SMT and POR beat Counter Abstraction
Parameterized Model Checking of Threshold-Based Distributed Algorithms

Igor Konnov  Helmut Veith  Josef Widder

Alpine Verification Meeting

May 4-6, 2015
Why fault-tolerant (FT) distributed algorithms

faults not in the control of system designer

- bit-flips in memory
- power outage
- disconnection from the network
- intruders take control over some computers
Why fault-tolerant (FT) distributed algorithms

faults not in the control of system designer

- bit-flips in memory
- power outage
- disconnection from the network
- intruders take control over some computers

distributed algorithms to make systems more reliable even in the presence of faults

- replicate processes
- exchange messages
- do coordinated computation
- goal: keep replicated processes in “good state”
- $n$ processes communicate by messages

$n$ processes communicate by messages
Fault-tolerant distributed algorithms

- $n$ processes communicate by messages
- all processes know that at most $t$ of them might be faulty
Fault-tolerant distributed algorithms

- $n$ processes communicate by messages
- all processes know that at most $t$ of them might be faulty
- $f$ are actually faulty, e.g., Byzantine
- resilience condition, e.g., $n > 3t \land t \geq f \geq 0$
- no masquerading: the processes know the origin of incoming messages
Distributed algorithms: computational model and faults

The classic model by [Fischer, Lynch, Paterson’85]

Environment:

- Asynchronous processes (no rounds, non-deterministic fair scheduler)
- Reliable asynchronous message passing (non-blocking send and receive)

Faults:

- crashes and clean crashes,
- omission faults,
- symmetric faults,
- Byzantine faults
if initiator then send INIT to all;

while true do
  if received INIT from at least 1 distinct proc.
  then send ECHO to all;

  if received ECHO from at least \( t + 1 \) distinct proc.
     and not sent ECHO before
  then send ECHO to all;

  if received ECHO from at least \( n - t \) distinct proc.
  then accept;

od
Reliable Broadcast: Sample Execution

\[init \geq t + 1 \geq n - t \]

\[accept \geq n - t \]

Igor Konnov
Reliable Broadcast: Sample Execution

init

≥ t

≥ n − t

accept

≥ n − t

accept
Reliable Broadcast: Sample Execution

\[
\text{init} \geq t + 1 \
\]
Reliable Broadcast: Sample Execution

\[ \geq n - t \]
\[ \text{accept} \]

\[ \geq t + 1 \]
Unforgeability: If no correct process sends \texttt{<INIT>} (broadcasts), then no correct process ever accepts.

Verification perspective: check, whether a bad state is reachable.
Unforgeability: If no correct process sends <INIT> (broadcasts), then no correct process ever accepts.

Verification perspective: check, whether a bad state is reachable.
Threshold-based fault-tolerant distributed algorithms

- The parameters \((n, t, f)\) are fixed in each run
- Main loop with the body executed atomically
- Processes are anonymous (no identifiers)

- Receiving messages, counting them and comparing to thresholds, e.g.,
  
  \[
  \text{if received } <\text{ECHO}> \text{ from } t + 1 \text{ distinct processes}
  \]
  
  \[
  \text{then } \ldots
  \]

- Sending messages to all processes, e.g.,
  
  \[
  \text{send } <\text{ECHO}> \text{ to all} \]
1 **Threshold automata (TA):**
formalization of process code using shared variables

2 **Counter systems with acceleration:**
computational model for parameterized systems of TA

3 **Parameterized reachability:**
safety properties stated formally

4 **Counter abstraction and acceleration:**
other approaches

5 **Representatives and schemas:**
parameterized bounded model checking with SMT
Preliminaries
Threshold automata (TA)

Every **correct** process follows the control flow graph \((L, E)\):

Processes move from one location to another along the edges labeled with:

- **Threshold guards**, e.g., \(x \geq (t + 1) - f\) compare a shared variable to a linear combination of parameters.
- **Updates**, e.g., \(x++\) increment shared variables (or do nothing).

(multiple guards and increments are allowed)
Threshold automata (TA)

Every correct process follows the control flow graph \((L, E)\):

\[
x \geq (n - t) - f \mapsto x++
\]

\[
x \geq (t + 1) - f \mapsto x++
\]

\[
true \mapsto x++
\]

\[
x \geq (n - t) - f
\]

Processes move from one location to another along the edges labeled with:

- **Threshold guards**, e.g., \(x \geq (t + 1) - f\) compare a shared variable to a linear combination of parameters.

- **Updates**, e.g., \(x++\) increment shared variables (or do nothing).

(multiple guards and increments are allowed)
Intuition: threshold automata and threshold-based DAs?

Crash faults:
run \( n \) processes,
\[
\ldots \rightarrow \ell_i \quad n_{faulty} < f, \quad n_{faulty}++ \rightarrow \ell_c \quad \text{crashed here}
\]

Byzantine faults:
run \( n - f \) processes,
count messages modulo Byzantine processes, e.g., \( x + f \geq (t + 1) \)

Warning:
This requires preliminary abstraction of message counters [FMCAD'13]
Intuition: threshold automata and threshold-based DAs?

\[
x \geq (n - t) - f \mapsto x++ \\
x \geq (t + 1) - f \mapsto x++ \\
true \mapsto x++
\]

send \(<x>\) to all if received \(<x>\) from at least \(t + 1\) distinct correct processes

Crash faults:

run \(n\) processes,

\[
\ldots \rightarrow \ell_i \rightarrow \ell_c \text{ crashed here}
\]

\(nfaulty < f, nfaulty++\)

Byzantine faults:

run \(n - f\) processes,

count messages modulo Byzantine processes, e.g., \(x + f \geq (t + 1)\)

Warning:

This requires preliminary abstraction of message counters [FMCAD'13]
Intuition: threshold automata and threshold-based DAs?

\[ x \geq (n - t) - f \rightarrow x++ \]

\[ x \geq (t + 1) - f \rightarrow x++ \]

Crash faults:
- run \( n \) processes,
- \( \ldots \rightarrow l_i \rightarrow l_c \)
- \( n_{faulty} < f, n_{faulty}++ \)
- \( l_c \) crashed here

Byzantine faults:
- run \( n - f \) processes,
- count messages modulo Byzantine processes, e.g., \( x + f \geq (t + 1) \)

Warning:
This requires preliminary abstraction of message counters [FMCAD’13]

Igor Konnov 23/64
Recall how processes count messages:

\[
\text{if received } <\text{ECHO}> \text{ from } t + 1 \text{ distinct processes}
\]

The case studies lead us to the natural restrictions on threshold automata:

**Restriction 1:** Every process changes a shared variable at most once

**Restriction 2:** The edges in cycles do not change the shared variables
**Counter system with acceleration!**

**Counter system** is a transition system simulating every system $P(p)^{N(p)}$.

**Configuration** $\sigma = (\kappa, g, p)$:
- $\kappa_i$ counts processes at location $\ell_i$ with $\kappa_1 + \cdots + \kappa_{|L|} = N(p)$,
- $g_j$ is the value of the shared variable $x_j$,
- $p$ are the values of the parameters.

![Diagram of counter system with acceleration]

**One transition $r^1$ (interleaving):**

- $x \geq (n - t) - f \rightarrow x++$
- $x \geq (t + 1) - f \rightarrow x++$
- $\text{true} \rightarrow x++$
- $x \geq (n - t) - f$

**Accelerated transition $r^3$:**

- $x \geq (n - t) - f$
- $\sigma_1 \rightarrow \kappa_1 \geq 1 \rightarrow \sigma_2$
- $\kappa_1--, \kappa_4++, x++$

-Igor Konnov
Reachability and parameterized reachability

**Reachability (fixed parameters):**
Fix the parameters, e.g., $n = 4$, $t = 1$, $f = 1$, $N = n - f = 3$.
Fix configurations $\sigma$ and $\sigma'$ of $P^N$.

**Question:** is $\sigma'$ reachable from $\sigma$ in $P^N$?

**Parameterized reachability:**
Fix properties $S$ and $S'$ on configurations, e.g., $S : \kappa_1 = N(p) = n - f$ and $S' : \kappa_4 \neq 0$.

**Question:** are there parameter values $p$ and configurations $\sigma$, $\sigma'$ of $P^N(p)$:
- parameters $p$ satisfy the resilience condition $RC(p)$,
- $\sigma \models S$ and $\sigma' \models S'$,
- $\sigma'$ is reachable from $\sigma$ in $P^N(p)$. 
Reachability and parameterized reachability

**Reachability (fixed parameters):**
Fix the parameters, e.g., $n = 4$, $t = 1$, $f = 1$, $N = n - f = 3$.
Fix configurations $\sigma$ and $\sigma'$ of $P^N$.

**Question:** is $\sigma'$ reachable from $\sigma$ in $P^N$?

**Parameterized reachability:**
Fix properties $S$ and $S'$ on configurations,
e.g., $S : \kappa_1 = N(p) = n - f$ and $S' : \kappa_4 \neq 0$.

**Question:** are there parameter values $p$ and configurations $\sigma$, $\sigma'$ of $P^N(p)$:
- parameters $p$ satisfy the resilience condition $RC(p)$,
- $\sigma \models S$ and $\sigma' \models S'$,
- $\sigma'$ is reachable from $\sigma$ in $P^N(p)$.
Parameterized reachability: Example 1

![Diagram]

Resilience condition 1: \( n > 3t \) and \( t \geq f \geq 0 \).

Can the faulty processes forge the broadcast by a correct process? that is, can correct processes reach \( l_4 \), if they start at \( l_1 \)? \( \text{NO} \)

\[
(t + 1) - f > 0 = x
\]

\[
(n - t) - f \geq n - t - t > t \geq 0 = x
\]
Parameterized reachability: Example 1

Resilience condition 1: \( n > 3t \) and \( t \geq f \geq 0 \).

Can the faulty processes forge the broadcast by a correct process? that is, can correct processes reach \( \ell_4 \), if they start at \( \ell_1 \)? NO

\[
(t + 1) - f > 0 = x
\]

\[
(n - t) - f \geq n - t - t > t \geq 0 = x
\]
Parameterized reachability: Example 2

Resilience condition 2: \( n > 3t \) and \( t + 1 \geq f \geq 0 \).

Can the faulty processes forge the broadcast by a correct process?

that is, can correct processes reach \( \ell_4 \), if they start at \( \ell_1 \)? **YES**
Parameterized reachability: Example 2

Resilience condition 2: $n > 3t$ and $t + 1 \geq f \geq 0$.

Can the faulty processes forge the broadcast by a correct process? that is, can correct processes reach $\ell_4$, if they start at $\ell_1$? YES
Parameterized reachability:

counter abstraction and acceleration
Way 1: Counter abstraction

Use counter abstraction to get a finite system $\mathcal{A}$.

Counters $\kappa_i$ are mapped to a finite domain $\hat{D}$, e.g.,

- $\{0, 1, \infty\}$ by [Pnueli, Xu, Zuck’02].
- Domain of parametric intervals extracted from thresholds, e.g., $\{[0, 1), [1, t + 1), [t + 1, n - t), [n - t, \infty)\}$, see [FMCAD’13].

Use a finite-state model checker, e.g., NuSMV or Spin

Warning:
Sometimes, abstraction refinement is needed [FMCAD’13]
Way 1: Counter abstraction

Use counter abstraction to get a finite system $\mathcal{A}$.

Counters $\kappa_i$ are mapped to a finite domain $\hat{D}$, e.g.,

- $\{0, 1, \infty\}$ by [Pnueli, Xu, Zuck'02].
- Domain of parametric intervals extracted from thresholds, e.g., $\{[0, 1), [1, t + 1), [t + 1, n - t), [n - t, \infty)\}$, see [FMCAD'13].

Use a finite-state model checker, e.g., NuSMV or Spin

Warning:

Sometimes, abstraction refinement is needed [FMCAD’13]
Bounded diameter

Fix a threshold automaton \( TA \) and a size function \( N \).

**Theorem [CONCUR'14]**

For each \( p \) with \( RC(p) \), the diameter of an accelerated counter system is independent of parameters and is less than or equal to \( |E| \cdot (|C| + 1) + |C| \):

- \( |E| \) is the number of edges in \( TA \) (self-loops excluded).
- \( |C| \) is the number of edge conditions in \( TA \) that can be unlocked (locked) by an edge appearing later (resp. earlier) in the control flow, or by a parallel edge.

In our example:

\( |E| = 4, |C| = 1 \).

Thus, \( d \leq 9 \).
Bounded diameter

Fix a threshold automaton TA and a size function \( N \).

**Theorem [CONCUR'14]**

For each \( p \) with \( RC(p) \), the diameter of an accelerated counter system is independent of parameters and is less than or equal to \( |E| \cdot (|C| + 1) + |C| \):

- \( |E| \) is the number of edges in TA (self-loops excluded).
- \( |C| \) is the number of edge conditions in TA that can be unlocked (locked) by an edge appearing later (resp. earlier) in the control flow, or by a parallel edge.

In our example:

\(|E| = 4, |C| = 1.\)

Thus, \( d \leq 9 \).
Use counter abstraction to get a finite system $\mathcal{A}$.

Counters $\kappa_i$ are mapped to a finite domain $\hat{D}$, e.g.,

- $\{0, 1, \infty\}$ by [Pnueli, Xu, Zuck’02].
- Domain of parametric intervals extracted from thresholds, e.g., $\{[0, 1), [1, t + 1), [t + 1, n - t), [n - t, \infty)\}$, see [FMCAD’13].

Once we know the diameter $d$ of the accelerated counter system, we know the diameter of the abstract system:

$$diam(\mathcal{A}) \leq d \cdot (|\hat{D}| - 1)$$
Threshold automata are a special case of counter automata.

Apply symbolic acceleration techniques for counter automata, e.g., FAST [Bardin, Finkel, Leroux et al.'08].

The diameter bound implies that the threshold automata are flattable.

Thus, FAST always terminates on threshold automata (in theory).
Accelerated systems: partial order reduction and SMT
Partial orders and SMT beat counter abstraction

Time to verify an instance, sec. (logscale)

Number of checked benchmarks

SMT
SAT
BDD
FAST

Igor Konnov
Partial orders and SMT beat counter abstraction (2)

Memory to verify an instance, MB (logscale)

Number of checked benchmarks

Memory to verify an instance, MB (logscale)

SMT
SAT
BDD
FAST

Igor Konnov
Our new solution

Our new solution consists of the key ingredients:

**Contexts**: In every execution, evaluation of a guard changes at most once.

  e.g., $x \geq t + 1 - f$ is initially false and later turns to true.

  A context keeps track of all unlocked guards.

**Representatives**: As before, transform every execution to a representative by reordering and accelerating the rules with the same context.

  the schedule $r_1^1 r_2^1 r_1^1 r_2^1$ becomes $r_1^2 r_2^3$.

**Schemas**: Representatives are generated by schemas.

  e.g., $r_1 r_2$ generates schedule $r_1^2 r_2^3$ by picking acceleration factors 2 and 3.
Our new solution

Our new solution consists of the key ingredients:

**Contexts**: In every execution, evaluation of a guard changes at most once.  

E.g., \( x \geq t + 1 - f \) is initially false and later turns to true. 

A *context* keeps track of all unlocked guards.

**Representatives**: As before, transform every execution to a representative by reordering and accelerating the rules with the same context.

The schedule \( r_1^1 r_2^1 r_1^1 r_2^1 \) becomes \( r_1^2 r_2^3 \).

**Schemas**: Representatives are generated by schemas.

E.g., \( r_1 r_2 \) generates schedule \( r_1^2 r_2^3 \) by picking acceleration factors 2 and 3.

Igor Konnov
Our new solution consists of the key ingredients:

**Contexts**: In every execution, evaluation of a guard changes at most once.

For example, $x \geq t + 1 - f$ is initially false and later turns to true. A context keeps track of all unlocked guards.

**Representatives**: As before, transform every execution to a representative by reordering and accelerating the rules with the same context.

The schedule $r_1^1 r_2^1 r_1^1 r_2^1 r_2^1$ becomes $r_1^2 r_2^3$.

**Schemas**: Representatives are generated by schemas.

For example, $r_1 r_2$ generates schedule $r_1^2 r_2^3$ by picking acceleration factors 2 and 3.
Our new solution

Our new solution consists of the key ingredients:

**Contexts:** In every execution, evaluation of a guard changes at most once.  

E.g., \( x \geq t + 1 - f \) is initially false and later turns to true.  

A *context* keeps track of all unlocked guards.

**Representatives:** As before, transform every execution to a representative by reordering and accelerating the rules with the same context.

The schedule \( r_1^1 r_2^1 r_1^1 r_2^1 r_2^1 \) becomes \( r_1^2 r_2^3 \).

**Schemas:** Representatives are generated by schemas.

E.g., \( r_1 r_2 \) generates schedule \( r_1^2 r_2^3 \) by picking acceleration factors 2 and 3.

*offline partial order reduction*
Contexts and representatives
Φ is the set of all threshold guards of TA, e.g., \( \Phi = \{ \varphi_1, \varphi_2 \} \)

A subset \( \Omega \subseteq \Phi \) is a **context**, e.g., \( \emptyset \), \( \{ \varphi_1 \} \), and \( \{ \varphi_1, \varphi_2 \} \) are contexts
Every execution defines a monotonically increasing sequence of contexts: e.g., for a configuration $\sigma$ with $n = 5$, $t = 1$, $f = 1$ and $\kappa_1 = 1$, $\kappa_2 = 3$

Transitions $r_1, r_1, r_1, r_1, r_1$ applied to $\sigma$ define the sequence of contexts

$$
\emptyset \subset \{\varphi_1\} \subset \{\varphi_1, \varphi_2\}.
$$

Or, annotated, 

$$
\emptyset \ {\{\varphi_1\}} \ {r_1} \ {r_1} \ {r_1} \ {\{\varphi_1, \varphi_2\}} \ {r_4} \ {\{\varphi_1, \varphi_2\}}
$$
Constructing short representatives

\[ r_3 : \varphi_2 \mapsto x++ \]
\[ r_2 : \varphi_1 \mapsto x++ \]
\[ r_1 : tt \mapsto x++ \]
\[ r_4 : \varphi_2 \]

\[ \varphi_1 \equiv x \geq t + 1, \quad \varphi_2 \equiv x \geq n - t \]

\[ \{} r_1^1 \{ \varphi_1 \} r_1^1, r_1^2, r_1^1 \{ \varphi_1, \varphi_2 \} r_4^1 \{ \varphi_1, \varphi_2 \} \]

the transitions with the same context are sorted, e.g., if \( r_1 \preceq^\text{lin} r_2 \preceq^\text{lin} r_4 \):

\[ \{} r_1^1 \{ \varphi_1 \} r_1^1, r_1^2, r_1^1 \{ \varphi_1, \varphi_2 \} r_4^1 \{ \varphi_1, \varphi_2 \} \]

and the instances of the same rule are accelerated:

\[ \{} r_1^1 \{ \varphi_1 \} r_1^2, r_2^1 \{ \varphi_1, \varphi_2 \} r_4^1 \{ \varphi_1, \varphi_2 \} \]
By applying sorting and acceleration, we prove:

**Proposition 9 [CAV’15]**

Given a threshold automaton, a configuration $\sigma$, and schedule $\tau$ applicable to $\sigma$, there exists a schedule $\text{rep}[\sigma, \tau]$ with the following properties:

1. $\text{rep}[\sigma, \tau]$ is applicable to $\sigma$, and $\text{rep}[\sigma, \tau](\sigma) = \tau(\sigma)$,
2. $|\text{rep}[\sigma, \tau]| \leq 2 \cdot |\mathcal{R}| \cdot (|\Phi| + 1) + |\Phi|$.

where

- $\mathcal{R}$ is the set of rules (edges of TA),
- $\Phi$ is the set of all threshold guards used in $\mathcal{R}$. 
Schemas

(the new ingredient)
What can we do with the representatives?

To check reachability, we have to explore all the representatives.

For a monotonically increasing sequence of contexts,

e.g., $\emptyset$, $\{\varphi_1\}$, $\{\varphi_1, \varphi_2\}$

all representatives follow the same pattern:

$$\emptyset \overset{r_1}{\rightarrow} \{\varphi_1\} \overset{r_1}{\rightarrow} r_1, r_2 \overset{\{\varphi_1, \varphi_2\}}{\rightarrow} r_1, r_2, r_3, r_4 \overset{\{\varphi_1, \varphi_2\}}{\rightarrow}$$
A schema is a sequence of contexts and rule sequences:

\[ S = \{\Omega_0\} \rho_1 \{\Omega_1\} \ldots \{\Omega_{m-1}\} \rho_m \{\Omega_m\} \]

A schema generates paths (including the representatives):

e.g., \{\} \ r_1 \ {\varphi_1} \ r_1, r_3, r_4 \ {\varphi_1, \varphi_2} \\

generates

\{\} \ r_1^2 \ {\varphi_1} \ r_1^1, r_3^3, r_4^3 \ {\varphi_1, \varphi_2} \\

\{\} \ r_1^2 \ {\varphi_1} \ r_1^0, r_3^0, r_4^2 \ {\varphi_1, \varphi_2} \\

How to find a feasible path that reaches a bad state?
A **schema** is a sequence of contexts and rule sequences:

\[ S = \{\Omega_0\} \rho_1 \{\Omega_1\} \ldots \{\Omega_{m-1}\} \rho_m \{\Omega_m\} \]

A schema **generates** paths (including the representatives):

e.g., \{\} \ r_1 \ \{\varphi_1\} \ r_1, r_3, r_4 \ \{\varphi_1, \varphi_2\}

generates

\{\} \ r_1^2 \ \{\varphi_1\} \ r_1^1, r_3^3, r_4^3 \ \{\varphi_1, \varphi_2\}

\{\} \ r_1^2 \ \{\varphi_1\} \ r_1^0, r_3^0, r_4^2 \ \{\varphi_1, \varphi_2\}

How to find a feasible path that reaches a bad state?
A **schema** is a sequence of contexts and rule sequences:

$$S = \{\Omega_0\} \rho_1 \{\Omega_1\} \ldots \{\Omega_{m-1}\} \rho_m \{\Omega_m\}$$

A schema **generates** paths (including the representatives):

- e.g., $$\{\} \ r_1 \ \{\varphi_1\} \ r_1, r_3, r_4 \ \{\varphi_1, \varphi_2\}$$

  generates

$$\{\} \ r_1^2 \ \{\varphi_1\} \ r_1^1, r_3^3, r_4^3 \ \{\varphi_1, \varphi_2\}$$

$$\{\} \ r_1^2 \ \{\varphi_1\} \ r_1^0, r_3^0, r_4^2 \ \{\varphi_1, \varphi_2\}$$

**How to find a feasible path that reaches a bad state?**
Checking feasibility with SMT

It is easy to check with SMT, whether a schema generates a feasible path:

\[
\{\} \rightarrow r_1 \{\varphi_1\} \rightarrow r_2 \{\varphi_1, \varphi_2\} \rightarrow r_4 \{\varphi_1, \varphi_2\}
\]

<table>
<thead>
<tr>
<th>$\kappa_1$</th>
<th>$\kappa_1^0 = n - f$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\kappa_2$</td>
<td>$\kappa_2^0 = 0$</td>
</tr>
<tr>
<td>$\kappa_3$</td>
<td>$\kappa_3^0 = 0$</td>
</tr>
<tr>
<td>$\kappa_4$</td>
<td>$\kappa_4^0 = 0$</td>
</tr>
<tr>
<td>$x$</td>
<td>$x^0 = 0$</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\kappa_1^1 & = \kappa_1^0 - \delta_1 \\
\kappa_2^1 & = \kappa_2^0 - \delta_1 \\
\kappa_3^1 & = \kappa_3^0 + \delta_1 \\
\kappa_4^1 & = \kappa_4^0 + \delta_1 \\
x^1 & = x^0 + \delta_1 \\
x^2 & = x^2 + \delta_2
\end{align*}
\]

\[
\begin{align*}
\kappa_1^2 & = \kappa_1^0 - \delta_2 \\
\kappa_2^2 & = \kappa_2^1 + \delta_2 \\
\kappa_3^2 & = \kappa_3^1 + \delta_2 \\
\kappa_4^2 & = \kappa_4^0 + \delta_3 \\
x^1 & \geq (t + 1) - f \\
x^2 & \geq (n - t) - f \\
\kappa_4^3 & = n - f
\end{align*}
\]
Complete parameterized reachability checking

Sound and complete algorithm for parameterized reachability in TA:

For each monotonically increasing sequence $\Omega$ of contexts:
- construct a schema $S$ for $\Omega$
- if there is a path $\pi$ generated by $S$ that reaches a bad state,
  then report $\pi$ as a counterexample

Theorem 1 [CAV’15]
For a threshold automaton, there is a complete schema set of cardinality at most $|\Phi|!$, where the length of each schema does not exceed $(3 \cdot |\Phi| + 2) \cdot |\mathcal{R}|$.

Note:
This result also holds for the guards like $nfaulty < f$
Complete parameterized reachability checking

Sound and complete algorithm for parameterized reachability in TA:

For each monotonically increasing sequence $\Omega$ of contexts:
   - construct a schema $S$ for $\Omega$
   - if there is a path $\pi$ generated by $S$ that reaches a bad state,
     then report $\pi$ as a counterexample

Theorem 1 [CAV’15]

For a threshold automaton, there is a complete schema set of cardinality at most $|\Phi|!$, where the length of each schema does not exceed $(3 \cdot |\Phi| + 2) \cdot |R|$. 

Note:
This result also holds for the guards like $nfaulty < f$
Complete parameterized reachability checking

Sound and complete algorithm for parameterized reachability in TA:

For each monotonically increasing sequence $\Omega$ of contexts:
construct a schema $S$ for $\Omega$
if there is a path $\pi$ generated by $S$ that reaches a bad state,
then report $\pi$ as a counterexample

Theorem 1 [CAV’15]
For a threshold automaton, there is a complete schema set of cardinality at most $|\Phi|!$, where the length of each schema does not exceed $(3 \cdot |\Phi| + 2) \cdot |\mathcal{R}|$.

Note:
This result also holds for the guards like $n_{faulty} < f$
Results

Now we can verify **safety** of the **parameterized** algorithms:

- Reliable broadcast (FRB, STRB, ABA)
- Non-blocking atomic commit with failure detectors (NBAC, NBACG)
- Condition-based consensus (CBC)
- One-step consensus (CF1S, C1CS, BOSCO)

<table>
<thead>
<tr>
<th></th>
<th>ABA</th>
<th>STRB</th>
<th>FRB</th>
<th>NBAC</th>
<th>CBC, C1CS</th>
<th>CF1S, FBC</th>
<th>BOSCO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Values</td>
<td>85</td>
<td>87</td>
<td>96</td>
<td>97</td>
<td>01</td>
<td>02</td>
<td>06</td>
</tr>
</tbody>
</table>

**Liveness?**

“...when looking for errors, most of your effort should be devoted to examining the safety part.”


“Liveness is whatever prevents an empty system from being correct.”

Results

Now we can verify safety of the parameterized algorithms:

- Reliable broadcast (FRB, STRB, ABA)
- Non-blocking atomic commit with failure detectors (NBAC, NBACG)
- Condition-based consensus (CBC)
- One-step consensus (CF1S, C1CS, BOSCO)

Liveness?

“...when looking for errors, most of your effort should be devoted to examining the safety part.” Leslie Lamport. Specifying Systems (2002)

“Liveness is whatever prevents an empty system from being correct.” Orna Kupferman. Beyond Safety Workshop (2004)
Conclusions

Standard model checkers are not tuned to the computational models of fault-tolerant distributed algorithms.

Computational primitives in FTDAs are simpler than the standard ones.

This and parameterization helped us to develop efficient techniques to check FTDAs used in the cloud: variations of Paxos, RAFT, etc.?
Conclusions

Standard model checkers are not tuned to the computational models of fault-tolerant distributed algorithms.

Computational primitives in FTDAs are simpler than the standard ones.

This and parameterization helped us to develop efficient techniques.

Check FTDAs used in the cloud: variations of Paxos, RAFT, etc.?
Thank you!

[ http://forsyte.at/software/bymc ]

_SMT and POR beat Counter Abstraction:_
_Parameterized Model Checking of Threshold-Based Distributed Algorithms._

To appear at CAV’15.