Specifying and checking an extension of Tendermint consensus in TLA+

TlaConf 2022 @ St. Louis, MO

Jure Kukovec, **Daniel Cason**, **Igor Konnov**, **and Josef Widder**

informal SYSTEMS

Who We Are

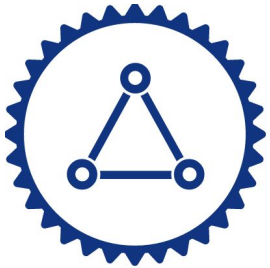
Verifiable distributed
systems *and* **organizations.**

We envision an open-source ecosystem of cooperatively owned and governed distributed organizations running on reliable distributed systems.

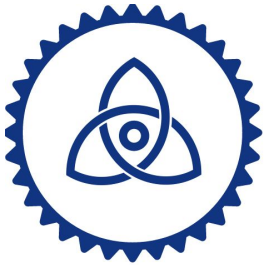
informal SYSTEMS

Who We Are

Blockchain Infrastructure



Tendermint-rs

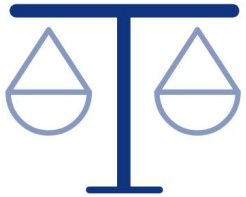


ibc-rs

Formal Verification Tools

Apalache

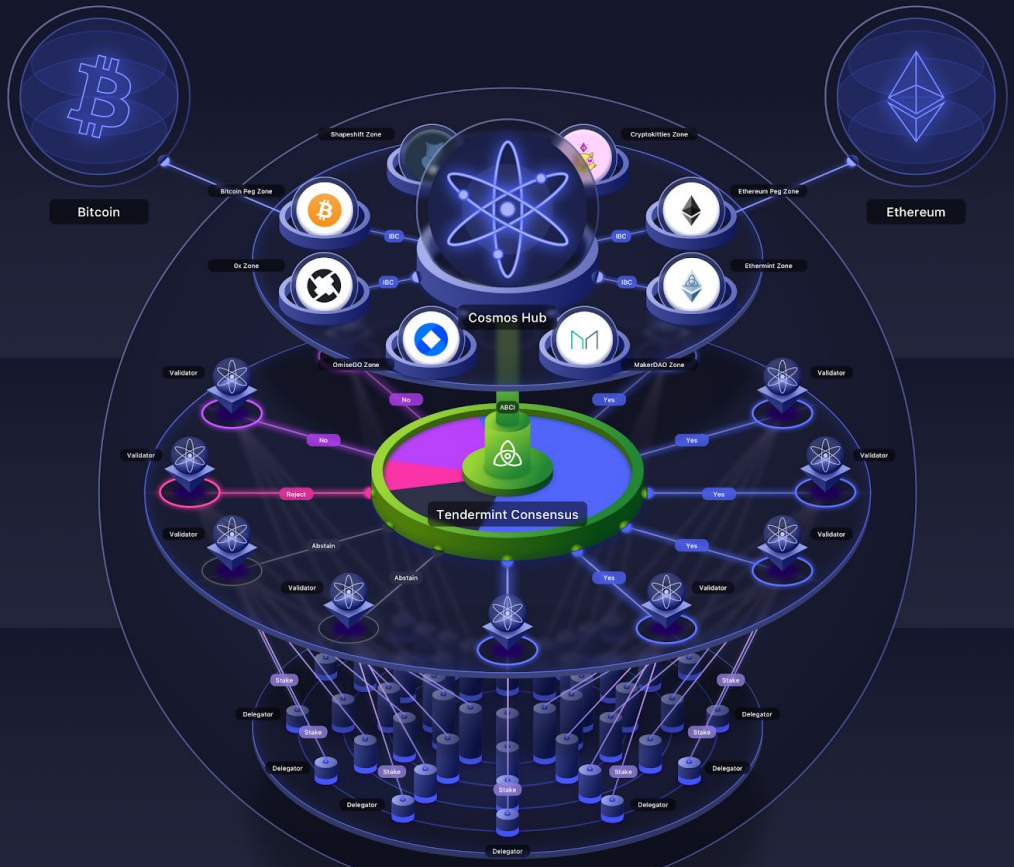
Business Operations Tools



Themis Contract

Our Infrastructure powers the COSMOS Network

informal SYSTEMS



COSMOS

INTERNET OF BLOCKCHAINS

https://cosmos.network

by @zcpeng

The Tendermint consensus protocol

informal SYSTEMS

“The latest gossip on BFT consensus” (2018)

Algorithm 1 Tendermint consensus algorithm

```
1:  Initialization:
2:    h_p := 0                                   /* current height, or consensus instance we are currently executing */
3:    round_p := 0                               /* current round number */
4:    step_p ∈ {propose, prevote, precommit}
5:    decision_p[] := nil
6:    lockedValue_p := nil
7:    lockedRound_p := −1
8:    validValue_p := nil
9:    validRound_p := −1
10: upon start do StartRound(0)
11: Function StartRound(round) :
12:   round_p ← round
13:   step_p ← propose
14:   if proposer(h_p, round_p) = p then
15:     if validValue_p ≠ nil then
16:       proposal ← validValue_p
17:     else
18:       proposal ← getValue()
19:     broadcast ⟨PROPOSAL, h_p, round_p, proposal, validRound_p⟩
20:   else
21:     schedule OnTimeoutPropose(h_p, round_p) to be executed after timeoutPropose(round_p)
```

informal SYSTEMS

Tendermint consensus

Byzantine fault tolerant (BFT), but *no* time guarantees
Assuming n > 3f

No requirements for validators' clocks; no synchrony

Block time deterministically determined from times of votes (weighted average) - *NOT* a part of the consensus

informal SYSTEMS

Time, for a change?

informal SYSTEMS

Why change Tendermint?

Preempt attacks based on apps' assumptions of time

Users wanted it:

- **They used the time in the block as if it was related to real-time**
- **Unbonding period depends on real-time**

informal SYSTEMS

Example change

- Rule on lines 28-35

arXiv paper

```
upon timely((PROPOSAL, h_p, round_p, v, vr))
 from proposer(h_p, round_p)
 AND 2f + 1 (PREVOTE, h_p, vr, id(v))
while step_p = propose ∧ (vr ≥ 0 ∧ vr < round_p) do {
 if valid(v) ∧ (lockedRound_p ≤ vr ∨ lockedValue_p = v) {

  broadcast (PREVOTE, h_p, round_p, id(v))
 } else {
  broadcast (PREVOTE, hp, round_p, nil)
 }
}
```

Proposer-based time

```
upon timely((PROPOSAL, h_p, round_p, (v, tprop), vr))
 from proposer(h_p, round_p)
 AND 2f + 1 (PREVOTE, h_p, vr, id(v, tvote))
 while step_p = propose ∧ (vr ≥ 0 ∧ vr < round_p) do {
  if valid(v) ∧ (lockedRound_p ≤ vr ∨ lockedValue_p = v) {
   // send hash of v and tprop in PREVOTE message
   broadcast (PREVOTE, h_p, round_p, id(v, tprop))
  } else {
   broadcast (PREVOTE, hp, round_p, nil)
  }
 }
```

https://github.com/tendermint/tendermint/blob/main/spec/consensus/proposer-based-timestamp/pbts_001_draft.md

informal SYSTEMS

From English to TLA+

https://github.com/tendermint/tendermint/tree/main-pbts/spec/consensus/proposer-based-timestamp/tla

informal SYSTEMS

Synchrony assumptions and validity

Bounded drift (between correct processes)
Bounded end-to-end latency

```
localClock \in [Corr -> MinTimestamp..(MinTimestamp + Precision)]
```

External validity - when is a value sensible?
Messages get rejected if their timestamps are invalid

```
\* @type: (TIME, TIME) => Bool;
IsTimely(processTime, messageTime) ==
  /\ processTime >= messageTime - Precision
  /\ processTime <= messageTime + Precision + Delay
```

informal SYSTEMS

PBTS: Observations

No impact on the consensus over payload

Main concern: liveness, not safety

Better liveness FT threshold: n > 2f + synchrony

```
Inv ==
    /\ AgreementOnValue
    /\ ConsensusValidValue
    /\ ConsensusTimeValid
    /\ TimeLiveness
```

informal SYSTEMS

Mastering Apalache

informal SYSTEMS

Modeling time

How should one model time in a specification?

a. Time ticks by 1:

   Next ≜ … ∧ t' = t + 1

b. Time increases by an arbitrary amount:

   Next ≜ … ∧ ∃tt ∈ Int: tt > t ∧ t' = tt (alt. ∃tt ∈ min..max)

informal SYSTEMS

Modeling time - Example

```
\* advance the global clock
\* @type: Bool;
AdvanceRealTime ==
    /\ ValidTime(realTime)
    /\ \E t \in Timestamps:
      /\ t > realTime
      /\ realTime' = t
      /\ localClock' = [p \in Corr |-> localClock[p] + (t - realTime)]
    /\ UNCHANGED <<coreVars, bookkeepingVars, beginRound>>
    /\ action' = "AdvanceRealTime"
```

informal SYSTEMS

DIY action composition

What is the best granularity of actions?

a. **One-by-one:**

   ∃ x ∈ DOMAIN f:

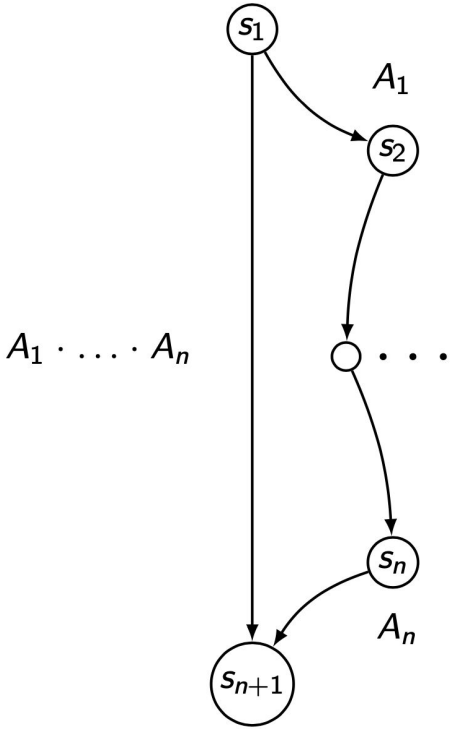
   f' = [f EXCEPT ![x] = F(x)]

b. **Many-at-once:**

   ∃ S ∈ SUBSET DOMAIN f:

   f' = [x ∈ DOMAIN f ↦

   IF x ∈ S THEN F(x) ELSE f[x]]



informal SYSTEMS

Simulation succeeds surprisingly swiftly

1.2. Simulator command-line parameters

The simulator can be run as follows:

```
$ apalache-mc simulate
    [all-checker-options] [--max-run=NUM] <myspec>.tla
```

The arguments are as follows:

- Special parameters:
  - `--max-run=NUM`: but produce up to `NUM` simulation runs (unless `--max-error` errors have been found), default: `100`

https://apalache.informal.systems/docs/apalache/running.html
#12-simulator-command-line-parameters

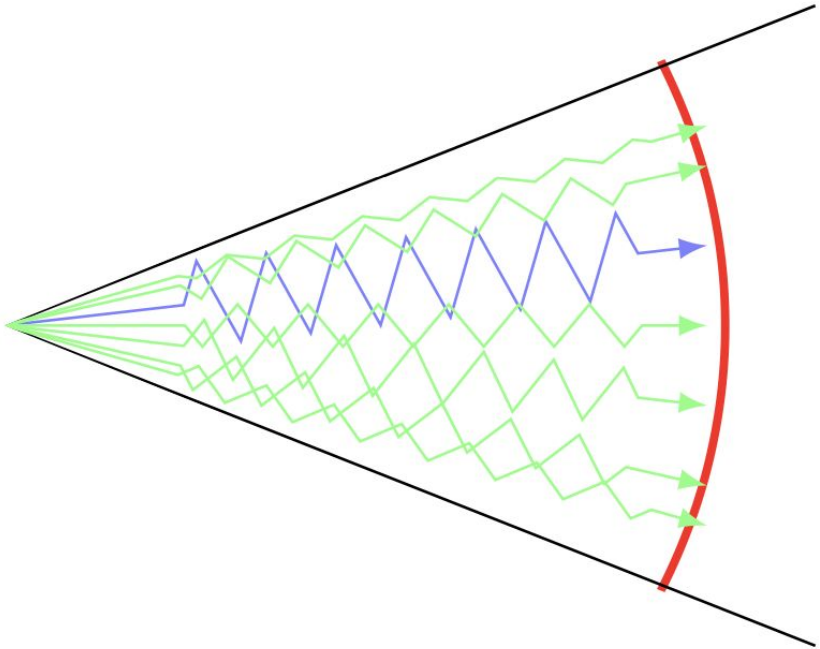
informal SYSTEMS

Simulation succeeds surprisingly swiftly

“check”: exhaustive, slow

“simulate”: incomplete, fast

- **At each step, pick one *symbolic transition* at random**
- **Repeat N times or until an invariant is violated**



informal SYSTEMS

Next-Gen technology

The Apalache module defines Gen(_) - bounded value generators

- **Gen(_): Int ... Unconstrained integer**
- **Gen(N): Set(t) ... A set of up to N elements produced by Gen(N): t**

Term	Potential values
Gen(15): Int	…, -1, 0, 1, …
Gen(2): Set(Str)	{}, {“a”}, {“cake”}, {“TLA”, “plus”}
Gen(3): Set(Set(Int))	{}, {{}}, {{0,42}, {-99}, {3, 7, 88}}, {{1,2,3}, {4,5,6},{7,8,9}}

informal SYSTEMS

Next-Gen technology - Example

```
InitGen ==
  /\ InitState
  /\ msgsPropose \in [Rounds -> Gen(N_GEN)]
  /\ msgsPrevote \in [Rounds -> Gen(N_GEN)]
  /\ msgsPrecommit \in [Rounds -> Gen(N_GEN)]
  /\ BenignAndSubset(msgsPropose, AllFaultyProposals)
  /\ BenignAndSubset(msgsPrevote, AllFaultyPrevotes)
  /\ BenignAndSubset(msgsPrecommit, AllFaultyPrecommits)
```

informal SYSTEMS

Experiments

Experiments

Results

- 3 correct, 1 faulty
  - No invariant violation
    - Expected outcome
  - ~75h
- 2 correct, 2 faulty
  - Counterexample found
    - Length 8
    - Expected outcome
  - ~49h

Machine & CMD

- Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz
- 32GB ram

Apalache command:

```
check
--length=8
--inv=Inv
--cinit=CInit
--discard-disabled=false
```

https://github.com/tendermint/tendermint/blob/main-pbts/spec/consensus/proposer-based-timestamp/tla/experiment_log.md

informal SYSTEMS

informal SYSTEMS

Correct by induction

BMC is complete for inductive invariants

Inductive invariants are HARD AND TIME-CONSUMING to write

Companies have time budgets, to balance verification against other work

```
\* a combination of all lemmas
Inv ==
    /\ EvidenceContainsMessages
    /\ AllNoFutureMessagesSent
    /\ AllIfInPrevoteThenSentPrevote
    /\ AllIfInPrecommitThenSentPrecommit
    /\ AllIfInDecidedThenReceivedProposal
    /\ AllIfInDecidedThenReceivedTwoThirds
    /\ AllIfInDecidedThenValidDecision
    /\ AllLockedRoundIffLockedValue
    /\ AllIfLockedRoundThenSentCommit
    /\ AllLatestPrecommitHasLockedRound
    /\ AllIfSentPrevoteThenReceivedProposalOrTwoThirds
    /\ IfSentPrecommitThenSentPrevote
    /\ IfSentPrecommitThenReceivedTwoThirds
    /\ AllNoEquivocationByCorrect
    /\ PrecommitsLockValue
```

https://github.com/tendermint/tendermint/blob/main/spec/light-client/accountability/TendermintAccInv_004_draft.tla

informal SYSTEMS

Correct by induction (cont.)

```
\* The final piece by Josef Widder:
\* if T + 1 processes precommit on
\* the same value in a round,
\* then in the future rounds there are
\* less than 2T + 1 prevotes for another value
PrecommitsLockValue ==
  \A r \in Rounds:
    \A v \in ValidValues \union {NilValue}:
      \/ LET Precommits ==  {m \in msgsPrecommit[r]: m.id = v}
         IN
         Cardinality(Senders(Precommits)) < THRESHOLD1
      \/ \A fr \in { rr \in Rounds: rr > r }:  \* future rounds
           \A w \in (ValuesOrNil) \ {v}:
             LET Prevotes == {m \in msgsPrevote[fr]: m.id = w}
             IN
             Cardinality(Senders(Prevotes)) < THRESHOLD2
```

https://github.com/tendermint/tendermint/blob/main/spec/light-client/accountability/TendermintAccInv_004_draft.tla

informal SYSTEMS

Proofs

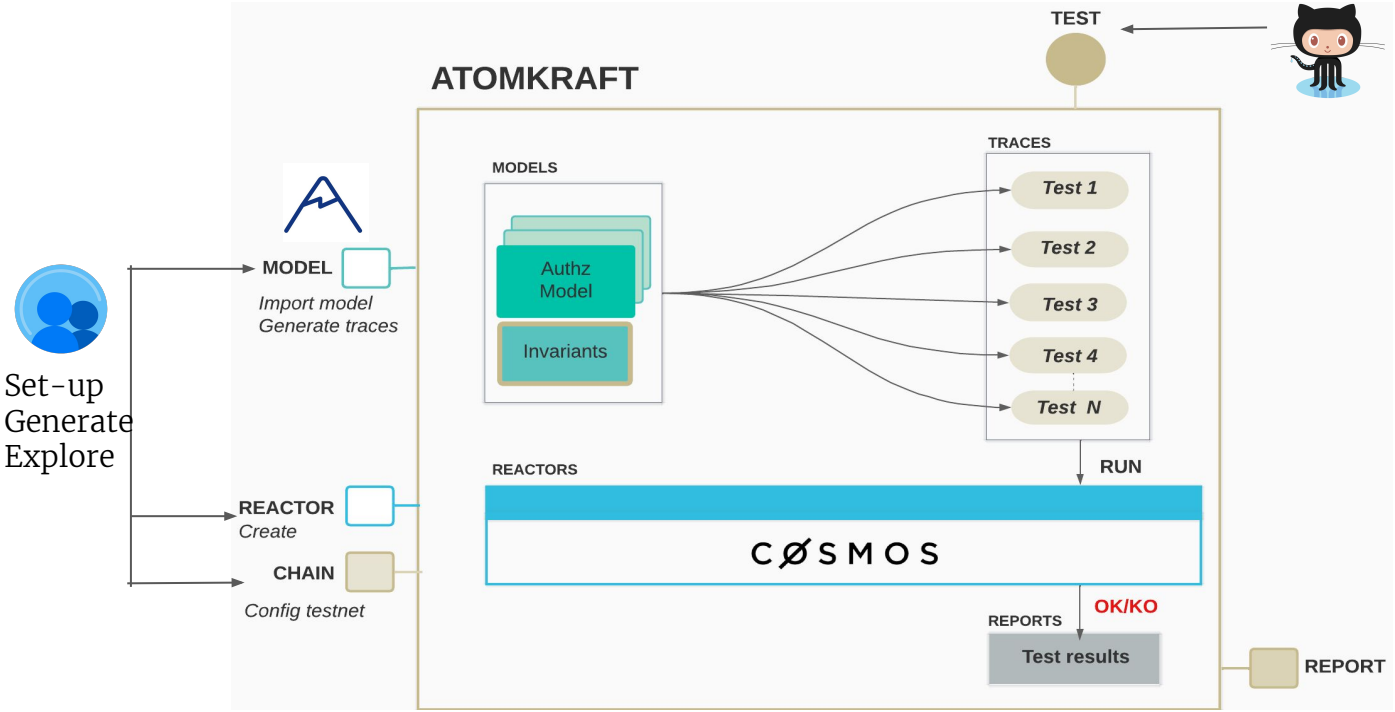
- TLAPS?
  - How to deal with Int?
- Ivy?
  - Extending existing proofs for base Tendermint by Galois
- reTLA?
  - Catch my talk at 16:30

```
action broadcast_prevote(r:round, v:value) = {
    var m: msg;
    m.m_kind := msg_kind.prevote;
    m.m_src := n;
    m.m_round := r;
    m.m_value := v;
    call shim.broadcast(n,m);
}
```

https://github.com/tendermint/tendermint/tree/main/spec/ivy-proofs

informal SYSTEMS

Atomkraft-Cosmos



https://github.com/informalsystems/atomkraft

informal SYSTEMS

Thanks!

informal.systems

apalache.informal.systems

informal SYSTEMS