Model checking of distributed algorithms:
from classics towards Tendermint blockchain

Igor Konnov

VMCAI winter school, January 16-18, 2020
Swiss non-profit foundation

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

Main focus:
- the Cosmos Network
- Tendermint consensus
Cosmos

A decentralized network of independent blockchains

Blockchains are powered by BFT consensus like Tendermint

They communicate over Inter-Blockchain Communication protocol

[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**

Theory: [arxiv.org/abs/1807.04938]
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
Timeline

Introduction to fault-tolerant distributed algorithms

Verifying synchronous threshold-guarded algorithms

Verifying asynchronous threshold-guarded algorithms

Can we verify Tendermint consensus?
Please send me some money

I will transfer you 100 atoms
Send 100 ATOMs to cosmos1wze...
Send 100 ATOMs to cosmos1wze...
Features of the system

Distributed

logically and geographically

Fault-tolerant

individual machines may crash and even act malicious

Safe and live

e.g., no double spending

every transaction is eventually committed
How to build such a system?
sequential code:

```c
int i = 0, j = 1000;

while (true) {
    begin_tx();

    if (recv(ItoJ))
        { i -= 100; j += 100; }

    if (recv(JtoI))
        { i += 100; j -= 100; }

    if (i < 0 || j < 0)
        abort_tx();
    else
        commit_tx();
}
```
sequential code:

```c
int i = 0, j = 1000;
while (true) {
    begin_tx();

    if (recv(ItoJ))
        { i -= 100; j += 100; }

    if (recv(JtoI))
        { i += 100; j -= 100; }

    if (i < 0 || j < 0)
        abort_tx();
    else
        commit_tx();
}
```

state machine:

```

```

```
Central server

I2J

J2I

read(i)

crash!

Igor Konnov 13 of 39
Replication is the solution
Replicated state machine

How to coordinate them?
Replicated state machine

How to coordinate them?
Replicated state machine

How to coordinate them?

Igor Konnov
Replicated state machine

How to coordinate them?
Two-phase commit

Transaction manager:

1. `send <INIT, txid> to ALL`
2. `ncommits = 0`
3. `while ncommits < N {
   on <ABORT> from i {
      send <ABORT> to ALL;
      break
   }
   on <COMMIT> from i {
      ncommits++
   }
   if ncommits == N
      send <COMMIT> to ALL
}

Replica i of N:

1. `on <INIT, txid> from mgr {
   begin_tx(txid)
   /* processing... */
   if error()
      send <ABORT> to mgr
   else send <COMMIT> to mgr
   receive m from mgr
   if m == <ABORT>
      abort_tx(txid)
   else
      commit_tx(txid)
}

if there are crashes?
Two-phase commit

Transaction manager:

```plaintext
send <INIT, txid> to ALL
ncommits = 0
while ncommits < N {
    on <ABORT> from i {
        send <ABORT> to ALL;
        break
    }
    on <COMMIT> from i
        ncommits++
    if ncommits == N
        send <COMMIT> to ALL
}
```

Replica i of N:

```plaintext
on <INIT, txid> from mgr {
    begin_tx(txid)
    /* processing... */
    if error()
        send <ABORT> to mgr
    else send <COMMIT> to mgr
    receive m from mgr
    if m == <ABORT>
        abort_tx(txid)
    else
        commit_tx(txid)
}
```

if there are crashes? 🔥
Distributed consensus
Idea of consensus

A distributed algorithm for $N$ replicas

every replica proposes a value $w \in V$

Termination

every correct replica eventually decides on a value $v \in V$

Agreement

if a replica decides on $v$, no replica decides on $V \setminus \{v\}$

Validity

if a replica decides on $v$, the value $v$ was proposed earlier
How is consensus useful?

1. propose (ItoJ)
2. decide (JtoI)
1. propose (JtoI)
2. decide (JtoI)
How is consensus useful?

1. propose(ItoJ)
2. decide(JtoI)

1. propose(JtoI)
2. decide(JtoI)
How is consensus useful?

1. propose(ItoJ)
2. decide(JtoI)

1. propose(JtoI)
2. decide(JtoI)
Blockchain with classical consensus

In practice, multiple user transactions are packed together

Consensus decides on block hashes
Let’s write some algorithms
**Termination**

every replica eventually decides on a value $v \in V$

**Agreement**

if a replica decides on $v$, no replica decides on $V \setminus \{v\}$

**Validity**

if a replica decides on $v$, the value $v$ was proposed earlier
Consensus without termination

The algorithm: do nothing!

1. propose(ItoJ)
1. propose(JtoI)
Termination
   every replica eventually decides on a value $v \in V$

Agreement
   if a replica decides on $v$, no replica decides on $V \setminus \{v\}$

Validity
   if a replica decides on $v$, the value $v$ was proposed earlier
Consensus without agreement

The algorithm: decide on own value!
**Termination**

every replica eventually decides on a value \( v \in V \)

**Agreement**

if a replica decides on \( v \), no replica decides on \( V \setminus \{v\} \)

**Validity**

if a replica decides on \( v \), the value \( v \) was proposed earlier
Consensus without validity

The algorithm: decide on a fixed value!

1. decide(bob)

1. decide(bob)

1. decide(bob)

1. decide(bob)
Termination
  every replica eventually decides on a value \( v \in V \)

Agreement
  if a replica decides on \( v \), no replica decides on \( V \setminus \{v\} \)

Validity
  if a replica decides on \( v \), the value \( v \) was proposed earlier

is there an algorithm?
Synchronous distributed consensus
Synchronous rounds

a) send post on Monday, receive post on Thursday, and compute on Friday

b) DHL delivers the post in 48 hours

<table>
<thead>
<tr>
<th>Round 1</th>
<th>Round 2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replica 1: send/receive/compute</td>
<td>send/receive/compute</td>
<td>...</td>
</tr>
<tr>
<td>Replica 2: send/receive/compute</td>
<td>send/receive/compute</td>
<td>...</td>
</tr>
<tr>
<td>Replica 3: send/receive/compute</td>
<td>send/receive/compute</td>
<td>...</td>
</tr>
<tr>
<td>Replica 4: send/receive/compute</td>
<td>send/receive/compute</td>
<td>...</td>
</tr>
</tbody>
</table>

a) in every round, a replica executes send/receive/compute

b) every message sent in round $k$ is received in round $k$
Synchronous rounds

a) send post on Monday, receive post on Thursday, and compute on Friday

b) DHL delivers the post in 48 hours

<table>
<thead>
<tr>
<th>Round 1</th>
<th>Round 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replica 1: send/receive/compute</td>
<td>send/receive/compute</td>
</tr>
<tr>
<td>Replica 2: send/receive/compute</td>
<td>send/receive/compute</td>
</tr>
<tr>
<td>Replica 3: send/receive/compute</td>
<td>send/receive/compute</td>
</tr>
<tr>
<td>Replica 4: send/receive/compute</td>
<td>send/receive/compute</td>
</tr>
</tbody>
</table>

a) in every round, a replica executes send/receive/compute

b) every message sent in round $k$ is received in round $k$
Naïve algorithm

1 \textit{round}_1:
2 \textbf{send} \{\textit{my\_value}_i\} \textbf{to} \textbf{ALL}
3 \textbf{receive} \textit{S}_j \textbf{from} \textit{r}_j: \ 1 \leq j \leq N
4 \textit{V}_i := \bigcup_{1 \leq j \leq N} \textit{S}_j
5 \textbf{decide} (\text{min}(\textit{V}_i))
Naïve algorithm

round$_1$:

1. send $\{my\_value_i\}$ to ALL
2. receive $S_j$ from $r_j$: $1 \leq j \leq N$
3. $V_i := \bigcup_{1 \leq j \leq N} S_j$
4. decide(min($V_i$))

\[
my\_value_1 = 10 \quad my\_value_2 = 0 \quad my\_value_3 = 10
\]
Assumptions about faults

- $f$ replicas crash (unknown)

- $t < n$ is an upper bound on $f$ (known)

Igor Konnov 32 of 39
Every replica \( r_i \) for \( i \in \{1, \ldots, N\} \) executes the algorithm:

1. **init**: 
   
   \[ best_i := my\_value_i \]

2. **round** \( k \): \( 1 \leq k \leq t + 1 \)
   
   - **send** \( best_i \) to ALL
   - **receive** \( b_j \) from \( r_j \): \( 1 \leq j \leq N \)
   
   \[ best_i := \min \{ b_1, \ldots, b_N \} \]

3. **if** \( k = t + 1 \) **then** decide(\( best_i \))
Every replica $r_i$ for $i \in \{1, \ldots, N\}$ executes the algorithm:

1. \textit{init}:
   \[ \text{best}_i := \text{my\_value}_i \]

2. \textit{round}_k: $1 \leq k \leq t + 1$
   \[ \text{send} \ \text{best}_i \ \text{to} \ \text{ALL} \]
   \[ \text{receive} \ \text{b}_j \ \text{from} \ r_j: 1 \leq j \leq N \]
   \[ \text{best}_i := \min \ \{\text{b}_1, \ldots, \text{b}_N\} \]

3. \textbf{if} $k = t + 1$ \textbf{then} \text{decide(\text{best}_i)}

\textbf{Termination} \ ✔
Every replica $r_i$ for $i \in \{1, \ldots, N\}$ executes the algorithm:

1. **init**: 
   
   $best_i := my\_value_i$

2. **round**$_k$: $1 \leq k \leq t + 1$
   
   - send $best_i$ to ALL
   - receive $b_j$ from $r_j$: $1 \leq j \leq N$

3. 

   $best_i := \min \{b_1, \ldots, b_N\}$

4. if $k = t + 1$ then decide($best_i$)

Termination ✔️  Validity ✔️

$best_i \in \bigcup_{1 \leq j \leq N} \{my\_value_j\}$
Every replica \( r_i \) for \( i \in \{1, \ldots, N\} \) executes the algorithm:

1. **init:**
   
   \[ best_i := my\_value_i \]

2. **round** \( k \): \( 1 \leq k \leq t + 1 \)
   
   - **send** \( best_i \) to ALL
   - **receive** \( b_j \) from \( r_j \): \( 1 \leq j \leq N \)
   - \( best_i := \min \{b_1, \ldots, b_N\} \)
   - **if** \( k = t + 1 \) then decide(\( best_i \))

**Termination** ✓  **Validity** ✓  **Agreement** ?

\[
best_i \in \bigcup_{1 \leq j \leq N} \{my\_value_j\}
\]
4 \text{round}_k: 1 \leq k \leq t + 1 \\
5 \text{send } \text{best}_i \text{ to } \text{ALL} \\
6 \text{receive } b_j \text{ from } r_j: 1 \leq j \leq N \\
7 \text{best}_i := \min \{b_1, \ldots, b_N\} \\
8 \text{if } k = t + 1 \text{ then } \text{decide}(\text{best}_i)

Assume \textbf{agreement} is violated:
- Two replicas \(r_i\) and \(r_j\) call decide(v_i) and decide(v_j) in line 8
- assume \(v_i < v_j\)
- \(r_j\) never received \(v_i\) in line 6
- by assumption, there are most \(t\) crashes
- hence, no crashes happen in some round \(m \leq t + 1\)
- each replica receives best_1, \ldots, best_N in round \(m\) (lines 5–7)
- hence, if \(r_i\) received \(v_i\), then \(r_j\) received \(v_i\) in round \(m\)
Proving agreement (pencil & paper)

4 \textit{round}_k : 1 \leq k \leq t + 1
5 \textbf{send} \textit{best}_i \textbf{to ALL}
6 \textbf{receive} \textit{b}_j \textbf{from} \textit{r}_j : 1 \leq j \leq N
7 \textit{best}_i := \min \{b_1, \ldots, b_N\}
8 \textbf{if} k = t + 1 \textbf{then} \textbf{decide}(\textit{best}_i)

Assume \textbf{agreement} is violated:

- Two replicas \textit{r}_i and \textit{r}_j call \textbf{decide}(\textit{v}_i) and \textbf{decide}(\textit{v}_j) in line 8
- assume \textit{v}_i < \textit{v}_j
- \textit{r}_j never received \textit{v}_i in line 6
- by assumption, there are most \( t \) crashes
- hence, no crashes happen in some round \( m \leq t + 1 \)
- each replica receives \textit{best}_1, \ldots, \textit{best}_N in round \( m \) (lines 5–7)
- hence, if \( r_i \) received \textit{v}_i, then \( r_j \) received \textit{v}_i in round \( m \) ☑
fewer constraints?
Asynchronous systems

$r_1$ sends/receives on Monday/Thursday, computes on Friday

$r_2$ sends/receives/computes once a month

$r_3$ went for a two-month vacation

$r_4$ left job without notice

$r_1$ uses DHL, $r_2$ uses LA POSTE, $r_3$ uses Post
Consensus in asynchronous systems

Various processor speeds

Various message delays, unbounded but finite

Consensus is not solvable [Fischer, Lynch, Paterson, 1985]

Practical consensus algorithms:

- termination is the engineering problem, Paxos
- or restrict asynchrony, DLS88, Tendermint
- or prove almost-sure termination Ben-Or
Consensus in asynchronous systems

Various processor speeds

Various message delays, unbounded but finite

Consensus is not solvable [Fischer, Lynch, Paterson, 1985]

Practical consensus algorithms:

- termination is the engineering problem, 
  \textbf{Paxos}

- or restrict asynchrony, 
  \textbf{DLS88, Tendermint}

- or prove almost-sure termination 
  \textbf{Ben-Or}
Beyond crashes

What if some replicas lie?

This is **Byzantine** behavior [Lamport, Shostak, Pease, 1982]

More than two-thirds must be correct: $n > 3t$

e.g., Tendermint
Beyond crashes

What if some replicas lie?

This is Byzantine behavior

More than two-thirds must be correct: $n > 3t$

[λamport, Shostak, Pease, 1982]

e.g., Tendermint

Igor Konnov
Conclusions for Part I

Distributed consensus provides fault tolerance

Interaction of multiple peers, fraction of them faulty

Various assumptions about computations

Are the fault-tolerant algorithms bug-free?
Model checking of distributed algorithms:
from classics towards Tendermint blockchain

part II

Igor Konnov

VMCAI winter school, January 16-18, 2020
Timeline

- Introduction to **fault-tolerant** distributed algorithms
- Verifying **synchronous** threshold-guarded algorithms
- Verifying **asynchronous** threshold-guarded algorithms
- Can we verify **Tendermint consensus**?
Verifying **synchronous** threshold-guarded distributed algorithms

[Stoilkovska, K., Widder, Zuleger. TACAS 2019]
Formalizing pseudo-code with threshold automata

Recall FloodMin:

1. **init:**
   \[ \text{best}_i := \text{my\_value}_i \]

2. **round** \( k \): \( 1 \leq k \leq t + 1 \)
   - send \( \text{best}_i \) to ALL
   - receive \( b_j \) from \( r_j \): \( 1 \leq j \leq N \)
   - \( \text{best}_i := \min \{ b_1, \ldots, b_N \} \)
   - if \( k = t + 1 \) then decide(\( \text{best}_i \))

\[ \phi_1 \equiv \#\{V_0, C_0\} > 0 \]
\[ \phi_2 \equiv \#\{V_0\} = 0 \]
Formalizing pseudo-code with threshold automata

Recall FloodMin:

init:

\[ \text{best}_i := \text{my\_value}_i \]

round<sub>k</sub>: \( 1 \leq k \leq t + 1 \)

send best<sub>i</sub> to ALL

receive \( b_j \) from \( r_j \): \( 1 \leq j \leq N \)

\[ \text{best}_i := \min \{ b_1, \ldots, b_N \} \]

if \( k = t + 1 \) then decide(best<sub>i</sub>)

\[ \phi_1 \equiv \#\{\text{v0, c0}\} > 0 \]

\[ \phi_2 \equiv \#\{\text{v0}\} = 0 \]
Recall FloodMin:

\[ \text{init:} \]
\[ \text{best}_i := \text{my\_value}_i \]

\[ \text{round}_k: 1 \leq k \leq t + 1 \]
\[ \text{send } \text{best}_i \text{ to ALL} \]
\[ \text{receive } b_j \text{ from } r_j: 1 \leq j \leq N \]
\[ \text{best}_i := \min \{b_1, \ldots, b_N\} \]
\[ \text{if } k = t + 1 \text{ then } \text{decide} (\text{best}_i) \]

\[
\phi_1 \equiv \# \{V_0, C_0\} > 0 \]
\[
\phi_2 \equiv \# \{V_0\} = 0
\]
Semantics of synchronous threshold automata

\[ \begin{align*}
  \phi_1 \text{ is } #\{v_0, c_0\} &> 0 \\
  \phi_2 \text{ is } #\{v_0\} &= 0
\end{align*} \]

Counter system: \((\Sigma, I, T)\)
Semantics of synchronous threshold automata

Counter system: \((Σ, I, T)\)

\[ \phi_1 \text{ is } \#\{V_0, C_0\} > 0 \]
\[ \phi_2 \text{ is } \#\{V_0\} = 0 \]

\[ n = 5, \quad t = 2, \quad f = 2 \]

\[ \tau(r_1) + \cdots + \tau(r_9) = n \]
Semantics of synchronous threshold automata

Counter system: \((\Sigma, I, T)\)

\[ n = 5, \quad t = 2, \quad f = 2 \]

\[ \tau(r_2) = 1 \]
\[ \tau(r_3) = 2 \]
\[ \tau(r_5) = 1 \]
\[ \tau(r_7) = 1 \]

\[ \tau(r_1) + \cdots + \tau(r_9) = n \]

\( \phi_1 \) is \( \#\{v_0, c_0\} > 0 \)
\( \phi_2 \) is \( \#\{v_0\} = 0 \)
Semantics of synchronous threshold automata

Counter system: \((\Sigma, I, T)\)

\(\phi_1\) is \(#\{v_0, c_0\} > 0\)

\(\phi_2\) is \(#\{v_0\} = 0\)

\(n = 5, \quad t = 2, \quad f = 2\)

\(\tau(r_2) = 1\)

\(\tau(r_3) = 2\)

\(\tau(r_5) = 1\)

\(\tau(r_7) = 1\)

\(\tau(r_1) + \cdots + \tau(r_9) = n\)
An execution of the counter system

A configuration is a tuple of counters $\kappa_{V0}, \kappa_{V1}, \kappa_{SE}, \kappa_{AC}$

An execution is a sequence of configurations (related by transitions)
An execution of the counter system

A configuration is a tuple of counters $\kappa_{V0}$, $\kappa_{V1}$, $\kappa_{SE}$, $\kappa_{AC}$

An execution is a sequence of configurations

(related by transitions)
Can we verify safety?

e.g., agreement
\( \forall n, t, f \) satisfying the resilience condition (e.g., \( n > t \))

\[
\begin{align*}
P(n, t) \parallel P(n, t) \parallel \ldots \parallel P(n, t) \parallel \text{Faulty} \parallel \ldots \parallel \text{Faulty} & \models \varphi \\
n - f & \text{correct} \\
f & \text{faulty}
\end{align*}
\]
Parameterized reachability

Input:
- synchronous threshold automaton TA
- Boolean formula \( \phi \) over counter equalities \( \sum_{\ell \in L} \kappa[\ell] \geq a \cdot p + b \)

Problem:
- find an initial configuration \( \sigma_{\text{init}} \) and a final configuration \( \sigma_{\text{fin}} \)
- there is an exection from \( \sigma_{\text{init}} \) to \( \sigma_{\text{fin}} \)
- formula \( \phi \) holds in \( \sigma_{\text{fin}} \)
Parameterized reachability for STA is undecidable

Reduction to non-halting of a two-counter machine

control flow processes

register processes

\[ \ell_{\text{stuck}} \]

\[ \ell_{1} \]

\[ \ell_{j} \]

\[ \ell_{j+1} \]

\[ \ell_{m} = \ell_{\text{halt}} \]

\[ \#\{\ell_{A}\} \neq 1 \]

\[ \#\{\ell_{j}\} > 0 \]
Parameterized reachability for STA is undecidable

Reduction to non-halting of a two-counter machine

```latex
\ell_m = \ell_{halt}
\ell_j \rightarrow \ell_{j+1}
\#\{\ell_A\} = 1
\#\{\ell_i\} \neq 1
\ell_{stuck}
\ell_i \rightarrow \ell_{d_A}
\ell_{store}
\#\{\ell_j\} > 0
\ell_A \rightarrow \ell_B
\ell^* \rightarrow \ell_j
\ell_{halt} \rightarrow \ell_A \rightarrow \ell_B
```

```latex
\text{inst}_j : \text{inc } A
```

Igor Konnov
Parameterized reachability for STA is undecidable

Reduction to non-halting of a two-counter machine

control flow
processes

\[ \ell_1 \rightarrow \ell_j \rightarrow \ell_{j+1} \rightarrow \ldots \rightarrow \ell_{\text{stuck}} \]

\[ \#\{\ell_i^j\} \neq 1 \quad \#\{\ell_i^j\} = 1 \]

\[ \ell_j : \text{inc } A \]

\[ \ell_j : \text{zero}_k(A) \quad \#\{\ell_A\} = 0 \]

register
processes

\[ \ell^i_A \rightarrow \ell_A \rightarrow \ell_A^d \rightarrow \ldots \rightarrow \ell_B \]

\[ \#\{\ell_i\} > 0 \quad \#\{\ell_{\text{halt}}\} > 0 \]

\[ \ell_m = \ell_{\text{halt}} \]

iff 2CM halts

Igor Konnov 10 of 20
Semi-decision procedure
Long vs. short executions

\[
\sigma_0 \rightarrow \sigma_1 \rightarrow \sigma_2 \rightarrow \sigma_3 \rightarrow \sigma_{t+1}
\]

V0 \rightarrow V1
C0 \rightarrow C1

r4 \rightarrow r2 \rightarrow r3 \rightarrow r4 \rightarrow r2 \rightarrow r3 \rightarrow r7 \rightarrow r2 \rightarrow r3 \rightarrow r7 \rightarrow r9

\sigma_0 \rightarrow \sigma'_1 \rightarrow \sigma'_2

v0 \rightarrow v1
C0 \rightarrow C1

r4 \rightarrow r2 \rightarrow r3 \rightarrow r4 \rightarrow r2 \rightarrow r3 \rightarrow r7 \rightarrow r4 \rightarrow r2 \rightarrow r3 \rightarrow r7 \rightarrow r9

r_1 : true
r_2 : \phi_1
r_3 : \phi_2
r_4 : true
r_5 : \phi_1
r_6 : \phi_2
r_7 : true
r_8 : true
r_9 : true
Long vs. short executions

\[
\begin{align*}
\sigma_0 & \quad \sigma_1 & \quad \sigma_2 & \quad \sigma_3 & \quad \sigma_{t+1} \\
V_0 & \quad V_1 & \quad C_0 & \quad C_1
\end{align*}
\]

\[
\begin{align*}
r_2 & \quad r_2 & \quad r_2 & \quad r_2 & \quad \cdots \\
r_4 & \quad r_4 & \quad r_4 & \quad r_4 & \\
r_3 & \quad r_3 & \quad r_3 & \\
r_7 & \quad r_7 & \\
r_9 & \\
\end{align*}
\]
Bounded executions for reachability

Is there a number $d$ such that we can always shorten executions to executions of length $\leq d$?

Yes, for several textbook algorithms
Is there a number $d$ such that we can always shorten executions to executions of length $\leq d$?

Yes, for several textbook algorithms
### Diameters computed with SMT

<table>
<thead>
<tr>
<th>algorithm</th>
<th>locations</th>
<th>resilience condition</th>
<th>d</th>
<th>z3 sec.</th>
<th>cvc4 sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>rb</td>
<td>4</td>
<td>$n &gt; 3t$</td>
<td>2</td>
<td>0.27</td>
<td>0.99</td>
</tr>
<tr>
<td>rb_hybrid</td>
<td>8</td>
<td>$n &gt; 3b + 2s$</td>
<td>2</td>
<td>1.16</td>
<td>37.6</td>
</tr>
<tr>
<td>rb_omit</td>
<td>8</td>
<td>$n &gt; 2t$</td>
<td>2</td>
<td>0.43</td>
<td>2.47</td>
</tr>
<tr>
<td>fair_cons</td>
<td>11</td>
<td>$n &gt; t$</td>
<td>2</td>
<td>0.97</td>
<td>10.9</td>
</tr>
<tr>
<td>floodmin, $k = 1$</td>
<td>5</td>
<td>$n &gt; t$</td>
<td>2</td>
<td>0.21</td>
<td>0.86</td>
</tr>
<tr>
<td>floodmin, $k = 2$</td>
<td>7</td>
<td>$n &gt; t$</td>
<td>2</td>
<td>0.53</td>
<td>7.43</td>
</tr>
<tr>
<td>floodset</td>
<td>7</td>
<td>$n &gt; t$</td>
<td>2</td>
<td>0.36</td>
<td>3.01</td>
</tr>
<tr>
<td>kset_omit, $k = 1$</td>
<td>4</td>
<td>$n &gt; t$</td>
<td>1</td>
<td>0.08</td>
<td>0.09</td>
</tr>
<tr>
<td>kset_omit, $k = 2$</td>
<td>6</td>
<td>$n &gt; t$</td>
<td>1</td>
<td>0.17</td>
<td>0.27</td>
</tr>
<tr>
<td>phase_king</td>
<td>34</td>
<td>$n &gt; 3t$</td>
<td>4</td>
<td>12.9</td>
<td>50.5</td>
</tr>
<tr>
<td>phase_queen</td>
<td>24</td>
<td>$n &gt; 4t$</td>
<td>3</td>
<td>1.78</td>
<td>17.7</td>
</tr>
</tbody>
</table>

Byzantine, Send Omission, Crash
Computing the diameter $d$

Reach every configuration in a predefined number of steps?

$d$ is the diameter of the system
Safety of synchronous fault-tolerant algorithms

Input STA

Compute diameter

Use BMC

using SMT (Z3)
SMT encoding

d is the diameter bound iff \( \Phi(d) \) holds true:

\[
\forall n, t, f. \ \forall \sigma_0, \ldots, \sigma_{d+1}. \ \exists \sigma'_0, \ldots, \sigma'_d.
\]

\[
\sigma_0 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_{d+1}} \sigma_{d+1} \quad \Rightarrow
\]

\[
(\sigma_0 = \sigma'_0) \land \sigma'_0 \xrightarrow{\tau'_1} \ldots \xrightarrow{\tau'_d} \sigma'_d \land \bigvee_{i=0}^{d} \sigma'_i = \sigma_{d+1}
\]

1. initialize \( d \) to 1
2. check if \( \neg \Phi(d) \) is unsatisfiable
3. if yes, output \( d \) and terminate
4. if no, increment \( d \), jump to step 2
SMT encoding

\[ d \text{ is the diameter bound iff } \Phi(d) \text{ holds true:} \]

\[
\forall n, t, f. \forall \sigma_0, \ldots, \sigma_{d+1}. \exists \sigma'_0, \ldots, \sigma'_d.
\]

\[
\begin{align*}
\sigma_0 &\xrightarrow{\tau_1} \cdots \xrightarrow{\tau_{d+1}} \sigma_{d+1} \\
\sigma'_0 &\xrightarrow{\tau'_1} \cdots \xrightarrow{\tau'_d} \sigma'_d \land \bigvee_{i=0}^{d} \sigma'_i = \sigma_{d+1}
\end{align*}
\]

1. initialize \( d \) to 1
2. check if \( \neg \Phi(d) \) is unsatisfiable
3. if yes, output \( d \) and terminate
4. if no, increment \( d \), jump to step 2
SMT encoding

\( d \) is the diameter bound iff \( \Phi(d) \) holds true:

\[
\forall n, t, f. \ \forall \sigma_0, \ldots, \sigma_{d+1}. \ \exists \sigma'_0, \ldots, \sigma'_d.
\]

\[
\sigma_0 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_{d+1}} \sigma_{d+1} \quad \Rightarrow
\]

\[
(\sigma_0 = \sigma'_0) \land \sigma'_0 \xrightarrow{\tau'_1} \ldots \xrightarrow{\tau'_d} \sigma'_d \land \bigvee_{i=0}^{d} \sigma'_i = \sigma_{d+1}
\]

1. initialize \( d \) to 1
2. check if \( \neg \Phi(d) \) is unsatisfiable
3. if yes, output \( d \) and terminate
4. if no, increment \( d \), jump to step 2
SMT encoding

d is the diameter bound iff $\Phi(d)$ holds true:

\[
\forall n, t, f. \ \forall \sigma_0, \ldots, \sigma_{d+1}. \ \exists \sigma'_0, \ldots, \sigma'_d.
\]

\[
\sigma_0 \xrightarrow{\tau_1} \ldots \xrightarrow{\tau_{d+1}} \sigma_{d+1} \quad \Rightarrow \quad (\sigma_0 = \sigma'_0) \land \sigma'_0 \xrightarrow{\tau'_1} \ldots \xrightarrow{\tau'_d} \sigma'_d \land \bigvee_{i=0}^{d} \sigma'_i = \sigma_{d+1}
\]

1. initialize $d$ to 1
2. check if $\neg \Phi(d)$ is unsatisfiable
3. if yes, output $d$ and terminate
4. if no, increment $d$, jump to step 2

Igor Konnov 17 of 20

LIA
## Bounded model checking with SMT

<table>
<thead>
<tr>
<th>algorithm</th>
<th>locations</th>
<th>RC</th>
<th>z3 sec.</th>
<th>cvc sec.</th>
</tr>
</thead>
<tbody>
<tr>
<td>rb</td>
<td>4</td>
<td>$n &gt; 3t$</td>
<td>0.08</td>
<td>0.08</td>
</tr>
<tr>
<td>rb_hybrid</td>
<td>8</td>
<td>$n &gt; 3b + 2s$</td>
<td>0.09</td>
<td>0.15</td>
</tr>
<tr>
<td>rb_omit</td>
<td>8</td>
<td>$n &gt; 2t$</td>
<td>0.09</td>
<td>0.14</td>
</tr>
<tr>
<td>fair_cons</td>
<td>11</td>
<td>$n &gt; t$</td>
<td>0.27</td>
<td>0.47</td>
</tr>
<tr>
<td>floodmin, $k = 1$</td>
<td>5</td>
<td>$n &gt; t$</td>
<td>0.18</td>
<td>0.29</td>
</tr>
<tr>
<td>floodmin, $k = 2$</td>
<td>7</td>
<td>$n &gt; t$</td>
<td>0.22</td>
<td>0.52</td>
</tr>
<tr>
<td>floodset</td>
<td>7</td>
<td>$n &gt; t$</td>
<td>0.21</td>
<td>0.49</td>
</tr>
<tr>
<td>kset_omit, $k = 1$</td>
<td>4</td>
<td>$n &gt; t$</td>
<td>0.04</td>
<td>0.03</td>
</tr>
<tr>
<td>kset_omit, $k = 2$</td>
<td>6</td>
<td>$n &gt; t$</td>
<td>0.04</td>
<td>0.07</td>
</tr>
<tr>
<td>phase_king</td>
<td>34</td>
<td>$n &gt; 3t$</td>
<td>1.41</td>
<td>5.12</td>
</tr>
<tr>
<td>phase_queen</td>
<td>24</td>
<td>$n &gt; 4t$</td>
<td>0.36</td>
<td>1.92</td>
</tr>
</tbody>
</table>

Byzantine, Send Omission, Crash
Actual bug in [BGP89a], corrected in [BGP89b]

for \( k := 1 \) to \( t+1 \) begin
  (* universal exchange *)
  send(V);
  for \( j := 0 \) to \( 1 \) do
    \( C[j] := \) the number of received 0's;
    (* universal exchange 2 *)
  end;
  \( V := D[1] > t; \) (* King's broadcast *)
  if \( k = p \) then send(V);
  if \( D[V] < n-t \) then
    \( V := \) the received message;
end;

Fig. 2. The Phase King protocol: code for processor \( i \).

...al exchanges are needed to achieve this.

2: Phase King solves the Distributed Consensus problem in \( 2t \) rounds and two-bit messages (or \( 4(t+1) \) rounds and single-bit messages).
Actual bug in [BGP89a], corrected in [BGP89b]

Our technique reported a counterexample. Corrected by changing inequality to $\geq$.

Fig. 4. The Phase King protocol: code for processor $i$. 
Conclusions for Part II

Synchronous threshold automata to model the algorithms

Bounded model checking of counter systems

Completeness due to the diameter bounds

Diameters are not always bounded
Model checking of distributed algorithms:
from classics towards Tendermint blockchain

part III

Igor Konnov

VMCAI winter school, January 16-18, 2020
Timeline

- Introduction to fault-tolerant distributed algorithms
- Verifying synchronous threshold-guarded algorithms
- Verifying asynchronous threshold-guarded algorithms
- Can we verify Tendermint consensus?
Verifying **asynchronous** threshold-guarded distributed algorithms

[K., Veith, Widder. CAV’15]
[K., Lazić, Veith, Widder. POPL’17]
[K., Lazić, Veith, Widder. FMSD’17]
[K., Widder. ISoLA’18]

...
Asynchronous systems

\( r_1 \) sends/receives on Monday/Thursday, computes on Friday

\( r_2 \) sends/receives/computes once a month

\( r_3 \) went for a two-month vacation

\( r_4 \) left job without notice

\( r_1 \) uses \( \text{DHL} \), \( r_2 \) uses \( \text{La Poste} \), \( r_3 \) uses \( \text{Post} \)
Fault-tolerant distributed algorithms

- **n** processes send messages **asynchronously**
- **f** processes are faulty (unknown)
- **t** is an upper bound on **f** (known)

Resilience condition on **n**, **t**, and **f**, e.g., \( n > 3t \land t \geq f \geq 0 \)
Faults and communication

Byzantine behavior:

More than two-thirds must be correct: \( n > 3t \)

Communication is **reliable**: if a correct process sends a message \( m \), \( m \) is eventually delivered to all correct processes
Faults and communication

Byzantine behavior: [Lamport, Shostak, Pease, 1982]

\[
\text{propose}(0) \quad \text{propose}(1)
\]

More than two-thirds must be correct: \( n > 3t \) (resilience)

Communication is **reliable**: if a correct process sends a message \( m \), \( m \) is eventually delivered to all correct processes [Fischer, Lynch, Paterson, 1985]
Byzantine model checker

[forsyte.at/software/bymc]
(source code, benchmarks, virtual machines, etc.)

10 parameterized fault-tolerant distributed algorithms:
An example
One-step Byzantine asynchronous consensus

every process starts with a value $v_i \in \{0, 1\}$

**agreement**: no two processes decide differently

**validity**: if a correct process decides on $v$, then $v$ was the initial value of at least one process

**unanimity**: if all correct processes are initialized with $v$, every deciding correct process must decide on $v$

**termination**: all correct processes eventually decide

decide in one communication step, when there are “not too many faults”
One-step Byzantine asynchronous consensus

every process starts with a value \( v_i \in \{0, 1\} \)

**agreement**: no two processes decide differently

**validity**: if a correct process decides on \( v \),
then \( v \) was the initial value of at least one process

**unanimity**: if all correct processes are initialized with \( v \),
every deciding correct process must decide on \( v \)

**termination**: all correct processes eventually decide

decide in one communication step,
when there are “not too many faults”
input $v_p$

send $\langle VOTE, v_p \rangle$ to all processors;

wait until $n - t$ VOTE messages have been received;

if more than $\frac{n + 3t}{2}$ VOTE messages contain the same value $v$
then DECIDE($v$);

if more than $\frac{n - t}{2}$ VOTE messages contain the same value $v$, and there is only one such value $v$
then $v_p \leftarrow v$;

call Underlying-Consensus($v_p$);

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

**fast termination:** when $n > 5t$ and $f = 0$ and $n > 7t$
Formalizing pseudo-code
Many ways to formalize distributed algorithms

**General languages**

- for instance, TLA$^+$
  - *model checking is hard*

**Parametric Promela**

- relatively easy to understand
  - *supported by ByMC via abstraction*

**Threshold automata**

- special input for ByMC
  - *efficient model checking with SMT*
(Asynchronous) threshold automata

\[ \phi_A \land s_1 < \tau_D + s_1 < \tau_D + s_0 + f \geq \tau_U + s_1 + f \geq \tau_U \]

threshold guards, e.g., \( \phi_A \) is defined as \( s_0 + s_1 + f \geq n - t \)

increments of shared variables, e.g., \( s_0++ \)

run \( n - f \) copies provided that there are \( f \leq t \) Byzantine faults and \( n > 3t \)
Verifying the asynchronous algorithms
Verifying these algorithms?

Parameterized verification problem:

\[ \forall n, f. \quad n - f \text{ copies of } \models \varphi \]

Our approach:

(I) Counting processes,
(II) Acceleration,
(III) Bounded model checking, and
(IV) Schemas
Threshold guards (e.g., $s_0 + s_1 + f \geq n - t$) do not use process ids

A transition by a single process:

$$\{ \kappa_{V1} = 4 \land \kappa_{\text{SENT}} = 1 \land s_0 = 1 \}$$

$$\kappa_{V1} -- ; \kappa_{\text{SENT}}++ ; S_0++ ;$$

$$\{ \kappa_{V1} = 3 \land \kappa_{\text{SENT}} = 2 \land s_0 = 2 \}$$
(II) Acceleration

The same transition by unboundedly many processes in one step:

\[ \kappa V_1 \leftarrow 4 \]
\[ \kappa \text{SENT} \leftarrow 4 \]
\[ s_0 \leftarrow 4 \]
\[ \kappa V_1 \rightarrow \]
\[ \kappa \text{SENT} \rightarrow \]
\[ s_0 \rightarrow \]

Acceleration factor can be any natural number \( \delta \)

Acceleration factor can be any natural number \( \delta \)
A transition by $\delta_i$ processes (in linear integer arithmetic):

\[
T(\sigma_i, \sigma_{i+1}, \delta_i) = \begin{cases} 
  \kappa_{\text{v1}}^{i+1} = \kappa_{\text{v1}}^i - \delta_i \land \\
  \kappa_{\text{SENT}}^{i+1} = \kappa_{\text{SENT}}^i + \delta_i \land \\
  s_0^{i+1} = s_0^i + \delta_i
\end{cases}
\]

Execution:

\[
\sigma_0 \to \sigma_1 \to \sigma_2 \to \ldots \to \sigma_{k-1} \to \sigma_k
\]

SMT formula:

\[
T(\sigma_0, \sigma_1, \delta_0) \land T(\sigma_1, \sigma_2, \delta_1) \land \cdots \land T(\sigma_{k-1}, \sigma_k, \delta_{k-1}) \land \text{Spec}
\]

how long should the executions be?
Completeness of bounded model checking

What we want to do:
\[ \models \varphi \iff \models \varphi \]

What we can do:
Complete and efficient BMC for:
- reachability
- safety and liveness

[I., Veith, Widder: CAV’15]
[K., Lazić, Veith, Widder: POPL ’17]

Igor Konnov 19 of 34
threshold guards, e.g., $\phi_A$ is defined as $s_0 + s_1 + f \geq n - t$

increments of shared variables, e.g., $s_0++$

run $n - f$ copies provided that there are $f \leq t$ Byzantine faults and $n > 3t$
Mover analysis

Exploring all bounded executions is inefficient

\[ s_0 + f \geq \tau_1 \]
\[ s_1 + f \geq \tau_2 \]

The argument contains:

- reordering: \( s_0++; s_1++; s_0++ \) becomes \( s_0++; s_0++; s_1++ \)

- acceleration

\[ s_0++; s_0++; s_1++ \) becomes \( s_0 += 2; s_1++ \)
(IV) Schemas — encoding representatives

Schema: \{\text{pre}_1\} \text{ actions}_1 \{\text{post}_1\} \ldots \{\text{pre}_k\} \text{ actions}_k \{\text{post}_k\}

Example:
{}
\text{(V0 \rightarrow SE0)}^{\delta_1} \{s_0 + f \geq \tau_{D0}\} \text{ (V1 \rightarrow SE1)}^{\delta_2} \{\ldots, s_1 + f \geq \tau_{D1}\} \text{ (V0 \rightarrow SE0)}^{\delta_3}, \text{ (V1 \rightarrow SE1)}^{\delta_4} \{\ldots, \phi_A\} \text{ (SE0 \rightarrow D0)}^{\delta_5}, \text{ (SE1 \rightarrow D1)}^{\delta_6} \{\kappa^{6}_{D0} \neq 0 \land \kappa^{6}_{D1} \neq 0\}

SMT solver tries to find: parameters $n, t, f$, acceleration factors $\delta(1), \ldots, \delta(6)$, counters $\kappa^{i}_{D0}, \kappa^{i}_{D1}, \ldots$

(a) the schema does not violate the property (UNSAT), or
(b) there is a counterexample (SAT)
SMT solver tries to find: parameters $n, t, f$, acceleration factors $\delta(1), \ldots, \delta(6)$, counters $\kappa_{D0}^i, \kappa_{D1}^i, \ldots$

(a) the schema does not violate the property (UNSAT), or 
(b) there is a counterexample (SAT)
(IV) Schemas — encoding representatives

Schema: \{\textit{pre}_1\} actions_1 \{\textit{post}_1\} \ldots \{\textit{pre}_k\} actions_k \{\textit{post}_k\}

Example:
\[
\begin{align*}
\{\} & (V_0 \rightarrow SE_0)^{\delta_1} \{s_0 + f \geq \tau_{D_0}\} (V_1 \rightarrow SE_1)^{\delta_2} \{\ldots, s_1 + f \geq \tau_{D_1}\} \\
(V_0 \rightarrow SE_0)^{\delta_3}, (V_1 \rightarrow SE_1)^{\delta_4} & \{\ldots, \phi_A\} (SE_0 \rightarrow D_0)^{\delta_5}, (SE_1 \rightarrow D_1)^{\delta_6} \\
& \{\kappa_{D_0}^6 \neq 0 \land \kappa_{D_1}^6 \neq 0\}
\end{align*}
\]

SMT solver tries to find: parameters \(n, t, f,\) acceleration factors \(\delta(1), \ldots, \delta(6),\) counters \(\kappa_{D_0}^i, \kappa_{D_1}^i, \ldots\)

(a) the schema does not violate the property (UNSAT), or
(b) there is a counterexample (SAT)
(IV) Schemas — encoding representatives

Schema: \{\textit{pre}_1\} \textit{actions}_1 \{\textit{post}_1\} \ldots \{\textit{pre}_k\} \textit{actions}_k \{\textit{post}_k\}

Example:

\{
\} (V0 \rightarrow \text{SE0})^\delta_1 \{s_0 + f \geq \tau_{D0}\} (V1 \rightarrow \text{SE1})^\delta_2 \{\ldots, s_1 + f \geq \tau_{D1}\}

(V0 \rightarrow \text{SE0})^\delta_3, (V1 \rightarrow \text{SE1})^\delta_4 \{\ldots, \phi_A\} (\text{SE0} \rightarrow \text{D0})^\delta_5, (\text{SE1} \rightarrow \text{D1})^\delta_6

\{\kappa^6_{D0} \neq 0 \land \kappa^6_{D1} \neq 0\}

SMT solver tries to find: parameters \(n, t, f,\)
acceleration factors \(\delta(1), \ldots, \delta(6),\)
counters \(\kappa^i_{D0}, \kappa^i_{D1}, \ldots\)

(a) the schema does not violate the property (UNSAT), or
(b) there is a counterexample (SAT)
(IV) Schemas — encoding representatives

Schema: \{pre_1\} actions_1 \{post_1\} \ldots \{pre_k\} actions_k \{post_k\}

Example:

\{
\}
(V0 \rightarrow SE0)_{\delta_1}
\{s_0 + f \geq \tau_{D_0}\}
(V1 \rightarrow SE1)_{\delta_2}
\{\ldots, s_1 + f \geq \tau_{D_1}\}
(V0 \rightarrow SE0)_{\delta_3}, (V1 \rightarrow SE1)_{\delta_4}
\{\ldots, \phi_A\}
(SE0 \rightarrow D0)_{\delta_5}, (SE1 \rightarrow D1)_{\delta_6}
\{\kappa_{D_0}^6 \neq 0 \land \kappa_{D_1}^6 \neq 0\}

SMT solver tries to find: parameters \(n, t, f\), acceleration factors \(\delta(1), \ldots, \delta(6)\), counters \(\kappa_{D_0}^i, \kappa_{D_1}^i, \ldots\)

(a) the schema does not violate the property (UNSAT), or
(b) there is a counterexample (SAT)
From reachability to safety & liveness

A) A temporal logic for bad executions

\[ E (\varphi_1 \land \Box (\varphi_2 \lor \varphi_3)) \]

B) Enumerating shapes of counterexamples

C) Property specific mover analysis

Details in [K., Lazić, Veith, Widder. POPL’17]
Overview of the verification algorithm

Threshold automaton $\rightarrow$ schemas $\{S_1, \ldots, S_k\}$

$\mathbb{Z}_3 \models S_1$
$\mathbb{Z}_3 \models S_2$
$\ldots$
$\mathbb{Z}_3 \models S_k$

sat

counterexample

unsat?
Overview of the verification algorithm

Threshold automaton $\rightarrow$ schemas $\{S_1, \ldots, S_k\}$

\[
\begin{align*}
Z3 &\models S_1 \\
Z3 &\models S_2 \\
\ldots &\\
Z3 &\models S_k
\end{align*}
\]

sat

counterexample

unsat?
Overview of the verification algorithm

Threshold automaton $\rightarrow$ schemas $\{S_1, \ldots, S_k\}$

\[
\begin{align*}
Z_3 &\models S_1 \\
Z_3 &\models S_2 \\
& \quad \vdots \\
Z_3 &\models S_k \\
\end{align*}
\]

Counterexample

unsat?

Vienna Scientific Cluster
Every lasso can be transformed into a bounded one. The bound depends on the process code and the specification, not the parameters.
Experiments
Byzantine model checker

[forsyte.at/software/bymc]
(source code, benchmarks, virtual machines, etc.)

10 parameterized fault-tolerant distributed algorithms:
More threshold guards...

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Condition</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable broadcast</td>
<td>$x \geq t + 1$</td>
<td>[Srikanth, Toueg’86]</td>
</tr>
<tr>
<td></td>
<td>$x \geq n - t$</td>
<td></td>
</tr>
<tr>
<td>Hybrid broadcast</td>
<td>$x \geq t_b + 1$</td>
<td>[Widder, Schmid’07]</td>
</tr>
<tr>
<td></td>
<td>$x \geq n - t_b - t_c$</td>
<td></td>
</tr>
<tr>
<td>Byzantine agreement</td>
<td>$x \geq \lceil \frac{n}{2} \rceil + 1$</td>
<td>[Bracha, Toueg’85]</td>
</tr>
<tr>
<td>Non-blocking atomic commitment</td>
<td>$x \geq n$</td>
<td>[Raynal’97], [Guerraoui’01]</td>
</tr>
<tr>
<td>Condition-based consensus</td>
<td>$x \geq n - t$</td>
<td>[Mostéfaoui, Mourgaya, Parvedy, Raynal’03]</td>
</tr>
<tr>
<td></td>
<td>$x \geq \lceil \frac{n}{2} \rceil + 1$</td>
<td></td>
</tr>
<tr>
<td>Consensus in one communication step</td>
<td>$x \geq n - t$</td>
<td>[Brasileiro, Greve, Mostéfaoui, Raynal’03]</td>
</tr>
<tr>
<td></td>
<td>$x \geq n - 2t$</td>
<td></td>
</tr>
<tr>
<td>Byzantine one-step consensus</td>
<td>$x \geq \lceil \frac{n+3t}{2} \rceil + 1$</td>
<td>[Song, van Renesse’08]</td>
</tr>
</tbody>
</table>

In general, there is a resilience condition, e.g., $n > 3t$, $n > 7t$
## Benchmarks

Each benchmark has two versions:

1. Threshold automaton \textit{hand-written}
2. Promela code \textit{automatic abstraction}

<table>
<thead>
<tr>
<th>Condition-based consensus</th>
<th>Consensus in one comm. step</th>
</tr>
</thead>
<tbody>
<tr>
<td>One-step consensus</td>
<td>BOSCO</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Non-blocking atomic commitment</th>
<th>(2 versions)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable broadcast</td>
<td>Folklore broadcast</td>
</tr>
</tbody>
</table>

Asynchronous Byzantine agreement
Time to check the algorithms

- Promela abstractions
- Threshold automata
Sequential vs. parallel (256 MPI cores)

Time to verify (sec., log2 scale)

Sequential

MPI

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

10000

100

1
sometimes, the number of schemas is smaller than the number of cores (256)
Promela vs. threshold automata: input

Number of automata locations

- BOSCO
- C1CS
- CF1S
- CBC
- ABA
- NBACR
- NBACG
- STRB
- FRB

Legend:
- Orange: Hand-written threshold automata
- Blue: Promela abstractions
Promela vs. threshold automata: input

Number of automata locations

<table>
<thead>
<tr>
<th>Description</th>
<th>Number of Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>BOSCO</td>
<td>2000</td>
</tr>
<tr>
<td>C1CS</td>
<td>1600</td>
</tr>
<tr>
<td>CF1S</td>
<td>1200</td>
</tr>
<tr>
<td>CBC</td>
<td>200</td>
</tr>
<tr>
<td>ABA</td>
<td>50</td>
</tr>
<tr>
<td>NBACR</td>
<td>100</td>
</tr>
<tr>
<td>NBACG</td>
<td>5</td>
</tr>
<tr>
<td>STRB</td>
<td>50</td>
</tr>
<tr>
<td>FRB</td>
<td>5</td>
</tr>
</tbody>
</table>

Hand-written threshold automata:  
Promela abstractions
Conclusions for Part III

Threshold automata to model asynchronous algorithms

Bounded model checking of counter systems

Completeness due to the bounds

...for safety and liveness
Extending threshold automata

<table>
<thead>
<tr>
<th>Standard TA</th>
<th>Increments in loops (NCTA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n &gt; x, x++$</td>
<td>$x++, n \leq x$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Piecewise monotone (PMTA)</th>
<th>Bounded difference (BDTA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n &gt; x^2, x++$</td>
<td>$1 &gt; x - y, x++$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reversible (RTA)</th>
<th>Reversal bounded (RBTA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$1 &gt; x, x++$</td>
<td>Like reversible automata, but increments and decrements of variables may alternate a bounded number of times.</td>
</tr>
<tr>
<td>$1 \leq x, x--$</td>
<td>$1 \leq x - y, y++$</td>
</tr>
</tbody>
</table>
### All flavors of threshold automata

<table>
<thead>
<tr>
<th>Level</th>
<th>Reversals</th>
<th>Canonical</th>
<th>Bounded Diameter</th>
<th>Flattable</th>
<th>Decidable Reachability</th>
<th>Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x$</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>TA</td>
</tr>
<tr>
<td>p.m. $f(x)$</td>
<td>0</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>PMTA</td>
</tr>
<tr>
<td>$x$</td>
<td>$\leq k$</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>RBTA</td>
</tr>
<tr>
<td>$x$</td>
<td>0</td>
<td>×</td>
<td>×</td>
<td>✓</td>
<td>✓</td>
<td>NCTA</td>
</tr>
<tr>
<td>$x - y$</td>
<td>0</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>BDTA</td>
</tr>
<tr>
<td>$x$</td>
<td>$\infty$</td>
<td>✓</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>RTA</td>
</tr>
</tbody>
</table>

Jure Kukovec  I.K.  Josef Widder
Randomized consensus algorithm Ben-Or

bool v := input_value({0, 1});
int r := 1;
while (true) do
    send (R,r,v) to all;
    wait for n - t messages (R,r,*);
    if received (n + t) / 2 messages (R,r,w)
    then send (P,r,w,D) to all;
    else send (P,r,?) to all;
    wait for n - t messages (P,r,*);
    if received at least t + 1
    messages (P,r,w,D) then {
        v := w; /* enough support -> update estimate */
        if received at least (n + t) / 2
        messages (P,r,w,D)
        then decide w; /* strong majority -> decide */
    } else v := random({0,1}); /* unclear -> coin toss */
    r := r + 1;
od

[Ben-Or, PODC 1983]
Randomized consensus algorithm Ben-Or

bool \( v := \text{input}_\text{value}({0, 1}) \);
int \( r := 1 \);
while (true) do
    send (R,r,v) to all;
    wait for \( n - t \) messages (R,r,*);
    if received \( (n + t) / 2 \) messages (R,r,w)
then send (P,r,w,D) to all;
else send (P,r,?) to all;
wait for \( n - t \) messages (P,r,*);
    if received at least \( t + 1 \) messages (P,r,w,D)
then {
    v := w; /* enough support \rightarrow update estimate */
    if received at least \( (n + t) / 2 \) messages (P,r,w,D)
then decide w; /* strong majority \rightarrow decide */
} else \( v := \text{random}({0,1}) \); /* unclear \rightarrow coin toss */
    \( r := r + 1 \);
od

[Ben-Or, PODC 1983]
Randomized consensus algorithm Ben-Or

```plaintext
bool v := input_value({0, 1});
int r := 1;
while (true) do
    send (R,r,v) to all;
    wait for n - t messages (R,r,*)

    if received (n + t) / 2 messages (R,r,w)
    then send (P,r,w,D) to all;
    else send (P,r,?) to all;
    wait for n - t messages (P,r,*);

    if received at least t + 1 messages (P,r,w,D) then {
        v := w; /* enough support -> update estimate */
        if received at least (n + t) / 2 messages (P,r,w,D)
        then decide w; /* strong majority -> decide */
    } else v := random({0,1}); /* unclear -> coin toss */
    r := r + 1;
end

[Ben-Or, PODC 1983]
```
Randomized consensus algorithm Ben-Or

bool v := input_value({0, 1});
int r := 1;
while (true) do
    send (R,r,v) to all;
    wait for n - t messages (R,r,*);

    if received (n + t) / 2 messages (R,r,w)
    then send (P,r,w,D) to all;
    else send (P,r,?) to all;
    wait for n - t messages (P,r,*);

    if received at least t + 1
    messages (P,r,w,D) then {
        v := w; /* enough support -> update estimate */
        if received at least (n + t) / 2
        messages (P,r,w,D)
        then decide w; /* strong majority -> decide */
    } else v := random({0,1}); /* unclear -> coin toss */
    r := r + 1;
od

[Ben-Or, PODC 1983]
No consensus algorithm for asynchronous systems (FLP’85)

Coin toss to break ties: \( \text{value} := \text{random}(\{0, 1\}) \)

Ben-Or’s, Bracha’s consensus, RS-Bosco, \( k \)-set agreement

Compositional reasoning and reduction for multiple rounds

ByMC to reason about a single round
Model checking of distributed algorithms:
from classics towards Tendermint blockchain

part IV

Igor Konnov

VMCAI winter school, January 16-18, 2020
Timeline

- Introduction to **fault-tolerant** distributed algorithms
- Verifying **synchronous** threshold-guarded algorithms
- Verifying **asynchronous** threshold-guarded algorithms
- Can we verify **Tendermint consensus**?
Tendermint consensus

New Height

Propose

Invalid block or not received in time

Valid block

Prevote Nil

Prevote Block

Commit

New Round

+2/3 precommit for

no +2/3 precommit for

Wait for precommits from +2/3

Precommit Nil

no +2/3 prevote for block

Precommit Block

+2/3 prevote for block

Wait for prevotes from +2/3
Algorithm 1 Tendermint consensus algorithm

1. **Initialization:**
   1. \( h_p \leftarrow 0 \)
   2. \( \text{round}_p = 0 \)

3. **step, \( \in \{\text{propose, prevote, precommit}\} \) \( \text{decision}_p \leftarrow \text{nil} \)
4. \( \text{lockedRound}_p \leftarrow \text{nil} \)
5. \( \text{validRound}_p \leftarrow \text{nil} \)
6. \( \text{validValue}_p \leftarrow \text{nil} \)
7. \( \text{proposer} \leftarrow \text{nil} \)

8. **StartRound(round):**
9. \( \text{round} \leftarrow \text{round} + 1 \)
10. **upon start do**
11. \( \text{StartRound}(0) \)
12. **Function**
13. \( \text{StartRound}(\text{round}) \)
14. **upon**
15. \( \text{StepRound}(\text{round}) \)
16. **else**
17. \( \text{schedule} \) \( OnTimeoutPrevote(h_p, \text{round}_p) \) **to be executed after** \( timeoutPrevote(\text{round}_p) \)
18. **upon** \( \{\text{PROPOSAL}, h_p, \text{round}_p, \text{proposer}, \text{validRound}_p\} \)
19. **broadcast** \( \langle \text{PROPOSAL}, h_p, \text{round}_p, \text{proposer}, \text{validRound}_p \rangle \)
20. **else**
21. \( \text{schedule} \) \( OnTimeoutPropose(h_p, \text{round}_p) \) **to be executed after** \( timeoutPropose(\text{round}_p) \)
22. **upon** \( \{\text{PROPOSAL}, h_p, \text{round}_p, v, -1\} \) **from** \( \text{proposer}(h_p, \text{round}_p) \) **while** \( \text{step}_p = \text{propose} \) **do**
23. **if** \( \text{valid}(v) \land (\text{lockedRound}_p = \text{nil} \
\lor \text{validValue}_p = v) \) **then**
24. **broadcast** \( \langle \text{PROPOSAL, PROPOSAL}, h_p, \text{round}_p, \text{proposer}, \text{validRound}_p \rangle \)
25. **else**
26. **broadcast** \( \langle \text{PREVOTE, h_p, \text{round}_p, \text{nil} \rangle \)
27. **step}_p \leftarrow \text{prevote} \)
28. **upon** \( \{\text{PROPOSAL, h_p, \text{round}_p, v, v}\} \) **from** \( \text{proposer}(h_p, \text{round}_p) \) **AND** \( 2f + 1 \) \( \langle \text{PREVOTE, h_p, v, \text{id}(v) \rangle} \) **while** \( \text{step}_p = \text{propose} \land (\text{validValue}_p = v) \) **do**
29. **if** \( \text{valid}(v) \land (\text{lockedRound}_p \leq \text{validRound}_p = v) \) **then**
30. **broadcast** \( \langle \text{PRECOMMIT, h_p, \text{round}_p, \text{id}(v) \rangle} \)
31. **else**
32. **broadcast** \( \langle \text{PREVOTE, h_p, \text{round}_p, \text{nil} \rangle \)
33. **step}_p \leftarrow \text{prevote} \)
34. **upon** \( 2f + 1 \) \( \langle \text{PREVOTE, h_p, \text{round}_p, \text{nil} \rangle} \) **while** \( \text{step}_p = \text{prevote} \) **for the first time**
35. **broadcast** \( \langle \text{PRECOMMIT, h_p, \text{round}_p, \text{nil} \rangle} \)
36. **upon** \( \{\text{PROPOSAL, h_p, \text{round}_p, v, v}\} \) **from** \( \text{proposer}(h_p, \text{round}_p) \) **AND** \( 2f + 1 \) \( \langle \text{PREVOTE, h_p, v, \text{id}(v) \rangle} \) **while** \( \text{valid}(v) \land \text{step}_p = \text{prevote} \) **for the first time**
37. **if** \( \text{step}_p = \text{propose} \) **then**
38. **tickedValue}_p \leftarrow v \)
39. **lockedRound}_p \leftarrow \text{round}_p \)
40. **broadcast** \( \langle \text{PRECOMMIT, h_p, \text{round}_p, \text{id}(v) \rangle} \)
41. **step}_p \leftarrow \text{precommit} \)
42. **validValue}_p \leftarrow v \)
43. **lockedRound}_p \leftarrow \text{round}_p \)
44. **upon** \( 2f + 1 \) \( \langle \text{PREVOTE, h_p, \text{round}_p, \text{nil} \rangle} \) **while** \( \text{step}_p = \text{prevote} \) **do**
45. **broadcast** \( \langle \text{PRECOMMIT, h_p, \text{round}_p, \text{nil} \rangle} \)
46. **step}_p \leftarrow \text{precommit} \)
47. **upon** \( 2f + 1 \) \( \langle \text{PRECOMMIT, h_p, \text{round}_p, \text{nil} \rangle} \) **for the first time**
48. **broadcast** \( \langle \text{PRECOMMIT, h_p, \text{round}_p, \text{nil} \rangle} \)
49. **upon** \( \{\text{PROPOSAL, h_p, v, v}\} \) **from** \( \text{proposer}(h_p, v) \) **AND** \( 2f + 1 \) \( \langle \text{PRECOMMIT, h_p, v, \text{id}(v) \rangle} \) **while** \( \text{decision}_p[h_p] = \text{nil} \) **do**
50. **if** \( \text{valid}(v) \) **then**
51. **decision}_p[h_p] \leftarrow v \)
52. \( h_p \leftarrow h_p + 1 \)
53. **broadcast** \( \langle \text{PRECOMMIT, h_p, \text{round}_p, \text{nil} \rangle} \)
54. **StartRound(0) \)
55. **upon** \( 2f + 1 \) \( \langle \text{*, h_p, \text{round}_p, \text{nil} \rangle} \) **with** \( \text{round} \geq \text{round}_p \) **do**
56. **StartRound(\text{round}) \)
57. **Function**
58. \( OnTimeoutPropose(h_p, \text{round}_p) \)
59. **upon** \( \text{OnTimeoutPrepropose}(h_p, \text{round}_p) \) **do**
60. **if** \( \text{height} = h_p \land \text{round} = \text{round}_p \land \text{step}_p = \text{propose} \) **then**
Challenges for ByMC

Unbounded height of the blockchain

Unbounded number of rounds within one height

Rotating coordinator, breaking symmetry

Partial synchrony to guarantee liveness

Correct processes have more than 2/3 of voting power
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

TLC takes forever...

Run **APALACHE** for fixed parameters
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

TLC takes forever...

Run **APALACHE** for fixed parameters
Symbolic model checker for TLA$^+$

TLA$^+$ \rightarrow \text{Reduction rules} \rightarrow \text{SMT (Z3)}

Focus on distributed algorithms

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

[forSyTe.at/research/apalache/]
What we were doing in the last months...

Specifying several Tendermint protocols in TLA⁺:

- fast synchronization
- light client
- consensus, tuned for fork detection

[github.com/interchainio/verification]
Today's highlights

**Functional Programming features in Scala**
I've been exploring functional programming with Scala and its eco system for the past few months.

Kevin Lawrence in Towards Data Science  ★  6 min read

**How to understand your program’s memory**
When coding in a language like C or C++ you can interact with your memory in a more low-level way. Sometimes…

Tiago Antunes in freeCodeCamp.org  6 min read

**Ethereum Classic (ETC) is currently being 51% attacked**
On 1/5/2019, Coinbase detected a deep reorg of the Ethereum Classic blockchain that included a double spend…

Mark Nespitt in The Coinbase Blog  7 min read
Fork accountability

Detect the peers that caused a fork — violation of agreement

Ran Apalache: 4 peers, 2 faults, fault threshold is 1:

☑️ found **equivocation**, 2 hours

☑️ found **amnesia**, 2 hours

➕ no other scenarios up to 15 steps, 7 CPU cores, 6.5 hours

Proving that no other scenarios exist? … for all parameters?
Fork accountability

Detect the peers that caused a fork — violation of agreement

Ran Apalache: 4 peers, 2 faults, fault threshold is 1:

- ✓ found **equivocation**, 2 hours
- ✓ found **amnesia**, 2 hours
- ✷ no other scenarios up to 15 steps, 7 CPU cores, 6.5 hours

Proving that no other scenarios exist? ... for all parameters?
Conclusions

Reasoning about fault-tolerant algorithms is hard

... but fun!

Practical algorithms are even harder

Threshold guards are everywhere

Specialized tools for narrow classes, e.g., ByMC vs. General tools for broader classes, e.g., Apalache
Future

Supporting as many features as in TLC

TLA⁺ users specify industrial-scale distributed protocols

all kinds of Paxos, Raft, key-value stores, group membership

These are large and complex specifications [Newcombe et al.’14]

Amazon used 80 CPU cores to find a trace of 35 steps

Semi-automated techniques that would get help from the user

Reduction arguments, abstractions, etc.