sets

MODULE *LiveSym*

CONSTANT S

VARIABLE x

$Spec \triangleq (x \in S) \wedge \Box[x' \in S]_x$

- Not supported by liveness checking

TLC: Liveness under Symmetry

- TLA$^+$ actions (labeled arcs) hard to account for in quotient graph
  - Abandoned approach resulted in incompleteness of liveness checking
- Idea: Use quotient graph to find SCCs, re-generate actual SCC for all elements of symmetry set
  - Inefficient if SCCs are large

TLC: Partial Order Reduction

- (Static) POR - similar to SPIN's implementation - explored by S. Merz
  - $\Rightarrow$ Didn't work too well
  - SPIN fine-grained atomicity similar to programming language
  - TLA$^+$ due to abstractions coarse-grained atomicity
    - Not looked at PlusCal (more fine-grained atomicity)
- Dynamic POR might be different (open question)

Statistical Properties

https://www.youtube.com/watch?v=cYenTPD7740