Model Checking of Fault-Tolerant Distributed Algorithms

Part IV: Parameterized Model Checking of Fault-tolerant Distributed Algorithms by Abstraction

Annu Gmeiner Igor Konnov Ulrich Schmid
Helmut Veith Josef Widder

TU WIEN

for(syte) Formal Methods in Systems Engineering

RiSE Rigorous Systems Engineering

VTSA 2014, Luxembourg

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

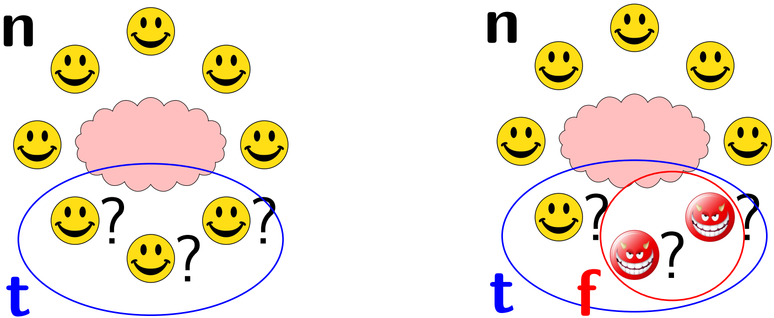
  fault-tolerant clock synchronization

Model checking problem for fault-tolerant DA algorithms

Parameterized model checking problem:

- given a distributed algorithm and spec. φ
- show for all n, t, and f satisfying $n > 3t \wedge t \geq f \geq 0$

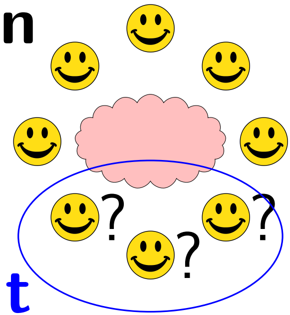
  $M(n, t, f) \models \varphi$
- every $M(n, t, f)$ is a system of $n - f$ correct processes

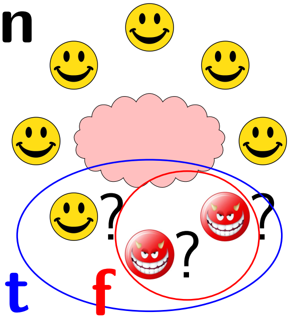


Model checking problem for fault-tolerant DA algorithms

Parameterized model checking problem:

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





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**!)

Control Flow Automata

Variables of process i

v_i : {0, 1} **init with** 0 **or** 1

$accept_i$: {0, 1} **init with** 0

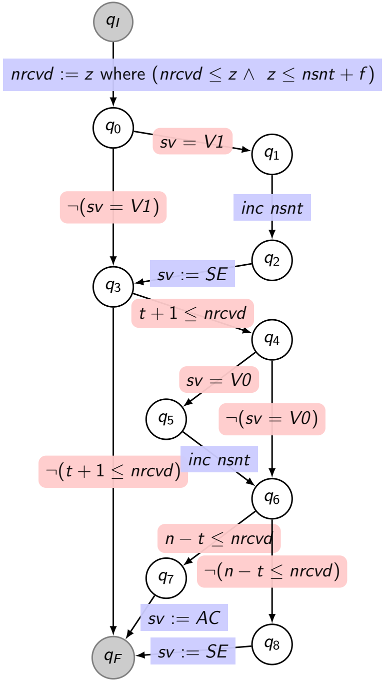
An indivisible step:

if $v_i = 1$
then send (echo) to all;

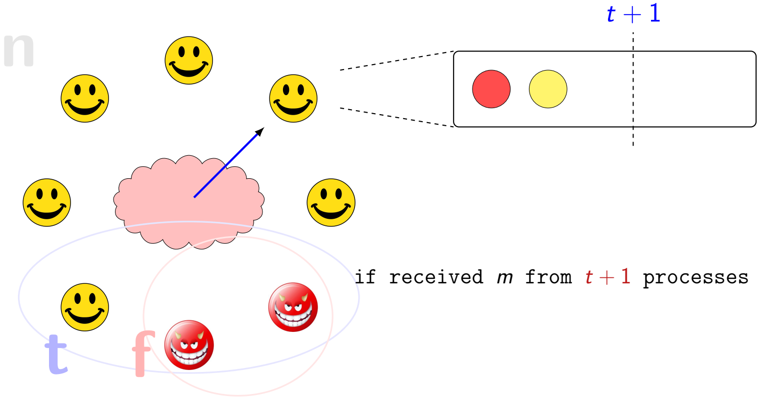
if received (echo) from at least
t + 1 distinct processes
and not sent (echo) before
then send (echo) to all;

if received (echo) from at least
n - t distinct processes
then $accept_i$:= 1;

$n - f$ copies of the process



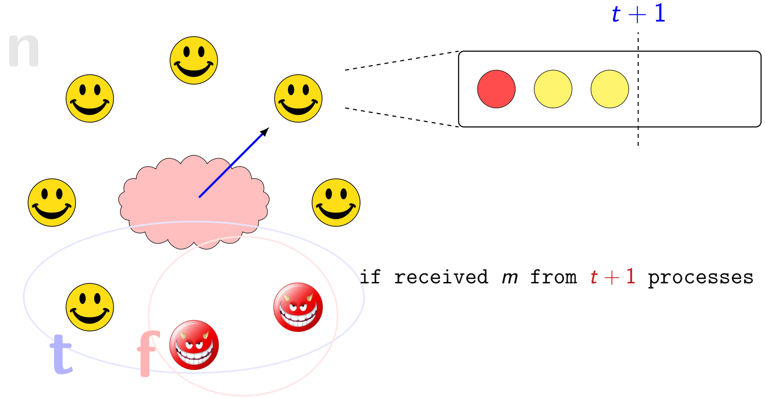
Counting argument in threshold-guarded algorithms



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

Correct processes count distinct incoming messages

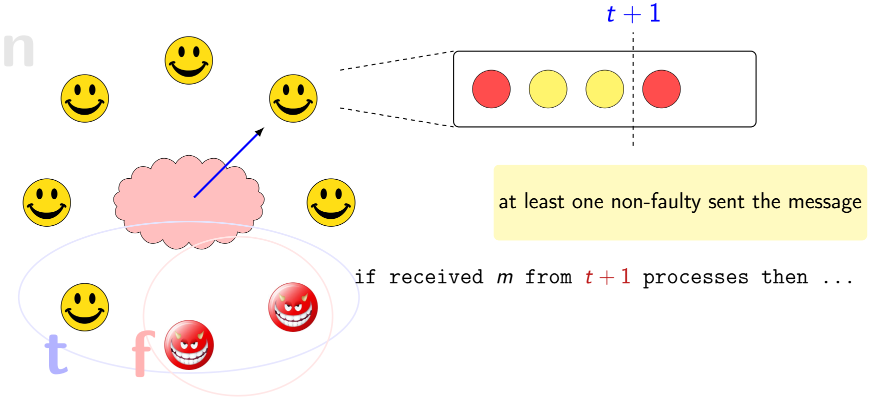
Counting argument in threshold-guarded algorithms



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

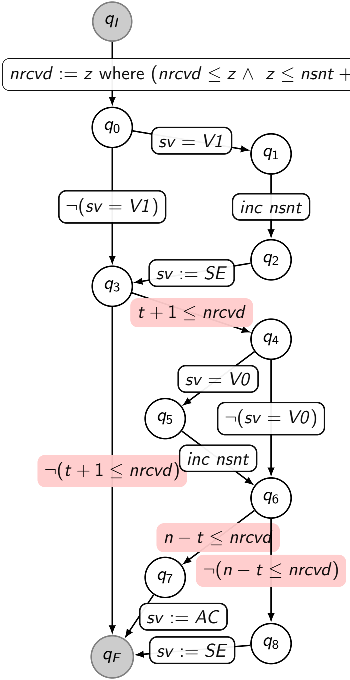
Correct processes count distinct incoming messages

Counting argument in threshold-guarded algorithms

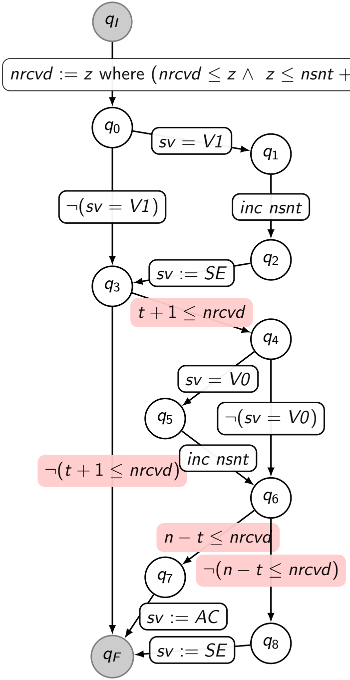


`if received` m `from` $t+1$ `processes then ...`

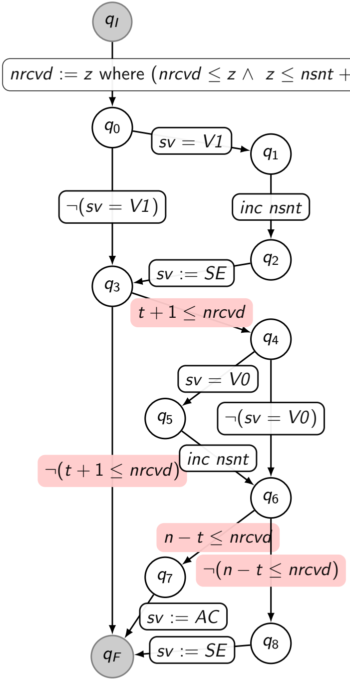
Correct processes count distinct incoming messages



- concrete values are not important
- thresholds are essential:
  $0,\ 1,\ t + 1,\ n - t$



- concrete values are not important
- thresholds are essential:
  $0, \ 1, \ t+1, \ n-t$
- intervals with symbolic boundaries:
  - $\mathrm{I}_0 \quad = [0, 1)$
  - $\mathrm{I}_1 \quad = [1, t+1)$
  - $\mathrm{I}_{t+1} = [t+1, n-t)$
  - $\mathrm{I}_{n-t} = [n-t, \infty)$



- concrete values are not important
- thresholds are essential: $0,\ 1,\ t+1,\ n-t$
- intervals with symbolic boundaries:
  - $I_0 \quad = [0, 1)$
  - $I_1 \quad = [1, t+1)$
  - $I_{t+1} = [t+1, n-t)$
  - $I_{n-t} = [n-t, \infty)$
- Parameteric Interval Abstraction (PIA)
- Similar to interval abstraction: $[t+1, n-t)$ rather than $[4, 10)$.
- Total order: $0 < 1 < t+1 < n-t$ for all parameters satisfying RC: $n > 3t, t \geq f \geq 0$.

Technical challenges

We have to reduce the verification of an infinite number of instances where

1. the process code is parameterized
2. the number of processes is parameterized

to one finite state model checking instance

Technical challenges

We have to reduce the verification of an infinite number of instances where

1. the process code is parameterized
2. the number of processes is parameterized

to one finite state model checking instance

We do that by:

1. PIA data abstraction
2. PIA counter abstraction

Technical challenges

We have to reduce the verification of an infinite number of instances where

1. the process code is parameterized
2. the number of processes is parameterized

to one finite state model checking instance

We do that by:

1. PIA data abstraction
2. PIA counter abstraction

abstraction is an over approximation $\Rightarrow$ possible abstract behavior that does not correspond to a concrete behavior.

3. Refining spurious counter-examples

Abstraction overview

Parameterized family

$$\{M(n,t,f) = \underbrace{\mathsf{P}(n,t,f) \parallel \cdots \parallel \mathsf{P}(n,t,f)}_{N(n,t,f)\text{ processes}} : n > 3t, t \geq f, f \geq 0\}$$

EXTRACT

Parametric Interval Domain $\widehat{D}$

PARAMETRIC INTERVAL DATA ABSTRACTION

Uniform parameterized family

$$\{\hat{M}(n,t,f) = \underbrace{\hat{\mathsf{P}} \parallel \cdots \parallel \hat{\mathsf{P}}}_{N(n,t,f)\text{ processes}} : n > 3t, t \geq f, f \geq 0\}$$

$\widehat{P}$ does not depend on n, t, f

$\widehat{P}$ simulates $P(n,t,f)$

CHANGE REPRESENTATION

Counter representation

PARAMETRIC INTERVAL COUNTER ABSTRACTION

one abstract system $\mathcal{A}$ that simulates for every n, t, f the behavior of $M(n,t,f)$

Abstraction overview

Parameterized family

$$\left\{ M(n,t,f) = \underbrace{\mathsf{P}(n,t,f) \parallel \cdots \parallel \mathsf{P}(n,t,f)}_{N(n,t,f) \text{ processes}} : n > 3t, t \geq f, f \geq 0 \right\}$$

$\longrightarrow$ EXTRACT

Parametric Interval Domain $\widehat{D}$

PARAMETRIC INTERVAL DATA ABSTRACTION

Uniform parameterized family

$$\left\{ \hat{M}(n,t,f) = \underbrace{\hat{\mathsf{P}} \parallel \cdots \parallel \hat{\mathsf{P}}}_{N(n,t,f) \text{ processes}} : n > 3t, t \geq f, f \geq 0 \right\}$$

$\widehat{P}$ does not depend on n, t, f

$\widehat{P}$ simulates $P(n,t,f)$

CHANGE REPRESENTATION

replay the counter-example

Counter representation

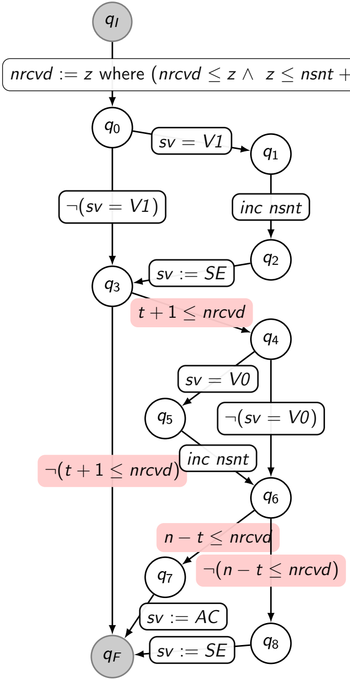
refine the system

PARAMETRIC INTERVAL COUNTER ABSTRACTION

$\longrightarrow$ one abstract system $\mathcal{A}$ that simulates for every n, t, f the behavior of $M(n,t,f)$

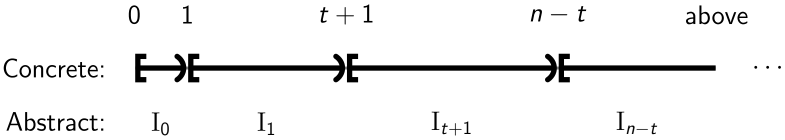
finite-state model checkin

Data abstraction



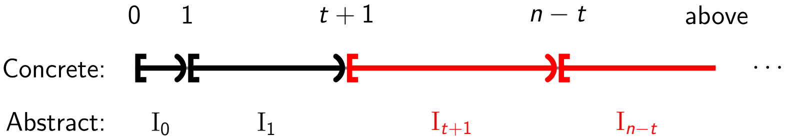
- concrete values are not important
- thresholds are essential: $0,\ 1,\ t+1,\ n-t$
- intervals with symbolic boundaries:
  - $\mathrm{I}_0 = [0, 1)$
  - $\mathrm{I}_1 = [1, t+1)$
  - $\mathrm{I}_{t+1} = [t+1, n-t)$
  - $\mathrm{I}_{n-t} = [n-t, \infty)$

Abstract operations



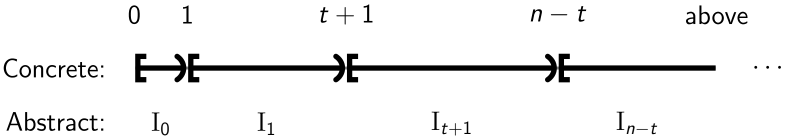
Concrete $t + 1 \le x$

Abstract operations



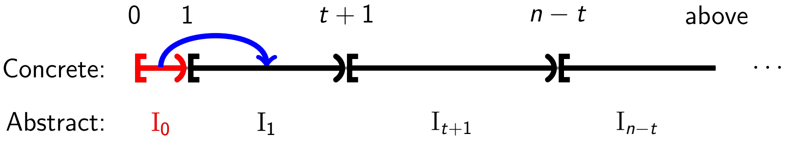
Concrete $t+1 \leq x$ is abstracted as $x = \mathrm{I}_{t+1} \vee x = \mathrm{I}_{n-t}$.

Abstract operations



Concrete $t + 1 \leq x$ is abstracted as $x = \mathrm{I}_{t+1} \vee x = \mathrm{I}_{n-t}$.
Concrete $x' = x + 1$,

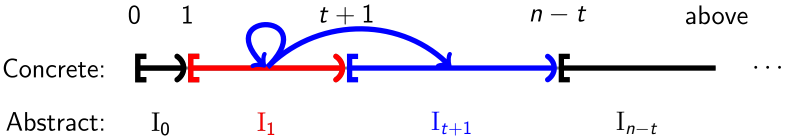
Abstract operations



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:

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

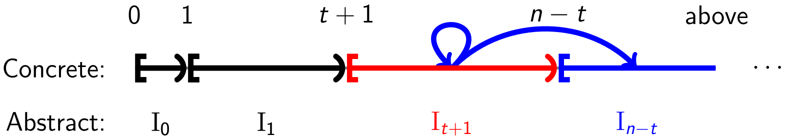
Abstract operations



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:

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

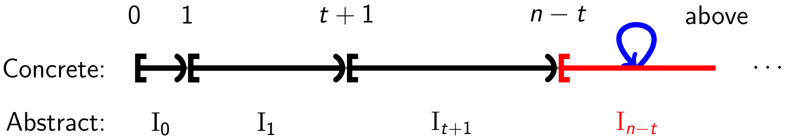
Abstract operations



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:

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

Abstract operations

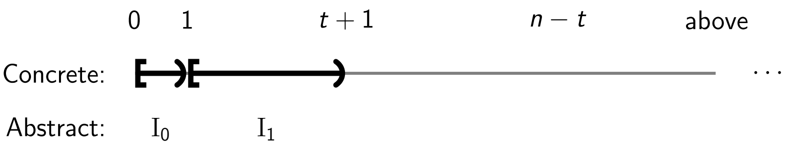


Concrete $t+1 \leq x$ is abstracted as $x = \mathrm{I}_{t+1} \vee x = \mathrm{I}_{n-t}$.

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

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

Abstract operations

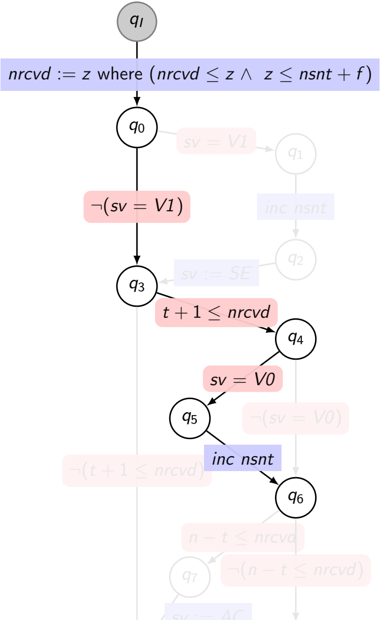


Concrete $t+1 \leq x$ is abstracted as $x = \mathrm{I}_{t+1} \vee x = \mathrm{I}_{n-t}$.
Concrete $x' = x + 1$, is abstracted as:

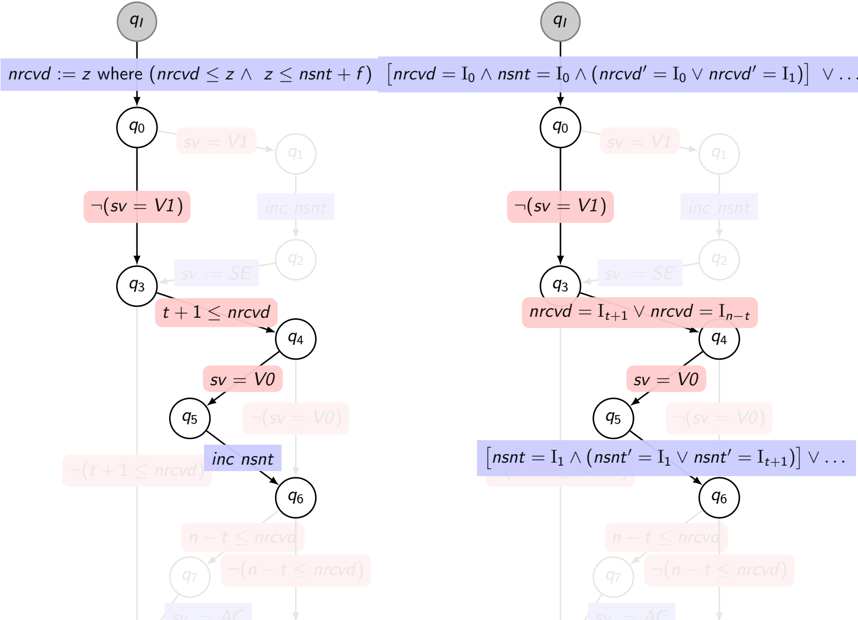
$$\begin{aligned} & x = \mathrm{I}_0 && \wedge\ x' = \mathrm{I}_1 \\ \vee & x = \mathrm{I}_1 && \wedge (x' = \mathrm{I}_1 \vee x' = \mathrm{I}_{t+1}) \\ \vee & x = \mathrm{I}_{t+1} && \wedge (x' = \mathrm{I}_{t+1} \vee x' = \mathrm{I}_{n-t}) \\ \vee & x = \mathrm{I}_{n-t} && \wedge\ x' = \mathrm{I}_{n-t}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

abstract increase may keep the same value!

Abstract CFA



Abstract CFA



Abstraction overview

Parameterized family

$$\left\{ M(n,t,f) = \underbrace{\mathsf{P}(n,t,f) \parallel \cdots \parallel \mathsf{P}(n,t,f)}_{N(n,t,f)\text{ processes}} : n > 3t, t \geq f, f \geq 0 \right\}$$

→ EXTRACT → Parametric Interval Domain $\widehat{D}$

↓

PARAMETRIC INTERVAL DATA ABSTRACTION

↓

Uniform parameterized family

$$\left\{ \hat{M}(n,t,f) = \underbrace{\hat{\mathsf{P}} \parallel \cdots \parallel \hat{\mathsf{P}}}_{N(n,t,f)\text{ processes}} : n > 3t, t \geq f, f \geq 0 \right\}$$

$\widehat{P}$ does not depend on n, t, f

$\widehat{P}$ simulates $P(n,t,f)$

↓

CHANGE REPRESENTATION

↓

Counter representation

↓

PARAMETRIC INTERVAL COUNTER ABSTRACTION

→ one abstract system $\mathcal{A}$ that simulates for every n, t, f the behavior of $M(n,t,f)$

Counter abstraction

Classic $(0, 1, \infty)$-counter abstraction

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

- finitely many local states,
  e.g., $\{N, T, C\}$.
- based on counter representation:
  for each local states count how many processes are in it

Classic $(0, 1, \infty)$-counter abstraction

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

- finitely many local states,
  e.g., $\{N, T, C\}$.
- based on counter representation:
  for each local states count how many processes are in it
- 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., $\mathbf{G}\,(K(C) = \mathbf{0} \vee K(C) = \mathbf{1})$.

Limits of $(0, 1, \infty)$-counter abstraction

Our parametric data + counter abstraction:

- we require finer counting of processes:
  - $t + 1$ processes in a specific state can force global progress,
  - t processes cannot
- mapping t, $t + 1$, and $n - t$ to **"many"** is too coarse.

Limits of $(0, 1, \infty)$-counter abstraction

Our parametric data + counter abstraction:

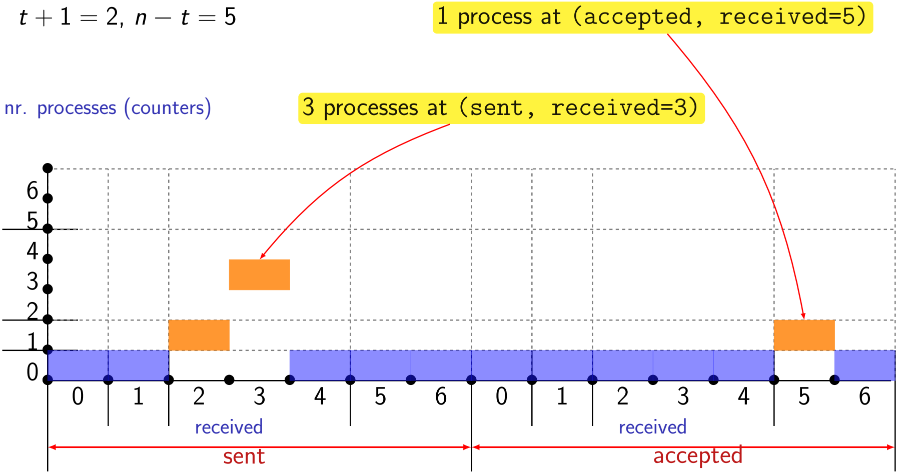
- we require finer counting of processes:
  - $t + 1$ processes in a specific state can force global progress,
  - t processes cannot
- mapping t, $t + 1$, and $n - t$ to **"many"** is too coarse.

starting point of our approach...

Data + counter abstraction over parametric intervals

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

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

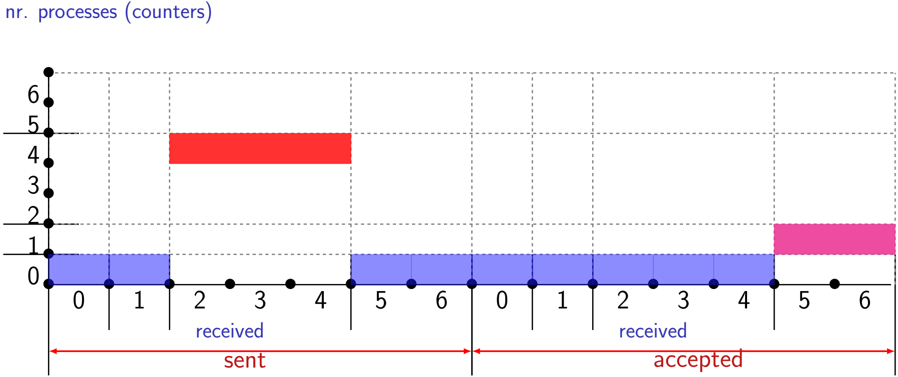


Local state is $(sv, nrcvd)$,
where $sv \in \{sent, accepted\}$ and $0 \leq rcvd \leq n$

Data + counter abstraction over parametric intervals

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

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

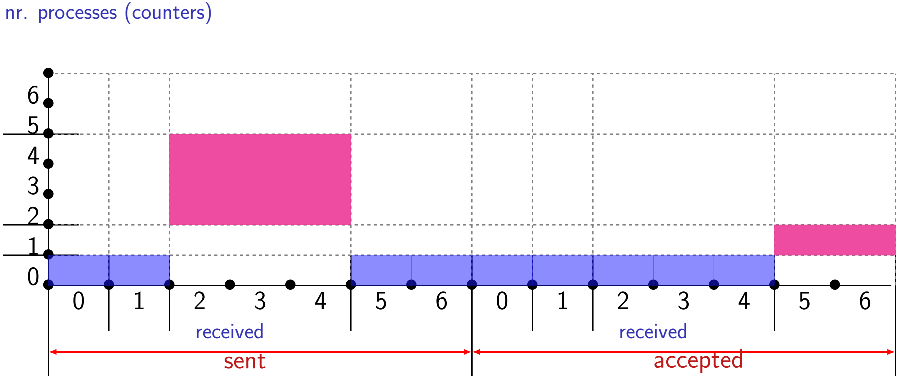


Local state is $(sv, nrcvd)$,
where $sv \in \{sent, accepted\}$ and $0 \leq rcvd \leq n$

Data + counter abstraction over parametric intervals

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

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



Local state is $(sv, nrcvd)$,
where $sv \in \{sent, accepted\}$ and $0 \leq rcvd \leq n$

Data + counter abstraction over parametric intervals

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

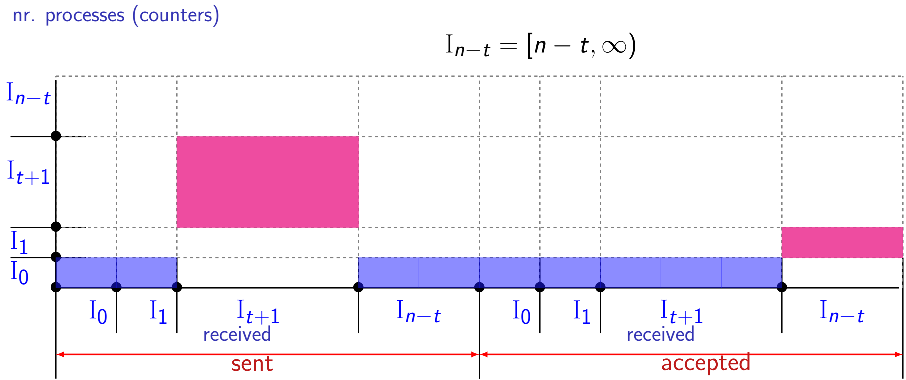
$n > 3 \cdot t \wedge t \geq f$

Parametric intervals:

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

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

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



A local state is $(sv, nrcvd)$,
where $sv \in \{sent, accepted\}$ and $nrcvd \in \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}$

Data + counter abstraction over parametric intervals

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

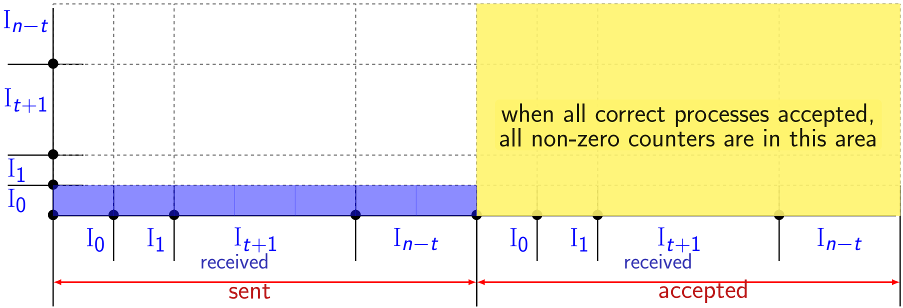
Parametric intervals:

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

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

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

nr. processes (counters)



A local state is $(sv, nrcvd)$,
where $sv \in \{sent, accepted\}$ and $nrcvd \in \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}$

Abstraction refinement

Spurious behavior

abstraction adds behaviors (e.g., `x'=x+1` may lead to `x'` being equal to `x`)

Spurious behavior

abstraction adds behaviors (e.g., `x'=x+1` may lead to `x'` being equal to `x`)

$\Rightarrow$ specs that hold in concrete system may be violated in abstract system

- spurious counterexamples
- we have to reduce the behaviors of the abstract system
  make it more concrete
- ... based on the counterexamples = CEGAR

Spurious behavior

abstraction adds behaviors (e.g., `x'=x+1` may lead to `x'` being equal to `x`)

$\Rightarrow$ specs that hold in concrete system may be violated in abstract system

- spurious counterexamples
- we have to reduce the behaviors of the abstract system
  make it more concrete
- ... based on the counterexamples = CEGAR

Three sources of spurious behavior

- # processes decreasing or increasing
- # messages sent $\neq$ # processes which have sent a message
- unfair loops

Spurious behavior

abstraction adds behaviors (e.g., `x'=x+1` may lead to `x'` being equal to `x`)

$\Rightarrow$ specs that hold in concrete system may be violated in abstract system

- spurious counterexamples
- we have to reduce the behaviors of the abstract system
  make it more concrete
- ... based on the counterexamples = CEGAR

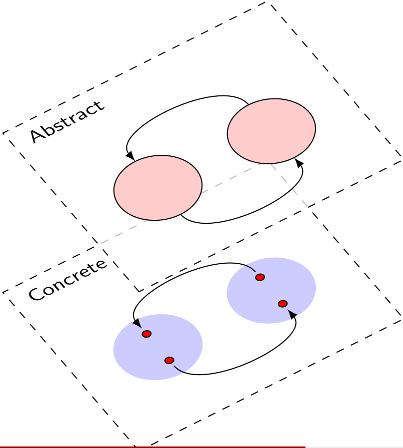
Three sources of spurious behavior

- # processes decreasing or increasing
- # messages sent $\neq$ # processes which have sent a message
- unfair loops

... and a new abstraction phenomenon

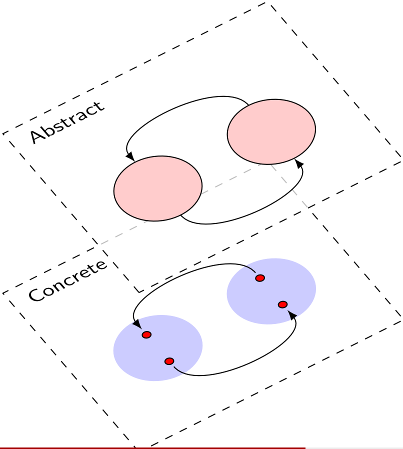
Parametric abst. refinement — uniformly spurious paths

Classic case:

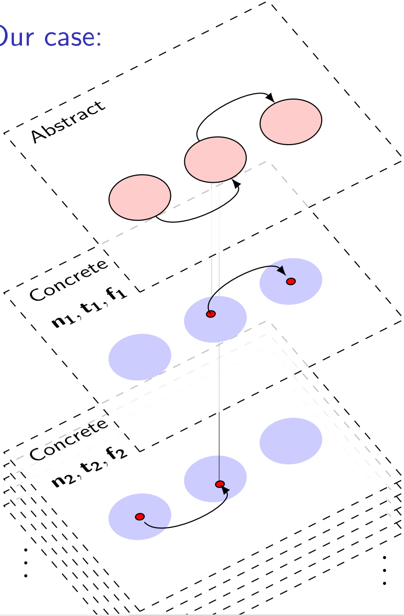


Parametric abst. refinement — uniformly spurious paths

Classic case:



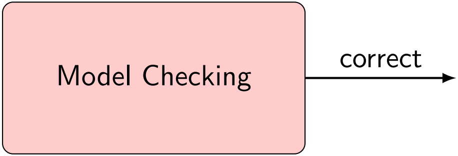
Our case:



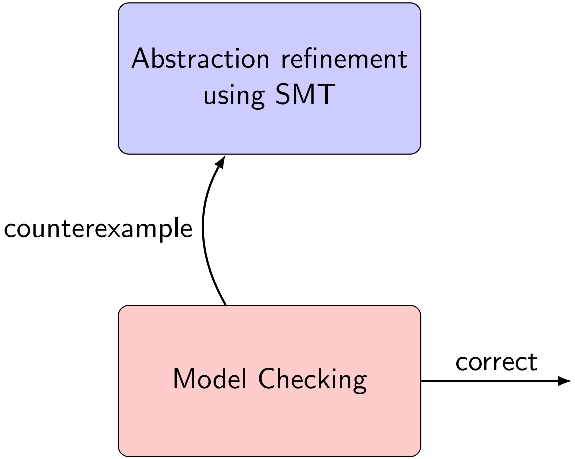
CEGAR — automated workflow

Model Checking

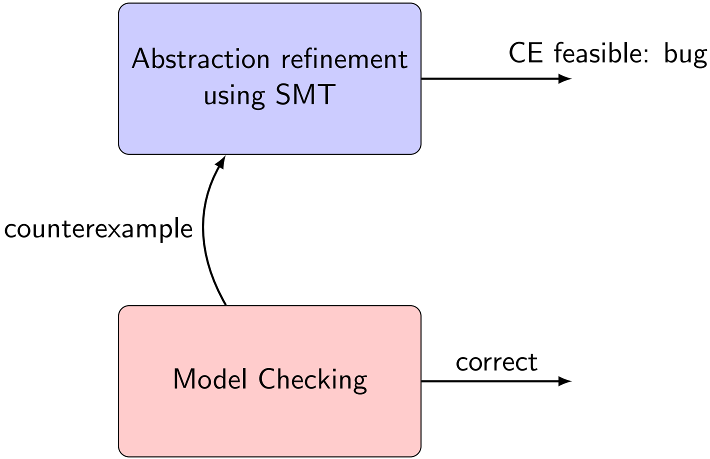
CEGAR — automated workflow



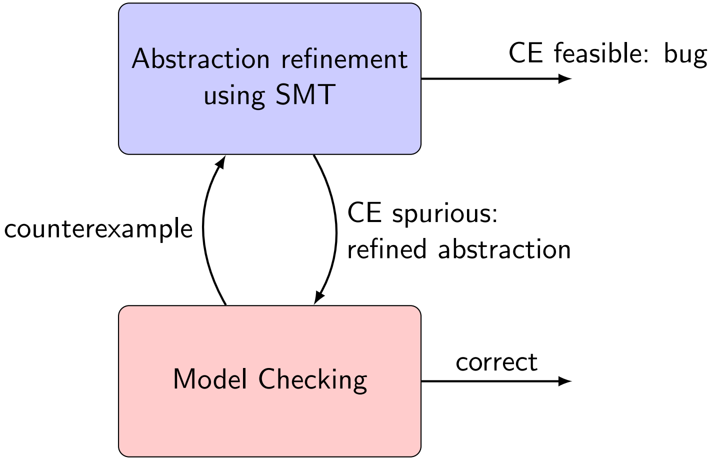
CEGAR — automated workflow



CEGAR — automated workflow



CEGAR — automated workflow



What is SMT?

recall SAT:

- given a Boolean formula, e.g., $(\neg a \vee \neg b \vee c) \wedge (\neg a \vee b \vee d \vee e)$
- is there an assignment of TRUE and FALSE to variables a, b, c, d, e such that the formula evaluates to TRUE?

What is SMT?

recall SAT:

- given a Boolean formula, e.g., $(\neg a \vee \neg b \vee c) \wedge (\neg a \vee b \vee d \vee e)$
- is there an assignment of TRUE and FALSE to variables a, b, c, d, e such that the formula evaluates to TRUE?

Satisfiability Modulo Theories (SMT) :

- here just linear arithmetics
- given a formula, e.g.,

$$x = y \;\wedge\; y = z \wedge u \neq x \wedge (x + y \leq 1 \wedge 2x + y = 1) \vee 3x + 2y \geq 3$$

- is there an assignment of values to u, x, y, z such that formula evaluates to TRUE?
- practically efficient tools: YICES, Z3

Counter example: losing processes

Output of data abstraction: 16 local states: $L = \{(sv, \widehat{nrcvd})\}$

with $sv \in \{v0, v1, sent, accepted\}$ and $\widehat{rcvd} \in \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}\}$

An abstract global state is $(\hat{k}, \widehat{nsnt})$,

where $\widehat{nsnt} \in \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}$ and $\hat{k} : L \to \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}$

Consider an abstract trace:

$$\widehat{nsnt}_1 = \mathrm{I}_0 \qquad \hat{k}_1(\ell) = \begin{cases} \mathrm{I}_{n-t}, & \text{if } \ell = (v1, \mathrm{I}_0) \\ \mathrm{I}_0, & \text{otherwise} \end{cases}$$

$$\widehat{nsnt}_2 = \mathrm{I}_1 \qquad \hat{k}_2(\ell) = \begin{cases} \mathrm{I}_{n-t}, & \text{if } \ell = (v1, \mathrm{I}_0) \\ \mathrm{I}_1, & \text{if } \ell = (sent, \mathrm{I}_0) \\ \mathrm{I}_0, & \text{otherwise} \end{cases}$$

$$\widehat{nsnt}_3 = \mathrm{I}_{t+1} \qquad \hat{k}_3(\ell) = \begin{cases} \mathrm{I}_{n-t}, & \text{if } \ell = (v1, \mathrm{I}_0) \\ \mathrm{I}_{t+1}, & \text{if } \ell = (sent, \mathrm{I}_0) \\ \mathrm{I}_0, & \text{otherwise} \end{cases}$$

Encode the last state in SMT as a conjunction T of the constraints:

resilience condition	$n > 3t \wedge t \geq f \wedge f \geq 0$	
zero counters	$(i \neq 4 \wedge i \neq 8) \to 0 \leq k_3[i] < 1$	UNSAT
non-zero counters	$n - t \leq k_3[4] \wedge t + 1 \leq k_3[8] < n - t$	
system size	$n - f = k_3[0] + k_3[1] + \cdots + k_3[15]$	

Counter example: losing processes

Output of data abstraction: 16 local states: $L = \{(sv, \hat{nrcvd})$
with $sv \in \{v0, v1, sent, accepted\}$ and $\hat{rcvd} \in \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}\}$

An abstract global state is $(\hat{k}, \hat{nsnt})$,
where $\hat{nsnt} \in \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}$ and $\hat{k} : L \to \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}$

Consider an abstract trace:

$$\hat{nsnt}_1 = \mathrm{I}_0 \qquad \hat{k}_1(\ell) = \begin{cases} \mathrm{I}_{n-t}, & \text{if } \ell = (v1, \mathrm{I}_0) \\ \mathrm{I}_0, & \text{otherwise} \end{cases}$$

$$\hat{nsnt}_2 = \mathrm{I}_1 \qquad \hat{k}_2(\ell) = \begin{cases} \mathrm{I}_{n-t}, & \text{if } \ell = (v1, \mathrm{I}_0) \\ \mathrm{I}_1, & \text{if } \ell = (sent, \mathrm{I}_0) \\ \mathrm{I}_0, & \text{otherwise} \end{cases}$$

$$\hat{nsnt}_3 = \mathrm{I}_{t+1} \qquad \hat{k}_3(\ell) = \begin{cases} \mathrm{I}_{n-t}, & \text{if } \ell = (v1, \mathrm{I}_0) \\ \mathrm{I}_{t+1}, & \text{if } \ell = (sent, \mathrm{I}_0) \\ \mathrm{I}_0, & \text{otherwise} \end{cases}$$

Encode the last state in SMT as a conjunction T of the constraints:

resilience condition	$n > 3t \wedge t \geq f \wedge f \geq 0$
zero counters	$(i \neq 4 \wedge i \neq 8) \to 0 \leq k_3[i] < 1$
non-zero counters	$n - t \leq k_3[4] \wedge t + 1 \leq k_3[8] < n - t$
system size	$n - f = k_3[0] + k_3[1] + \cdots + k_3[15]$

UNSAT

Counter example: losing processes

Output of data abstraction: 16 local states: $L = \{(sv, n\hat{rcvd})\}$
with $sv \in \{v0, v1, sent, accepted\}$ and $\hat{rcvd} \in \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}\}$

An abstract global state is $(\hat{k}, \hat{nsnt})$,
where $\hat{nsnt} \in \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}$ and $\hat{k} : L \to \{\mathrm{I}_0, \mathrm{I}_1, \mathrm{I}_{t+1}, \mathrm{I}_{n-t}\}$

Consider an abstract trace:

$$\hat{nsnt}_1 = \mathrm{I}_0 \qquad \hat{k}_1(\ell) = \begin{cases} \mathrm{I}_{n-t}, & \text{if } \ell = (v1, \mathrm{I}_0) \\ \mathrm{I}_0, & \text{otherwise} \end{cases}$$

$$\hat{nsnt}_2 = \mathrm{I}_1 \qquad \hat{k}_2(\ell) = \begin{cases} \mathrm{I}_{n-t}, & \text{if } \ell = (v1, \mathrm{I}_0) \\ \mathrm{I}_1, & \text{if } \ell = (sent, \mathrm{I}_0) \\ \mathrm{I}_0, & \text{otherwise} \end{cases}$$

$$\hat{nsnt}_3 = \mathrm{I}_{t+1} \qquad \hat{k}_3(\ell) = \begin{cases} \mathrm{I}_{n-t}, & \text{if } \ell = (v1, \mathrm{I}_0) \\ \mathrm{I}_{t+1}, & \text{if } \ell = (sent, \mathrm{I}_0) \\ \mathrm{I}_0, & \text{otherwise} \end{cases}$$

Encode the last state in SMT as a conjunction T of the constraints:

resilience condition	$n > 3t \wedge t \geq f \wedge f \geq 0$	
zero counters	$(i \neq 4 \wedge i \neq 8) \to 0 \leq k_3[i] < 1$	UNSAT
non-zero counters	$n - t \leq k_3[4] \wedge t + 1 \leq k_3[8] < n - t$	
system size	$n - f = k_3[0] + k_3[1] + \cdots + k_3[15]$	

Remove transitions

- We ask the SMT solver:
  is there a satisfiable assignment for T?
- if yes,

  then the state is OK, may be part of a real counterexample
- if not, then the state is spurious

  remove transitions to that state in the abstract system

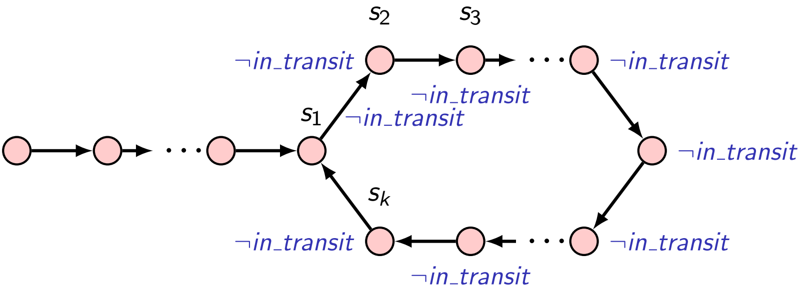
Liveness

- distributed algorithm requires reliable communication
- every message sent is eventually received
- $\neg in\_transit \equiv [\forall i.\ nrcvd_i \geq nsnt]$
- fairness $\mathbf{F}\,\mathbf{G}\,\neg in\_transit$ necessary to verify liveness,
  e.g., $\Big(\mathbf{F}\,\mathbf{G}\,\neg in\_transit \rightarrow \big(\mathbf{G}\,([\forall i.\ sv_i = v1] \rightarrow \mathbf{F}\,[\forall i.\ sv_i = accept])\big)\Big)$

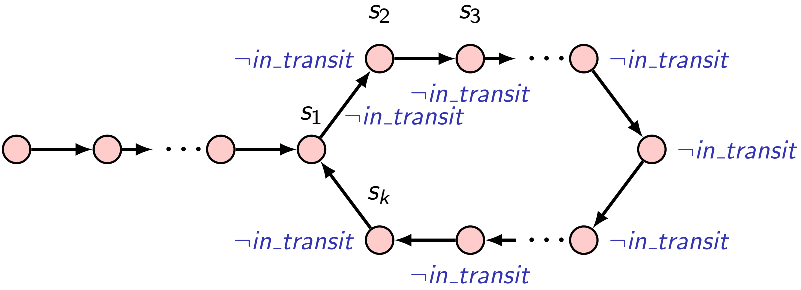
Liveness

- distributed algorithm requires reliable communication
- every message sent is eventually received
- $\neg in\_transit \equiv [\forall i.\ nrcvd_i \geq nsnt]$
- fairness $\mathbf{F}\,\mathbf{G}\,\neg in\_transit$ necessary to verify liveness,
  e.g., $\Big(\mathbf{F}\,\mathbf{G}\,\neg in\_transit \rightarrow \big(\mathbf{G}\,([\forall i.\ sv_i = v1] \rightarrow \mathbf{F}\,[\forall i.\ sv_i = accept])\big)\Big)$

counter example (lasso):

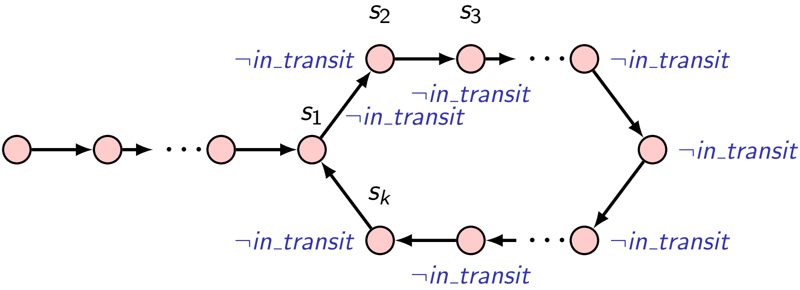


Liveness — fairness suppression



if there is a spurious s_j (all its concretizations violate $\neg in\_transit$), then the loop is spurious.

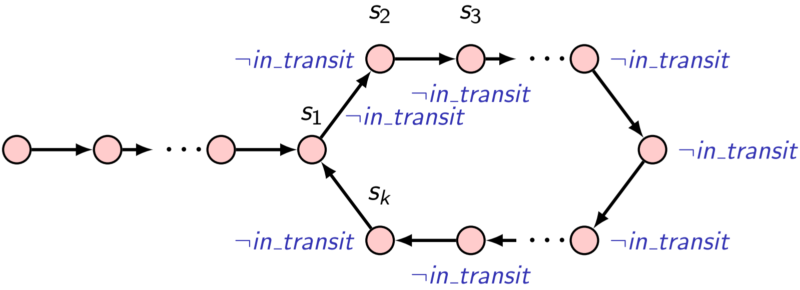
Liveness — fairness suppression



if there is a spurious s_j (all its concretizations violate $\neg in\_transit$), then the loop is spurious.

refine fairness to $\mathbf{F}\,\mathbf{G}\,\neg in\_transit \;\wedge\; \mathbf{G}\,\mathbf{F}\left(\bigwedge_{1\le j\le k} \text{“out of } s_j''\right)$

Liveness — fairness suppression



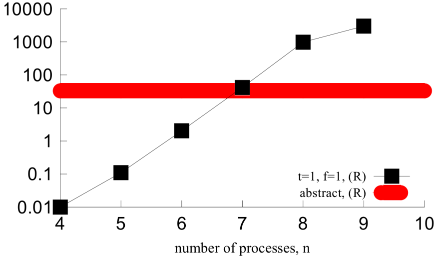
if there is a spurious s_j (all its concretizations violate $\neg in\_transit$), then the loop is spurious.

refine fairness to $\mathbf{F}\,\mathbf{G}\,\neg in\_transit \;\wedge\; \mathbf{G}\,\mathbf{F}\left(\bigwedge_{1 \le j \le k} \text{“out of } s_j\text{''}\right)$

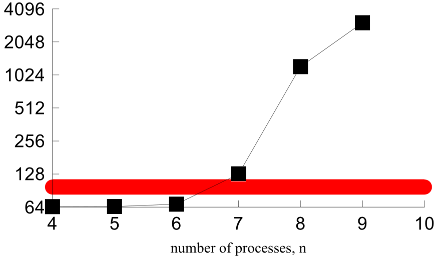
experimental evaluation

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.

Experimental results at a glance

Algorithm	Fault	Resilience	Property	Valid?	#Refinements	Time
ST87	Byz	$n > 3t$	**U**	✓	0	4 sec.
ST87	Byz	$n > 3t$	**C**	✓	10	32 sec.
ST87	Byz	$n > 3t$	**R**	✓	10	24 sec.
ST87	Symm	$n > 2t$	**U**	✓	0	1 sec.
ST87	Symm	$n > 2t$	**C**	✓	2	3 sec.
ST87	Symm	$n > 2t$	**R**	✓	12	16 sec.
ST87	Omit	$n > 2t$	**U**	✓	0	1 sec.
ST87	Omit	$n > 2t$	**C**	✓	5	6 sec.
ST87	Omit	$n > 2t$	**R**	✓	5	10 sec.
ST87	Clean	$n > t$	**U**	✓	0	2 sec.
ST87	Clean	$n > t$	**C**	✓	4	8 sec.
ST87	Clean	$n > t$	**R**	✓	13	31 sec.
CT96	Clean	$n > t$	**U**	✓	0	1 sec.
CT96	Clean	$n > t$	**A**	✓	0	1 sec.
CT96	Clean	$n > t$	**R**	✓	0	1 sec.
CT96	Clean	$n > t$	**C**	✗	0	1 sec.

When resilience condition is wrong...

Algorithm	Fault	Resilience	Property	Valid?	#Refinements	Time
ST87	Byz	$n > 3t \wedge f \leq t{+}1$	**U**	✗	9	56 sec.
ST87	Byz	$n > 3t \wedge f \leq t{+}1$	**C**	✗	11	52 sec.
ST87	Byz	$n > 3t \wedge f \leq t{+}1$	**R**	✗	10	17 sec.
ST87	Byz	$n \geq 3t \wedge f \leq t$	**U**	✓	0	5 sec.
ST87	Byz	$n \geq 3t \wedge f \leq t$	**C**	✓	9	32 sec.
ST87	Byz	$n \geq 3t \wedge f \leq t$	**R**	✗	30	78 sec.
ST87	Symm	$n > 2t \wedge f \leq t{+}1$	**U**	✗	0	2 sec.
ST87	Symm	$n > 2t \wedge f \leq t{+}1$	**C**	✗	2	4 sec.
ST87	Symm	$n > 2t \wedge f \leq t{+}1$	**R**	✓	8	12 sec.
ST87	Omit	$n \geq 2t \wedge f \leq t$	**U**	✓	0	1 sec.
ST87	Omit	$n \geq 2t \wedge f \leq t$	**C**	✗	0	2 sec.
ST87	Omit	$n \geq 2t \wedge f \leq t$	**R**	✗	0	2 sec.

Summary of results

- Abstraction tailored for distributed algorithms
    - threshold-based
    - fault-tolerant
    - allows to express different fault assumptions

- Verification of threshold-based fault-tolerant algorithms
    - with threshold guards that are widely used
    - Byzantine faults (and other)
    - for all system sizes

Related work: non-parameterized

Model checking of the small size instances:

- clock synchronization [Steiner, Rushby, Sorea, Pfeifer 2004]
- consensus [Tsuchiya, Schiper 2011]
- asynchronous agreement, folklore broadcast, condition-based consensus [John, Konnov, Schmid, Veith, Widder 2013]
- and more...

Related work: parameterized case

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: parameterized case

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:

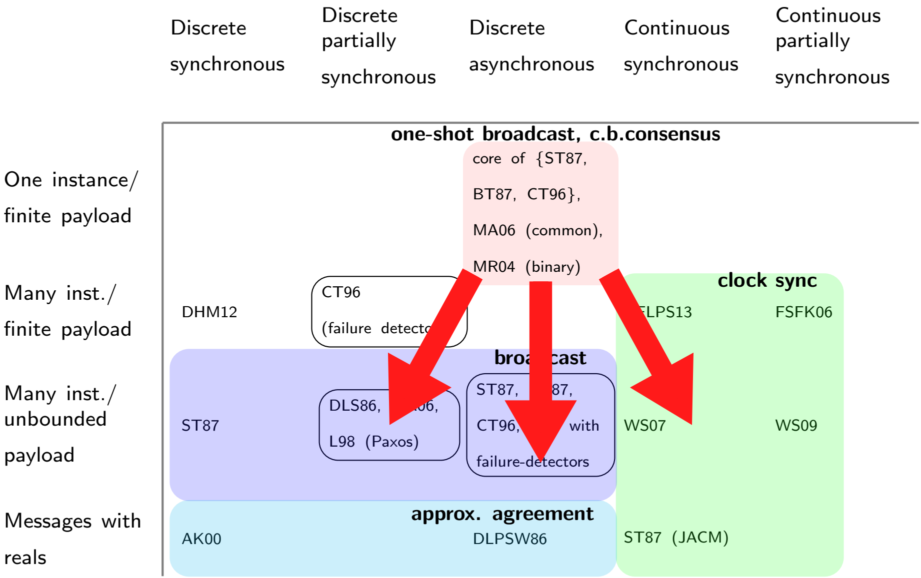
[Alberti, Ghilardi, Pagani, Ranise, Rossi 2010-2012]

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

Our current work

	Discrete synchronous	Discrete partially synchronous	Discrete asynchronous	Continuous synchronous	Continuous partially synchronous
			one-shot broadcast, c.b.consensus		
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

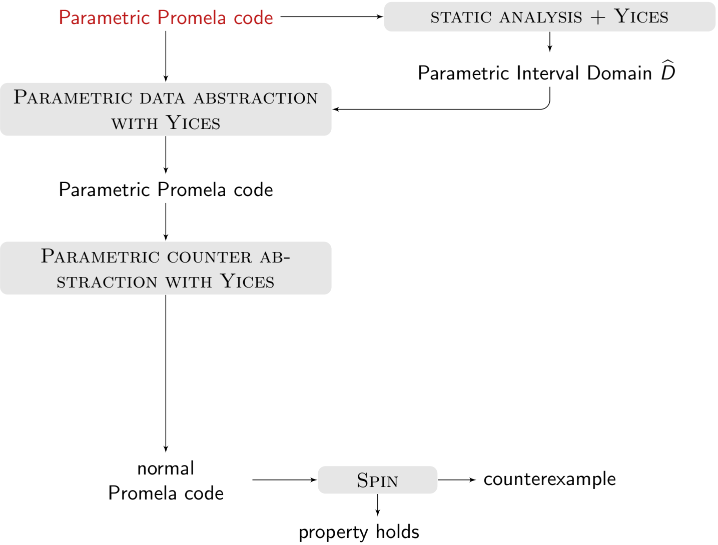


Thank you!

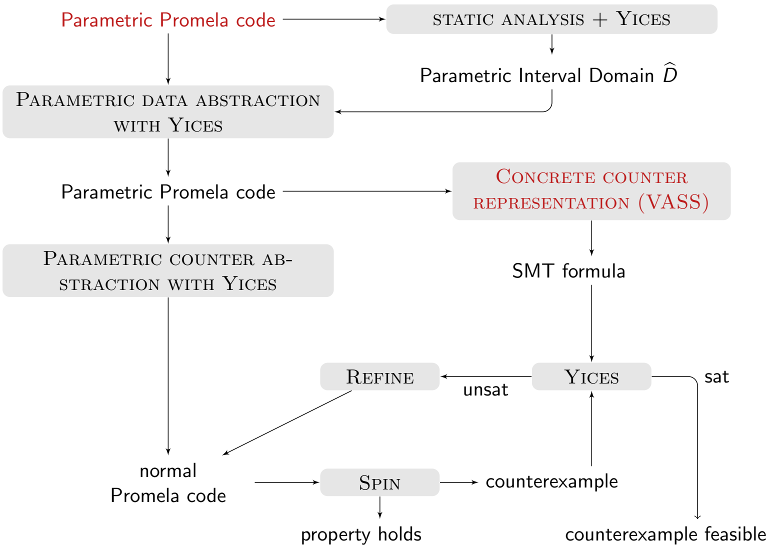
[http://forsyte.at/software/bymc]

the implementation

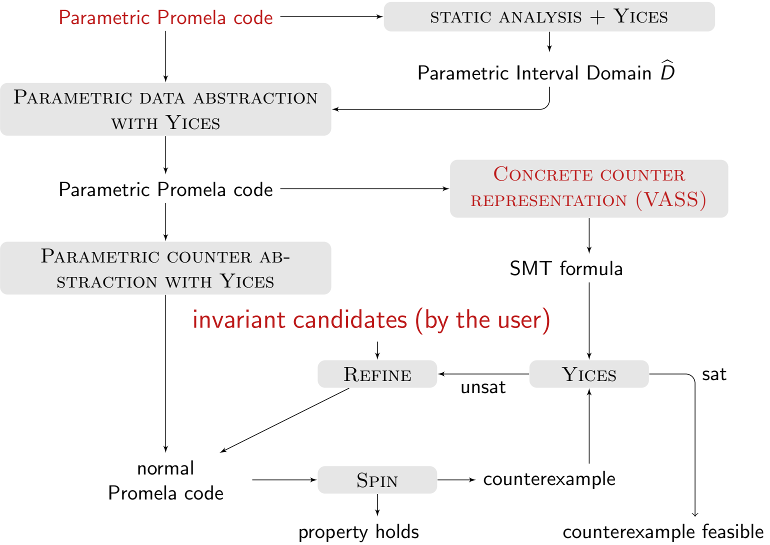
Tool Chain: ByMC



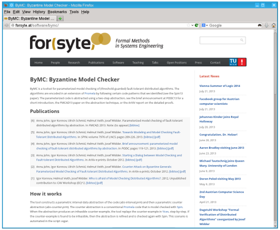
Tool Chain: ByMC

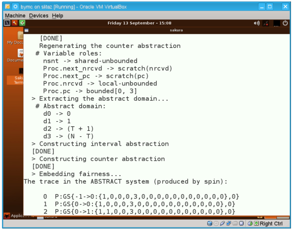


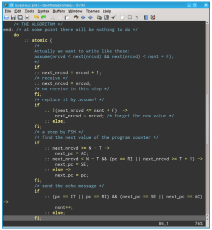
Tool Chain: ByMC



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

Running the tool — concrete case

- user specifies parameter value
- useful to check whether the code behaves as expected
- `$bymc/verifyco-spin "N=4,T=1,F=1" bcast-byz.pml relay`
  - model checking problem in directory "`./x/spin-bcast-byz-relay-N=4,T=1,F=1`"
  - in `concrete.prm`
    - parameters are replaced by numbers
    - process prototype is replaced with $N - F = 3$ active processes

Running the tool — parameterized model checking

- PIA data and counter abstraction
- finite-state model checking on abstract model
- `$bymc/verifypa-spin bcast-omit.pml relay`
  - model checking problem in directory "`./x/bcast-byz-relay-yymmdd-HHMM.*`"
  - directory contains
    - `abs-interval.prm`: result of the data abstraction;
    - `abs-counter.prm`: result of the counter abstraction;
    - `abs-vass.prm`: auxiliary abstraction for abstraction refinement;
    - `mc.out`: the last output by Spin;
    - `cex.trace`: the counterexample (if there is one);
    - `yices.log`: communication log with Yices.

Fairness, Refinement, and Invariants

- In the Byzantine case we have *in_transit* : $\forall i. (nrcvd_i \geq nsnt)$ and $\mathbf{G}\,\mathbf{F}\,\neg in\_transit$.
- In this case communication fairness implies computation fairness.
- But in the abstract version *nsnt* can deviate from the number of processes who sent the echo message.
- In this case the user formulates a simple state invariant candidate, e.g., $nsnt = K([sv = SE \vee sv = AC])$ (on the level of the original concrete system).
- The tool checks automatically, whether the candidate is actually a state invariant.
- After the abstraction the abstract version of the invariant restricts the behavior of the abstract transition system.

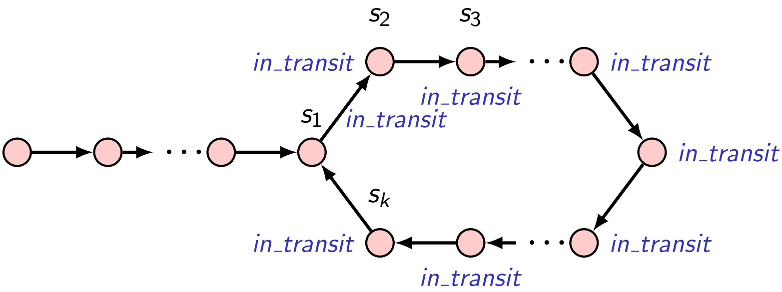
Parametric abstraction refinement — justice suppression

justice $\mathbf{G}\,\mathbf{F}\,\neg \mathit{in\_transit}$ necessary to verify liveness

Parametric abstraction refinement — justice suppression

justice $\mathbf{G}\,\mathbf{F}\,\neg \mathit{in\_transit}$ necessary to verify liveness

counter example:

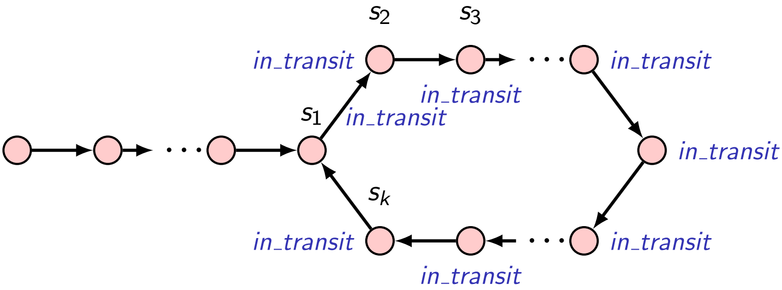


if $\forall j$ all concretizations of s_j violate $\neg \mathit{in\_transit}$, then CE is spurious.

Parametric abstraction refinement — justice suppression

justice $\mathbf{G}\,\mathbf{F}\,\neg \mathit{in\_transit}$ necessary to verify liveness

counter example:

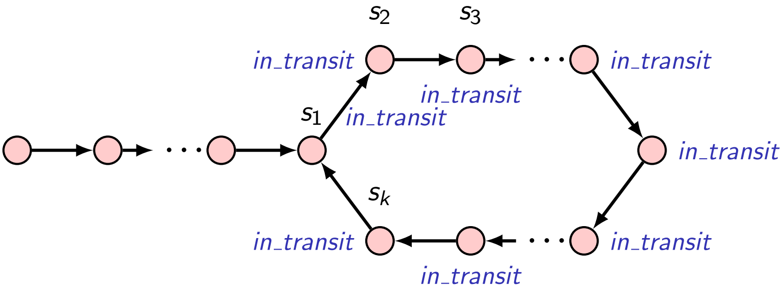


if $\forall j$ all concretizations of s_j violate $\neg \mathit{in\_transit}$, then CE is spurious.

refine justice to $\mathbf{G}\,\mathbf{F}\,\neg \mathit{in\_transit} \;\wedge\; \mathbf{G}\,\mathbf{F}\left(\bigvee_{1\le j\le k} \neg at(s_j)\right)$

Parametric abstraction refinement — justice suppression

justice $\mathbf{G}\,\mathbf{F}\,\neg\mathit{in\_transit}$ necessary to verify liveness
counter example:



if $\forall j$ all concretizations of s_j violate $\neg\mathit{in\_transit}$, then CE is spurious.

refine justice to $\mathbf{G}\,\mathbf{F}\,\neg\mathit{in\_transit} \;\wedge\; \mathbf{G}\,\mathbf{F}\left(\bigvee_{1\le j\le k} \neg at(s_j)\right)$

...we use unsat cores to refine several loops at once

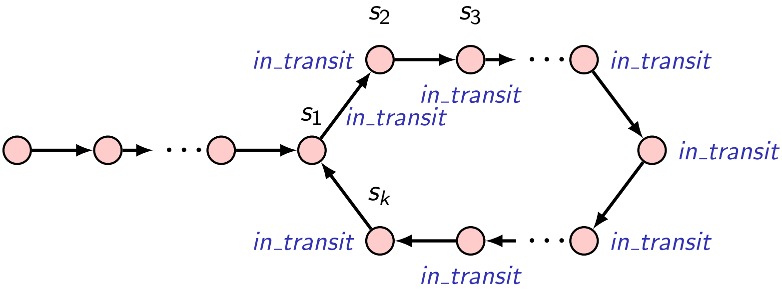
Parametric abstraction refinement — justice suppression

justice $\mathbf{G}\,\mathbf{F}\,\neg \mathit{in\_transit}$ necessary to verify liveness

Parametric abstraction refinement — justice suppression

justice **G F** $\neg$*in_transit* necessary to verify liveness

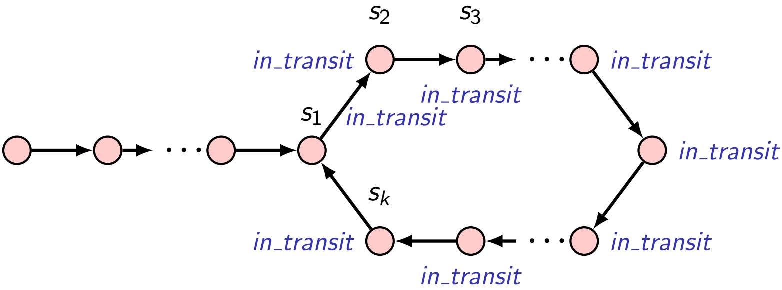
counter example:



if $\forall j$ all concretizations of s_j violate $\neg$*in_transit*, then CE is spurious.

Parametric abstraction refinement — justice suppression

justice $\mathbf{G}\,\mathbf{F}\,\neg\mathit{in\_transit}$ necessary to verify liveness
counter example:

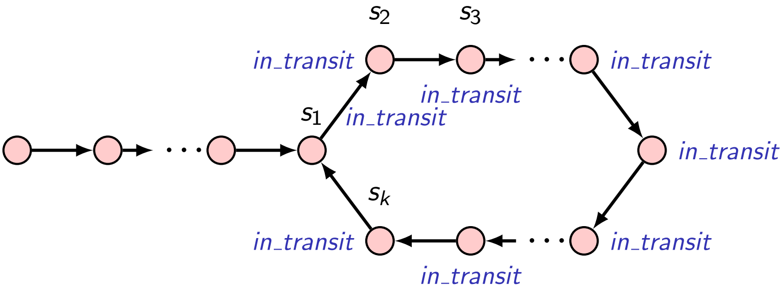


if $\forall j$ all concretizations of s_j violate $\neg\mathit{in\_transit}$, then CE is spurious.

refine justice to $\mathbf{G}\,\mathbf{F}\,\neg\mathit{in\_transit} \;\wedge\; \mathbf{G}\,\mathbf{F}\left(\bigvee_{1\leq j\leq k} \neg at(s_j)\right)$

Parametric abstraction refinement — justice suppression

justice $\mathbf{G}\,\mathbf{F}\,\neg \mathit{in\_transit}$ necessary to verify liveness
counter example:



if $\forall j$ all concretizations of s_j violate $\neg \mathit{in\_transit}$, then CE is spurious.

refine justice to $\mathbf{G}\,\mathbf{F}\,\neg \mathit{in\_transit} \;\wedge\; \mathbf{G}\,\mathbf{F}\left(\bigvee_{1\le j\le k} \neg at(s_j)\right)$

. . . we use unsat cores to refine several loops at once

asynchronous reliable broadcast (srikanth & toueg 1987)

the core of the classic broadcast algorithm from the da literature.
it solves an agreement problem depending on the inputs v_i.

Variables of process i

v_i: {0, 1} **init with** 0 **or** 1
$accept_i$: {0, 1} **init with** 0

An indivisible step:

```
if v_i = 1
then send (echo) to all;

if received (echo) from at least
   t + 1 distinct processes
   and not sent (echo) before
then send (echo) to all;

if received (echo) from at least
   n - t distinct processes
then accept_i := 1;
```

asynchronous reliable broadcast (srikanth & toueg 1987)

the core of the classic broadcast algorithm from the da literature.
it solves an agreement problem depending on the inputs v_i.

Variables of process i

```
v_i : {0, 1} init with 0 or 1
accept_i : {0, 1} init with 0
```

An indivisible step:

```
if v_i = 1
then send (echo) to all;

if received (echo) from at least
   t + 1 distinct processes
   and not sent (echo) before
then send (echo) to all;

if received (echo) from at least
   n - t distinct processes
then accept_i := 1;
```

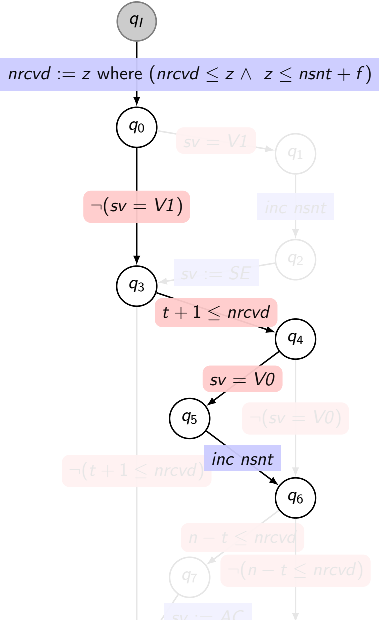
asynchronous

t byzantine faults

correct if $n > 3t$
resilience condition rc

parameterized process
skeleton $p(n, t)$

Abstract CFA



Abstract CFA