Let TLA$^+$ RiSE

Markus A. Kuppe

August 16, 2018

Outline

- TLA$^+$
  - TLA$^+$
  - Community
  - TLA Toolbox, Tools, and TLAPS
- Past, Present, Future Projects
  - Scalability
    - Vertical Scaling
    - Horizontal Scaling
  - Performance
  - State Space Reduction
    - Randomization
    - Symmetry Reduction
- Conclusion & Outlook

TLA$^+$

- High-level specification language
  - Design above the code level
- Distributed and concurrent systems

- $\Rightarrow$ Team wrote *two* RTOS [?]
  - (First version flew on the Rosetta spacecraft)
  - "*We witnessed first hand the brain washing done by years of C programming.*"
  - "*The TLA$^+$ abstraction helped a lot in coming to a much cleaner architecture.*"
  - "*One of the results was that the code size is about 10x less than the previous version.*"

TLA$^+$

- Untyped
- Zermelo Fraenkel set theory with Choice
- Linear-time framework: Temporal Logic of Actions (TLA)

MODULE M

VARIABLE v

$Init \triangleq \ldots$ Defines initial states

$Next \triangleq v' = v + 1 \land \ldots$ Constrains allowed transitions

$Spec \triangleq Init \land \Box[Next]_v$ Defines system executions

$\land F$ and optionally weak or strong Fairness

$Safety \triangleq \Box \ldots$

$Liveness \triangleq \Diamond\Box \ldots$

PlusCal

- Imperative-style pseudo-code but precise
- Atomicity via labels
- Can embed TLA$^+$
- Transpiles to TLA$^+$
  - $\Rightarrow$ Checkable with TLC
- "*A gateway drug for programmers*" (C. Newcombe)

--**algorithm** *Euclid* {
  variables $x = M$, $y = N$ **;** {
    while $(x \neq y)$ {
      if $(x < y)$ $\{y := y - x\}$
      else $\{x := x - y\}$
    }
  } Sequential algorithm needs no labels
}

Outline

- TLA$^+$
  - TLA$^+$
  - Community
  - TLA Toolbox, Tools, and TLAPS
- Past, Present, Future Projects
  - Scalability
    - Vertical Scaling
    - Horizontal Scaling
  - Performance
  - State Space Reduction
    - Randomization
    - Symmetry Reduction
- Conclusion & Outlook

Some Adopters

- Microsoft
- Amazon
- Google
- Intel
- Oracle
- Huawei
- ARM
- Mongo
- Thales
- ...

Community at large

- “tlaplus” Google Group with ~900 members (almost self-sustaining)
- Microsoft internal “TLA Plus” group with ~150 members
- GitHub (~500 stars) and ~10 contributors
- Twitter, Reddit, Youtube, ...

PGo

- PlusCal to Go transpiler (Beschastnikh et. al.)
- Very early stages
  - Single PlusCal Process
- `https://github.com/UBC-NSS/pgo`

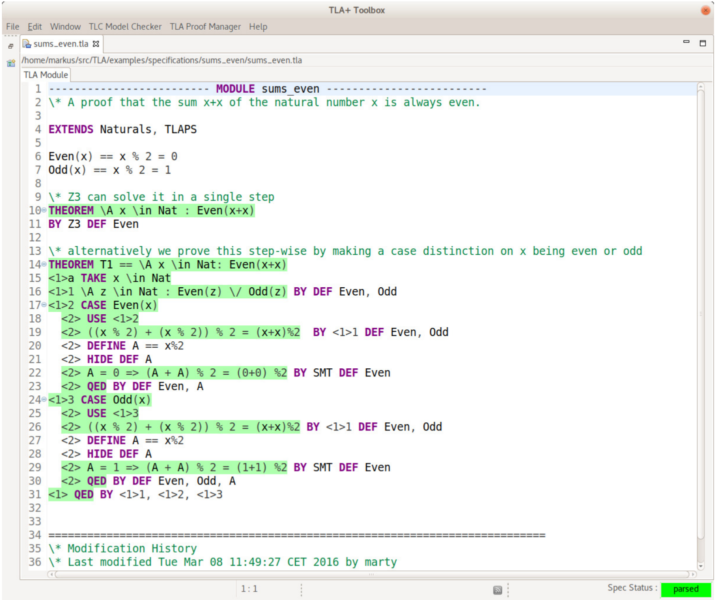
BMCMT: Bounded Model Checking of TLA$^+$ with SMT

- Abstraction-based parameterized TLA+ checker
  - Uses Z3
- Challenge: “Rich language. Specifications in TLA$^+$ are considerably more expressive than standard software: TLA$^+$ is untyped, it allows quantification over sets, comparison of cardinalities, and comparison and updates of the states of concurrent components.” [?]
- Recording of TLA+ community event talk:
  - TLC faster for small models (especially when bound unknown)

Outline

- TLA^+
  - TLA^+
  - Community
  - TLA Toolbox, Tools, and TLAPS
- Past, Present, Future Projects
  - Scalability
    - Vertical Scaling
    - Horizontal Scaling
  - Performance
  - State Space Reduction
    - Randomization
    - Symmetry Reduction
- Conclusion & Outlook

TLA Toolbox, Tools, and TLAPS



TLA+ Toolbox

File Edit Window TLC Model Checker TLA Proof Manager Help

sums_even.tla

/home/markus/src/TLA/examples/specifications/sums_even/sums_even.tla

TLA Module

```
------------------------- MODULE sums_even -------------------------
\* A proof that the sum x+x of the natural number x is always even.

EXTENDS Naturals, TLAPS

Even(x) == x % 2 = 0
Odd(x) == x % 2 = 1

\* Z3 can solve it in a single step
THEOREM \A x \in Nat : Even(x+x)
BY Z3 DEF Even

\* alternatively we prove this step-wise by making a case distinction on x being even or odd
THEOREM T1 == \A x \in Nat: Even(x+x)
<1>a TAKE x \in Nat
<1>1 \A z \in Nat : Even(z) \/ Odd(z) BY DEF Even, Odd
<1>2 CASE Even(x)
  <2> USE <1>2
  <2> ((x % 2) + (x % 2)) % 2 = (x+x)%2  BY <1>1 DEF Even, Odd
  <2> DEFINE A == x%2
  <2> HIDE DEF A
  <2> A = 0 => (A + A) % 2 = (0+0) %2 BY SMT DEF Even
  <2> QED BY DEF Even, A
<1>3 CASE Odd(x)
  <2> USE <1>3
  <2> ((x % 2) + (x % 2)) % 2 = (x+x)%2 BY <1>1 DEF Even, Odd
  <2> DEFINE A == x%2
  <2> HIDE DEF A
  <2> A = 1 => (A + A) % 2 = (1+1) %2 BY SMT DEF Even
  <2> QED BY DEF Even, Odd, A
<1> QED BY <1>1, <1>2, <1>3


=====================================================================
\* Modification History
\* Last modified Tue Mar 08 11:49:27 CET 2016 by marty
```

1:1

Spec Status : parsed

Figure: TLAPS

TLA Toolbox, Tools, and TLAPS

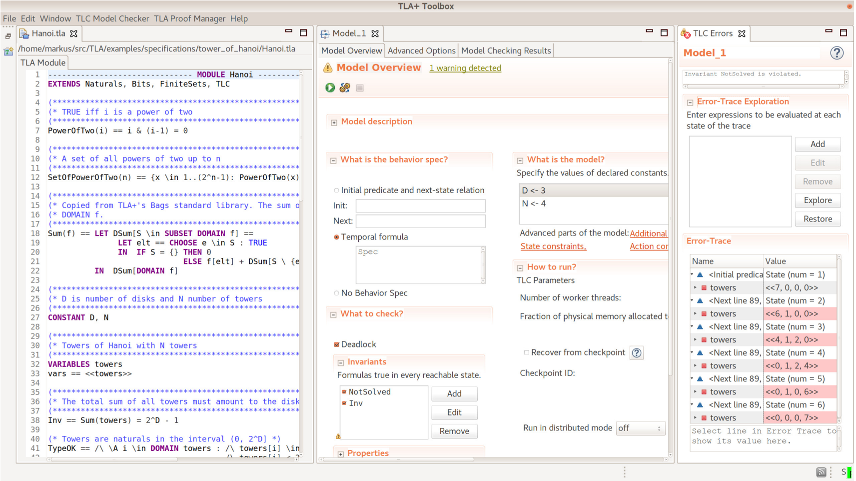


Figure: Toolbox Model Checking

TLC

- Explicit-state model checker for TLA$^+$
- Disk-based (but you don't want it to go to disk)
- Handles a subclass of TLA$^+$ that seems to be useful in practice
  - E.g. no Temporal Existential Quantification, Composition of Actions, ...

TLC

- Safety checking corresponds to Breadth-First search over on-the-fly generated state graph
  - Fingerprints $\sim 2^{64}$ (long)
- Liveness checking corresponds to Depth-First search over (partial) behavior graph [?]
  - Behavior graph is state graph x tableaux
  - Technically limited to $\sim 2^{32}$ vertices

Outline

TLA$^+$

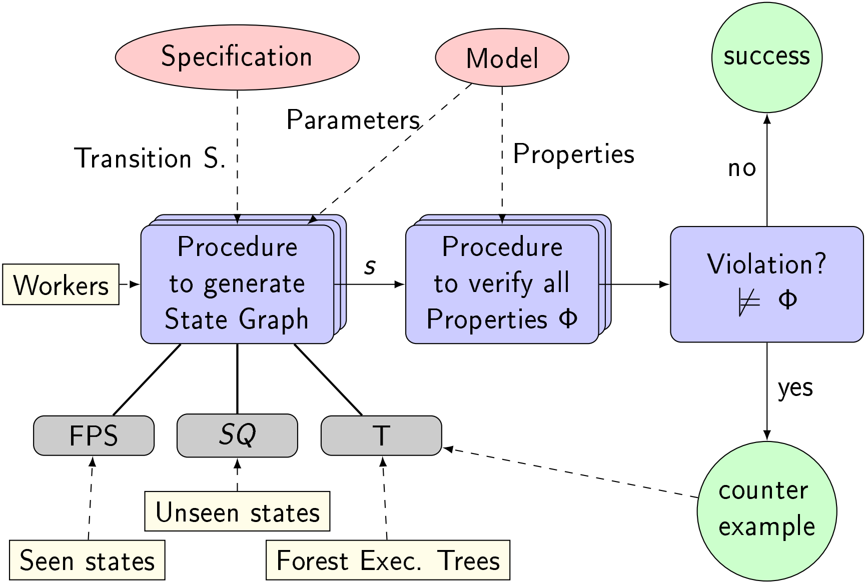
- TLA$^+$
- Community
- TLA Toolbox, Tools, and TLAPS

Past, Present, Future Projects

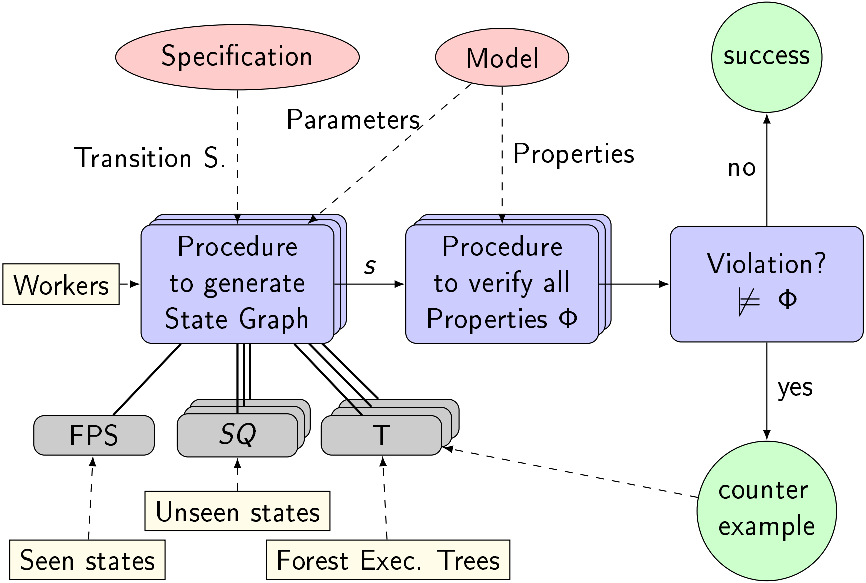
- Scalability
  - Vertical Scaling
  - Horizontal Scaling
- Performance
- State Space Reduction
  - Randomization
  - Symmetry Reduction

Conclusion & Outlook

Schematic TLC



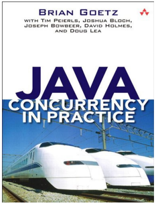
Schematic TLC



Lock-Striping FPS

1. A global lock for *FPS* to guard **concurrent** find-or-puts does not scale due to *lock* ***contention***
2. $\Rightarrow$ Partition *FPS* and use one lock per partition

"[...] *lock striping seems much more promising because the size of the stripe set can be increased as processor counts increase.*"



Lock-Striping FPS

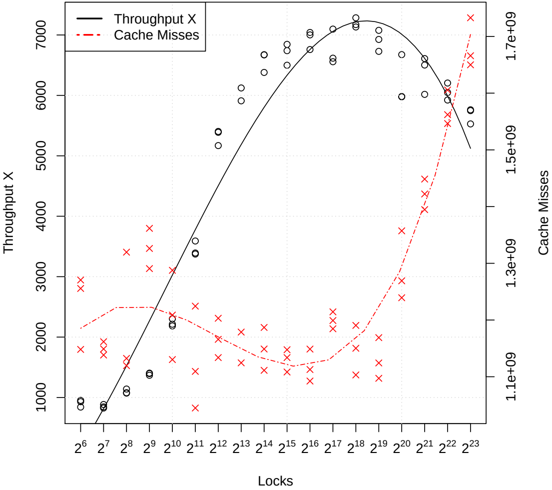


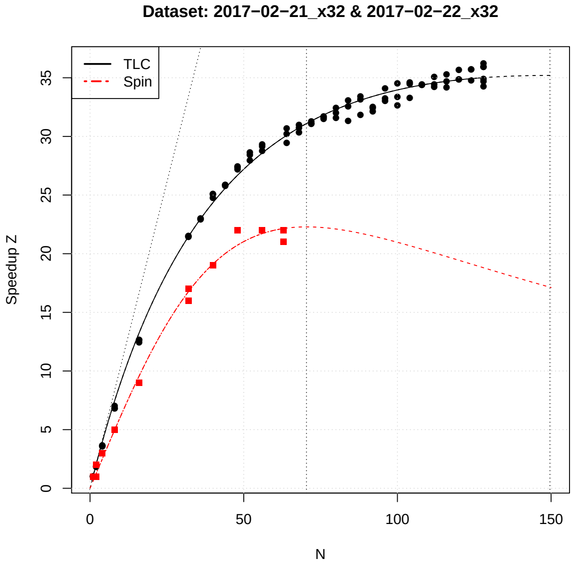
Figure: Lock striping exhibits lock *coherence*

Lock-Free & Shared-Nothing

Minimize worker contention via:

- Lock-Free (CAS) Partitioned/Sharded FPS
  - Parallel adaptive sort & parallel eviction to disk
  - Raw memory to avoid GC
- Shared-Nothing Trace T per worker
- (Shared-Nothing State Queue SQ)
  - Overly optimistic assumptions about average shape/properties of state graphs!

Lock-Free & Shared-Nothing



- TLC beats scalability of SPIN
  - (Bakery spec)
- SPIN outperforms throughput of TLC

Liveness Checking

- Check Liveness: Find and check lassos for fulfilling cycles
- Strongly Connected Components with **?**
  - DFS, linear time, implemented iteratively
- Liveness checking runs periodically (stops safety)

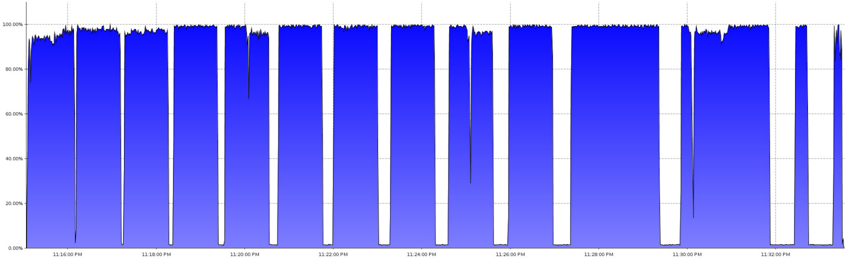


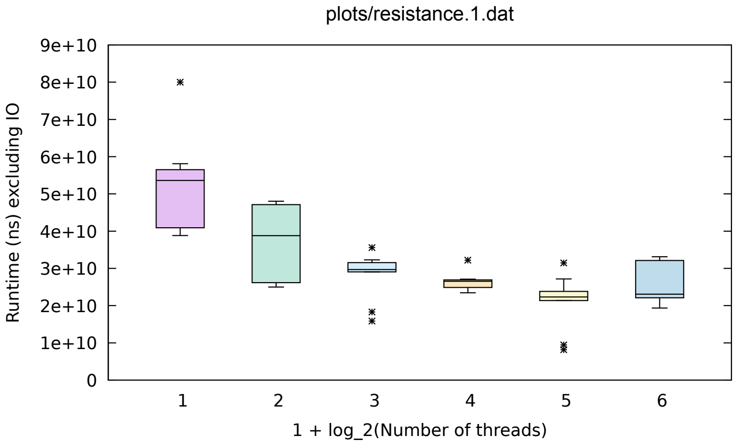
Figure: CPU usage with periodic liveness checking (32 core machine)

Concurrent SCC

- R.E. Tarjan drafted a concurrent algorithm for us
  - Scalability of prototype not promising, abandoned idea for lower hanging fruits
- "Multi-Core on-the-Fly SCC Decomposition" [?]
  - GSoC student Parv Mor implemented prototype this summer
  - Results look more promising

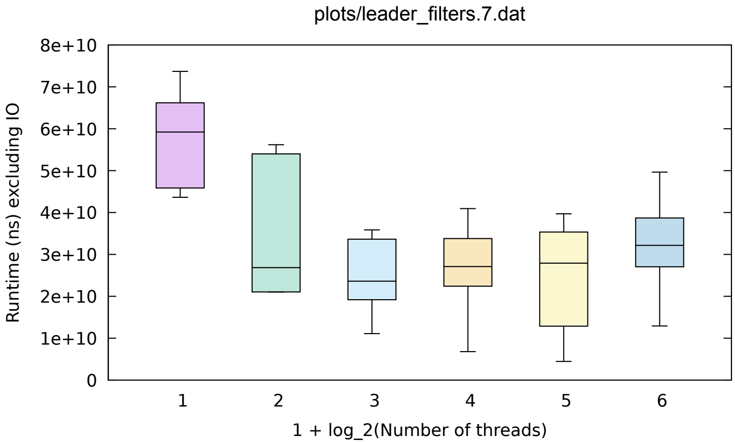
  - Contention/Coherence Union find data-structure?!
  - "A Randomized Concurrent Algorithm for Disjoint Set Union" [?]

Concurrent SCC



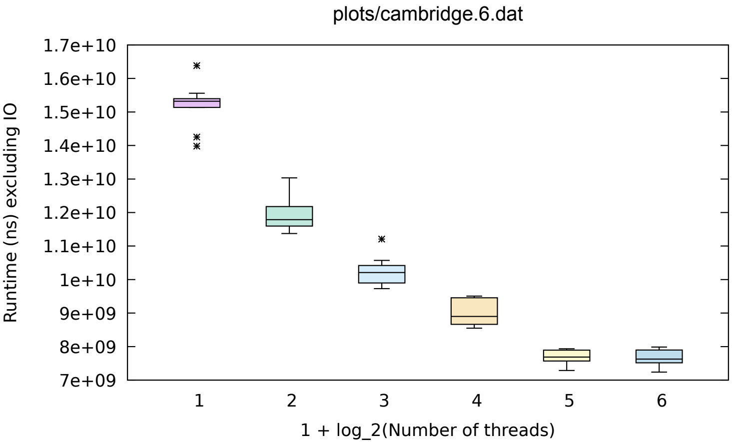
Vertices: 13.8M, Arcs: 32.1M, SCCs: 3 (max 13.8M), Diameter: 60391

Concurrent SCC



Vertices: 26.3M, Arcs: 91.7M, SCCs: 26.3M (max 1), Diameter: 71

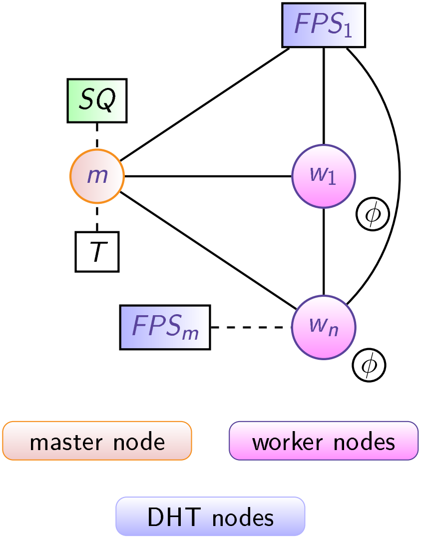
Concurrent SCC



Vertices: 3.4M, Arcs: 9.5M, SCCs: 8413 (max 3.3M), Diameter: 418

Distributed TLC

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



Distributed TLC

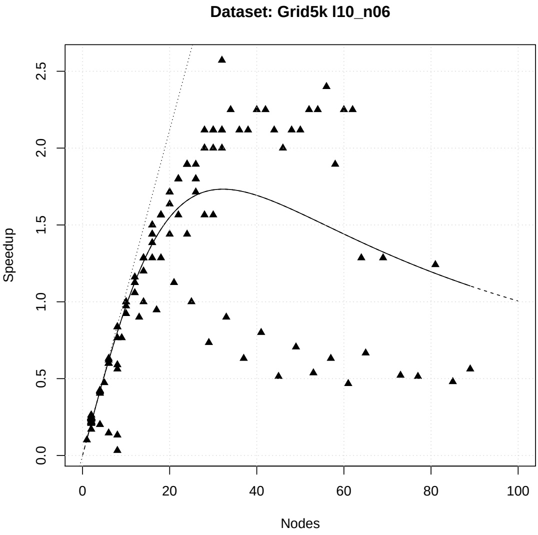


Figure: Scalability distributed TLC: $Cost/State = 2^{10}$

Distributed TLC

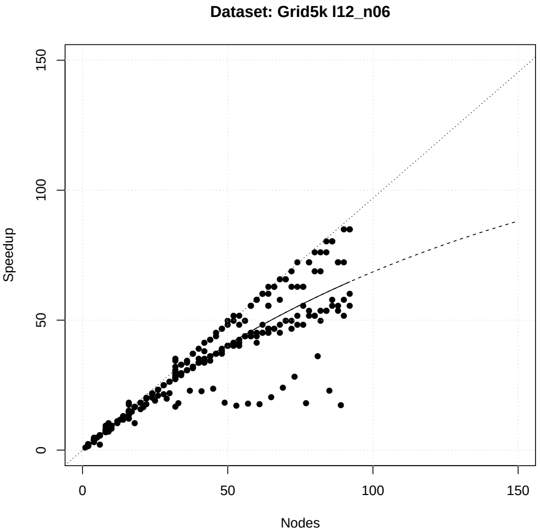


Figure: Scalability distributed TLC: $Cost/State = 2^{12}$

Distributed TLC

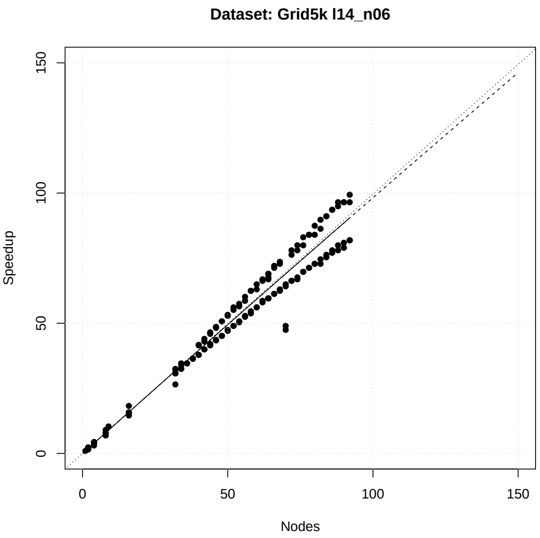


Figure: Scalability distributed TLC: $Cost/State = 2^{14}$

Outline

- TLA^+
  - TLA^+
  - Community
  - TLA Toolbox, Tools, and TLAPS
- Past, Present, Future Projects
  - Scalability
    - Vertical Scaling
    - Horizontal Scaling
  - Performance
  - State Space Reduction
    - Randomization
    - Symmetry Reduction
- Conclusion & Outlook

Cloud TLC

- Push-Button model checking in the cloud
  - Support for Azure and AWS
    - Just compute APIs for portability reasons
  - Hide away idiosyncrasies of TLC and cloud platform
- Support for single node TLC and Distributed TLC
- Can be started from Toolbox and CLI within seconds
  - Cold-start in the range of minutes
- Easily check several models concurrently
  - Instance count is elastic with regards to resource demand
- Instances dispose automatically after inactivity

Outline

TLA$^+$

- TLA$^+$
- Community
- TLA Toolbox, Tools, and TLAPS

Past, Present, Future Projects

- Scalability
  - Vertical Scaling
  - Horizontal Scaling
- Performance
- State Space Reduction
  - Randomization
  - Symmetry Reduction

Conclusion & Outlook

Performance Next-State

- No intermediate language, no compiler, just AST interpreter
- Simple left-to-right evaluation of expressions
  - Recursion but no tail call optimization in Java
- $\Rightarrow$ Evaluation of next-state at least two orders magnitude slower compared to SPIN

/home/markus/dump/syncthing/work/LabInt

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

(see online)

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

$$NextOuch \triangleq \begin{array}{l} \land\ x' \in 1 \,..\, 100 \\ \land\ y' = expensiveOp(23) \end{array}$$

$$NextYeah \triangleq \begin{array}{l} \land\ y' = expensiveOp(23) \\ \land\ x' \in 1 \,..\, 100 \end{array}$$

TLA$^+$ Compiler

- "The Truffle language development framework allows running programming languages efficiently on GraalVM."[1]
- "The guest language developer gets a high-performance language implementation, but does not need to be a compiler expert." [?]
  - Speedup of evaluation at runtime over special-purpose compilers:
    - Ruby 3.8x
    - R 5x
- Translate AST emitted by SANY to Truffle AST
- $\Rightarrow$ Partial Evaluation for TLA$^+$

[1] GraalVM is a just-in-time compiler for OpenJDK.

Outline

TLA$^+$
- TLA$^+$
- Community
- TLA Toolbox, Tools, and TLAPS

Past, Present, Future Projects
- Scalability
  - Vertical Scaling
  - Horizontal Scaling
- Performance
- State Space Reduction
  - Randomization
  - Symmetry Reduction

Conclusion & Outlook

Find Inductive Invariant candidates with TLC

Goal: Proof invariance of I with TLAPS
Find inductive invariant Inv that satisfies:

1. $Init \Rightarrow Inv$, which means that Inv is true in all initial states.
2. $Inv \Rightarrow I$, which means that I is true in every state on which Inv is true.
3. $Inv \land Next \Rightarrow Inv'$, which means if Inv is true on any state s, then it is true on any state reachable from s by a $Next$ step.
    3.1 Let TLC check:
    $CheckInductiveSpec \triangleq Inv \land \square[Next]_{vars}$

[?]

Find Inductive Invariant candidates with TLC

MODULE *Foo*

EXTENDS *Naturals*

VARIABLE x

$TypeOK \triangleq x \in \text{SUBSET}\ (1 \,..\, 500)$ or x $\in$ Nat, ...

$H \triangleq \ldots$ "interesting part"

$Inv \triangleq TypeOK \land H$

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

New Standard Module Randomization

MODULE *Randomization*

RandomSubset(k, S) equals a randomly chosen subset of S containing k elements, where 0 < k < Cardinality(S).

$RandomSubset(k, S) \triangleq \text{CHOOSE } T \in \text{SUBSET } S : Cardinality(T) = T$

RandomSetOfSubsets(k, n, S) equals a pseudo-randomly chosen set of subsets of S – that is, a randomly chosen subset of SUBSET S. Thus, each element T of this set is a subset of S. Each such T is chosen so that each element of S has a probability n / Cardinality(S) of being in T. Thus, the average number of elements in each chosen subset T is n. The set RandomSetOfSubsets(k, n, S) is obtained by making k such choices of T. Because this can produce duplicate choices, the number of elements T in this set may be less than k.

$RandomSetOfSubsets(k, n, S) \triangleq$
$\quad \text{CHOOSE } T \in \text{SUBSET SUBSET } S : Cardinality(T) \leq k$

New Standard Module Randomization

MODULE *Foo*

EXTENDS *Integers*, *Randomization*

VARIABLE x

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

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

Symmetry Reduction

- Chooses a representative of equivalence classes (orbit) of states
- Constructive Orbit Problem - in general - is NP-hard [see ?]

MODULE *Symmetry*

EXTENDS *FiniteSets*
VARIABLE x
CONSTANT S
ASSUME $(Cardinality(S) \geq 9)$
$Spec \triangleq (x \in S) \land \Box[x' \in S]_{\langle x \rangle}$ Without symmetry: 9 states, without: 1

- For each state enumerate $|vars| * |A|! * |B|!$ where A and B are two symmetry sets
- Not supported by liveness checking (TLC prints warning)

Liveness under Symmetry

- TLA$^+$ actions (labeled arcs) hard to account for in quotient graph
  - Approach resulted in incompleteness of liveness checking
- $\Rightarrow$ Abandoned idea
- Maybe: Use quotient graph to find SCCs, re-generate actual SCC for all elements of symmetry set
  - Inefficient if SCCs are large (which they tend to be)

Partial Order Reduction for TLA$^+$?

- (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 (fine-grained atomicity)
- Dynamic POR might be different (open question)

Conclusion & Outlook

- Continue to focus on scalability of parallel and distributed TLC
  - Concurrent SCC search with lock-free union find
  - Scalability StateQueue
- "TLA+ compiler" to speed-up evaluation of next-state relation
- Shift Toolbox maintenance to community
- Machine Learning combined with Cloud TLC
  - Optimize scalability and performance => Less manual tuning of TLC
  - Predict size of state graph/time to check => User defines "when"
- Start new with TLC-Next instead of continue with existing TLC
  - Feature cost and technical debt to drag on
- OTS data-structures not (yet?) ready for multicore "revolution"
- Scalability & Performance too much of an art

Q&A

Q&A

Bibliography I