Making TLA^{+} Model Checking Symbolic

Igor Konnov

Joining Interchain Foundation in August

Jure Kukovec

Thanh-Hai Tran

VeriDis + Matryoushka seminar, Amsterdam, June 2019

Loria Laboratoire lorrain de recherche en informatique et ses applications

Inria informatics mathematics

logics

WWTF VIENNA SCIENCE AND TECHNOLOGY FUND

Why TLA$^+$?

Rich specification language

TLA$^+$ is used in industry, e.g., amazon

TLA$^+$ tools maintained at Inria and Microsoft Research

- an interactive proof system (TLAPS)
- a model checker (TLC), state enumeration

Raft

Paxos (Synod), Egalitarian Paxos, Flexible Paxos

Apache Kafka

several bugs found

TLA$^+$

First-order logic with sets (ZFC)

Rich expression syntax:

- operations on sets, functions, tuples, records, sequences

Temporal operators:

- $\Box$ (always), $\Diamond$ (eventually), $\leadsto$ (leads-to), no *Nexttime*

Practice: safety properties, $\Box$*Invariant*

APALACHE-MC 0.5.0

github.com/konnov/apalache

Symbolic model checker that works under the assumptions of TLC:

Fixed and finite constants (parameters)

Finite sets, function domains and **co-domains**

TLC's restrictions on formula structure

Bounded model checking to check safety

As few language restrictions as possible

Technically,

Quantifier-free formulas in SMT: `QF_UFNIA`

Unfolding quantified expressions: $\forall x \in S : P$ as $\bigwedge_{c \in S} P[c/x]$

APALACHE-MC 0.5.0

github.com/konnov/apalache

Symbolic model checker that works under the assumptions of TLC:

- **Fixed and finite constants** (parameters)
- **Finite sets, function domains** and **co-domains**
- TLC's restrictions on formula structure
- Bounded model checking to check safety
- **As few language restrictions as possible**

Technically,

- Quantifier-free formulas in SMT: `QF_UFNIA`
- Unfolding quantified expressions: $\forall x \in S : \; P$ as $\bigwedge_{c \in S} P[c/x]$

an example

A service for reliable broadcast

one process broadcasts a message **bcast**

unforgeability: if no correct process received **bcast**, then no correct process ever accepts **bcast** $000\dots0$

---

correctness: if all correct processes received **bcast**, then some correct process eventually accepts **bcast** $111\dots1$

relay: if a correct process accepts **bcast**, then all correct processes eventually accept **bcast** $011\dots1$

Reliable broadcast by Srikanth & Toueg 87

```
local  $myval_i \in \{0, 1\}$        -- did process $i$ receive bcast?

while true do
 if  $myval_i = 1$ and not sent ECHO before
 then send ECHO to all

 if  received ECHO from at least n-2t    distinct processes
    and not sent ECHO before
 then send ECHO to all

 if  received ECHO from at least n - t    distinct processes
 then accept
od
```

resilience: of $n > 3t$ processes, $f \leq t$ processes are Byzantine

How to check its properties?

I read that paper about **Byzantine Model Checker**

Model the algorithm as a threshold automaton

Verify safety and liveness for all $n, t, f : n > 3t \land t \geq f \geq 0$

I have heard this talk by Leslie Lamport

Let's write it in TLA$^+$

Run the **TLC model checker** for fixed parameters

Declaration and initialization

EXTENDS *Integers*, *FiniteSets*

$N \triangleq 12 \qquad T \triangleq 3 \qquad F \triangleq 3$

$Corr \triangleq 1 \,..\, (N - F - 1) \qquad Faulty \triangleq (N - F) \,..\, N$

VARIABLES *pc*, *rcvd*, *sent*

$$
\begin{aligned}
Init \triangleq\ & \land\ pc \in [Corr \to \{\text{“}V0\text{“}, \text{“}V1\text{“}\}] && \textit{some processes receive the broadcast} \\
& \land\ sent = \{\} && \textit{no messages sent initially} \\
& \land\ rcvd \in [Corr \to \{\}] && \textit{no messages received initially}
\end{aligned}
$$

Transition relation

$$
\begin{array}{l}
\mathit{Next} \triangleq \\
\quad \exists p \in \mathit{Corr} : \\
\qquad \land\ \mathit{Receive}(p) \\
\qquad \land\ \lor\ \mathit{UponV1}(p) \\
\qquad \quad\ \ \lor\ \mathit{UponNonFaulty}(p) \\
\qquad \quad\ \ \lor\ \mathit{UponAccept}(p) \\
\qquad \quad\ \ \lor\ \text{UNCHANGED}\ \langle \mathit{pc}, \mathit{sent} \rangle
\end{array}
$$

$$
\begin{array}{l}
\mathit{Receive}(p) \triangleq \\
\quad \exists \mathit{newMessages} \in \text{SUBSET}(\mathit{sent} \cup \mathit{Faulty}) : \\
\qquad \mathit{rcvd}' = [\mathit{rcvd}\ \text{EXCEPT}\ ![\mathit{self}] = \mathit{rcvd}[p] \cup \mathit{newMessages}]
\end{array}
$$

Actions

$$\mathit{UponV1}(p) \triangleq$$

$$\land\ pc[p] = \text{"V1"}$$

$$\land\ pc' = [pc \ \text{EXCEPT}\ ![p] = \text{"SE"}] \quad \land \quad sent' = sent \cup \{p\}$$

$$\mathit{UponNonFaulty}(p) \triangleq$$

$$\land\ pc[p] \in \{\text{"V0"}, \text{"V1"}\} \quad \land \quad \mathit{Cardinality}(rcvd'[p]) >= N - 2 * T$$

$$\land\ pc' = [pc \ \text{EXCEPT}\ ![p] = \text{"SE"}] \quad \land \quad sent' = sent \cup \{p\}$$

$$\mathit{UponAccept}(p) \triangleq$$

$$\land\ pc[p] \in \{\text{"V0"}, \text{"V1"}, \text{"SE"}\} \quad \land \quad \mathit{Cardinality}(rcvd'[p]) >= N - T$$

$$\land\ pc' = [pc \ \text{EXCEPT}\ ![p] = \text{"AC"}]$$

$$\land\ sent' = sent \cup (\text{IF}\ pc[p] \neq \text{"SE"}\ \text{THEN}\ \{p\}\ \text{ELSE}\ \{\})$$

Safety?

unforgeability: if no correct process received **bcast**, then no correct process ever accepts **bcast** $000\ldots0$

* *a non-inductive invariant*

$Unforg \triangleq \forall p \in Corr : pc[p] \neq \text{“}AC\text{”}$

* *restricted initial states*

$InitNoBcast \triangleq Init \land pc \in [Corr \rightarrow \{\text{“}V0\text{”}\}]$

Check that every state reachable from *InitNoBcast* satisfies *Unforg*

Breaking unforgeability

12 processes, 4 faults $n = 3f$

APALACHE-MC: a counterexample in **5 minutes**

- 12K SMT constants, 34K SMT assertions depth 6

TLC: a counterexample after **2 hrs 21 min**

- 600M states depth 6

how does it work?

What is hard about TLA$^+$?

Rich data

sets of sets, functions, records, tuples, sequences

No types

TLA$^+$ is not a programming language

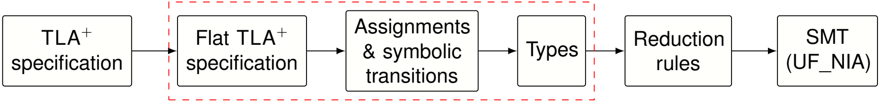
No imperative statements like assignments

TLA$^+$ is not a programming language

No standard control flow

TLA$^+$ is not a programming language

Essential steps



Extracting assignments and symbolic transitions

similar to TLC — treat some $x' \in \{\dots\}$ as assignments

Simple type inference

propagate types at every step — $x : \text{Int}$ gives us $\{x\} : \text{Set[Int]}$

Bounded model checking

overapproximate the contents of data structures

assignments & symbolic transitions

Symbolic transitions

[Kukovec, K., Tran, ABZ'18]

$$
\begin{aligned}
Next \triangleq\ & \exists p \in Corr : \\
& \land\ Receive(p) \\
& \land\ \lor\ UponV1(p) \\
& \quad\ \lor\ UponNonFaulty(p) \\
& \quad\ \lor\ UponAccept(p) \\
& \quad\ \lor\ \text{UNCHANGED}\ \langle pc, sent \rangle
\end{aligned}
$$

Automatically partitioning *Next* into four transitions:

$$
\begin{aligned}
& \exists p \in Corr : \\
& \quad \land\ Receive(p) \\
& \quad \land\ UponV1(p)
\end{aligned}
$$

$$
\begin{aligned}
& \exists p \in Corr : \\
& \quad \land\ Receive(p) \\
& \quad \land\ UponNonFaulty(p)
\end{aligned}
$$

$$
\begin{aligned}
& \exists p \in Corr : \\
& \quad \land\ Receive(p) \\
& \quad \land\ UponAccept(p)
\end{aligned}
$$

$$
\begin{aligned}
& \exists p \in Corr : \\
& \quad \land\ Receive(p) \\
& \quad \land\ \text{UNCHANGED}\ \langle pc, sent \rangle
\end{aligned}
$$

Symbolic transitions

[Kukovec, K., Tran, ABZ'18]

$$
\begin{aligned}
\mathit{Next} \triangleq\ & \exists p \in \mathit{Corr} : \\
& \land\ \mathit{Receive}(p) \\
& \land\ \lor\ \mathit{UponV1}(p) \\
& \quad\ \lor\ \mathit{UponNonFaulty}(p) \\
& \quad\ \lor\ \mathit{UponAccept}(p) \\
& \quad\ \lor\ \text{UNCHANGED}\ \langle \mathit{pc}, \mathit{sent} \rangle
\end{aligned}
$$

Automatically partitioning *Next* into four transitions:

$$
\begin{aligned}
& \exists p \in \mathit{Corr} : \\
& \quad \land\ \mathit{Receive}(p) \\
& \quad \land\ \mathit{UponV1}(p)
\end{aligned}
$$

$$
\begin{aligned}
& \exists p \in \mathit{Corr} : \\
& \quad \land\ \mathit{Receive}(p) \\
& \quad \land\ \mathit{UponNonFaulty}(p)
\end{aligned}
$$

$$
\begin{aligned}
& \exists p \in \mathit{Corr} : \\
& \quad \land\ \mathit{Receive}(p) \\
& \quad \land\ \mathit{UponAccept}(p)
\end{aligned}
$$

$$
\begin{aligned}
& \exists p \in \mathit{Corr} : \\
& \quad \land\ \mathit{Receive}(p) \\
& \quad \land\ \text{UNCHANGED}\ \langle \mathit{pc}, \mathit{sent} \rangle
\end{aligned}
$$

How does TLC find assignments?

TLC detects assignments as it explores a formula:

- from left to right:

$$x' = 1 \land x' \in \{1, 2, 3\}$$

- treating action-level disjunctions as non-deterministic choice

$$\big(x' = 1 \lor x' = 2\big) \land x' \geq 2$$

- expecting the same kind of assignments on all branches

$$(x' = 1 \land y' = 2) \lor x' = 3$$

Finding symbolic assignments (with SMT)

Looking for assignment strategies that:

- cover every Boolean branch
- have exactly one assignment per variable per branch
- do not contain cyclic assignments

$$\big((\underline{y' = x'} \land x' \in \{2, 3, y'\}) \lor (x' = 2 \land \underline{y' \in \{x'\}})\big) \land \underline{x' = 3}$$

Sometimes, we do better than TLC (above)

Sometimes, worse, e.g., when $x = 0$:

$$x > 0 \lor (x' = x + 1 \lor y' = x - 1)$$

Definitions and the framework in: [Kukovec, K., Tran, ABZ'18]

Finding symbolic assignments (with SMT)

Looking for assignment strategies that:

- cover every Boolean branch
- have exactly one assignment per variable per branch
- do not contain cyclic assignments

$$\big((\underline{y' = x'} \land x' \in \{2, 3, y'\}) \lor (x' = 2 \land \underline{y' \in \{x'\}})\big) \land \underline{x' = 3}$$

Sometimes, we do better than TLC (above)

Sometimes, worse, e.g., when $x = 0$:

$$x > 0 \lor (x' = x + 1 \lor y' = x - 1)$$

Definitions and the framework in: [Kukovec, K., Tran, ABZ'18]

Simple types

Types: scalars and functions

Basic:

constants: *Const*	“a”, “hello”
integers: *Int*	-1, 1024
Booleans: *Bool*	FALSE, TRUE

Finite sets:

$\mathit{Set}[\tau]$	$\mathit{Set}[\mathit{Set}[\mathit{Int}]]$

Function-like:

functions: $\tau_1 \to \tau_2$	$\mathit{Int} \to \mathit{Bool}$
tuples: $\tau_1 \times \cdots \times \tau_n$	$\mathit{Int} \times \mathit{Bool} \times (\mathit{Int} \to \mathit{Int})$
records: $[\mathit{Const} \mapsto \tau_1, \dots, \mathit{Const} \mapsto \tau_n]$	$[\text{“a”} \mapsto \mathit{Int}, \text{“b”} \mapsto \mathit{Bool}]$
sequences: $\mathit{Seq}(\tau)$	$\mathit{Seq}[\mathit{Int}]$

Simple type inference

Knowing the types at the current state

Compute the types of the expressions and of the primed variables

if X has type $\mathit{Set}[\mathit{Int}]$

- $X' \in [X \to X]$ has type $\mathit{Int} \to \mathit{Int}$
- y in $\{y \in X : y > 0\}$ has type Int

$\{\}$ and $\langle\rangle$ are polymorphic constructors for sets and sequences

- hence, we ask the user to specify the type, e.g., $\{\} <: \{\mathit{Int}\}$
- records also require type annotations

Bounded model checking

Old recipe for bounded symbolic computations

Two symbolic transitions that assign values to x

$$\mathit{Next} \triangleq A \lor B$$

Translate TLA$^+$ expressions to SMT with some $\llbracket \cdot \rrbracket$

state 0		state 1		state 2		…
$\llbracket \mathit{Init} \rrbracket$	$x \mapsto i_0$	$\llbracket A[i_0/x] \rrbracket$	$x' \mapsto a_1$	$\llbracket A[c_1/x] \rrbracket$	$x' \mapsto a_2$	
		$\llbracket B[i_0/x] \rrbracket$	$x' \mapsto b_1$	$\llbracket B[c_1/x] \rrbracket$	$x' \mapsto b_2$	…
		$\llbracket x' \in \{a_1, b_1\} \rrbracket$	$x' \mapsto c_1$	$\llbracket x' \in \{a_2, b_2\} \rrbracket$	$x' \mapsto c_2$	

What is $[\![\cdot]\!]$?

Our idea

Mimic the semantics implemented by TLC

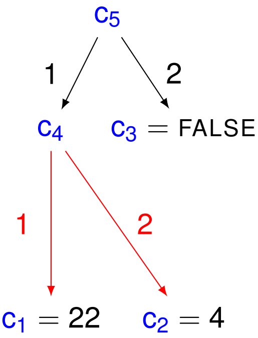
Compute layout of data structures, constrain contents with SMT

Define operational semantics by reduction rules (for finite models)

trade efficiency for expressivity

Static picture of TLA$^+$ values and relations between them

Arena:



SMT:

integer `sort Int`

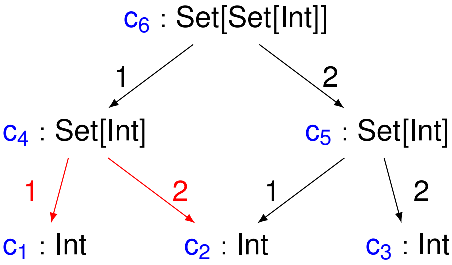
Boolean `sort Bool`

name, e.g., `"abc"`, uninterpreted sort

finite set:

- a constant c of uninterpreted sort set_τ
- propositional constants for members $in_{\langle c_1, c \rangle}, \dots, in_{\langle c_n, c \rangle}$

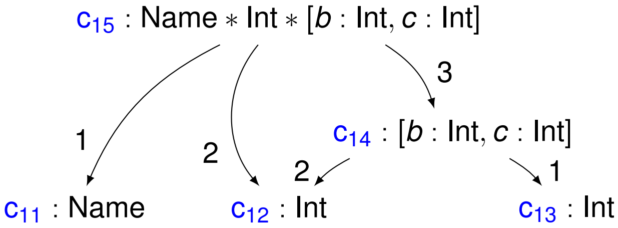
Arenas for sets: $\{\{1,2\},\{2,3\}\}$



SMT defines the contents, e.g., to get $\{\{1\},\{2\}\}$:

$$in_{\langle c_1,c_4\rangle} \land \neg in_{\langle c_2,c_4\rangle} \land in_{\langle c_2,c_5\rangle} \land \neg in_{\langle c_3,c_5\rangle}$$

Tuples and records: $\langle \text{"a"}, 3, [b \mapsto 0, c \mapsto 3] \rangle$



Arena and types precisely define the contents of tuples and records

A warning about records

It is common to combine records of different types, like in Paxos:

$$\{[\mathit{type} \mapsto \texttt{"1a"}, \mathit{bal} \mapsto 1]\} \cup \{[\mathit{type} \mapsto \texttt{"2a"}, \mathit{bal} \mapsto 3, \mathit{val} \mapsto 1]\}$$

The user annotates record constructors:

$$[\mathit{type} \mapsto \texttt{"1a"}, \mathit{bal} \mapsto 1] \\ <: [\mathit{type} \mapsto \textsc{String}, \mathit{bal} \mapsto \textsc{Int}, \mathit{val} \mapsto \textsc{Int}]$$

The unspecified fields may be assigned arbitrary values by SMT

Functions and sequences

a function $f : \tau_1 \to \tau_2$ is encoded with its relation:

$$\{\langle x, f[x] \rangle : x \in \text{DOMAIN } f\}$$

a sequence is encoded as a triple:

$$\langle \mathit{fun}, \mathit{start}, \mathit{end} \rangle$$

Abstract reduction system

A state is $\langle e \mid \mathrm{Ar} \mid \nu \mid \Phi \rangle$:

- a TLA^{+} expression e and arena Ar,
- a valuation $\nu : \mathit{Vars} \to \mathit{Cells} \cup \{\bot\}$
- SMT constraints Φ

Reduction rules:

- simplify the expression, enrich the arena and add constraints

A reduction sequence

$$\begin{tikzcd}
\{\} \in \{\{1\}\} \arrow[r, bend left] & c_1 \in \{\{1\}\} \arrow[r, bend right] & c_1 \in \{\{c_2\}\} \arrow[r, bend left] & c_1 \in \{c_3\} \arrow[r, bend right] & c_1 \in c_4 \arrow[r, bend left] & c_5
\end{tikzcd}$$

Arena:	$c_1,$	$c_2,$	$c_3,$ $c_3 \rightarrow c_2,$	$c_4,$ $c_4 \rightarrow c_3,$	c_5
SMT:	$c_1 : U_{SSI}$	$c_2 : \text{Int}$ $c_2 = 1$	$c_3 : U_{SI}$ $in_{\langle c_2, c_3 \rangle}$	$c_4 : U_{SSI}$ $in_{\langle c_3, c_4 \rangle}$	$c_5 \leftrightarrow$ $in_{\langle c_3, c_4 \rangle}$ $\wedge$ $c_1 = c_3$ $\ldots$

Equalities

Integers, Booleans, and string constants

SMT equality (=)

Sets, functions, records, tuples, and sequences

- **lazy**, define $X = Y$ when needed e.g., $X \subseteq Y \land Y \subseteq X$
- avoid redundant constraints
- use locality thanks to arenas, cache equalities

KERA$^+$: a core language of TLA$^+$ action operators

define reductions for a small set of operators

prove soundness only for these reductions

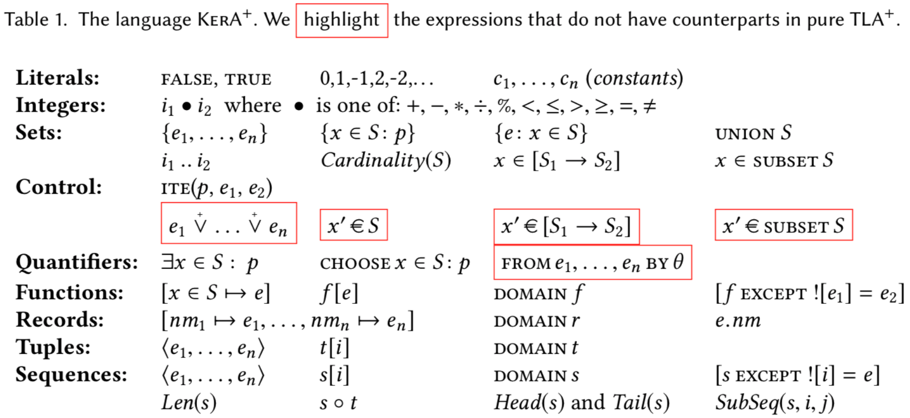


Table 1. The language KERA$^+$. We $\boxed{\text{highlight}}$ the expressions that do not have counterparts in pure TLA$^+$.

Literals:	FALSE, TRUE	0,1,-1,2,-2,...	$c_1, \ldots, c_n$ (*constants*)	
Integers:	$i_1 \bullet i_2$ where $\bullet$ is one of: $+, -, *, \div, \%, <, \leq, >, \geq, =, \neq$			
Sets:	$\{e_1, \ldots, e_n\}$	$\{x \in S : p\}$	$\{e : x \in S\}$	UNION S
	$i_1 .. i_2$	$Cardinality(S)$	$x \in [S_1 \rightarrow S_2]$	$x \in$ SUBSET S
Control:	ITE(p, e_1, e_2)			
	$\boxed{e_1 \overset{+}{\vee} \ldots \overset{+}{\vee} e_n}$	$\boxed{x' \in S}$	$\boxed{x' \in [S_1 \rightarrow S_2]}$	$\boxed{x' \in \text{SUBSET } S}$
Quantifiers:	$\exists x \in S : p$	CHOOSE $x \in S : p$	$\boxed{\text{FROM } e_1, \ldots, e_n \text{ BY } \theta}$	
Functions:	$[x \in S \mapsto e]$	$f[e]$	DOMAIN f	$[f$ EXCEPT $![e_1] = e_2]$
Records:	$[nm_1 \mapsto e_1, \ldots, nm_n \mapsto e_n]$		DOMAIN r	$e.nm$
Tuples:	$\langle e_1, \ldots, e_n \rangle$	$t[i]$	DOMAIN t	
Sequences:	$\langle e_1, \ldots, e_n \rangle$	$s[i]$	DOMAIN s	$[s$ EXCEPT $![i] = e]$
	$Len(s)$	$s \circ t$	$Head(s)$ and $Tail(s)$	$SubSeq(s, i, j)$

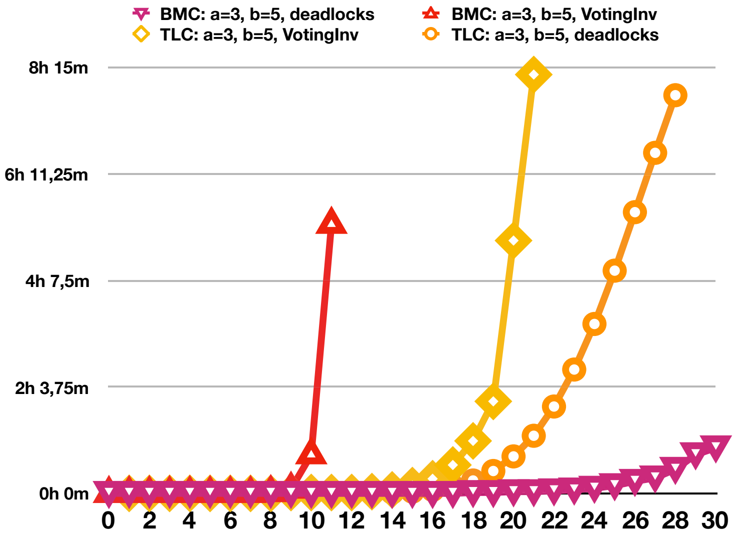
is it fast?

Are we faster than TLC?

Inductive invariants	APALACHE	TLC
TwoPhase, $n = 7$	4s	2h44m

Bounded model checking	APALACHE	TLC
TwoPhase, $n = 7$**,** $k = 10$	1h29m	13s
bcastByz, $n = 6$**,** $k = 11$	1h00m	3h42m
bcastFolk, $n = 20$**,** $k = 10$	41s	timeout
Paxos, $a = 3$**,** $b = 4$**,** $k = 13$	1h42m	< 1m

Safety of Paxos: 3 acceptors, 5 ballots



Performance of SMT solvers

We use Microsoft Z3

SMT solvers are fragile, jumping from hours to seconds and back

Removing uninterpreted functions and integers as much as possible

Mixture of propositional and integer constraints

Bottleneck = UNSAT + non-determinism

Carefully add quantifiers?

Performance of SMT solvers

We use Microsoft Z3

SMT solvers are fragile, jumping from hours to seconds and back

Removing uninterpreted functions and integers as much as possible

Mixture of propositional and integer constraints

Bottleneck = UNSAT + non-determinism

Carefully add quantifiers?

Problematic patterns

VARIABLE x

$Init \triangleq x = 0$

$Next \triangleq x' = 1 - x \lor x' = x$

$Invariant \triangleq x \neq 3$

executions, $k \leq 20$

incremental mode

z3:	44 sec
cvc4:	900 sec
yices2:	99 sec

SMT solvers do not like control non-determinism

Conclusions

Framework for TLA$^+$ model checking with SMT

Bounded model checking alone is not enough

Need for reductions, abstractions, etc.

TLC works surprisingly well