TLA+ Model Checking Made Symbolic

Igor Konnov

Interchain Foundation / Informal Systems

Jure Kukovec  Thanh-Hai Tran
Swiss non-profit foundation

Supports R&D of applications that are:
- secure and scalable
- decentralized

Main focus:
- the Cosmos Network
- Tendermint consensus
What is Cosmos?

Cosmos is a decentralized network of independent blockchains.

Blockchains are powered by BFT consensus like Tendermint.

They communicate over Inter-Blockchain Communication protocol.

[https://cosmos.network/ecosystem]
Tendermint

Byzantine fault-tolerant State Machine Replication middleware

Consensus protocol adapts DLS & PBFT for blockchains:

- wide area network
- hundreds of validators and thousands of nodes
- communication via gossip

**efficient and open source**
Verification-Driven Development of Tendermint:

1. PODC-style specifications in English

2. TLA$^+$ specifications (make English formal / fix it)
   - model checking for finding bugs in TLA$^+$ specs

3. Implementation in Rust
   - model-based testing of the implementation using TLA$^+$ specs

4. Automated verification of TLA$^+$ specs
TLA+ model checking made symbolic
Why TLA⁺?

Rich specification language

TLA⁺ is used in industry, e.g.,

- TLA⁺ tools maintained at Amazon and Inria
  - an interactive proof system (TLAPS)
  - a model checker (TLC), state enumeration

Raft

Paxos (Synod), Egalitarian Paxos, Flexible Paxos

Apache Kafka

several bugs found
First-order logic with sets (ZFC)

Rich expression syntax:
- operations on sets, functions, tuples, records, sequences

Temporal operators:
- □ (always), ◇ (eventually), ↷ (leads-to), no Nexttime

Practice: safety properties, □ Invariant
APALACHE 0.5.0

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: \( \text{QF\textunderscore UFNIA} \)
- Unfolding quantified expressions: \( \forall x \in S : P \) as \( \bigwedge_{c \in S} P[c/x] \)
APALACHE 0.5.0

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: $$\forall x \in S: P \text{ as } \bigwedge_{c \in S} P[c/x]$$
example from distributed algorithms
A service for reliable broadcast

one process broadcasts a message \texttt{bcast}

\textbf{unforgeability}: if no correct process received \texttt{bcast},
then no correct process ever \texttt{accepts bcast} \hspace{1cm} 000 \ldots 0

\textbf{correctness}: if all correct processes received \texttt{bcast},
then some correct process eventually accepts \texttt{bcast} \hspace{1cm} 111 \ldots 1

\textbf{relay}: if a correct process accepts \texttt{bcast},
then all correct processes eventually accept \texttt{bcast} \hspace{1cm} 011 \ldots 1
local \( myval_i \in \{0, 1\} \) -- did process \( i \) receive \textbf{bcast}? 

\[
\text{while true do}
\]
\[
\quad \text{if } \text{myval}_i = 1 \text{ and not sent ECHO before}
\]
\[
\text{then send ECHO to all}
\]
\[
\quad \text{if received ECHO from at least } n-2t \text{ distinct processes}
\]
\[
\quad \quad \text{and not sent ECHO before}
\]
\[
\quad \text{then send ECHO to all}
\]
\[
\quad \text{if received ECHO from at least } n - t \text{ distinct processes}
\]
\[
\quad \text{then accept}
\]
\[
\text{od}
\]

\textbf{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 $n, t, f : n > 3t \land t \geq f \geq 0$

[forzyte.at/software/bymc]

I have heard this talk by Leslie Lamport

Let’s write it in $\text{TLA}^+$

Run the TLC model checker for fixed parameters
Declaration and initialization

EXTENDS Integers, FiniteSets

\[ N \triangleq 12 \quad T \triangleq 3 \quad F \triangleq 3 \]
\[ \text{Corr} \triangleq 1 \ldots (N - F - 1) \quad \text{Faulty} \triangleq (N - F) \ldots N \]

VARIABLES \( pc \), \( rcvd \), \( sent \)

\[ \text{Init} \triangleq \wedge pc \in \text{[Corr \rightarrow \{“V0”, “V1”\}}} \quad \text{some processes receive the broadcast} \]
\[ \wedge sent = \{\} \quad \text{no messages sent initially} \]
\[ \wedge rcvd \in \text{[Corr \rightarrow \{\}}} \quad \text{no messages received initially} \]
Transition relation

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

Receive (p) \triangleq
\exists \text{newMessages} \in \text{SUBSET}(sent \cup \text{Faulty}) :
rcvd' = [rcvd \text{ EXCEPT } ![self] = rcvd[p] \cup \text{newMessages}]
Actions

\[ \text{Upon}V1(p) \triangleq \]
\[ \land pc[p] = \text{"V1"} \]
\[ \land pc' = [pc \text{ EXCEPT } ![p] = \text{"SE"}] \land sent' = sent \cup \{p\} \]

\[ \text{UponNonFaulty} (p) \triangleq \]
\[ \land pc[p] \in \{\text{"V0"}, \text{"V1"}\} \land \text{Cardinality(rcvd}'[p]) \geq N - 2 \times T \]
\[ \land pc' = [pc \text{ EXCEPT } ![p] = \text{"SE"}] \land sent' = sent \cup \{p\} \]

\[ \text{UponAccept} (p) \triangleq \]
\[ \land pc[p] \in \{\text{"V0"}, \text{"V1"}, \text{"SE"}\} \land \text{Cardinality(rcvd}'[p]) \geq N - T \]
\[ \land pc' = [pc \text{ EXCEPT } ![p] = \text{"AC"}] \]
\[ \land sent' = sent \cup (\text{IF } pc[p] \neq \text{"SE"} \text{ THEN } \{p\} \text{ ELSE } \{\}) \]
**unforgeability**: if no correct process received $\text{bcast}$, then no correct process ever accepts $\text{bcast}$

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

\* restricted initial states

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

Check that every state reachable from $\text{InitNoBcast}$ satisfies $\text{Unforg}$
Breaking unforgeability

12 processes, 4 faults  \( n = 3f \)

**APALACHE-MC**: a counterexample in \( 5 \text{ minutes} \)
- 12K SMT constants, 34K SMT assertions  \( \text{depth 6} \)

**TLC**: a counterexample after \( 2 \text{ hrs 21 min} \)
- 600M states  \( \text{depth 6} \)
how does APALACHE 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 $x' \in \{ \ldots \}$ as assignments

Simple type inference
Propagate types at every step
$x : \text{Int}$ gives us $\{x\} : \text{Set}[\text{Int}]$

Bounded model checking
Overapproximate data structures and use SMT
assignments & symbolic transitions
Symbolic transitions

\[ \text{Next} \triangleq \exists p \in \text{Corr} : \]
\[ \land \ \text{Receive}(p) \land \lor \ \text{UponV1}(p) \lor \ \text{UponNonFaulty}(p) \lor \ \text{UponAccept}(p) \lor \ \text{UNCHANGED} \langle pc, sent \rangle \]

Automatically partitioning \text{Next} into four transitions:

\[ \exists p \in \text{Corr} : \land \ \text{Receive}(p) \land \ \text{UponV1}(p) \]
\[ \exists p \in \text{Corr} : \land \ \text{Receive}(p) \land \ \text{UponNonFaulty}(p) \]
\[ \exists p \in \text{Corr} : \land \ \text{Receive}(p) \land \ \text{UponAccept}(p) \]
\[ \exists p \in \text{Corr} : \land \ \text{Receive}(p) \land \ \text{UNCHANGED} \langle pc, sent \rangle \]
Symbolic transitions

\[\text{Next} \triangleq \exists p \in \text{Corr} : \]
\[\land \text{Receive}(p)\]
\[\land \lor \text{UponV1}(p)\]
\[\lor \text{UponNonFaulty}(p)\]
\[\lor \text{UponAccept}(p)\]
\[\lor \text{UNCHANGED} \langle pc, sent \rangle\]

Automatically partitioning \textit{Next} into four transitions:

\[\exists p \in \text{Corr} : \]
\[\land \text{Receive}(p)\]
\[\land \text{UponV1}(p)\]

\[\exists p \in \text{Corr} : \]
\[\land \text{Receive}(p)\]
\[\land \text{UponNonFaulty}(p)\]

\[\exists p \in \text{Corr} : \]
\[\land \text{Receive}(p)\]
\[\land \text{UponAccept}(p)\]

\[\exists p \in \text{Corr} : \]
\[\land \text{Receive}(p)\]
\[\land \text{UNCHANGED} \langle pc, sent \rangle\]
Types
Types: scalars and functions

Basic:

- constants: \textit{Const} \hspace{1cm} \text{“a”, “hello”}
- integers: \textit{Int} \hspace{1cm} -1, 1024
- Booleans: \textit{Bool} \hspace{1cm} \text{FALSE, TRUE}

Finite sets:

\[ \text{Set}[\tau] \]

\[ \text{Set}[\text{Set}[\text{Int}]] \]

Function-like:

- functions: \( \tau_1 \to \tau_2 \) \hspace{1cm} \text{Int \to Bool}
- tuples: \( \tau_1 \times \cdots \times \tau_n \) \hspace{1cm} \text{Int \times Bool \times (Int \to Int)}
- records: \( [\text{Const} \mapsto \tau_1, \ldots, \text{Const} \mapsto \tau_n] \) \hspace{1cm} \text{[“a” \mapsto Int, “b” \mapsto Bool]}
- sequences: \( \text{Seq}(\tau) \) \hspace{1cm} \text{Seq[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 $\text{Set}[\text{Int}]$

$X' \in [X \to X]$ has type $\text{Int} \to \text{Int}$

$y$ in $\{ y \in X : y > 0 \}$ has type $\text{Int}$

{} and ⟨⟩ are polymorphic constructors for sets and sequences

hence, we ask the user to specify the type, e.g., $\{\} <: \{\text{Int}\}$

records also require type annotations
Bounded model checking
Old recipe for bounded symbolic computations

Two symbolic transitions that assign values to $x$

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

Translate TLA$^+$ expressions to SMT with some $\left[ \cdot \right]$
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:

- $c_5$
- $c_4$
- $c_3 = \text{FALSE}$
- $c_1 = 22$
- $c_2 = 4$

### SMT:

- Integer sort $\text{Int}$
- Boolean sort $\text{Bool}$
- Name, e.g., "abc", uninterpreted sort
- Finite set:
  - A constant $c$ of uninterpreted sort $\text{set}_T$
  - Propositional constants for members
    - $\text{in}_{\langle c_1, c \rangle}, \ldots, \text{in}_{\langle c_n, c \rangle}$
Arenas for sets: \{\{1, 2\}, \{2, 3\}\}

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

\[
in_{\langle c_1, c_4 \rangle} \land \lnot in_{\langle c_2, c_4 \rangle} \land in_{\langle c_2, c_5 \rangle} \land \lnot in_{\langle c_3, c_5 \rangle}
\]
Tuples and records: \(\langle "a", 3, [b \mapsto 0, c \mapsto 3]\rangle\)

Arena and types precisely define the contents of tuples and records
Functions and sequences

A function \( f : \tau_1 \rightarrow \tau_2 \) is encoded with its relation:

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

A sequence is encoded as a triple:

\[
\langle \text{fun}, \text{start}, \text{end} \rangle
\]
Abstract reduction system

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

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

Reduction rules:

- simplify the expression, enrich the arena and add constraints
A reduction sequence

\[
\begin{align*}
{} & \in \{\{1\}\} \\
c_1 & \in \{\{1\}\} \\
c_1 & \in \{\{c_2\}\} \\
c_1 & \in \{c_3\} \\
c_1 & \in c_4 \\
c_5 &
\end{align*}
\]

Arena:
\[
\begin{align*}
c_1, & \\
c_2, & \\
c_3, & \\
c_4, & \\
c_5
\end{align*}
\]

SMT:
\[
\begin{align*}
c_1 &: U_{SSI} \\
c_2 &: \text{Int} \\
c_3 &: U_{Sli} \\
c_4 &: U_{SSI} \\
c_5 & \leftrightarrow \\
& \text{in}_{\langle c_3, c_4 \rangle} \\
& \land \\
c_1 & = c_3 \\
& \ldots
\end{align*}
\]
Rewriting a set filter

\[ \{ x \in \{1, 2\} : p(x) \} \leadsto \{ x \in c_3 : p(x) \} \leadsto c_4 \]

Corresponding arena

\( c_3 : \text{Set[Int]} \)

(\text{empty}) \leadsto 

\( c_1 : \text{Int} \quad c_2 : \text{Int} \)

\( c_3 : \text{Set[Int]} \quad c_4 : \text{Set[Int]} \)

\( c_1 : \text{Int} \quad c_2 : \text{Int} \)

Igor Konnov
SMT constraints for set filter

\[ \text{in}_{\langle c_4, c_1 \rangle} \iff \text{in}_{\langle c_3, c_1 \rangle} \land c_5 = \text{true} \]

\[ \text{in}_{\langle c_4, c_2 \rangle} \iff \text{in}_{\langle c_3, c_2 \rangle} \land c_6 = \text{true} \]

Set filtering

\[ \{ x \in \{1, 2\} : p(x) \} \]

\[ p(1) \rightsquigarrow c_5 : \text{Bool} \]

\[ p(2) \rightsquigarrow c_6 : \text{Bool} \]
Equalities

Integers, Booleans, and string constants

SMT equality (=)

Sets, functions, records, tuples, and sequences

- **lazy**, define $X = Y$ when needed

- avoid redundant constraints

- use locality thanks to arenas, cache equalities

\[ X \subseteq Y \land Y \subseteq X \]
**KERA⁺**: a core language of TLA⁺ action operators

Table 1. The language KERA⁺. We highlight the expressions that do not have counterparts in pure TLA⁺.

<table>
<thead>
<tr>
<th>Category</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literals</td>
<td>FALSE, TRUE, 0, 1, 2, 3, ..., c₁, ..., cₙ (constants)</td>
</tr>
<tr>
<td>Integers</td>
<td>i₁ • i₂ where • is one of: +, −, *, ÷, %, &lt;, ≤, &gt;, ≥, =, ≠</td>
</tr>
<tr>
<td>Sets</td>
<td>{e₁, ..., eₙ}, {x ∈ S : p}, {e : x ∈ S}, UNION S, x ∈ SUBSET S</td>
</tr>
<tr>
<td>Control</td>
<td>ITE(p, e₁, e₂), x ∈ S, x ∈ [S₁ → S₂], x ∈ SUBSET S</td>
</tr>
<tr>
<td>Quantifiers</td>
<td>∃x ∈ S : p, CHOOSE x ∈ S : p, FROM e₁, ..., eₙ BY θ</td>
</tr>
<tr>
<td>Functions</td>
<td>[x ∈ S ↦ e], f[e], DOMAIN f</td>
</tr>
<tr>
<td>Records</td>
<td>[nm₁ ↦ e₁, ..., nmₙ ↦ eₙ], DOMAIN r, e. nm</td>
</tr>
<tr>
<td>Tuples</td>
<td>⟨e₁, ..., eₙ⟩, t[i], DOMAIN t</td>
</tr>
<tr>
<td>Sequences</td>
<td>⟨e₁, ..., eₙ⟩, s[i], DOMAIN s, Head(s) and Tail(s), SubSeq(s, i, j)</td>
</tr>
</tbody>
</table>

reduction rules + proofs
Experiments
## Benchmarks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCR-$n$</td>
<td>Leader election in rings with $n$ processes</td>
</tr>
<tr>
<td>Bakery-$n$</td>
<td>Bakery algorithm for mutual exclusion of $n$ processes</td>
</tr>
<tr>
<td>bcastByz-$n$</td>
<td>Reliable broadcast of $n$ processes</td>
</tr>
<tr>
<td>bcastFolk-$n$</td>
<td>Folklore broadcast of $n$ processes</td>
</tr>
<tr>
<td>EWD840-$n$</td>
<td>Termination detection in a ring of $n$ processes</td>
</tr>
<tr>
<td>Paxos-$n$</td>
<td>Paxos consensus for $n$ acceptors with crash faults</td>
</tr>
<tr>
<td>Prisoners-$n$</td>
<td>Puzzle of $n$ prisoners</td>
</tr>
<tr>
<td>Raft-$n$</td>
<td>Raft consensus for $n$ processes and crash faults</td>
</tr>
<tr>
<td>SimpAlloc-$c$-$r$</td>
<td>Simple resource allocator with $c$ clients and $r$ resources</td>
</tr>
<tr>
<td>Traffic</td>
<td>Traffic example</td>
</tr>
<tr>
<td>TwoPhase-$n$</td>
<td>Two-phase commit with $n$ resource managers</td>
</tr>
</tbody>
</table>

**Repository:** [github.com/tlaplus/Examples/](https://github.com/tlaplus/Examples/)
### Inductive invariant checking

#### Wrong invariant candidates

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>APALACHE</th>
<th>TLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bakery–5</td>
<td>1 min</td>
<td>N/A</td>
</tr>
<tr>
<td>TwoPhase–7</td>
<td>1 min</td>
<td>3 hrs</td>
</tr>
<tr>
<td>LCR–5</td>
<td>1 min</td>
<td>2 hrs</td>
</tr>
<tr>
<td>bcastByz–10</td>
<td>1 min</td>
<td>19 min</td>
</tr>
<tr>
<td>EWD840–11</td>
<td>1 min</td>
<td>12 min</td>
</tr>
</tbody>
</table>

#### Inductive invariants

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>APALACHE</th>
<th>TLC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bakery–5</td>
<td>1 min</td>
<td>N/A</td>
</tr>
<tr>
<td>TwoPhase–7</td>
<td>1 min</td>
<td>3 hrs</td>
</tr>
<tr>
<td>LCR–5</td>
<td>1 min</td>
<td>2 hrs</td>
</tr>
<tr>
<td>bcastByz–4</td>
<td>1 min</td>
<td>1 min</td>
</tr>
<tr>
<td>EWD840–10</td>
<td>1 min</td>
<td>1 min</td>
</tr>
</tbody>
</table>
Bounded model checking with invariants

(TO = 24 hours)
Symbolic model checker for TLA\(^+\)  [OOPSLA’19]

TLA\(^+\) → Reduction rules → SMT (Z3)

Distributed algorithms

- Invariants
- Inductive invariants
- Fixed parameters, bounded executions
- Fixed parameters

forsyte.at/research/apalache/
TLC does not scale to minimal interesting examples

- Byzantine faults
- very non-deterministic environment

Starting to use Apalache

Improvements in the encoding

Parallel model checker?

We are hiring!