A CEGAR approach to parameterized verification of distributed algorithms

Nathalie Bertrand

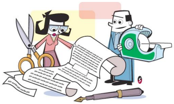
Inria & IRISA

joint work with Ocan Sankur, Bastien Thomas, Josef Widder

ETAPS SynCoP'23 April 23rd 2023

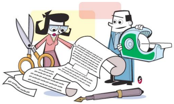
Outline

Model checking distributed systems

Does  satisfy  ?

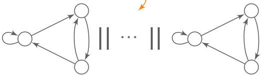
distributed system requirement

Model checking distributed systems

Does  satisfy  ?

distributed system

requirement

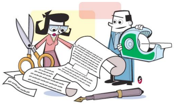


$\mathbf{AG}(\wedge_{i \neq j}(\mathsf{Crit}_i \longrightarrow \neg \mathsf{Crit}_j))$

model

property

Model checking distributed systems

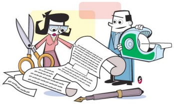
Does  satisfy  ?

distributed system — requirement

 $\models$ $\mathbf{AG}(\bigwedge_{i\neq j}(\mathsf{Crit}_i \longrightarrow \neg\mathsf{Crit}_j))$?

model — model-checking algorithm — property

Model checking distributed systems

Does  satisfy  ?

distributed system — requirement

 $\models$ $\mathbf{AG}(\wedge_{i \neq j}(\mathsf{Crit}_i \longrightarrow \neg \mathsf{Crit}_j))$?

model — model-checking algorithm — property

⚠ Limitations of standard model-checking techniques

- **state-space explosion**: product transition system is exponential in number of processes, and of variables
  → tools hardly scale to large number of processes
- models with **fixed number of processes**
  → yet correctness should be proven for arbitrarily many

Parameterized verification: to infinity and beyond!

from  to 

Parameterized verification: to infinity and beyond!

from  to 

- correctness should hold **for every number of components**

$$\forall n \quad \underbrace{C \,||\cdots||\, C}_{n \text{ times}} \,||\, S \quad \models \quad \varphi$$

- more generally: for every network topologies, for every potential failures $\Longrightarrow$ for every *parameter valuations*

⚠ model checking infinitely many instances at once

Parameterized verification for distributed algorithms

Many models for parameterized verification of distributed algorithms depending on:
communication mechanism, synchrony assumptions, fault model, etc.

- threshold automata [Konnov Lazić Veith Widder POPL'17]
- broadcast protocols [Esparza Finkel Mayr LICS'99] [Delzanno Sangnier Zavattaro Concur'10]
- global sync. protocols [Jaber Jacobs Wagner Kulkarni Samanta CAV'20]
- shared-memory models [Esparza Ganty Majumdar JACM 2016] [Bouyer Markey Randour Sangnier Stan ICALP'16]
- token-passing algorithms on lines/rings [Lin Rümmer CAV'16]
- population protocols [Esparza Ganty Leroux Majumdar Acta Inf. 2017]
- synchronous algorithms on rings [Aiswarya Bollig Gastin I&C 2018]

$\cdots$ and probably more

Outline

An asynchronous round-based consensus algorithm

Ben Or randomized consensus algorithm [Ben Or PODC'83]

- binary consensus robust to Byzantine processes
- n processes communicate by **broadcasts** in **asynchronous rounds**
- t is a known upper bound on unknown number of faulty processes f
- rounds consist of two phases
  processes broadcast their local state (phase, round, preference)

An asynchronous round-based consensus algorithm

Ben Or randomized consensus algorithm [Ben Or PODC'83]

- binary consensus robust to Byzantine processes
- n processes communicate by **broadcasts** in **asynchronous rounds**
- t is a known upper bound on unknown number of faulty processes f
- rounds consist of two phases
  processes broadcast their local state (phase, round, preference)

```
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 v := w;
 else v := ?;
 send (P,r,v) to all;
 wait for n − t messages (P,r,∗);
   if received at least t + 1 messages (P,r,w)
   then {v := w;                  /∗ enough support −> update estimate ∗/
     if received at least (n + t)/2 messages (P,r,w)
     then decide w;}                 /∗ strong majority −> decide ∗/
   else v := random(0, 1) ;     /∗ unclear −> coin toss ∗/
 r := r + 1 ;
od
```

An asynchronous round-based consensus algorithm

Ben Or randomized consensus algorithm [Ben Or PODC'83]

- binary consensus robust to Byzantine processes
- n processes communicate by **broadcasts** in **asynchronous rounds**
- t is a known upper bound on unknown number of faulty processes f
- rounds consist of two phases
  processes broadcast their local state (phase, round, preference) a_v^r, b_v^r, d_v^r

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

Formal semantics of Ben Or's algorithm

state	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	b_1	$\cdot$
	p_2	a_0	$b_?$	a_1	b_1	d_1
	p_3	a_1	$b_?$	a_0	$b_?$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_0)	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	$\cdot$	$\cdot$
	p_2	a_0	$\cdot$	a_1	$\cdot$	d_1
	p_3	a_1	$\cdot$	$\cdot$	$\cdot$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_1)	$\cdots$			$\cdots$		
received(p_2)	$\cdots$			$\cdots$		
received(p_3)	$\cdots$			$\cdots$		
received(p_4)	$\cdots$			$\cdots$		

Full configuration $n = 6, t = 1, f = 1$

stores for each process

- history of local states
- received messages

⚠ full configurations $\neq$ snapshots

⚠ layer indices $\neq$ timestamp

Formal semantics of Ben Or's algorithm

state	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	b_1	$\cdot$
	p_2	a_0	$b_?$	a_1	b_1	d_1
	p_3	a_1	$b_?$	a_0	$b_?$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_0)	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	$\cdot$	$\cdot$
	p_2	a_0	$\cdot$	a_1	$\cdot$	d_1
	p_3	a_1	$\cdot$	$\cdot$	$\cdot$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_1)	$\cdots$			$\cdots$		
received(p_2)	$\cdots$			$\cdots$		
received(p_3)	$\cdots$			$\cdots$		
received(p_4)	$\cdots$			$\cdots$		

state	p_0	a_0	b_0	a_0	b_1	$\cdot$
	p_1	a_0	$b_?$	a_1	b_1	$\cdot$
	p_2	a_0	$b_?$	a_1	b_1	d_1
	p_3	a_1	$b_?$	a_0	$b_?$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_0)	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	$\cdot$	$\cdot$
	p_2	a_0	$\cdot$	a_1	b_1	d_1
	p_3	a_1	$b_?$	$\cdot$	$\cdot$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_1)	$\cdots$			$\cdots$		
received(p_2)	$\cdots$			$\cdots$		
received(p_3)	$\cdots$			$\cdots$		
received(p_4)	$\cdots$			$\cdots$		

Full configuration $n = 6, t = 1, f = 1$

stores for each process

- history of local states
- received messages

⚠ full configurations $\neq$ snapshots

⚠ layer indices $\neq$ timestamp

Step

for one process

- reception of some messages
- state update according to thresholds on received messages
- broadcast of new state

Message abstraction

Full Configuration	$n = 6, t = 1, f = 1$					
state	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	b_1	$\cdot$
	p_2	a_0	$b_?$	a_1	b_1	d_1
	p_3	a_1	$b_?$	a_0	$b_?$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_0)	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	$\cdot$	$\cdot$
	p_2	a_0	$\cdot$	a_1	$\cdot$	d_1
	p_3	a_1	$\cdot$	$\cdot$	$\cdot$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_1)	$\cdots$			$\cdots$		
received(p_2)	$\cdots$			$\cdots$		
received(p_3)	$\cdots$			$\cdots$		
received(p_4)	$\cdots$			$\cdots$		

Message abstraction

Full Configuration $n = 6, t = 1, f = 1$

state	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	b_1	$\cdot$
	p_2	a_0	$b_?$	a_1	b_1	d_1
	p_3	a_1	$b_?$	a_0	$b_?$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_0)	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	$\cdot$	$\cdot$
	p_2	a_0	$\cdot$	a_1	$\cdot$	d_1
	p_3	a_1	$\cdot$	$\cdot$	$\cdot$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_1)	$\cdots$			$\cdots$		
received(p_2)	$\cdots$			$\cdots$		
received(p_3)	$\cdots$			$\cdots$		
received(p_4)	$\cdots$			$\cdots$		

layered hyp.

Succinct Configuration

p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
p_1	a_0	$b_?$	a_1	b_1	$\cdot$
p_2	a_0	$b_?$	a_1	b_1	d_1
p_3	a_1	$b_?$	a_0	$b_?$	$\cdot$
p_4	a_1	$b_?$	a_1	b_1	$\cdot$

Message abstraction

Full Configuration $n = 6, t = 1, f = 1$

state	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	b_1	$\cdot$
	p_2	a_0	$b_?$	a_1	b_1	d_1
	p_3	a_1	$b_?$	a_0	$b_?$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_0)	p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
	p_1	a_0	$b_?$	a_1	$\cdot$	$\cdot$
	p_2	a_0	$\cdot$	a_1	$\cdot$	d_1
	p_3	a_1	$\cdot$	$\cdot$	$\cdot$	$\cdot$
	p_4	a_1	$b_?$	a_1	b_1	$\cdot$
received(p_1)	$\cdots$			$\cdots$		
received(p_2)	$\cdots$			$\cdots$		
received(p_3)	$\cdots$			$\cdots$		
received(p_4)	$\cdots$			$\cdots$		

layered hyp.

Succinct Configuration

p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
p_1	a_0	$b_?$	a_1	b_1	$\cdot$
p_2	a_0	$b_?$	a_1	b_1	d_1
p_3	a_1	$b_?$	a_0	$b_?$	$\cdot$
p_4	a_1	$b_?$	a_1	b_1	$\cdot$

Message abstraction is sound and complete for finite/infinite configurations

Completeness requires that threshold guards involve current-phase messages only

Counting abstraction

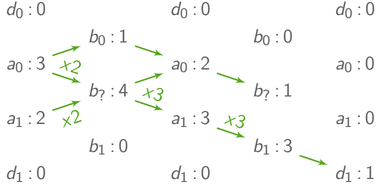
Succinct Configuration					
p_0	a_0	b_0	a_0	$\cdot$	$\cdot$
p_1	a_0	$b_?$	a_1	b_1	$\cdot$
p_2	a_0	$b_?$	a_1	b_1	d_1
p_3	a_1	$b_?$	a_0	$b_?$	$\cdot$
p_4	a_1	$b_?$	a_1	b_1	$\cdot$

Counting abstraction

Succinct Configuration

p_0	a_0	b_0	a_0	·	·
p_1	a_0	$b_?$	a_1	b_1	·
p_2	a_0	$b_?$	a_1	b_1	d_1
p_3	a_1	$b_?$	a_0	$b_?$	·
p_4	a_1	$b_?$	a_1	b_1	·

Counter Configuration, $n = 6,\ t = 1,\ f = 1$

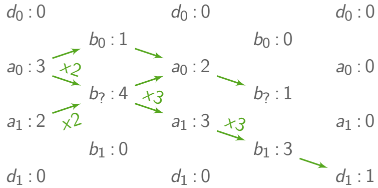


Counting abstraction

Succinct Configuration

p_0	a_0	b_0	a_0	·	·
p_1	a_0	$b_?$	a_1	b_1	·
p_2	a_0	$b_?$	a_1	b_1	d_1
p_3	a_1	$b_?$	a_0	$b_?$	·
p_4	a_1	$b_?$	a_1	b_1	·

Counter Configuration, $n = 6$, $t = 1$, $f = 1$

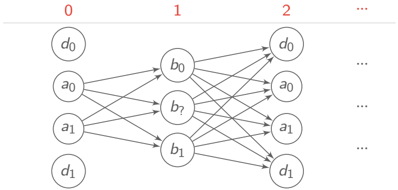


A counter configuration is reachable iff it respects the flow conditions and for positive flows the guards are satisfied.

- one can check the guards *a posteriori* and the order in which guards become true is irrelevant
- flow conditions and guard-coherence can be encoded in linear arithmetic formulas independent on the concrete parameter valuation

Layered threshold automata for counting abstraction

variant of threshold automata [Konnov Veith Widder CAV'15]



LTA represents correct processes

behaviour of Byzantine processes is dealt with in guards

- one model for all processes
- automaton with states arranged in unboundedly many layers
- threshold guards on transitions = constraint on current layer
  - a process can move to b_1 if it receives $\frac{n+t}{2}$ messages $(R, r, 1)$
  - these messages can be sent by processes in a_1 or Byzantine processes
  - $\mathbf{g}(a_0, b_1) = \mathbf{g}(a_1, b_1) := a_1 + f \geq \frac{n+t}{2}$

⚠ Guards are monotonous: once they hold, they hold forever

Outline

Model checking layered threshold automata

Input: an LTA, an LTL property φ (atomic propositions = linear expressions on number of processes in some states)
Output: yes iff for every parameter valuation every reachable full configuration satisfies φ

The parameterized model checking of layered threshold automata is **undecidable**, for **safety** properties already.

Model checking layered threshold automata

Input: an LTA, an LTL property φ (atomic propositions = linear expressions on number of processes in some states)
Output: yes iff for every parameter valuation every reachable full configuration satisfies φ

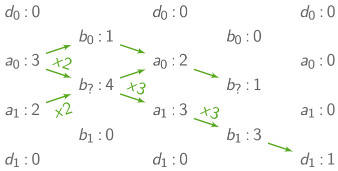
The parameterized model checking of layered threshold automata is **undecidable**, for **safety** properties already.

Our approach:

1. **predicate abstraction**: *guard automaton*
2. **CEGAR** counter-example guided abstraction refinement: abstraction refinement by automated synthesis of new predicates

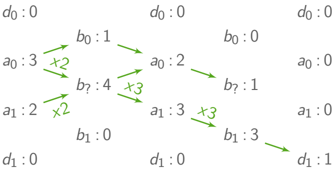
Predicate abstraction and guard automaton

Counter Configuration, $n = 6$, $t = 1$, $f = 1$



Predicate abstraction and guard automaton

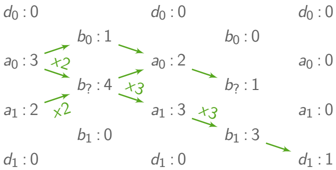
Counter Configuration, $n=6,\ t=1,\ f=1$



Predicate abstraction					
$a_0+d_0>0$	$\top$	–	$\top$	–	$\bot$
$a_1+d_1>0$	$\top$	–	$\top$	–	$\top$
$d_0+d_1>0$	$\bot$	–	$\bot$	–	$\top$
$a_0+a_1+f\le n$	$\top$	–	$\top$	–	$\top$
$b_0>0$	–	$\top$	–	$\bot$	–
$b_1>0$	–	$\top$	–	$\top$	–
$b_?+f\ge n$	–	$\bot$	–	$\bot$	–

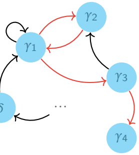
Predicate abstraction and guard automaton

Counter Configuration, $n = 6,\ t = 1,\ f = 1$



Predicate abstraction

$a_0 + d_0 > 0$	$\top$	–	$\top$	–	$\bot$
$a_1 + d_1 > 0$	$\top$	–	$\top$	–	$\top$
$d_0 + d_1 > 0$	$\bot$	–	$\bot$	–	$\top$
$a_0 + a_1 + f \leq n$	$\top$	–	$\top$	–	$\top$
$b_0 > 0$	–	$\top$	–	$\bot$	–
$b_1 > 0$	–	$\top$	–	$\top$	–
$b_? + f \geq n$	–	$\bot$	–	$\bot$	–
	γ_1	γ_2	γ_1	γ_3	γ_4

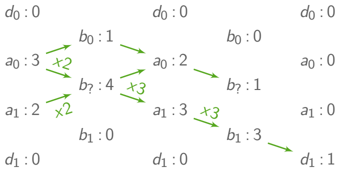


Guard automaton

- states = valuations of predicates
- transitions obtained via queries to SMT solver

Predicate abstraction and guard automaton

Counter Configuration, $n=6$, $t=1$, $f=1$

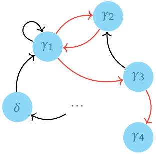


Predicate abstraction

	γ_1	γ_2	γ_1	γ_3	γ_4
$a_0 + d_0 > 0$	$\top$	–	$\top$	–	$\bot$
$a_1 + d_1 > 0$	$\top$	–	$\top$	–	$\top$
$d_0 + d_1 > 0$	$\bot$	–	$\bot$	–	$\top$
$a_0 + a_1 + f \leq n$	$\top$	–	$\top$	–	$\top$
$b_0 > 0$	–	$\top$	–	$\bot$	–
$b_1 > 0$	–	$\top$	–	$\top$	–
$b_? + f \geq n$	–	$\bot$	–	$\bot$	–

Guard automaton

- states = valuations of predicates
- transitions obtained via queries to SMT solver



The language of the guard automaton **overapproximates** the set of executions of the layered threshold automaton.

⚠ incomplete method and depends on the chosen predicates

Counter-example guided abstraction refinement

General principles [Clarke Grumberg Jha Lu Veith JACM'03]

1. generate initial abstraction with fixed set of predicates
2. if abstraction satisfies the property, then return property is **valid**
3. else, check realizability of the abstract counterexample
   - if counterexample can be realized, then return property is **invalid**
   - else, refine the abstraction by adding more predicates to remove spurious counterexample and goto 2.

Counter-example guided abstraction refinement

General principles

[Clarke Grumberg Jha Lu Veith JACM'03]

1. generate initial abstraction with fixed set of predicates
2. if abstraction satisfies the property, then return property is **valid**
3. else, check realizability of the abstract counterexample
   - if counterexample can be realized, then return property is **invalid**
   - else, refine the abstraction by adding more predicates to remove spurious counterexample and goto 2.

Specificities for layered threshold automata

- abstractions are guard automata
- realizability is checked with SMT solver trying to instantiate the parameters and counters to obtain concrete counterexample
- new predicates are obtained by interpolation

[Henzinger Jhala Majumdar McMillan POPL'04]

Outline

Introduction

CEGAR approach for fault-tolerant distributed algorithms

- Modelling broadcast fault-tolerant algorithms
- Model checking layered threshold automata
- Tool implementation: PyLTA

Conclusion

PyLTA Input Language on Ben Or example

Model definition

```
PARAMETERS: n, t, f
```

Define parameters

```
PARAMETER_RELATION: 5 * t < n
PARAMETER_RELATION: f <= t
```

Set resilience conditions

```
LAYERS: R, P, R
```

Define two repeating layers

```
STATES: R.d0, R.0, R.1, R.d1
STATES: P.0, P.u, P.1
```

Define states in each layer

```
CASE R.0:
 IF 2*(R.0 + f) >= n + t THEN P.0
 IF 2*(R.1 + f) >= n + t THEN P.1
 IF R.0 + R.1 + f >= n - t
     & 2*R.0 > n - 3*t & 2*R.1 > n - 3*t THEN P.u
CASE R.1: ...
```

Define guarded transitions

PyLTA Input Language on Ben Or example (2)

Specifying predicates and properties

```
WITH
  R.all0: R.0 + f == n & R.d0 + R.d1 + R.1 ==0
  R.decide1: R.d1 >0
VERIFY: R.all0 -> ! F (R & R.decide1)
```

Validity-0

PyLTA Input Language on Ben Or example (2)

Specifying predicates and properties

```
WITH
  R.all0: R.0 + f == n & R.d0 + R.d1 + R.1 ==0
  R.decide1: R.d1 >0
VERIFY: R.all0 -> ! F (R & R.decide1)
```

Validity-0

```
WITH
  R.initial: R.0 + R.1 == n & R.d0 + R.d1 == 0
  R.decide0: R.d0 > 0
  R.decide1: R.d1 > 0
VERIFY: R.initial -> !( F(R & R.decide0) & F(R & R.decide1) )
```

Agreement

PyLTA Input Language on Ben Or example (2)

Specifying predicates and properties

```
WITH
  R.all0: R.0 + f == n & R.d0 + R.d1 + R.1 ==0
  R.decide1: R.d1 >0
VERIFY: R.all0 -> ! F (R & R.decide1)
```

Validity-0

```
WITH
  R.initial: R.0 + R.1 == n & R.d0 + R.d1 == 0
  R.decide0: R.d0 > 0
  R.decide1: R.d1 > 0
VERIFY: R.initial -> !( F(R & R.decide0) & F(R & R.decide1) )
```

Agreement

```
WITH
  R.initial: R.0 + R.1 + f == n & R.d0 + R.d1 == 0
  R.fair: R.0 + R.1 >= n - t -> (
          R.0 == EDGE(R.0, P.0) + EDGE(R.0, P.u) + EDGE(R.0, P.1) &
          R.1 == EDGE(R.1, P.0) + EDGE(R.1, P.u) + EDGE(R.1, P.1) )
  P.fair: ...
  R.decided: R.d0 > 0 | R.d1 > 0
  R.unbalanced: 2*R.0 >= n + 3*t | 2*R.1 >= n + 3*t
VERIFY:  R.initial & G (R -> R.fair) & G (P -> P.fair)
           & F (R & R.unbalanced) -> F (R & R.decided)
```

Termination
under strong hyp.

PyLTA Implementation and Benchmarks

PyLTA performs counter abstraction, predicate abstraction and CEGAR

Implementation details

- written in Python
- BDD representation of transitions in guard automaton
- SPOT builds Büchi automaton from negation of LTL specification
- MathSat checks realizability of counter examples and produces interpolants to generate new predicates

A promising implementation

- benchmark on standard synchronous and asynchronous algorithms (Flood Min, Ben Or, Bosco, Phase King, reliable broadcast, 2-agreement) and bugged variants
- PyLTA answers within seconds
- up to a handful of refinement steps (each adding several predicates)
- some inconclusive cases

Outline

Summary

Parameterized verification techniques

- apply to **simple standard** distributed algorithms
- provide **automated correctness** proofs
  in contrast to error-prone manual proofs and non-exhaustive simulation
- many frameworks depending on targetted algorithms

Summary

Parameterized verification techniques

- apply to **simple standard** distributed algorithms
- provide **automated correctness** proofs
  in contrast to error-prone manual proofs and non-exhaustive simulation
- many frameworks depending on targetted algorithms

This talk: CEGAR approach for round-based threshold-based fault-tolerant distributed algorithms

- synchronous and asynchronous settings
- layered threshold automata
- LTL parameterized verification undecidable in general
- predicate abstraction and counterexample-guided refinement
- tool implementation: PyLTA

[**B.** Thomas Widder CONCUR'20] [Sankur Thomas TACAS'23]

Future work

In PyLTA

- use implicit predicate abstraction to improve performances
  [Tonetta FM'09]
- define ranking functions to remedy some inconclusive cases
  [Heismann Hoenicke Leike Podelski ATVA'13]

On theoretical side

- formalize model extraction from pseudo-code
- handle Paxos-like consensus algorithms
- extend to randomized algorithms to cover *e.g.* almost-sure termination of Ben Or Byzantine consensus algorithm

Future work

In PyLTA

- use implicit predicate abstraction to improve performances
  [Tonetta FM'09]
- define ranking functions to remedy some inconclusive cases
  [Heismann Hoenicke Leike Podelski ATVA'13]

On theoretical side

- formalize model extraction from pseudo-code
- handle Paxos-like consensus algorithms
- extend to randomized algorithms to cover *e.g.* almost-sure termination of Ben Or Byzantine consensus algorithm

Thanks for your attention!