Parameterized Model Checking of Fault-tolerant Distributed Algorithms

Annu Gmeiner Igor Konnov Ulrich Schmid Helmut Veith
Josef Widder

PV 2014
Rome, Sept 6, 2014

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

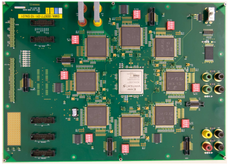
Why fault-tolerant (FT) distributed algorithms

faults not in the control of system designer

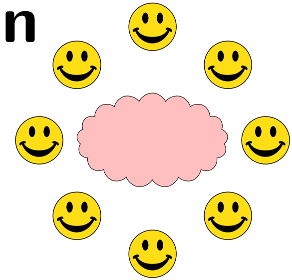
- bit-flips in memory
- power outage
- disconnection from the network
- intruders take control over some computers

distributed algorithms intended 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”

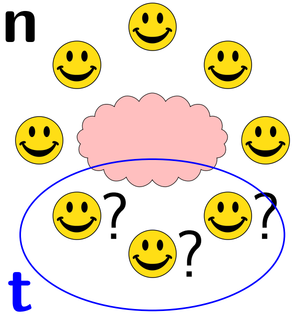


Fault-tolerant distributed algorithms



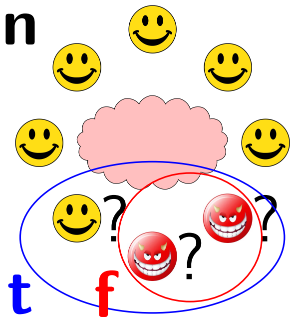
- n processes communicate by messages

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
- resilience conditions, e.g., $n > 3t \wedge t \geq f \geq 0$

Fault-tolerant DAs: Model Checking Challenges

- unbounded data types

  counting how many messages have been received

- parameterization in multiple parameters

  among n processes $f \leq t$ are faulty with $n > 3t$

- contrast to concurrent programs

  fault tolerance against adverse environments

- degrees of concurrency

  many degrees of partial synchrony

- continuous time

  fault-tolerant clock synchronization

Importance of liveness in distributed algorithms

Interplay of safety and liveness is a central challenge in DAs

- interplay of safety and liveness is non-trivial
- asynchrony and faults lead to impossibility results

Importance of liveness in distributed algorithms

Interplay of safety and liveness is a central challenge in DAs

- interplay of safety and liveness is non-trivial
- asynchrony and faults lead to impossibility results

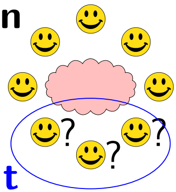
Rich literature to verify safety (e.g. in concurrent systems)

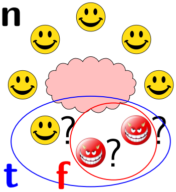
Distributed algorithms perspective:

- "doing nothing is always safe"
- "tools verify algorithms that actually might do nothing"

Model checking problem for fault-tolerant DA algorithms

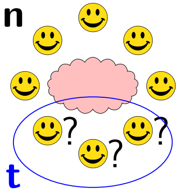
- given a distributed algorithm and spec. φ
- system description:
  $M(n,t,f) = P(n,t,f) \parallel P(n,t,f) \parallel \cdots \parallel P(n,t,f)$
- every $M(n,t,f)$ is a system of $n - f$ correct processes
- show for all n, t, and f satisfying $n > 3t \wedge t \geq f \geq 0$
  $M(n,t,f) \models \varphi$

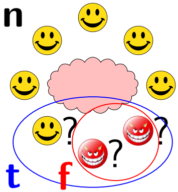




Model checking problem for fault-tolerant DA algorithms

- given a distributed algorithm and spec. φ
- system description:
  $M(n,t,f) = P(n,t,f) \parallel P(n,t,f) \parallel \cdots \parallel P(n,t,f)$
- every $M(n,t,f)$ is a system of $N(n,t,f)$ correct processes
- show for all n, t, and f satisfying *resilience condition*
  $M(n,t,f) \models \varphi$





Properties in Linear Temporal Logic

Unforgeability (U). If $v_i = 0$ for all correct processes i, then for all correct processes j, accept_j remains 0 forever.

$$\mathbf{G}\left(\left(\bigwedge_{i=1}^{n-f} v_i = 0\right) \rightarrow \mathbf{G}\ \left(\bigwedge_{j=1}^{n-f} accept_j = 0\right)\right)$$

Completeness (C). If $v_i = 1$ for all correct processes i, then there is a correct process j that eventually sets accept_j to 1.

$$\mathbf{G}\left(\left(\bigwedge_{i=1}^{n-f} v_i = 1\right) \rightarrow \mathbf{F}\left(\bigvee_{j=1}^{n-f} accept_j = 1\right)\right)$$

Relay (R). If a correct process i sets accept_i to 1, then eventually all correct processes j set accept_j to 1.

$$\mathbf{G}\left(\left(\bigvee_{i=1}^{n-f} accept_i = 1\right) \rightarrow \mathbf{F}\left(\bigwedge_{j=1}^{n-f} accept_j = 1\right)\right)$$

Properties in Linear Temporal Logic

Unforgeability (U). If $v_i = 0$ for all correct processes i, then for all correct processes j, accept_j remains 0 forever.

$$\mathbf{G}\left(\left(\bigwedge_{i=1}^{n-f} v_i = 0\right) \rightarrow \mathbf{G}\ \left(\bigwedge_{j=1}^{n-f} accept_j = 0\right)\right)$$

Safety

Completeness (C). If $v_i = 1$ for all correct processes i, then there is a correct process j that eventually sets accept_j to 1.

$$\mathbf{G}\left(\left(\bigwedge_{i=1}^{n-f} v_i = 1\right) \rightarrow \mathbf{F}\left(\bigvee_{j=1}^{n-f} accept_j = 1\right)\right)$$

Liveness

Relay (R). If a correct process i sets accept_i to 1, then eventually all correct processes j set accept_j to 1.

$$\mathbf{G}\left(\left(\bigvee_{i=1}^{n-f} accept_i = 1\right) \rightarrow \mathbf{F}\left(\bigwedge_{j=1}^{n-f} accept_j = 1\right)\right)$$

Liveness

Threshold-guarded fault-tolerant distributed algorithms

Threshold-guarded FTDAs

Fault-free construct: quantified guards ($t=f=0$)

- Existential Guard
  `if received` m `from` *some* `process then ...`
- Universal Guard
  `if received` m `from` *all* `processes then ...`

These guards allow one to treat the processes in a parameterized way

Threshold-guarded FTDAs

Fault-free construct: quantified guards (t=f=0)

- Existential Guard
  `if received` m `from` *some* `process then ...`
- Universal Guard
  `if received` m `from` *all* `processes then ...`

These guards allow one to treat the processes in a parameterized way

what if faults might occur? 

Threshold-guarded FTDAs

Fault-free construct: quantified guards (t=f=0)

- Existential Guard
  `if received` m `from` *some* `process then ...`
- Universal Guard
  `if received` m `from` *all* `processes then ...`

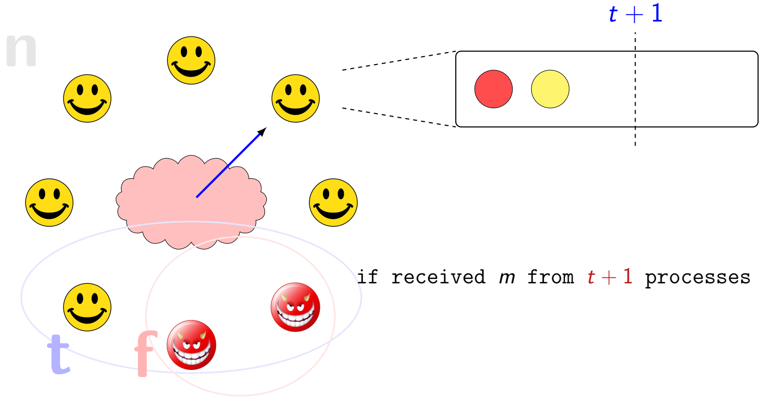
These guards allow one to treat the processes in a parameterized way

what if faults might occur? 

Fault-Tolerant Algorithms: n **processes, at most** t **are Byzantine**

- Threshold Guard
  `if received` m `from` $n - t$ `processes then ...`
- (the processes cannot refer to **f**!)

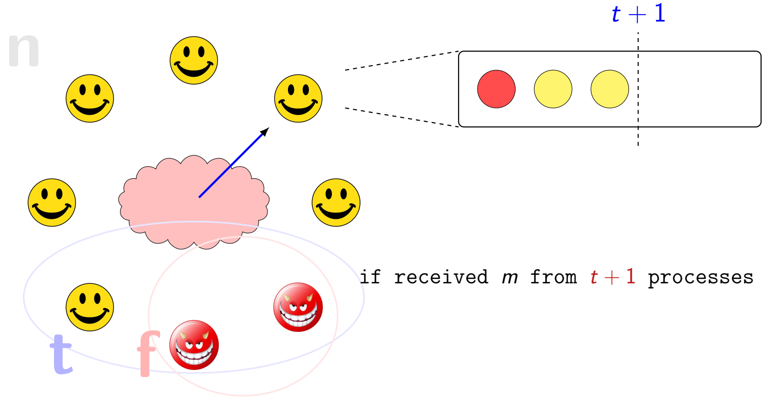
Counting argument in threshold-guarded algorithms



`if received` m `from` $t+1$ `processes then ...`

Correct processes count distinct incoming messages

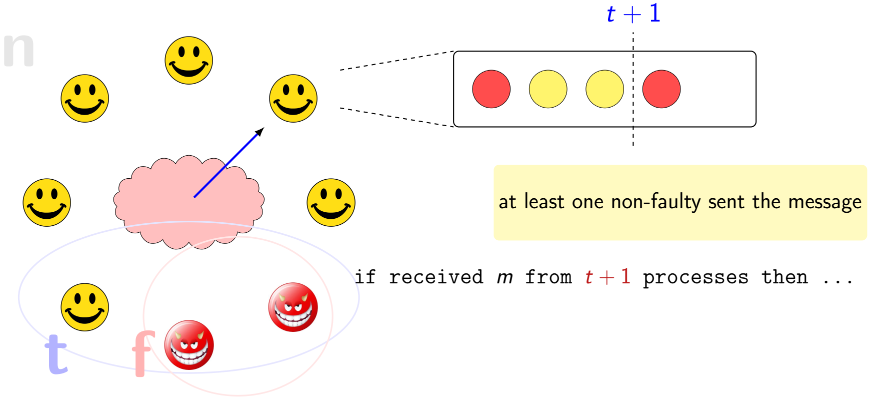
Counting argument in threshold-guarded algorithms



if received m from $t+1$ processes then ...

Correct processes count distinct incoming messages

Counting argument in threshold-guarded algorithms



Correct processes count distinct incoming messages

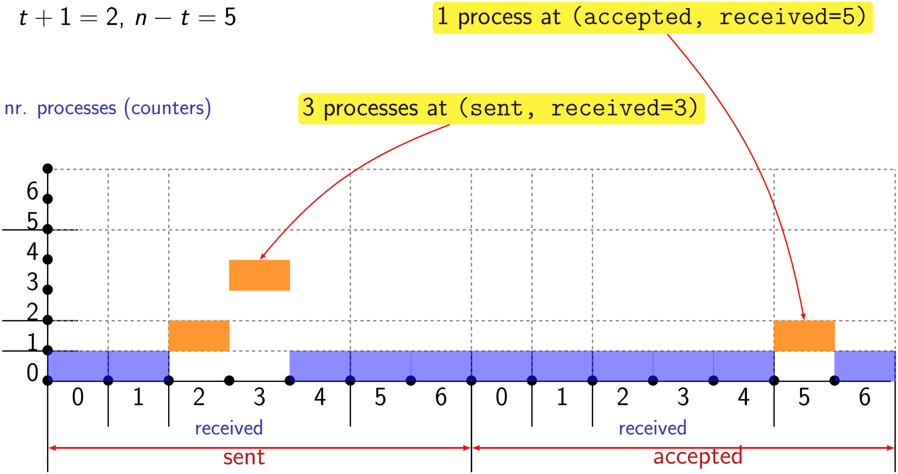
our abstractions

at a glance

Data + counter abstraction over parametric intervals

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

$t + 1 = 2,\ n - t = 5$

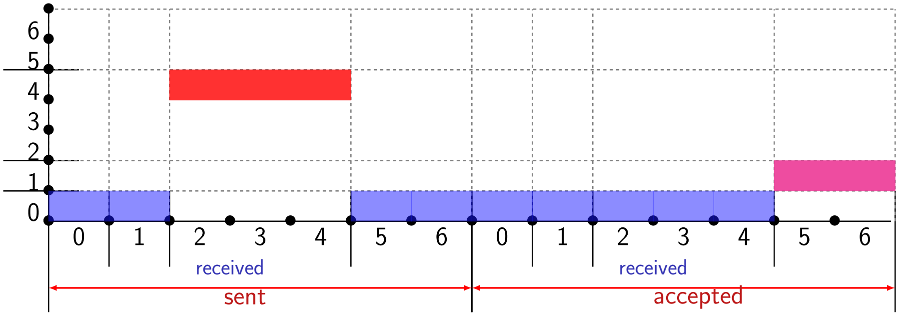


Data + counter abstraction over parametric intervals

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

$t + 1 = 2,\ n - t = 5$

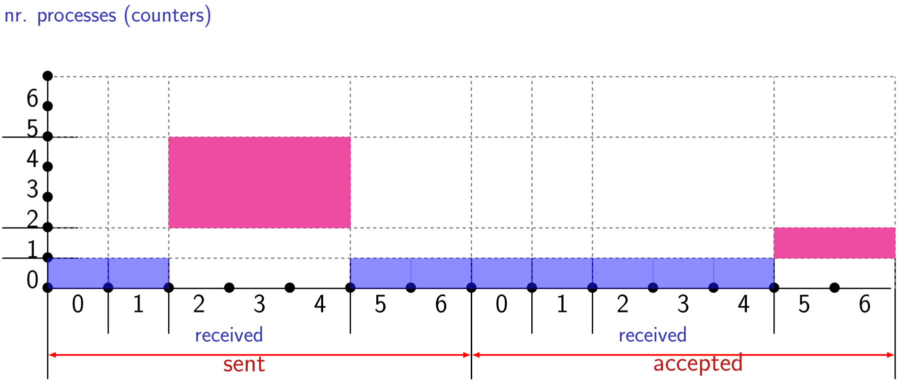
nr. processes (counters)



Data + counter abstraction over parametric intervals

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

$t + 1 = 2,\ n - t = 5$



Data + counter abstraction over parametric intervals

$\cancel{n=6}$, $\cancel{t=1}$, $\cancel{f=1}$

$n > 3 \cdot t \wedge t \geq f$

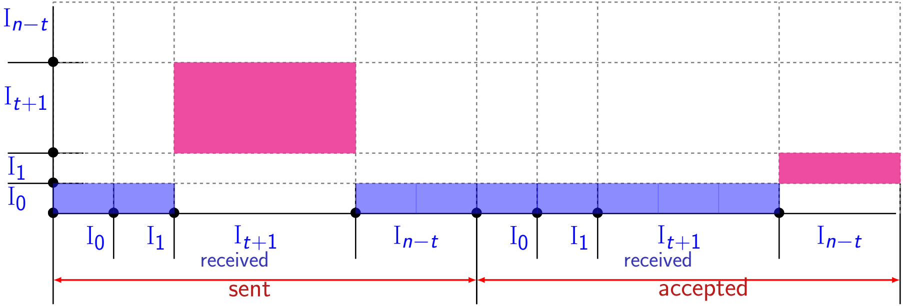
Parametric intervals:

$I_0 = [0, 1) \quad I_1 = [1, t+1)$

$I_{t+1} = [t+1, n-t)$

$I_{n-t} = [n-t, \infty)$

nr. processes (counters)



Data + counter abstraction over parametric intervals

$n > 3 \cdot t \wedge t \geq f$

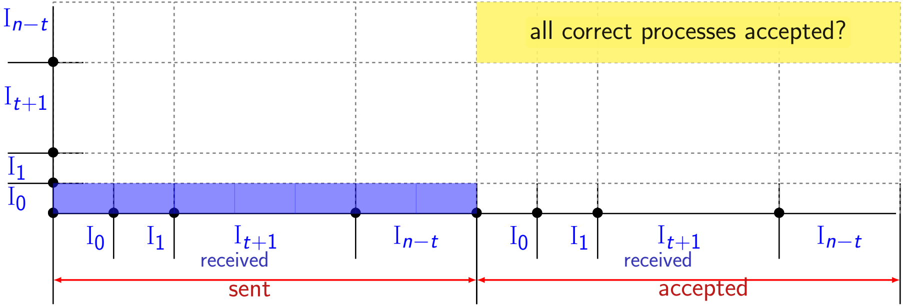
Parametric intervals:

$\mathrm{I}_0 = [0, 1) \quad \mathrm{I}_1 = [1, t+1)$

$\mathrm{I}_{t+1} = [t+1, n-t)$

$\mathrm{I}_{n-t} = [n-t, \infty)$

nr. processes (counters)



Related work: $(0, 1, \infty)$-counter abstraction

Pnueli, Xu, and Zuck (2001) introduced $(0, 1, \infty)$-counter abstraction:

- finitely many local states,
  e.g., $\{N, T, C\}$.
- abstract the number of processes in every state,
  e.g., $K: \quad C \mapsto \mathbf{0}, \quad T \mapsto \mathbf{1}, \quad N \mapsto \textbf{``many''}$.
- perfectly reflects mutual exclusion properties
  e.g., $\mathbf{G}\,(K(C) \neq \textbf{``many''})$.

Related work: $(0, 1, \infty)$-counter abstraction

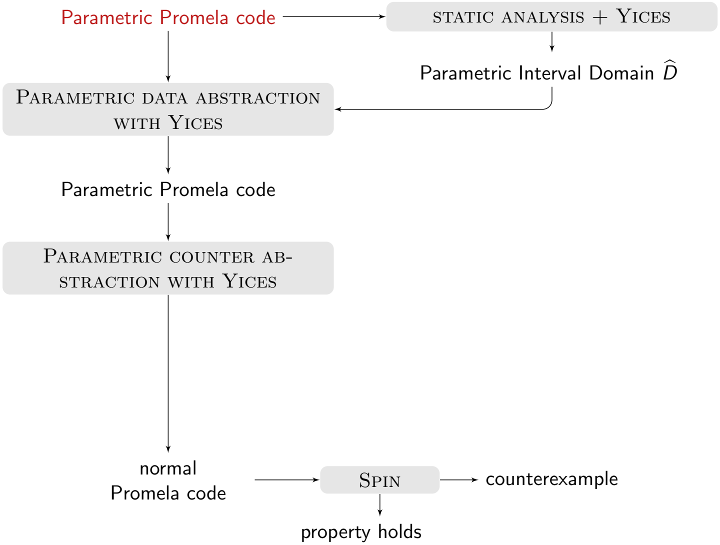
Pnueli, Xu, and Zuck (2001) introduced $(0, 1, \infty)$-counter abstraction:

- finitely many local states,
  e.g., $\{N, T, C\}$.
- abstract the number of processes in every state,
  e.g., $K: \quad C \mapsto \mathbf{0}, \quad T \mapsto \mathbf{1}, \quad N \mapsto$ **"many"**.
- perfectly reflects mutual exclusion properties
  e.g., $\mathbf{G}\,(K(C) \neq$ **"many"**$)$.

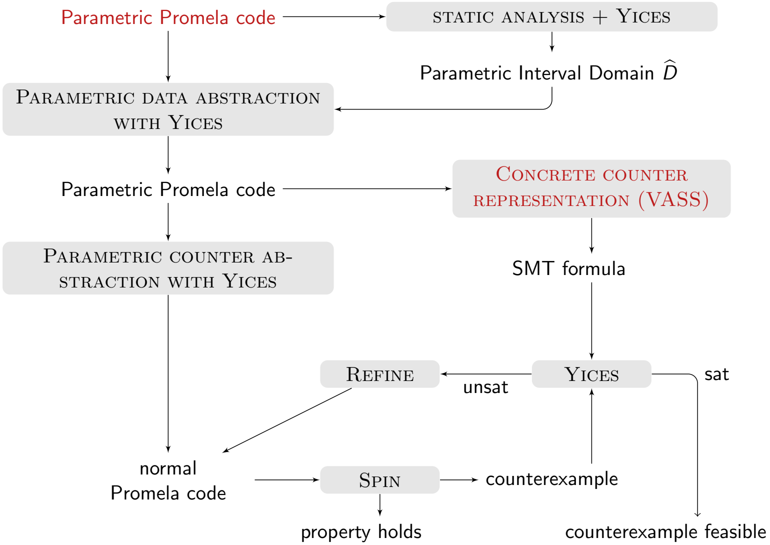
Our parametric data + counter abstraction:

- unboundendly many local states (nr. of received messages)
- finer counting of processes:
  $t + 1$ processes in a specific state can force global progress,
  while t processes cannot
- mapping t, $t + 1$, and $n - t$ to **"many"** is too coarse.

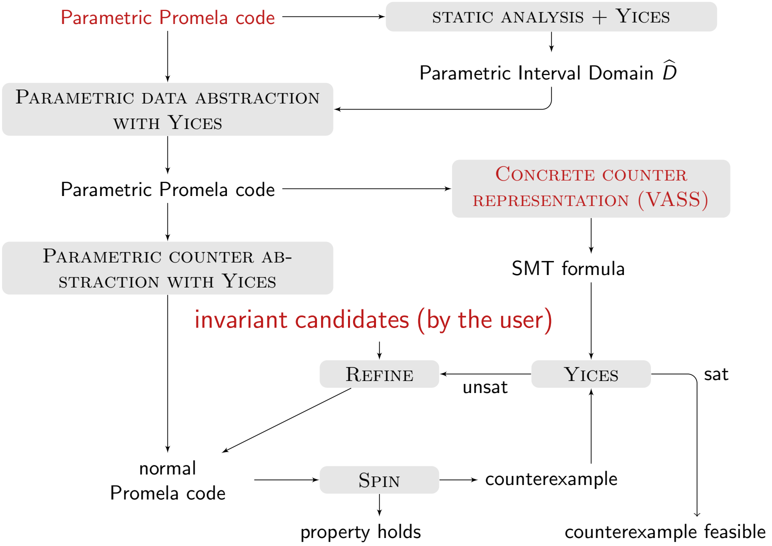
Tool Chain: ByMC



Tool Chain: ByMC

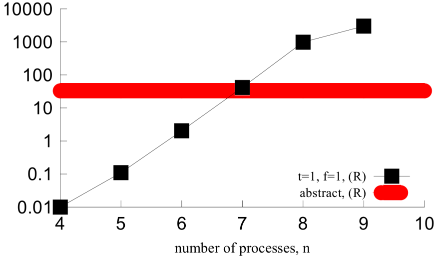


Tool Chain: ByMC

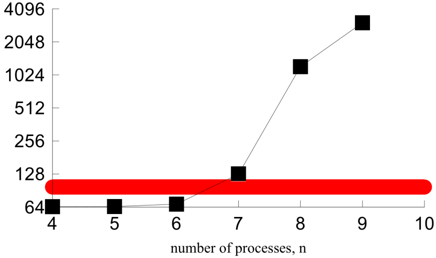


Concrete vs. parameterized (Byzantine case)

Time to check relay (sec, logscale)



Memory to check relay (MB, logscale)



- Parameterized model checking performs well (the red line).
- Experiments for fixed parameters quickly degrade ($n = 9$ runs out of memory).
- We found counter-examples for the cases $n = 3t$ and $f > t$, where the resilience condition is violated.

Completeness threshold for bounded model checking

Fix a threshold automaton TA and a size function N.

Theorem

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

- $|E|$ is the number of edges in TA (self-loops excluded).
- $|\mathcal{C}|$ 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.

Completeness threshold for bounded model checking

Fix a threshold automaton TA and a size function N.

Theorem

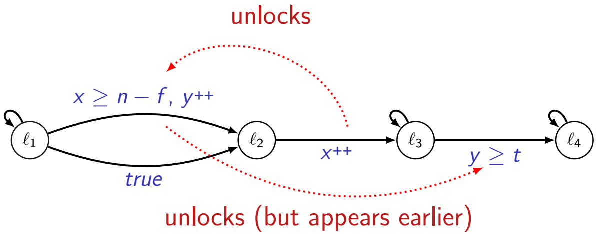
For each $\mathbf{p}$ with $RC(\mathbf{p})$, the diameter of an accelerated counter system is independent of parameters and is less than or equal to $|E| \cdot (|\mathcal{C}| + 1) + |\mathcal{C}|$:

- $|E|$ is the number of edges in TA (self-loops excluded).
- $|\mathcal{C}|$ 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:

$|E| = 4$, $|\mathcal{C}| = 1$.

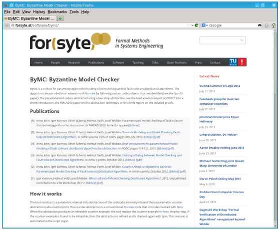
Thus, $d \leq 9$.

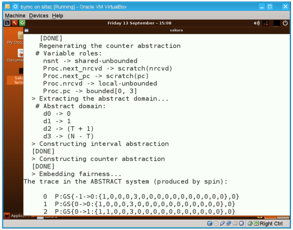


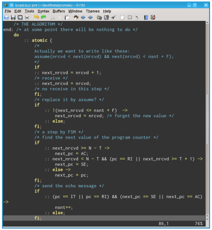
Can we reach the bound with NuSMV?

Timeout in abstraction refinement: NBAC (13200) and NBACC (16500).

Experimental setup







The tool (source code in OCaml),

the code of the distributed algorithms in Parametric Promela,

and a virtual machine with full setup

are available at: http://forsyte.at/software/bymc

Related work: PV of FTDAs

Regular model checking of fault-tolerant distributed protocols:

[Fisman, Kupferman, Lustig 2008]

- "First-shot" theoretical framework.
- No guards like $x \geq t + 1$, only $x \geq 1$.
- No implementation.
- Manual analysis applied to folklore broadcast (crash faults).

Related work: PV of FTDAs

Regular model checking of fault-tolerant distributed protocols:

[Fisman, Kupferman, Lustig 2008]

- "First-shot" theoretical framework.
- No guards like $x \geq t+1$, only $x \geq 1$.
- No implementation.
- Manual analysis applied to folklore broadcast (crash faults).

Backward reachability using SMT with arrays:

[Alberti, Ghilardi, Pagani, Ranise, Rossi 2010-2012]

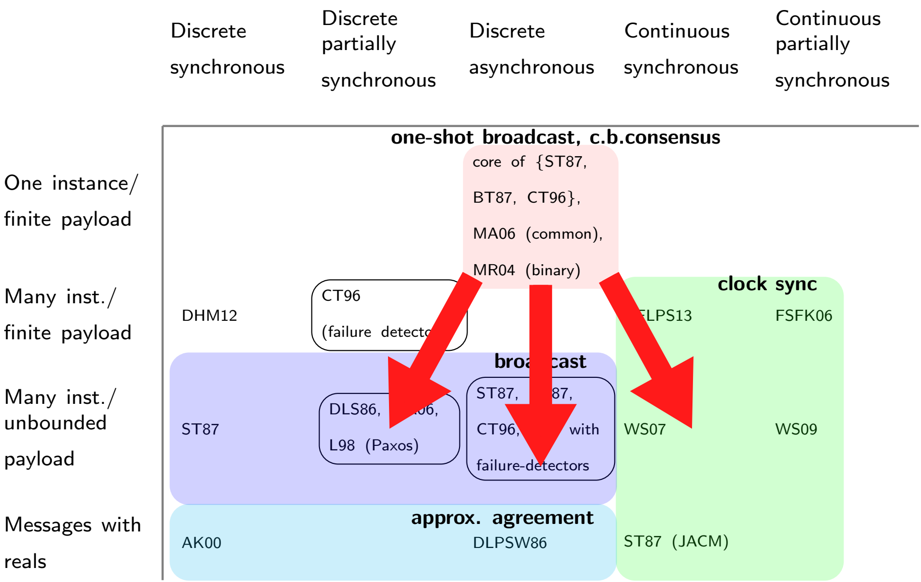
- Implementation.
- Experiments on Chandra-Toueg 1990.
- No resilience conditions like $n > 3t$.
- Safety only.

Our current work

one-shot broadcast, c.b.consensus

	Discrete synchronous	Discrete partially synchronous	Discrete asynchronous	Continuous synchronous	Continuous partially synchronous
One instance/ finite payload			core of {ST87, BT87, CT96}, MA06 (common), MR04 (binary)		
Many inst./ finite payload					
Many inst./ unbounded payload					
Messages with reals					

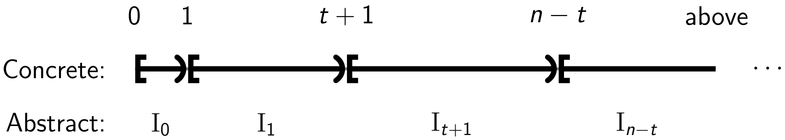
Future work: threshold guards + orthogonal features



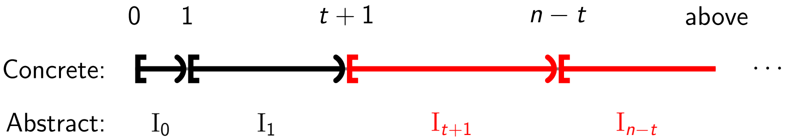
Thank you!

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

Abstract operations

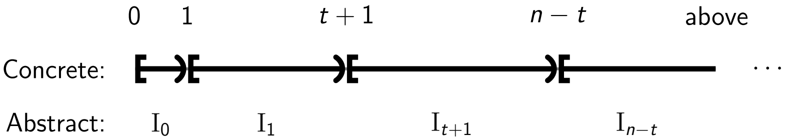


Concrete $t + 1 \leq x$



Concrete $t+1 \le x$ is abstracted as $x = \mathrm{I}_{t+1} \vee x = \mathrm{I}_{n-t}$.

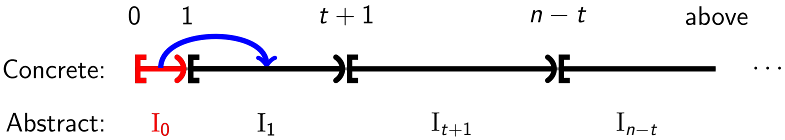
Abstract operations



Concrete $t+1 \le x$ is abstracted as $x = \mathrm{I}_{t+1} \vee x = \mathrm{I}_{n-t}$.

Concrete $x' = x + 1$,

Abstract operations

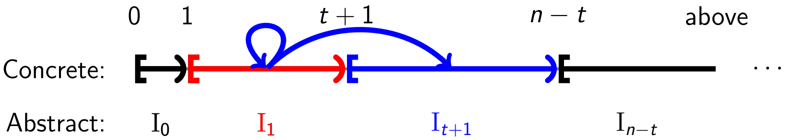


Concrete $t+1 \leq x$ is abstracted as $x = \mathrm{I}_{t+1} \vee x = \mathrm{I}_{n-t}$.

Concrete $x' = x + 1$, is abstracted as:

$$x = \mathrm{I}_0 \quad \wedge \; x' = \mathrm{I}_1 \; \ldots$$

Abstract operations

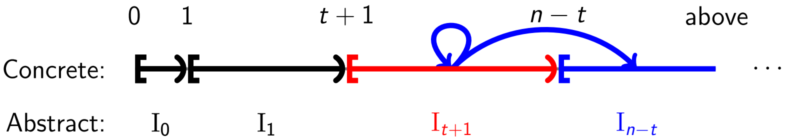


Concrete $t+1 \leq x$ is abstracted as $x = \mathrm{I}_{t+1} \vee x = \mathrm{I}_{n-t}$.

Concrete $x' = x + 1$, is abstracted as:

$$
\begin{aligned}
& x = \mathrm{I}_0 \quad \wedge \; x' = \mathrm{I}_1 \\
\vee & x = \mathrm{I}_1 \quad \wedge (x' = \mathrm{I}_1 \quad \vee x' = \mathrm{I}_{t+1}) \ \ldots
\end{aligned}
$$

Abstract operations

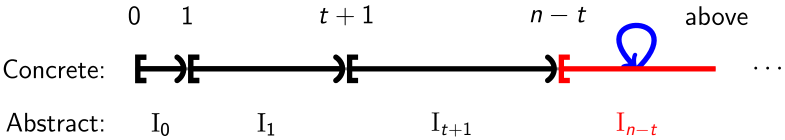


Concrete $t+1 \leq x$ is abstracted as $x = \mathrm{I}_{t+1} \vee x = \mathrm{I}_{n-t}$.

Concrete $x' = x + 1$, is abstracted as:

$$
\begin{aligned}
& x = \mathrm{I}_0 && \wedge\ x' = \mathrm{I}_1 \\
\vee & x = \mathrm{I}_1 && \wedge\ (x' = \mathrm{I}_1 \vee x' = \mathrm{I}_{t+1}) \\
\vee & x = \mathrm{I}_{t+1} && \wedge\ (x' = \mathrm{I}_{t+1} \vee x' = \mathrm{I}_{n-t}) \ \ldots
\end{aligned}
$$

Abstract operations



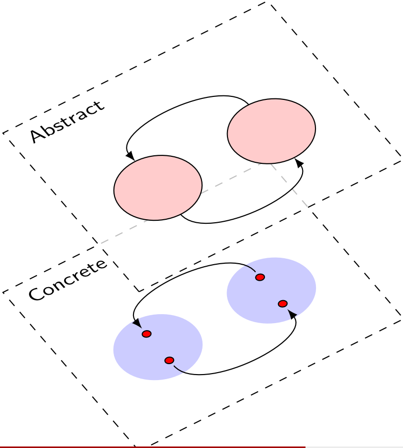
Concrete $t + 1 \leq x$ is abstracted as $x = \mathrm{I}_{t+1} \vee x = \mathrm{I}_{n-t}$.

Concrete $x' = x + 1$, is abstracted as:

$$
\begin{aligned}
& x = \mathrm{I}_0 \quad \wedge\ x' = \mathrm{I}_1 \\
\vee & x = \mathrm{I}_1 \quad \wedge (x' = \mathrm{I}_1 \quad \vee x' = \mathrm{I}_{t+1}) \\
\vee & x = \mathrm{I}_{t+1} \wedge (x' = \mathrm{I}_{t+1} \vee x' = \mathrm{I}_{n-t}) \\
\vee & x = \mathrm{I}_{n-t} \wedge\ x' = \mathrm{I}_{n-t}
\end{aligned}
$$

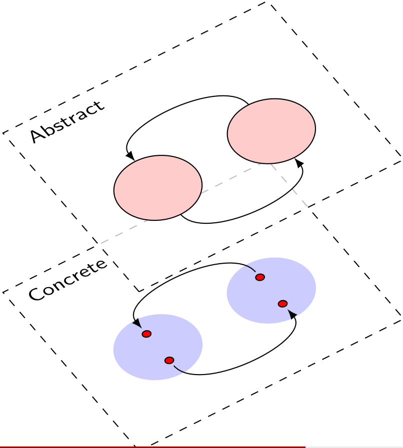
Parametric abst. refinement — uniformly spurious paths

Classical CEGAR:



Parametric abst. refinement — uniformly spurious paths

Classical CEGAR:



Our case: