Unmasking Fault Tolerance

Quantifying deterministic recovery dynamics in probabilistic environments

Nils Müllner

Fakultät II, Department für Informatik
Carl von Ossietzky Universität Oldenburg

February 26, 2014

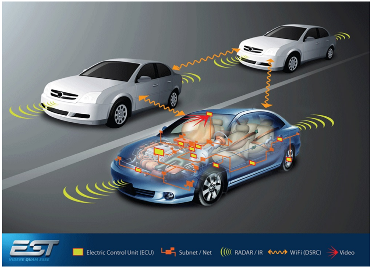
Agenda

1. Introduction
2. Concept
3. Computation
4. Composition
5. Conclusion
6. Literature

The world of fault tolerance

- Distributed systems are omnipresent,
  - like consumers of a power grid or
  - distributed sensors in a car or airplane.
- A distributed system comprises processes that can collaborate to provide a service, like providing energy or environmental data.

power grid, source: wordpress.com

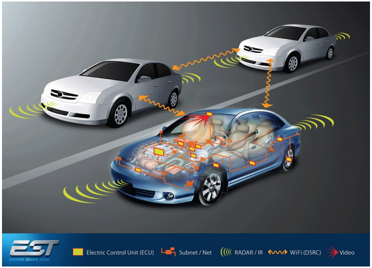


distributed sensors, source: mathworks.com

The world of fault tolerance

- Distributed systems are omnipresent,
  - like consumers of a power grid or
  - distributed sensors in a car or airplane.
- A distributed system comprises processes that can collaborate to provide a service, like providing energy or environmental data.

power grid, source: wordpress.com



distributed sensors, source: mathworks.com

The world of fault tolerance

- The service such a system provides
  a) can be critical (*hard* safety requirements)
     $\Rightarrow$ must never fail!
  b) or not (*soft* safety requirements)
     $\Rightarrow$ temporary downtimes are acceptable.
- Focus on category b).
  Less critical systems that can cope with temporary invalidation of safety.
- Systems that can recover from the effects of faults can run indefinitely.

The world of fault tolerance

- The service such a system provides
  a) can be critical (*hard* safety requirements)
     $\Rightarrow$ must never fail!
  b) or not (*soft* safety requirements)
     $\Rightarrow$ temporary downtimes are acceptable.
- Focus on category b).
  Less critical systems that can cope with temporary invalidation of safety.
- Systems that can recover from the effects of faults can run indefinitely.

The world of fault tolerance

- The service such a system provides
  a) can be critical (*hard* safety requirements)
     $\Rightarrow$ must never fail!
  b) or not (*soft* safety requirements)
     $\Rightarrow$ temporary downtimes are acceptable.
- Focus on category b).
  Less critical systems that can cope with temporary invalidation of safety.
- Systems that can recover from the effects of faults can run indefinitely.

The world of fault tolerance

- The service such a system provides
  a) can be critical (*hard* safety requirements)
  $\Rightarrow$ must never fail!
  b) or not (*soft* safety requirements)
  $\Rightarrow$ temporary downtimes are acceptable.
- Focus on category b).
  Less critical systems that can cope with temporary invalidation of safety.
- Systems that can recover from the effects of faults can run indefinitely.

The world of fault tolerance

- Distributed systems are prone to *sporadic transient faults*.
- Such faults occur *probabilistically* and can corrupt values stored in processes.
- Other *probabilistic* influence – like a central scheduler – might further influence the system.

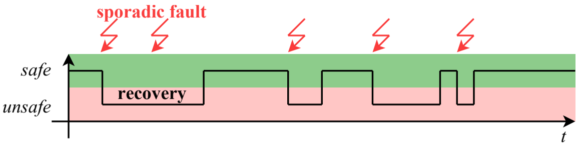
The world of fault tolerance

- Distributed systems are prone to *sporadic transient faults*.
- Such faults occur *probabilistically* and can corrupt values stored in processes.
- Other *probabilistic* influence – like a central scheduler – might further influence the system.

The world of fault tolerance

- Distributed systems are prone to *sporadic transient faults*.
- Such faults occur *probabilistically* and can corrupt values stored in processes.
- Other *probabilistic* influence – like a central scheduler – might further influence the system.

Key questions



- How well does such a distributed system *provide its service over time?*
- How well does such a distributed system *recover over time?*

Requirements

To answer these questions, we need:

1. a *measure* to quantify recovery and
2. a *method* to compute that measure.

Limiting window availability

. . . is the *probability*
that a system works according to its specification
(i.e. it is in a *safe* state)
at least once *within a time frame*,
considering the stationary as initial distribution.
[SSS2006,ATC2009,WAINA2011]

Limiting window availability

. . . is the *probability*
that a system works according to its specification
(i.e. it is in a *safe* state)
at least once *within a time frame*,
considering the stationary as initial distribution.
[SSS2006,ATC2009,WAINA2011]

Limiting window availability

. . . is the *probability*
that a system works according to its specification
(i.e. it is in a *safe* state)
at least once *within a time frame*,
considering the stationary as initial distribution.
[SSS2006,ATC2009,WAINA2011]

Limiting window availability

. . . is the *probability*
that a system works according to its specification
(i.e. it is in a *safe* state)
at least once *within a time frame*,
considering the stationary as initial distribution.
[SSS2006,ATC2009,WAINA2011]

Independent limit

Why *limiting*?

Generally, any arbitrary distribution is applicable.

For *indefinitely running* systems,
the stationary distribution is the interesting one.

Independent limit

Why *limiting*?

Generally, any arbitrary distribution is applicable.

For *indefinitely running* systems,
the stationary distribution is the interesting one.

Independent limit

Why *limiting*?

Generally, any arbitrary distribution is applicable.

For *indefinitely running* systems,
the stationary distribution is the interesting one.

System model and sporadic faults

A distributed system comprises *processes* taking serial execution steps.

π_1 π_2

Processes can *communicate to achieve common goal*, controlling a critical intersection.

An algorithm contains *both functional and recovery* instructions.

Sporadic *faults* let executing processes store arbitrary values.

Label	Guard enabled and $s = \pi_1$	Command	Label	Guard enabled and $s = \pi_2$	Command
a_1	$R_1 = \mathit{green} \wedge R_2 = \mathit{green}$	$R_1 := \mathit{red}_1$	a_{26}	$R_1 = \mathit{green} \wedge R_2 = \mathit{green}$	$R_2 := \mathit{red}_1$
a_2	$R_1 = \mathit{green} \wedge R_2 = \mathit{yellow}$	$R_1 := \mathit{red}_1$	a_{27}	$R_1 = \mathit{green} \wedge R_2 = \mathit{yellow}$	$R_2 := \mathit{red}_1$
a_3	$R_1 = \mathit{green} \wedge R_2 = \mathit{yellow}_1$	$R_1 := \mathit{red}_1$	a_{28}	$R_1 = \mathit{green} \wedge R_2 = \mathit{yellow}_1$	$R_2 := \mathit{red}_1$
a_4	$R_1 = \mathit{yellow} \wedge R_2 = \mathit{green}$	$R_1 := \mathit{red}_1$	a_{29}	$R_1 = \mathit{yellow} \wedge R_2 = \mathit{green}$	$R_2 := \mathit{red}_1$
a_5	$R_1 = \mathit{yellow}_1 \wedge R_2 = \mathit{green}$	$R_1 := \mathit{red}_1$	a_{30}	$R_1 = \mathit{yellow}_1 \wedge R_2 = \mathit{green}$	$R_2 := \mathit{red}_1$
a_6	$R_1 = \mathit{green} \wedge R_2 = \mathit{red}$	$R_1 := \mathit{red}$	a_{31}	$R_1 = \mathit{green} \wedge R_2 = \mathit{red}$	$R_2 := \mathit{red}_1$
a_7	$R_1 = \mathit{green} \wedge R_2 = \mathit{red}_1$	$R_1 := \mathit{yellow}_1$	a_{32}	$R_1 = \mathit{green} \wedge R_2 = \mathit{red}_1$	$R_2 := \mathit{red}_1$
a_8	$R_1 = \mathit{red} \wedge R_2 = \mathit{green}$	$R_1 := \mathit{red}_1$	a_{33}	$R_1 = \mathit{red} \wedge R_2 = \mathit{green}$	$R_2 := \mathit{red}$
a_9	$R_1 = \mathit{red}_1 \wedge R_2 = \mathit{green}$	$R_1 := \mathit{red}_1$	a_{34}	$R_1 = \mathit{red}_1 \wedge R_2 = \mathit{green}$	$R_2 := \mathit{yellow}_1$
a_{10}	$R_1 = \mathit{yellow} \wedge R_2 = \mathit{yellow}$	$R_1 := \mathit{red}_1$	a_{35}	$R_1 = \mathit{yellow} \wedge R_2 = \mathit{yellow}$	$R_2 := \mathit{red}_1$
a_{11}	$R_1 = \mathit{yellow} \wedge R_2 = \mathit{yellow}_1$	$R_1 := \mathit{red}_1$	a_{36}	$R_1 = \mathit{yellow} \wedge R_2 = \mathit{yellow}_1$	$R_2 := \mathit{red}_1$
a_{12}	$R_1 = \mathit{yellow}_1 \wedge R_2 = \mathit{yellow}$	$R_1 := \mathit{red}_1$	a_{37}	$R_1 = \mathit{yellow}_1 \wedge R_2 = \mathit{yellow}$	$R_2 := \mathit{red}_1$
a_{13}	$R_1 = \mathit{yellow}_1 \wedge R_2 = \mathit{yellow}_1$	$R_1 := \mathit{red}_1$	a_{38}	$R_1 = \mathit{yellow}_1 \wedge R_2 = \mathit{yellow}_1$	$R_2 := \mathit{red}_1$
a_{14}	$R_1 = \mathit{yellow} \wedge R_2 = \mathit{red}$	$R_1 := \mathit{red}$	a_{39}	$R_1 = \mathit{yellow} \wedge R_2 = \mathit{red}$	$R_2 := \mathit{red}_1$
a_{15}	$R_1 = \mathit{yellow}_1 \wedge R_2 = \mathit{red}$	$R_1 := \mathit{red}$	a_{40}	$R_1 = \mathit{yellow}_1 \wedge R_2 = \mathit{red}$	$R_2 := \mathit{red}_1$
a_{16}	$R_1 = \mathit{yellow} \wedge R_2 = \mathit{red}_1$	$R_1 := \mathit{green}$	a_{41}	$R_1 = \mathit{yellow} \wedge R_2 = \mathit{red}_1$	$R_2 := \mathit{red}_1$
a_{17}	$R_1 = \mathit{yellow}_1 \wedge R_2 = \mathit{red}_1$	$R_1 := \mathit{red}$	a_{42}	$R_1 = \mathit{yellow}_1 \wedge R_2 = \mathit{red}_1$	$R_2 := \mathit{red}_1$
a_{18}	$R_1 = \mathit{red} \wedge R_2 = \mathit{yellow}$	$R_1 := \mathit{red}_1$	a_{43}	$R_1 = \mathit{red} \wedge R_2 = \mathit{yellow}$	$R_2 := \mathit{red}$
a_{19}	$R_1 = \mathit{red} \wedge R_2 = \mathit{yellow}_1$	$R_1 := \mathit{red}_1$	a_{44}	$R_1 = \mathit{red} \wedge R_2 = \mathit{yellow}_1$	$R_2 := \mathit{red}$
a_{20}	$R_1 = \mathit{red}_1 \wedge R_2 = \mathit{yellow}$	$R_1 := \mathit{red}_1$	a_{45}	$R_1 = \mathit{red}_1 \wedge R_2 = \mathit{yellow}$	$R_2 := \mathit{green}$
a_{21}	$R_1 = \mathit{red}_1 \wedge R_2 = \mathit{yellow}_1$	$R_1 := \mathit{red}_1$	a_{46}	$R_1 = \mathit{red}_1 \wedge R_2 = \mathit{yellow}_1$	$R_2 := \mathit{red}_1$
a_{22}	$R_1 = \mathit{red} \wedge R_2 = \mathit{red}$	$R_1 := \mathit{red}_1$	a_{47}	$R_1 = \mathit{red} \wedge R_2 = \mathit{red}$	$R_2 := \mathit{red}$
a_{23}	$R_1 = \mathit{red}_1 \wedge R_2 = \mathit{red}$	$R_1 := \mathit{red}$	a_{48}	$R_1 = \mathit{red}_1 \wedge R_2 = \mathit{red}$	$R_2 := \mathit{yellow}$
a_{24}	$R_1 = \mathit{red} \wedge R_2 = \mathit{red}_1$	$R_1 := \mathit{green}$	a_{49}	$R_1 = \mathit{red} \wedge R_2 = \mathit{red}_1$	$R_2 := \mathit{red}$
a_{25}	$R_1 = \mathit{red}_1 \wedge R_2 = \mathit{red}_1$	$R_1 := \mathit{yellow}$	a_{50}	$R_1 = \mathit{red}_1 \wedge R_2 = \mathit{red}_1$	$R_2 := \mathit{green}$

Table: Traffic lights algorithm – guarded commands

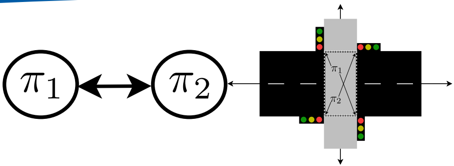
System model and sporadic faults

A distributed system comprises *processes* taking serial execution steps.

Processes can *communicate to achieve common goal*, controlling a critical intersection.

An algorithm contains *both functional and recovery* instructions.

Sporadic *faults* let executing processes store arbitrary values.



Label	Guard enabled and $s = \pi_1$	Command	Label	Guard enabled and $s = \pi_2$	Command
a_1	$R_1 = green \wedge R_2 = green$	$R_1 := red_1$	a_{26}	$R_1 = green \wedge R_2 = green$	$R_2 := red_1$
a_2	$R_1 = green \wedge R_2 = yellow$	$R_1 := red_1$	a_{27}	$R_1 = green \wedge R_2 = yellow$	$R_2 := red_1$
a_3	$R_1 = green \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{28}	$R_1 = green \wedge R_2 = yellow_1$	$R_2 := red_1$
a_4	$R_1 = yellow \wedge R_2 = green$	$R_1 := red_1$	a_{29}	$R_1 = yellow \wedge R_2 = green$	$R_2 := red_1$
a_5	$R_1 = yellow_1 \wedge R_2 = green$	$R_1 := red_1$	a_{30}	$R_1 = yellow_1 \wedge R_2 = green$	$R_2 := red_1$
a_6	$R_1 = green \wedge R_2 = red$	$R_1 := red$	a_{31}	$R_1 = green \wedge R_2 = red$	$R_2 := red_1$
a_7	$R_1 = green \wedge R_2 = red_1$	$R_1 := yellow_1$	a_{32}	$R_1 = green \wedge R_2 = red_1$	$R_2 := red_1$
a_8	$R_1 = red \wedge R_2 = green$	$R_1 := red_1$	a_{33}	$R_1 = red \wedge R_2 = green$	$R_2 := red$
a_9	$R_1 = red_1 \wedge R_2 = green$	$R_1 := red_1$	a_{34}	$R_1 = red_1 \wedge R_2 = green$	$R_2 := yellow_1$
a_{10}	$R_1 = yellow \wedge R_2 = yellow$	$R_1 := red_1$	a_{35}	$R_1 = yellow \wedge R_2 = yellow$	$R_2 := red_1$
a_{11}	$R_1 = yellow \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{36}	$R_1 = yellow \wedge R_2 = yellow_1$	$R_2 := red_1$
a_{12}	$R_1 = yellow_1 \wedge R_2 = yellow$	$R_1 := red_1$	a_{37}	$R_1 = yellow_1 \wedge R_2 = yellow$	$R_2 := red_1$
a_{13}	$R_1 = yellow_1 \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{38}	$R_1 = yellow_1 \wedge R_2 = yellow_1$	$R_2 := red_1$
a_{14}	$R_1 = yellow \wedge R_2 = red$	$R_1 := red$	a_{39}	$R_1 = yellow \wedge R_2 = red$	$R_2 := red_1$
a_{15}	$R_1 = yellow_1 \wedge R_2 = red$	$R_1 := red$	a_{40}	$R_1 = yellow_1 \wedge R_2 = red$	$R_2 := red_1$
a_{16}	$R_1 = yellow \wedge R_2 = red_1$	$R_1 := green$	a_{41}	$R_1 = yellow \wedge R_2 = red_1$	$R_2 := red_1$
a_{17}	$R_1 = yellow_1 \wedge R_2 = red_1$	$R_1 := red$	a_{42}	$R_1 = yellow_1 \wedge R_2 = red_1$	$R_2 := red_1$
a_{18}	$R_1 = red \wedge R_2 = yellow$	$R_1 := red_1$	a_{43}	$R_1 = red \wedge R_2 = yellow$	$R_2 := red$
a_{19}	$R_1 = red \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{44}	$R_1 = red \wedge R_2 = yellow_1$	$R_2 := red$
a_{20}	$R_1 = red_1 \wedge R_2 = yellow$	$R_1 := red_1$	a_{45}	$R_1 = red_1 \wedge R_2 = yellow$	$R_2 := green$
a_{21}	$R_1 = red_1 \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{46}	$R_1 = red_1 \wedge R_2 = yellow_1$	$R_2 := red_1$
a_{22}	$R_1 = red \wedge R_2 = red$	$R_1 := red_1$	a_{47}	$R_1 = red \wedge R_2 = red$	$R_2 := red$
a_{23}	$R_1 = red_1 \wedge R_2 = red$	$R_1 := red$	a_{48}	$R_1 = red_1 \wedge R_2 = red$	$R_2 := yellow$
a_{24}	$R_1 = red \wedge R_2 = red_1$	$R_1 := green$	a_{49}	$R_1 = red \wedge R_2 = red_1$	$R_2 := red$
a_{25}	$R_1 = red_1 \wedge R_2 = red_1$	$R_1 := yellow$	a_{50}	$R_1 = red_1 \wedge R_2 = red_1$	$R_2 := green$

Table: Traffic lights algorithm – guarded commands

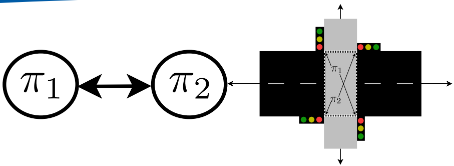
System model and sporadic faults

A distributed system comprises *processes* taking serial execution steps.

Processes can *communicate to achieve common goal*, controlling a critical intersection.

An algorithm contains *both functional and recovery* instructions.

Sporadic *faults* let executing processes store arbitrary values.



Label	Guard enabled and $\mathfrak{s} = \pi_1$	Command	Label	Guard enabled and $\mathfrak{s} = \pi_2$	Command
a_1	$R_1 = green \wedge R_2 = green$	$R_1 := red_1$	a_{26}	$R_1 = green \wedge R_2 = green$	$R_2 := red_1$
a_2	$R_1 = green \wedge R_2 = yellow$	$R_1 := red_1$	a_{27}	$R_1 = green \wedge R_2 = yellow$	$R_2 := red_1$
a_3	$R_1 = green \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{28}	$R_1 = green \wedge R_2 = yellow_1$	$R_2 := red_1$
a_4	$R_1 = yellow \wedge R_2 = green$	$R_1 := red_1$	a_{29}	$R_1 = yellow \wedge R_2 = green$	$R_2 := red_1$
a_5	$R_1 = yellow_1 \wedge R_2 = green$	$R_1 := red_1$	a_{30}	$R_1 = yellow_1 \wedge R_2 = green$	$R_2 := red_1$
a_6	$R_1 = green \wedge R_2 = red$	$R_1 := red$	a_{31}	$R_1 = green \wedge R_2 = red$	$R_2 := red_1$
a_7	$R_1 = green \wedge R_2 = red_1$	$R_1 := yellow_1$	a_{32}	$R_1 = green \wedge R_2 = red_1$	$R_2 := red_1$
a_8	$R_1 = red \wedge R_2 = green$	$R_1 := red_1$	a_{33}	$R_1 = red \wedge R_2 = green$	$R_2 := red$
a_9	$R_1 = red_1 \wedge R_2 = green$	$R_1 := red_1$	a_{34}	$R_1 = red_1 \wedge R_2 = green$	$R_2 := yellow_1$
a_{10}	$R_1 = yellow \wedge R_2 = yellow$	$R_1 := red_1$	a_{35}	$R_1 = yellow \wedge R_2 = yellow$	$R_2 := red_1$
a_{11}	$R_1 = yellow \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{36}	$R_1 = yellow \wedge R_2 = yellow_1$	$R_2 := red_1$
a_{12}	$R_1 = yellow_1 \wedge R_2 = yellow$	$R_1 := red_1$	a_{37}	$R_1 = yellow_1 \wedge R_2 = yellow$	$R_2 := red_1$
a_{13}	$R_1 = yellow_1 \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{38}	$R_1 = yellow_1 \wedge R_2 = yellow_1$	$R_2 := red_1$
a_{14}	$R_1 = yellow \wedge R_2 = red$	$R_1 := red$	a_{39}	$R_1 = yellow \wedge R_2 = red$	$R_2 := red_1$
a_{15}	$R_1 = yellow_1 \wedge R_2 = red$	$R_1 := red$	a_{40}	$R_1 = yellow_1 \wedge R_2 = red$	$R_2 := red_1$
a_{16}	$R_1 = yellow \wedge R_2 = red_1$	$R_1 := green$	a_{41}	$R_1 = yellow \wedge R_2 = red_1$	$R_2 := red_1$
a_{17}	$R_1 = yellow_1 \wedge R_2 = red_1$	$R_1 := red$	a_{42}	$R_1 = yellow_1 \wedge R_2 = red_1$	$R_2 := red_1$
a_{18}	$R_1 = red \wedge R_2 = yellow$	$R_1 := red_1$	a_{43}	$R_1 = red \wedge R_2 = yellow$	$R_2 := red$
a_{19}	$R_1 = red \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{44}	$R_1 = red \wedge R_2 = yellow_1$	$R_2 := red$
a_{20}	$R_1 = red_1 \wedge R_2 = yellow$	$R_1 := red_1$	a_{45}	$R_1 = red_1 \wedge R_2 = yellow$	$R_2 := green$
a_{21}	$R_1 = red_1 \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{46}	$R_1 = red_1 \wedge R_2 = yellow_1$	$R_2 := red_1$
a_{22}	$R_1 = red \wedge R_2 = red$	$R_1 := red_1$	a_{47}	$R_1 = red \wedge R_2 = red$	$R_2 := red$
a_{23}	$R_1 = red_1 \wedge R_2 = red$	$R_1 := red$	a_{48}	$R_1 = red_1 \wedge R_2 = red$	$R_2 := yellow$
a_{24}	$R_1 = red \wedge R_2 = red_1$	$R_1 := green$	a_{49}	$R_1 = red \wedge R_2 = red_1$	$R_2 := red$
a_{25}	$R_1 = red_1 \wedge R_2 = red_1$	$R_1 := yellow$	a_{50}	$R_1 = red_1 \wedge R_2 = red_1$	$R_2 := green$

Table: Traffic lights algorithm – guarded commands

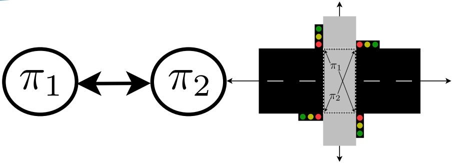
System model and sporadic faults

A distributed system comprises *processes* taking serial execution steps.

Processes can *communicate to achieve common goal*, controlling a critical intersection.

An algorithm contains *both functional and recovery* instructions.

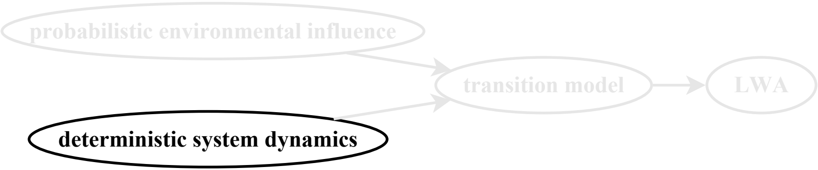
Sporadic *faults* let executing processes store arbitrary values.



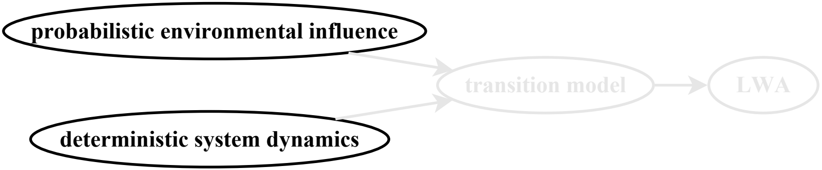
Label	Guard enabled and $\mathfrak{s} = \pi_1$	Command	Label	Guard enabled and $\mathfrak{s} = \pi_2$	Command
a_1	$R_1 = green \wedge R_2 = green$	$R_1 := red_1$	a_{26}	$R_1 = green \wedge R_2 = green$	$R_2 := red_1$
a_2	$R_1 = green \wedge R_2 = yellow$	$R_1 := red_1$	a_{27}	$R_1 = green \wedge R_2 = yellow$	$R_2 := red_1$
a_3	$R_1 = green \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{28}	$R_1 = green \wedge R_2 = yellow_1$	$R_2 := red_1$
a_4	$R_1 = yellow \wedge R_2 = green$	$R_1 := red_1$	a_{29}	$R_1 = yellow \wedge R_2 = green$	$R_2 := red_1$
a_5	$R_1 = yellow_1 \wedge R_2 = green$	$R_1 := red_1$	a_{30}	$R_1 = yellow_1 \wedge R_2 = green$	$R_2 := red_1$
a_6	$R_1 = green \wedge R_2 = red$	$R_1 := red$	a_{31}	$R_1 = green \wedge R_2 = red$	$R_2 := red_1$
a_7	$R_1 = green \wedge R_2 = red_1$	$R_1 := yellow_1$	a_{32}	$R_1 = green \wedge R_2 = red_1$	$R_2 := red_1$
a_8	$R_1 = red \wedge R_2 = green$	$R_1 := red_1$	a_{33}	$R_1 = red \wedge R_2 = green$	$R_2 := red$
a_9	$R_1 = red_1 \wedge R_2 = green$	$R_1 := red_1$	a_{34}	$R_1 = red_1 \wedge R_2 = green$	$R_2 := yellow_1$
a_{10}	$R_1 = yellow \wedge R_2 = yellow$	$R_1 := red_1$	a_{35}	$R_1 = yellow \wedge R_2 = yellow$	$R_2 := red_1$
a_{11}	$R_1 = yellow \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{36}	$R_1 = yellow \wedge R_2 = yellow_1$	$R_2 := red_1$
a_{12}	$R_1 = yellow_1 \wedge R_2 = yellow$	$R_1 := red_1$	a_{37}	$R_1 = yellow_1 \wedge R_2 = yellow$	$R_2 := red_1$
a_{13}	$R_1 = yellow_1 \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{38}	$R_1 = yellow_1 \wedge R_2 = yellow_1$	$R_2 := red_1$
a_{14}	$R_1 = yellow \wedge R_2 = red$	$R_1 := red$	a_{39}	$R_1 = yellow \wedge R_2 = red$	$R_2 := red_1$
a_{15}	$R_1 = yellow_1 \wedge R_2 = red$	$R_1 := red$	a_{40}	$R_1 = yellow_1 \wedge R_2 = red$	$R_2 := red_1$
a_{16}	$R_1 = yellow \wedge R_2 = red_1$	$R_1 := green$	a_{41}	$R_1 = yellow \wedge R_2 = red_1$	$R_2 := red_1$
a_{17}	$R_1 = yellow_1 \wedge R_2 = red_1$	$R_1 := red$	a_{42}	$R_1 = yellow_1 \wedge R_2 = red_1$	$R_2 := red_1$
a_{18}	$R_1 = red \wedge R_2 = yellow$	$R_1 := red_1$	a_{43}	$R_1 = red \wedge R_2 = yellow$	$R_2 := red$
a_{19}	$R_1 = red \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{44}	$R_1 = red \wedge R_2 = yellow_1$	$R_2 := red$
a_{20}	$R_1 = red_1 \wedge R_2 = yellow$	$R_1 := red_1$	a_{45}	$R_1 = red_1 \wedge R_2 = yellow$	$R_2 := green$
a_{21}	$R_1 = red_1 \wedge R_2 = yellow_1$	$R_1 := red_1$	a_{46}	$R_1 = red_1 \wedge R_2 = yellow_1$	$R_2 := red_1$
a_{22}	$R_1 = red \wedge R_2 = red$	$R_1 := red_1$	a_{47}	$R_1 = red \wedge R_2 = red$	$R_2 := red$
a_{23}	$R_1 = red_1 \wedge R_2 = red$	$R_1 := red$	a_{48}	$R_1 = red_1 \wedge R_2 = red$	$R_2 := yellow$
a_{24}	$R_1 = red \wedge R_2 = red_1$	$R_1 := green$	a_{49}	$R_1 = red \wedge R_2 = red_1$	$R_2 := red$
a_{25}	$R_1 = red_1 \wedge R_2 = red_1$	$R_1 := yellow$	a_{50}	$R_1 = red_1 \wedge R_2 = red_1$	$R_2 := green$

Table: Traffic lights algorithm – guarded commands

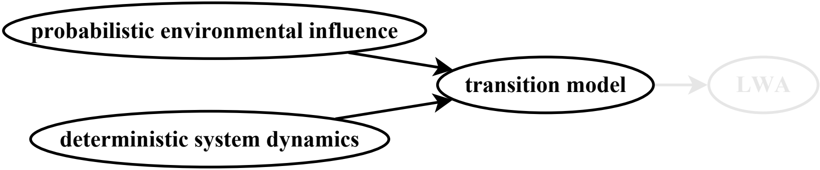
From system and environment to computing LWA



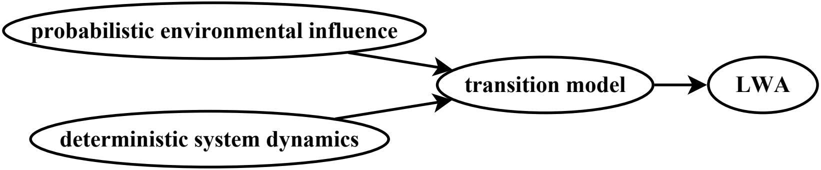
From system and environment to computing LWA



From system and environment to computing LWA



From system and environment to computing LWA



State space, transition model, safety

- *system state* $s_t = \langle g, r_1 \rangle$

- *state space*
  $S = \{\langle g, g\rangle, \langle g, y\rangle, \ldots, \langle r_1, r_1\rangle\}$

- *transition probability*
  $prob(\overrightarrow{\langle g, g\rangle, \langle g, r_1\rangle})$

- (state based) *safety*:
  at least one light shows, r or r_1

- partitions state space into *legal* and *illegal* states

State space, transition model, safety

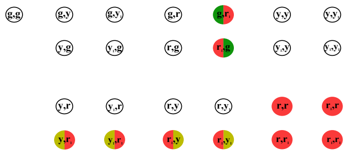
- *system state* $s_t = \langle g, r_1 \rangle$

- *state space*
  $\mathcal{S} = \{\langle g, g\rangle, \langle g, y\rangle, \ldots, \langle r_1, r_1\rangle\}$

- *transition probability*
  $prob(\overrightarrow{\langle g, g\rangle, \langle g, r_1\rangle})$

- (state based) *safety*:
  at least one light shows, r or r_1

- partitions state space into *legal* and *illegal* states



State space, transition model, safety

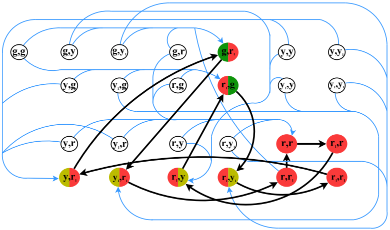
- *system state* $s_t = \langle g, r_1 \rangle$

- *state space*
  $\mathcal{S} = \{\langle g, g\rangle, \langle g, y\rangle, \ldots, \langle r_1, r_1\rangle\}$

- *transition probability*
  $prob(\overrightarrow{\langle g, g\rangle, \langle g, r_1\rangle})$

- (state based) *safety*:
  at least one light shows, r or r_1

- partitions state space into *legal* and *illegal* states



State space, transition model, safety

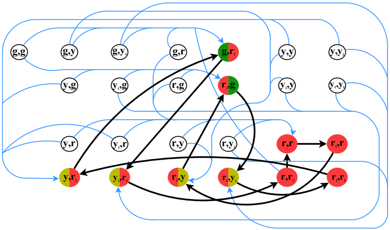
- *system state* $s_t = \langle g, r_1 \rangle$

- *state space*
  $\mathcal{S} = \{\langle g, g\rangle, \langle g, y\rangle, \ldots, \langle r_1, r_1\rangle\}$

- *transition probability*
  $prob(\overrightarrow{\langle g, g\rangle, \langle g, r_1\rangle})$

- (state based) *safety*:
  at least one light shows, r or r_1

- partitions state space into *legal* and *illegal* states



State space, transition model, safety

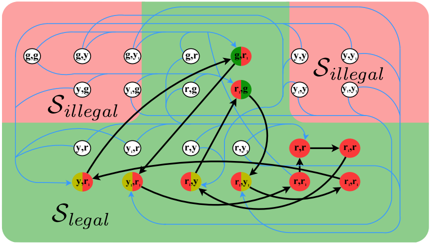
- *system state* $s_t = \langle g, r_1 \rangle$

- *state space*
  $\mathcal{S} = \{\langle g, g\rangle, \langle g, y\rangle, \dots, \langle r_1, r_1\rangle\}$

- *transition probability*
  $prob(\overrightarrow{\langle g, g\rangle, \langle g, r_1\rangle})$

- (state based) *safety*:
  at least one light shows, r or r_1

- partitions state space into *legal* and *illegal* states



Traffic light transition matrix

↓from/to→	g,g	g,y	g,y_1	y,g	y_1,g	g,r	g,r_1	r,g	r_1,g	y,y	y,y_1	y_1,y	y_1,y_1	y,r	y_1,r	y,r_1	y_1,r_1	r,y	r,y_1	r_1,y	r_1,y_1	r,r	r_1,r	r,r_1	r_1,r_1
g,g	2q	q	q	q	q	q	p+q	q	p+q																
g,y	q	2q	q			q	p+q			q		q						q		p+q					
g,y_1	q	q	2q			q	p+q				q		q						q		p+q				
y,g	q			2q	q			q	p+q	q	q			q		p+q									
y_1,g	q			q	2q			q	p+q			q	q		q		p+q								
g,r	q	q	q			2q	p+q							q	q							p+q	q		
g,r_1	q	q	q			q	p+2q									q	p+q							q	q
r,g	q			q	q			2q	p+q									q	q			p+q		q	
r_1,g	q			q	q			q	p+2q											q	p+q		q		q
y,y		q		q						2q	q	q		q		p+q		q		p+q					
y,y_1			q	q						q	2q		q	q		p+q			q		p+q				
y_1,y		q			q					q		2q	q		q		p+q	q		p+q					
y_1,y_1			q		q						q	q	2q		q		p+q		q		p+q				
y,r				q		q				q	q			2q	q	p+q						p+q	q		
y_1,r					q	q						q	q	q	2q		p+q					p+q	q		
y,r_1				q			p+q			q	q			q		p+2q	q							q	q
y_1,r_1					q		q					q	q		q	q	p+2q							p+q	q
r,y		q						q		q		q						2q	q	p+q		p+q		q	
r,y_1			q					q			q		q					q	2q		p+q	p+q		q	
r_1,y		q							p+q	q		q						q		p+2q	p		q		q
r_1,y_1			q						q		q		q						q	q	p+2q		q		p+q
r,r						q		q						q	q			q	q			p+2q	p+q	q	
r_1,r						q			q					q	q					p+q	q	q	p+2q		q
r,r_1							q	q								q	q	q	q			p+q		p+2q	q
r_1,r_1							q		p+q							p+q	q			q	q		q	q	2q

Table: Transition model $\mathcal{D}$ of traffic light example

number of states =

number of *possible values* to the power of

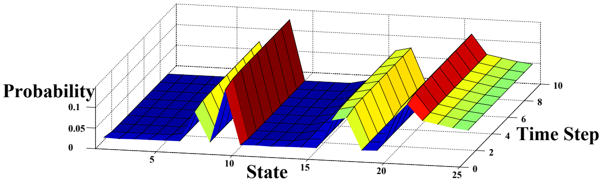
number of *registers* to the power of

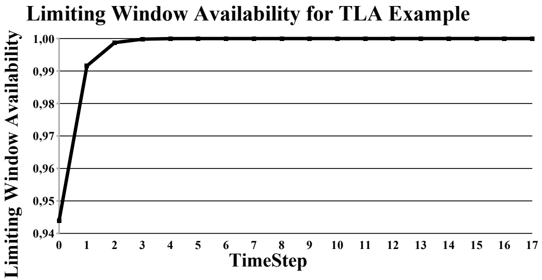
number of *processes*

here: 5 values in 1 register for each of 2 traffic lights processes:

$|\mathcal{S}| = 5^{1^2} = 25$

Traffic lights LWA





State space, transition model, safety

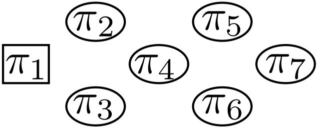
- small two process system for proof of concept:✓
- next step: larger, more complex system, different algorithm

State space, transition model, safety

- small two process system for proof of concept:✓
- next step: larger, more complex system, different algorithm

System model and sporadic faults

A distributed system comprises processes taking serial execution steps.



Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

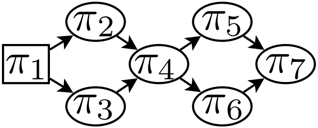
Sporadic faults let executing processes store wrong values (2).

const **id** := 0, *var* R, repeat { $R := 0$ }.	*const* **neighbors** := $\langle \pi_i, \ldots \rangle$, *const* **distance** :=min(**distance(neighbors)**)+1, *const* **set** := $\langle R_j, \ldots \rangle \mid \forall \pi_j :$ ($\pi_j \in$ **neighbors**)$\wedge$ (**distance**(π_j) =**distance**-1), *var* R, repeat{ $\neg((\exists R_i : \pi_i \in set \wedge R_i = 2)\, xor\, \exists R_i : \pi_i \in set \wedge R_i = 0))$ $\rightarrow R := 1;$ $\Box \exists R_i : \pi_i \in set \wedge R_i = 0$ $\rightarrow R := 0;$ $\Box \exists R_i : \pi_i \in set \wedge R_i = 2$ $\rightarrow R := 2$ }.

Figure: Broadcast algorithm – self-stabilizing (BASS)

System model and sporadic faults

A distributed system comprises processes taking serial execution steps.



Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

```
const id := 0,
var R,
repeat {
  R := 0
}.
```

```
const neighbors := ⟨π_i, ...⟩,
const distance
:=min(distance(neighbors))+1,
const set := ⟨R_j, ...⟩|∀π_j :
  (π_j ∈ neighbors)∧
  (distance(π_j) =distance−1),
var R,
repeat{
    ¬((∃R_i : π_i ∈ set ∧ R_i = 2)xor
      ∃R_i : π_i ∈ set ∧ R_i = 0))
      → R := 1;
  □∃R_i : π_i ∈ set ∧ R_i = 0
      → R := 0;
  □∃R_i : π_i ∈ set ∧ R_i = 2
      → R := 2
}.
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

Sporadic faults let executing processes store wrong values (2).

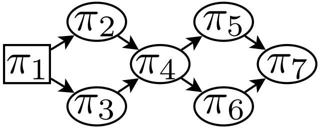
System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



const **id** := 0, *var* R, repeat { $R := 0$ }.	*const* **neighbors** $:= \langle \pi_i, \ldots \rangle$, *const* **distance** :=min(**distance**(**neighbors**))+1, *const* **set** $:= \langle R_j, \ldots \rangle \mid \forall \pi_j :$ ($\pi_j \in$ **neighbors**)$\wedge$ (**distance**(π_j) =**distance**-1), *var* R, repeat{ $\neg((\exists R_i : \pi_i \in set \wedge R_i = 2) xor$ $\exists R_i : \pi_i \in set \wedge R_i = 0))$ $\rightarrow R := 1;$ $\Box \exists R_i : \pi_i \in set \wedge R_i = 0$ $\rightarrow R := 0;$ $\Box \exists R_i : \pi_i \in set \wedge R_i = 2$ $\rightarrow R := 2$ }.

Figure: Broadcast algorithm – self-stabilizing (BASS)

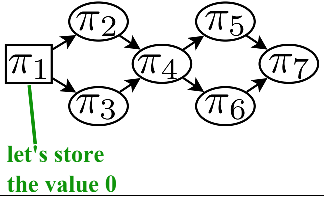
System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



const **id** := 0,
var R,
repeat {
  $R := 0$
}.

const **neighbors** $:= \langle \pi_i, \ldots \rangle$,
const **distance**
:=min(**distance**(**neighbors**))+1,
const **set** $:= \langle R_j, \ldots \rangle | \forall \pi_j :$
  ($\pi_j \in$ **neighbors**)$\wedge$
  (**distance**(π_j) =**distance**-1),
var R,
repeat{
  $\neg((\exists R_i : \pi_i \in set \wedge R_i = 2) xor$
    $\exists R_i : \pi_i \in set \wedge R_i = 0))$
    $\rightarrow R := 1;$
  $\Box \exists R_i : \pi_i \in set \wedge R_i = 0$
    $\rightarrow R := 0;$
  $\Box \exists R_i : \pi_i \in set \wedge R_i = 2$
    $\rightarrow R := 2$
}.

Figure: Broadcast algorithm – self-stabilizing (BASS)

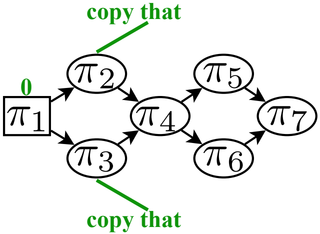
System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).




```
const id := 0,
var R,
repeat {
   R := 0
}.
```

```
const neighbors := ⟨π_i, ...⟩,
const distance
:=min(distance(neighbors))+1,
const set := ⟨R_j, ...⟩|∀π_j :
   (π_j ∈ neighbors)∧
   (distance(π_j) =distance−1),
var R,
repeat{
      ¬((∃R_i : π_i ∈ set ∧ R_i = 2)xor
         ∃R_i : π_i ∈ set ∧ R_i = 0))
         → R := 1;
     □∃R_i : π_i ∈ set ∧ R_i = 0
         → R := 0;
     □∃R_i : π_i ∈ set ∧ R_i = 2
         → R := 2
}.
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

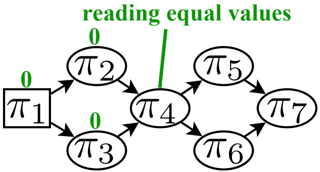
System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



const **id** := 0, *var* R, repeat { $R := 0$ }.	*const* **neighbors** $:= \langle \pi_i, \ldots \rangle$, *const* **distance** :=min(**distance**(**neighbors**))+1, *const* **set** $:= \langle R_j, \ldots \rangle \mid \forall \pi_j :$ ($\pi_j \in$ **neighbors**)$\wedge$ (**distance**(π_j) =**distance**-1), *var* R, repeat{ $\neg((\exists R_i : \pi_i \in set \wedge R_i = 2) xor$ $\exists R_i : \pi_i \in set \wedge R_i = 0))$ $\rightarrow R := 1;$ $\Box \exists R_i : \pi_i \in set \wedge R_i = 0$ $\rightarrow R := 0;$ $\Box \exists R_i : \pi_i \in set \wedge R_i = 2$ $\rightarrow R := 2$ }.

Figure: Broadcast algorithm – self-stabilizing (BASS)

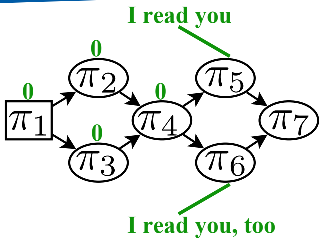
System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



const **id** := 0, *var* R, repeat { $R := 0$ }.	*const* **neighbors** $:= \langle \pi_i, \ldots \rangle$, *const* **distance** :=min(**distance**(**neighbors**))+1, *const* **set** $:= \langle R_j, \ldots \rangle \mid \forall \pi_j :$ ($\pi_j \in$ **neighbors**)$\wedge$ (**distance**(π_j) =**distance**-1), *var* R, repeat{ $\neg((\exists R_i : \pi_i \in set \wedge R_i = 2) xor$ $\exists R_i : \pi_i \in set \wedge R_i = 0))$ $\rightarrow R := 1;$ $\Box \exists R_i : \pi_i \in set \wedge R_i = 0$ $\rightarrow R := 0;$ $\Box \exists R_i : \pi_i \in set \wedge R_i = 2$ $\rightarrow R := 2$ }.

Figure: Broadcast algorithm – self-stabilizing (BASS)

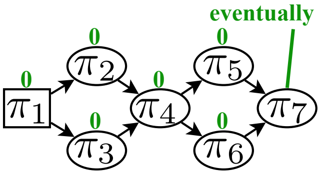
System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).



const **id** := 0, *var* R, repeat { $R := 0$ }.	*const* **neighbors** $:= \langle \pi_i, \ldots \rangle$, *const* **distance** :=min(**distance**(**neighbors**))+1, *const* **set** $:= \langle R_j, \ldots \rangle \vert \forall \pi_j :$ $(\pi_j \in$ **neighbors**$)\wedge$ (**distance**(π_j) =**distance**$-1)$, *var* R, repeat{ $\neg((\exists R_i : \pi_i \in set \wedge R_i = 2) xor$ $\exists R_i : \pi_i \in set \wedge R_i = 0))$ $\rightarrow R := 1;$ $\Box \exists R_i : \pi_i \in set \wedge R_i = 0$ $\rightarrow R := 0;$ $\Box \exists R_i : \pi_i \in set \wedge R_i = 2$ $\rightarrow R := 2$ }.

Figure: Broadcast algorithm – self-stabilizing (BASS)

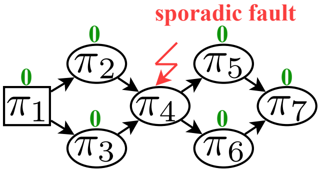
System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).




```
const id := 0,
var R,
repeat {
  R := 0
}.
```

```
const neighbors := ⟨π_i, ...⟩,
const distance
:=min(distance(neighbors))+1,
const set := ⟨R_j, ...⟩|∀π_j :
  (π_j ∈ neighbors)∧
  (distance(π_j) =distance−1),
var R,
repeat{
    ¬((∃R_i : π_i ∈ set ∧ R_i = 2)xor
      ∃R_i : π_i ∈ set ∧ R_i = 0))
      → R := 1;
  □∃R_i : π_i ∈ set ∧ R_i = 0
      → R := 0;
  □∃R_i : π_i ∈ set ∧ R_i = 2
      → R := 2
}.
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

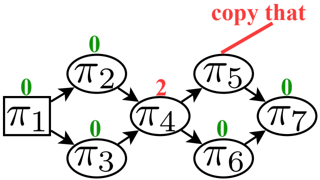
System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).




```
const id := 0,          const neighbors := ⟨π_i, ...⟩,
var R,                  const distance
repeat {                :=min(distance(neighbors))+1,
  R := 0                const set := ⟨R_j, ...⟩|∀π_j :
}.                        (π_j ∈ neighbors)∧
                          (distance(π_j) =distance−1),
                        var R,
                        repeat{
                              ¬((∃R_i : π_i ∈ set ∧ R_i = 2)xor
                                ∃R_i : π_i ∈ set ∧ R_i = 0))
                                → R := 1;
                           □∃R_i : π_i ∈ set ∧ R_i = 0
                                → R := 0;
                           □∃R_i : π_i ∈ set ∧ R_i = 2
                                → R := 2
                        }.
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

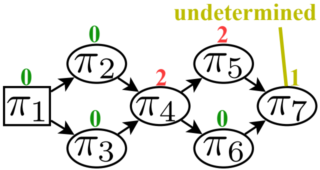
System model and sporadic faults

A distributed system comprises processes taking serial execution steps.

Processes can communicate to achieve common goal, like agreeing on one value.

An algorithm contains both functional and recovery instructions.

Sporadic faults let executing processes store wrong values (2).

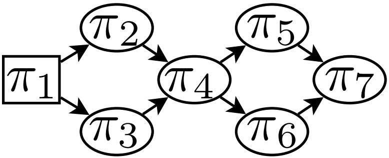



```
const id := 0,
var R,
repeat {
   R := 0
}.
```

```
const neighbors := ⟨πi, ...⟩,
const distance
:=min(distance(neighbors))+1,
const set := ⟨Rj, ...⟩|∀πj :
   (πj ∈ neighbors)∧
   (distance(πj) =distance−1),
var R,
repeat{
      ¬((∃Ri : πi ∈ set ∧ Ri = 2)xor
        ∃Ri : πi ∈ set ∧ Ri = 0))
        → R := 1;
   □∃Ri : πi ∈ set ∧ Ri = 0
        → R := 0;
   □∃Ri : πi ∈ set ∧ Ri = 2
        → R := 2
}.
```

Figure: Broadcast algorithm – self-stabilizing (BASS)

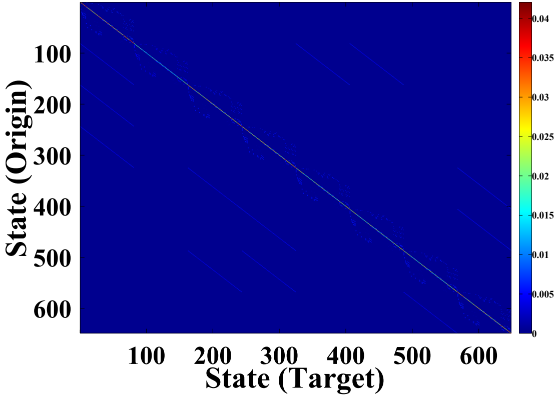
Broadcast algorithm – self-stabilizing



7 processes, 1 register each, 3 possible values[1]:
here: $|\mathcal{S}| = 2^3 \cdot 3^4 = 648$

[1] Processes $\pi_1 - \pi_3$ cannot derive 1.

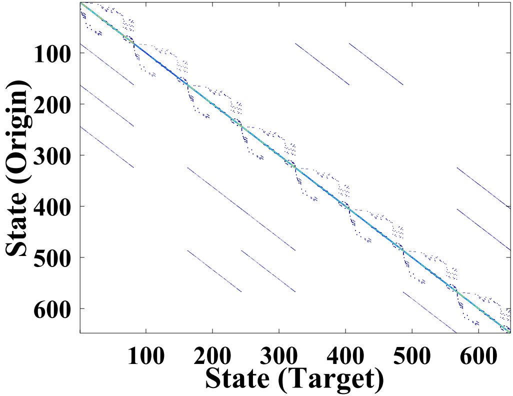
Broadcast algorithm – self-stabilizing



7 processes, 1 register each, 3 possible values[1]:
here: $|\mathcal{S}| = 2^3 \cdot 3^4 = 648$

[1] Processes $\pi_1 - \pi_3$ cannot derive 1.

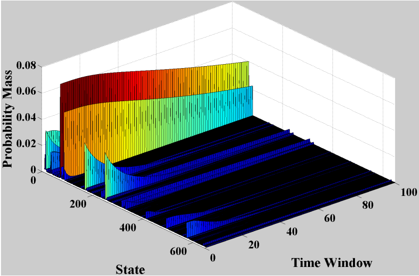
Broadcast algorithm – self-stabilizing

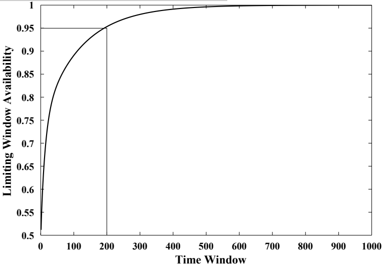


7 processes, 1 register each, 3 possible values[1]:
here: $|\mathcal{S}| = 2^3 \cdot 3^4 = 648$

[1] Processes $\pi_1 - \pi_3$ cannot derive 1.

LWA Examples



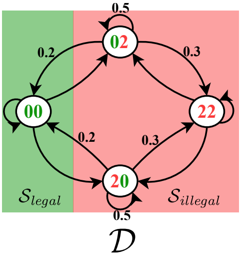


Concluding LWA

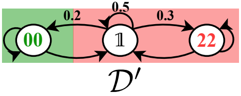
- ☺ a measure to quantify recovery
- ☺ a method to compute that measure
- ☹ yet, inherently confined by state space explosion

Lumping

- The first step in reducing the size of the state space is *lumping*.
- *Lumping* coalesces *bisimilar* states, i.e. states *that do the same* in a transition model



becomes



Decomposition

- Lumping requires a transition model.
- But a transition model is likely too large to be constructed at one go.
- Idea: Successive construction of transition model.
  - Mutually independent processes ⇒ already discussed [Boudali et al., 2010, AINA2014]
  - Hierarchically structured ⇒ challenging, but feasible [WAINA2011,AINA2012,JCSS2013]

Decomposition

- Lumping requires a transition model.
- But a transition model is likely too large to be constructed at one go.
- Idea: Successive construction of transition model.
  - Mutually independent processes $\Rightarrow$ already discussed [Boudali et al., 2010, AINA2014]
  - Hierarchically structured $\Rightarrow$ challenging, but feasible [WAINA2011,AINA2012,JCSS2013]

Decomposition

- Lumping requires a transition model.
- But a transition model is likely too large to be constructed at one go.
- Idea: Successive construction of transition model.
  - Mutually independent processes ⇒ already discussed [Boudali et al. , 2010, AINA2014]
  - Hierarchically structured ⇒ challenging, but feasible [WAINA2011,AINA2012,JCSS2013]

Decomposition

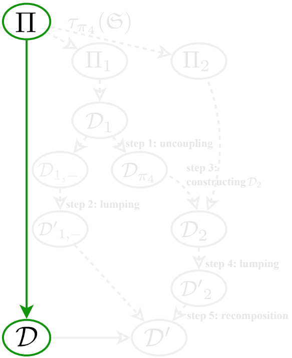
- Lumping requires a transition model.
- But a transition model is likely too large to be constructed at one go.
- Idea: Successive construction of transition model.
  - Mutually independent processes ⇒ already discussed [Boudali et al. , 2010, AINA2014]
  - Hierarchically structured ⇒ challenging, but feasible [WAINA2011,AINA2012,JCSS2013]

Decomposition

- Lumping requires a transition model.
- But a transition model is likely too large to be constructed at one go.
- Idea: Successive construction of transition model.
  - Mutually independent processes $\Rightarrow$ already discussed [Boudali et al. , 2010, AINA2014]
  - Hierarchically structured $\Rightarrow$ challenging, but feasible [WAINA2011,AINA2012,JCSS2013]

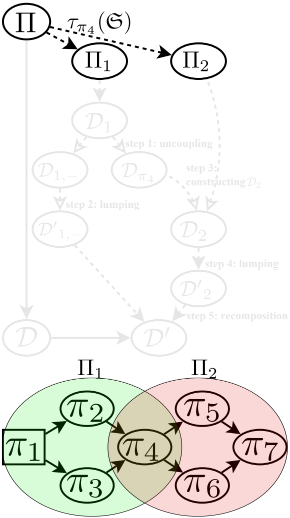
Tackling state space explosion

- Ideally, tractable system size.
- If not, then slice system.
- Build transition model of upper sub-system Π_1.
- Uncouple gateway process, i.e. $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$.
- Lump $\mathcal{D}_{1,-}$ to $\mathcal{D}'_{1,-}$.
- Build transition model of lower sub-system Π_2.
- Lump $\mathcal{D}_2$ to $\mathcal{D}'_2$.
- Recompose $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$.



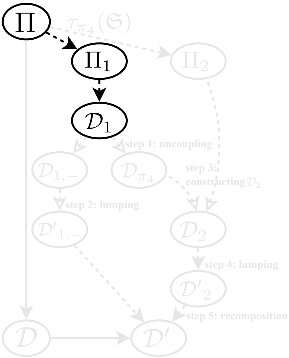
Tackling state space explosion

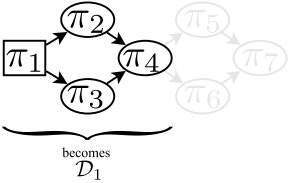
- Ideally, tractable system size.
- If not, then slice system.
- Build transition model of upper sub-system Π_1.
- Uncouple gateway process, i.e. $\mathcal{D}_1 \to \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$.
- Lump $\mathcal{D}_{1,-}$ to $\mathcal{D}'_{1,-}$.
- Build transition model of lower sub-system Π_2.
- Lump $\mathcal{D}_2$ to $\mathcal{D}'_2$.
- Recompose $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$.



Tackling state space explosion

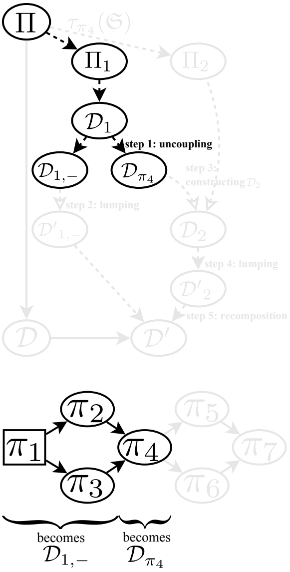
- Ideally, tractable system size.
- If not, then slice system.
- Build transition model of upper sub-system Π_1.
- Uncouple gateway process, i.e. $\mathcal{D}_1 \to \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$.
- Lump $\mathcal{D}_{1,-}$ to $\mathcal{D}'_{1,-}$.
- Build transition model of lower sub-system Π_2.
- Lump $\mathcal{D}_2$ to $\mathcal{D}'_2$.
- Recompose $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$.





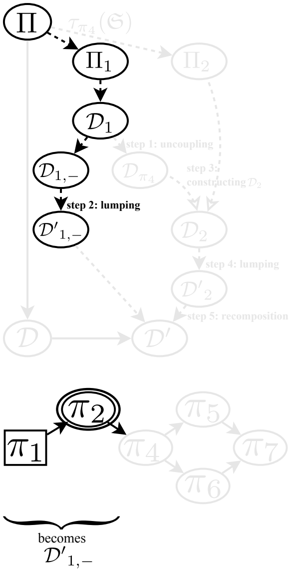
Tackling state space explosion

- Ideally, tractable system size.
- If not, then slice system.
- Build transition model of upper sub-system Π_1.
- Uncouple gateway process, i.e. $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$.
- Lump $\mathcal{D}_{1,-}$ to $\mathcal{D}'_{1,-}$.
- Build transition model of lower sub-system Π_2.
- Lump $\mathcal{D}_2$ to $\mathcal{D}'_2$.
- Recompose $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$.



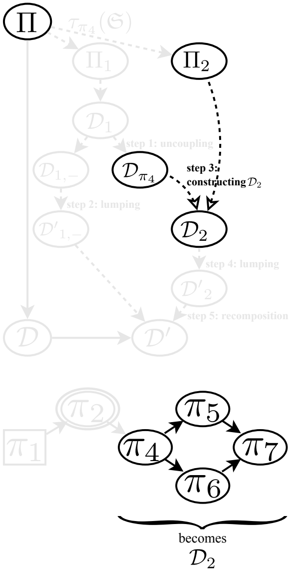
Tackling state space explosion

- Ideally, tractable system size.
- If not, then slice system.
- Build transition model of upper sub-system Π_1.
- Uncouple gateway process, i.e. $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$.
- Lump $\mathcal{D}_{1,-}$ to $\mathcal{D}'_{1,-}$.
- Build transition model of lower sub-system Π_2.
- Lump $\mathcal{D}_2$ to $\mathcal{D}'_2$.
- Recompose $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$.



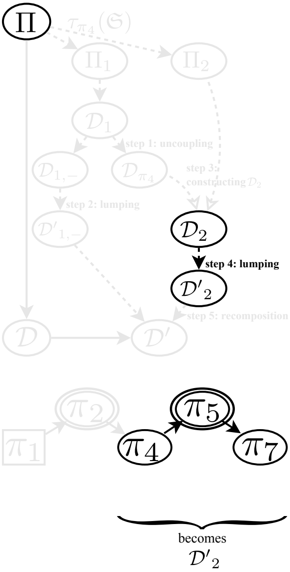
Tackling state space explosion

- Ideally, tractable system size.
- If not, then slice system.
- Build transition model of upper sub-system Π_1.
- Uncouple gateway process, i.e. $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$.
- Lump $\mathcal{D}_{1,-}$ to $\mathcal{D}'_{1,-}$.
- Build transition model of lower sub-system Π_2.
- Lump $\mathcal{D}_2$ to $\mathcal{D}'_2$.
- Recompose $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$.



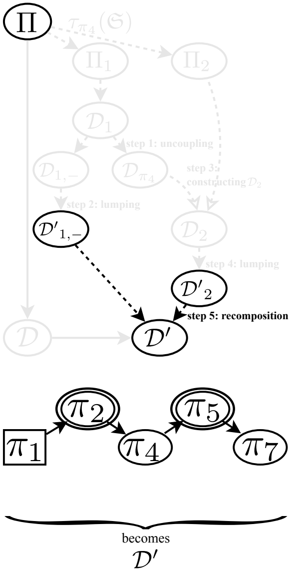
Tackling state space explosion

- Ideally, tractable system size.
- If not, then slice system.
- Build transition model of upper sub-system Π_1.
- Uncouple gateway process, i.e. $\mathcal{D}_1 \rightarrow \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$.
- Lump $\mathcal{D}_{1,-}$ to $\mathcal{D}'_{1,-}$.
- Build transition model of lower sub-system Π_2.
- Lump $\mathcal{D}_2$ to $\mathcal{D}'_2$.
- Recompose $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$.



Tackling state space explosion

- Ideally, tractable system size.
- If not, then slice system.
- Build transition model of upper sub-system Π_1.
- Uncouple gateway process, i.e. $\mathcal{D}_1 \to \mathcal{D}_{1,-} \otimes \mathcal{D}_{\pi_4}$.
- Lump $\mathcal{D}_{1,-}$ to $\mathcal{D}'_{1,-}$.
- Build transition model of lower sub-system Π_2.
- Lump $\mathcal{D}_2$ to $\mathcal{D}'_2$.
- Recompose $\mathcal{D}' = \mathcal{D}'_{1,-} \otimes \mathcal{D}'_2$.



Tackling state space explosion

- No state space larger than 81 states during computation:
- $|\mathcal{S}| = 648$ while $|\mathcal{S}'| = 324$, and
- only half the states and quarter the transitions!

Synopsis of decomposition

1. Fault tolerant systems often comprise uniform components $\Rightarrow$ High potential for lumping.
2. Structured systems are challenging, as propagation through gateway processes must be accounted for.
3. Self-stabilizing systems often rely on hierarchic structures and uniform processes to facilitate stabilization.
4. Combining decomposition and lumping can dampen state space explosion for analysis of self-stabilizing systems.

Synopsis of decomposition

1. Fault tolerant systems often comprise uniform components $\Rightarrow$ High potential for lumping.
2. Structured systems are challenging, as propagation through gateway processes must be accounted for.
3. Self-stabilizing systems often rely on hierarchic structures and uniform processes to facilitate stabilization.
4. Combining decomposition and lumping can dampen state space explosion for analysis of self-stabilizing systems.

Synopsis of decomposition

1. Fault tolerant systems often comprise uniform components
   $\Rightarrow$ High potential for lumping.
2. Structured systems are challenging, as propagation through gateway processes must be accounted for.
3. Self-stabilizing systems often rely on hierarchic structures and uniform processes to facilitate stabilization.
4. Combining decomposition and lumping can dampen state space explosion for analysis of self-stabilizing systems.

Synopsis of decomposition

1. Fault tolerant systems often comprise uniform components $\Rightarrow$ High potential for lumping.
2. Structured systems are challenging, as propagation through gateway processes must be accounted for.
3. Self-stabilizing systems often rely on hierarchic structures and uniform processes to facilitate stabilization.
4. Combining decomposition and lumping can dampen state space explosion for analysis of self-stabilizing systems.

Conclusion

- Area of application: non-critical dependable distributed systems exposed to sporadic transient faults.
- Goal: measure recovery
- Method: transition model analysis
- Challenge: state space explosion
- Solution: efficiently combining lumping and decomposition

Conclusion

- Area of application: non-critical dependable distributed systems exposed to sporadic transient faults.
- Goal: measure recovery
- Method: transition model analysis
- Challenge: state space explosion
- Solution: efficiently combining lumping and decomposition

Conclusion

- Area of application: non-critical dependable distributed systems exposed to sporadic transient faults.
- Goal: measure recovery
- Method: transition model analysis
- Challenge: state space explosion
- Solution: efficiently combining lumping and decomposition

Conclusion

- Area of application: non-critical dependable distributed systems exposed to sporadic transient faults.
- Goal: measure recovery
- Method: transition model analysis
- Challenge: state space explosion
- Solution: efficiently combining lumping and decomposition

Conclusion

- Area of application: non-critical dependable distributed systems exposed to sporadic transient faults.
- Goal: measure recovery
- Method: transition model analysis
- Challenge: state space explosion
- Solution: efficiently combining lumping and decomposition

Summarizing achievements

- Recovery is an important attribute to quantify.
- Lumping and decomposition are helpful assets from model checking,
- but required to be adapted in that matter.
- Methods have been successfully demonstrated on numerous examples:
  - traffic lights,
  - broadcast algorithm [AINA2012,JCSS2013],
  - power grids [WAINA2014] and
  - wireless sensor network [AINA2014].

Summarizing achievements

- Recovery is an important attribute to quantify.
- Lumping and decomposition are helpful assets from model checking,
- but required to be adapted in that matter.
- Methods have been successfully demonstrated on numerous examples:
  - traffic lights,
  - broadcast algorithm [AINA2012,JCSS2013],
  - power grids [WAINA2014] and
  - wireless sensor network [AINA2014].

Summarizing achievements

- Recovery is an important attribute to quantify.
- Lumping and decomposition are helpful assets from model checking,
- but required to be adapted in that matter.
- Methods have been successfully demonstrated on numerous examples:
  - traffic lights,
  - broadcast algorithm [AINA2012,JCSS2013],
  - power grids [WAINA2014] and
  - wireless sensor network [AINA2014].

Summarizing achievements

- Recovery is an important attribute to quantify.
- Lumping and decomposition are helpful assets from model checking,
- but required to be adapted in that matter.
- Methods have been successfully demonstrated on numerous examples:
  - traffic lights,
  - broadcast algorithm [AINA2012,JCSS2013],
  - power grids [WAINA2014] and
  - wireless sensor network [AINA2014].

Own publications

AnSS2008 Nils Müllner, Abhishek Dhama, and Oliver Theel. *Derivation of Fault Tolerance Measures of Self-Stabilizing Algorithms by Simulation.* In Proceedings of the 41st Annual Symposium on Simulation (AnSS2008), pages 183 – 192, Ottawa, ON, Canada, April 2008. IEEE Computer Society Press.

ATC2009 Nils Müllner, Abhishek Dhama, and Oliver Theel. *Deriving a Good Trade-off Between System Availability and Time Redundancy.* In Proceedings of the Symposia and Workshops on Ubiquitous, Automatic and Trusted Computing, number E3737 in Track "International Symposium on UbiCom Frontiers - Innovative Research, Systems and Technologies (Ufirst-09)", pages 61 – 67, Brisbane, QLD, Australia, July 2009. IEEE Computer Society Press.

WAINA2011 Nils Müllner and Oliver Theel. *The Degree of Masking Fault Tolerance vs. Temporal Redundancy.* In Proceedings of the 25th IEEE of the International Conference on Advanced Information Networking and Applications Workshops (WAINA2011), Track "The Seventh International Symposium on Frontiers of Information Systems and Network Applications (FINA2011)", pages 21 – 28, Biopolis, Singapore, 2011. IEEE Computer Society Press.

AINA2012 Nils Müllner, Oliver Theel, and Martin Fränzle. *Combining Decomposition and Reduction for State Space Analysis of a Self-Stabilizing System.* In Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA2012), pages 936 – 943, Fukuoka-shi, Fukuoka, Japan, March 2012. IEEE Computer Society Press. **Best Paper Award.**

JCSS2013 Nils Müllner, Oliver Theel, and Martin Fränzle. *Combining Decomposition and Reduction for the State Space Analysis of Self-Stabilizing Systems.* In Journal of Computer and System Sciences (JCSS), volume 79, pages 1113 – 1125. Elsevier Science Publishers B. V., November 2013. The paper is an extended version of AINA2012.

IREP2013 Maryam Kamgarpour, Christian Ellen, Sadegh Esmaeil Zadeh Soudjani, Sebastian Gerwinn, Johanna L. Mathieux, Nils Müllner, Alessandro Abate, Duncan S. Callaway, Martin Fränzle, and John Lygeros. *Modeling Options for Demand Side Participation of Thermostatically Controlled Loads.* In Proceedings of the IREP Symposium-Bulk Power System Dynamics and Control -IX (IREP), August 25-30, 2013, Rethymnon, Greece, 2013.

AINA2014 Nils Müllner, Oliver Theel, and Martin Fränzle. *Combining Decomposition and Lumping to Evaluate Semi-hierarchical Systems.* In Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications (AINA2014), Victoria, BC, Canada, 2014. accepted for publication.

WAINA2014 Nils Müllner, Oliver Theel, and Martin Fränzle. *Composing Thermostatically Controlled Loads to Determine the Reliability against Blackouts.* In Proceedings of the 28th IEEE International Conference on Advanced Information Networking and Applications Workshops (WAINA 2014), Victoria, BC, Canada, 2014. accepted for publication.

Questions?

SSS2006 Abhishek Dhama, Oliver Theel, and Timo Warns. *Reliability and Availability Analysis of Self-Stabilizing Systems*. In Proceedings of the Eighth International Conference on Stabilization, Safety, and Security of Distributed Systems (SSS2006), pages 244 – 261, 2006.

Boudali et al. , 2010 Hichem Boudali, Pepijn Crouzen, and Mariëlle Stoelinga. *A Rigorous, Compositional, and Extensible Framework for Dynamic Fault Tree Analysis.* IEEE Trans. Dependable Sec. Comput., 7(2):128 – 143, 2010.