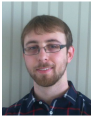
TLA$^+$ Model Checking Made Symbolic

Igor Konnov

Interchain Foundation / Informal Systems




Jure Kukovec   Thanh-Hai Tran

INTERCHAIN FOUNDATION
informatics mathematics Inria
logics
WWTF
VIENNA SCIENCE AND TECHNOLOGY FUND

INTERCHAIN
FOUNDATION

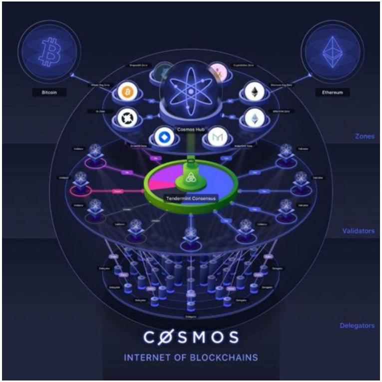
Swiss non-profit foundation

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., 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 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: `QF_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: `QF_UFNIA`

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

example from distributed algorithms

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** — $000\ldots0$

---

correctness: if all correct processes received **bcast**, then some correct process eventually accepts **bcast** — $111\ldots1$

relay: if a correct process accepts **bcast**, then all correct processes eventually accept **bcast** — $011\ldots1$

Reliable broadcast by Srikanth & Toueg 87

```
local  myval_i ∈ {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 $n, t, f : n > 3t \wedge t \geq f \geq 0$

[forsyte.at/software/bymc]

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*

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

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

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

$Init \triangleq \land\ pc \in [Corr \rightarrow \{\text{“}V0\text{“}, \text{“}V1\text{“}\}]$ *some processes receive the broadcast*

$\quad\quad\quad \land\ sent = \{\}$ *no messages sent initially*

$\quad\quad\quad \land\ rcvd \in [Corr \rightarrow \{\}]$ *no messages received initially*

Transition relation

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

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

Actions

$UponV1(p) \triangleq$
$\land\ pc[p] = \text{"V1"}$

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

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

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

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

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

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

Safety?

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

* *a non-inductive invariant*

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

* *restricted initial states*

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

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 APALACHE work?

What is hard about TLA$^+$?

Rich data

sets of sets, functions, records, tuples, sequences

No types

TLA$^+$ is not a programming language

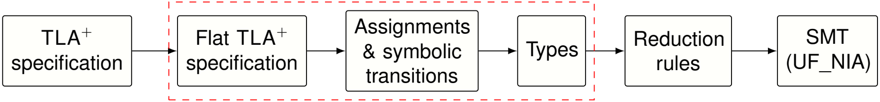
No imperative statements like assignments

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 $x' \in \{\dots\}$ as assignments

Simple type inference

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

Bounded model checking

overapproximate data structures and use SMT

assignments & symbolic transitions

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

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

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]

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

Automatically partitioning *Next* into four transitions:

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

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

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

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

Types

Types: scalars and functions

Basic:

constants: *Const* — "a", "hello"

integers: *Int* — -1, 1024

Booleans: *Bool* — FALSE, TRUE

Finite sets:

$Set[\tau]$ — $Set[Set[Int]]$

Function-like:

functions: $\tau_1 \to \tau_2$ — $Int \to Bool$

tuples: $\tau_1 \times \cdots \times \tau_n$ — $Int \times Bool \times (Int \to Int)$

records: $[Const \mapsto \tau_1, \dots, Const \mapsto \tau_n]$ — $[\text{“a”} \mapsto Int, \text{“b”} \mapsto Bool]$

sequences: $Seq(\tau)$ — $Seq[Int]$

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 $Set[Int]$

$X' \in [X \to X]$ has type $Int \to Int$

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., $\{\} <: \{Int\}$

records also require type annotations

Bounded model checking

Old recipe for bounded symbolic computations

Two symbolic transitions that assign values to x

$$Next \triangleq A \lor B$$

Translate TLA$^+$ expressions to SMT with some $[\![\cdot]\!]$

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

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

Our idea

Mimic the semantics implemented by TLC

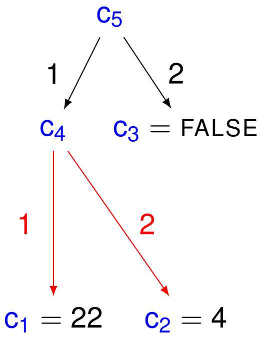
Compute layout of data structures, constrain contents with SMT

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`

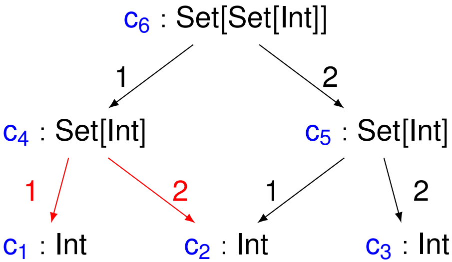
Boolean `sort Bool`

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

finite set:

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

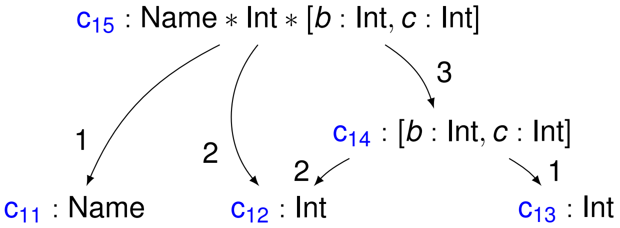
Arenas for sets: $\{\{1, 2\}, \{2, 3\}\}$



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: $\langle \texttt{"a"}, 3, [b \mapsto 0, c \mapsto 3] \rangle$



Arena and types precisely define the contents of tuples and records

Functions and sequences

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

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

a sequence is encoded as a triple:

$$\langle fun, start, end \rangle$$

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 $\nu : \mathit{Vars} \rightarrow \mathit{Cells} \cup \{\perp\}$

SMT constraints Φ

Reduction rules:

simplify the expression, enrich the arena and add constraints

A reduction sequence

$\{\} \in \{\{1\}\}$ $\quad$ $c_1 \in \{\{1\}\}$ $\quad$ $c_1 \in \{\{c_2\}\}$ $\quad$ $c_1 \in \{c_3\}$ $\quad$ $c_1 \in c_4$ $\quad$ c_5

Arena: $c_1,$ $\quad$ $c_2,$ $\quad$ $c_3,$ $\quad$ $c_4,$ $\quad$ c_5

$c_3 \to c_2,$ $\quad$ $c_4 \to c_3,$

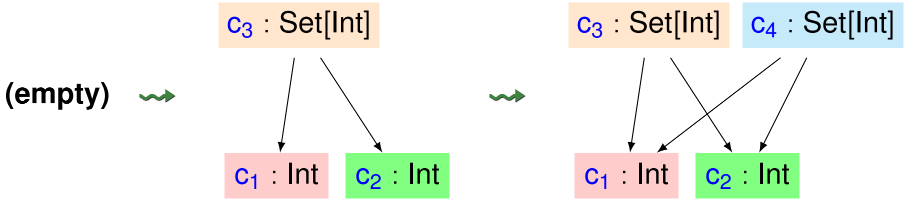
SMT: $c_1 : U_{SSI}$ $\quad$ $c_2 : \mathrm{Int}$ $\quad$ $c_3 : U_{SI}$ $\quad$ $c_4 : U_{SSI}$ $\quad$ $c_5 \leftrightarrow in_{\langle c_3, c_4 \rangle} \land c_1 = c_3 \ldots$

$c_2 = 1$ $\quad$ $in_{\langle c_2, c_3 \rangle}$ $\quad$ $in_{\langle c_3, c_4 \rangle}$

Rewriting a set filter

$$\{x \in \{1, 2\} : p(x)\} \rightsquigarrow \{x \in c_3 : p(x)\} \rightsquigarrow c_4$$

Corresponding arena



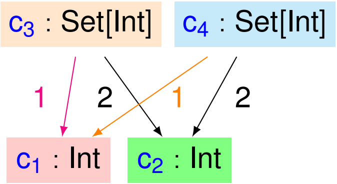
SMT constraints for set filter

$$in_{\langle c_4, c_1 \rangle} \Leftrightarrow in_{\langle c_3, c_1 \rangle} \land c_5 = true$$

$$in_{\langle c_4, c_2 \rangle} \Leftrightarrow in_{\langle c_3, c_2 \rangle} \land c_6 = true$$

Set filtering

$\{x \in \{ \boxed{1}, \boxed{2} \} : p(x)\}$



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

$p(2) \rightsquigarrow c_6 : \text{Bool}$

Equalities

Integers, Booleans, and string constants

SMT equality (=)

Sets, functions, records, tuples, and sequences

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

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
	$i_1 \mathrel{..} i_2$	*Cardinality*(S)	$x \in [S_1 \rightarrow S_2]$	$x \in$ SUBSET S
Control:	ITE(p, e_1, e_2)			
	$\boxed{e_1 \dot{\vee} \ldots \dot{\vee} e_n}$	$\boxed{x' \in S}$	$\boxed{x' \in [S_1 \rightarrow S_2]}$	$\boxed{x' \in \text{SUBSET } S}$
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)

reduction rules + proofs

Experiments

Benchmarks

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

Repository: `github.com/tlaplus/Examples/`

Inductive invariant checking

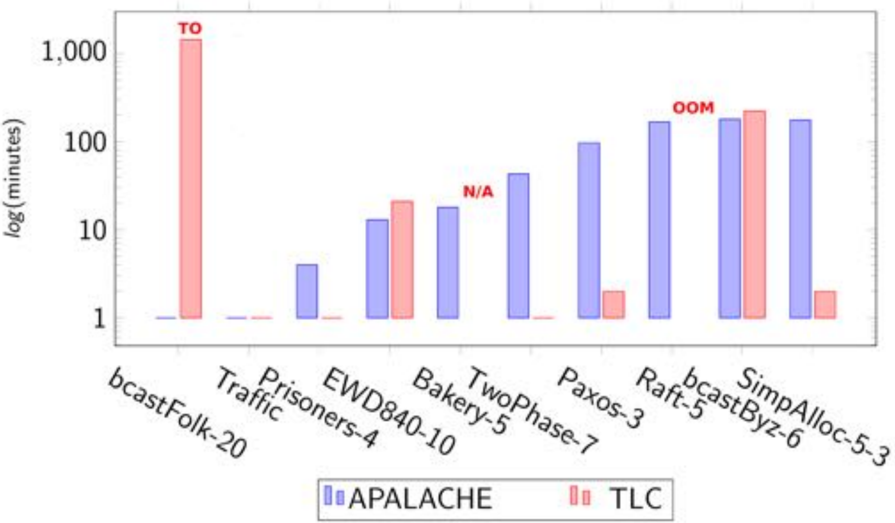
Wrong invariant candidates

Benchmark	**APALACHE**	TLC
Bakery–5	**1 min**	N/A
TwoPhase–7	**1 min**	3 hrs
LCR–5	**1 min**	2 hrs
bcastByz–10	**1 min**	19 min
EWD840–11	**1 min**	12 min

Inductive invariants

Benchmark	**APALACHE**	TLC
Bakery–5	**1 min**	N/A
TwoPhase–7	**1 min**	3 hrs
LCR–5	**1 min**	2 hrs
bcastByz–4	1 min	1 min
EWD840–10	1 min	1 min

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/`

INTERCHAIN FOUNDATION

[informal.systems]

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!