On the Relationship between Self-Stabilization and Fault Tolerance

Problems and Prospects

Felix Gärtner

TU DARMSTADT, GERMANY

Questions addressed in this talk:

A How do self-stabilization and fault tolerance relate?

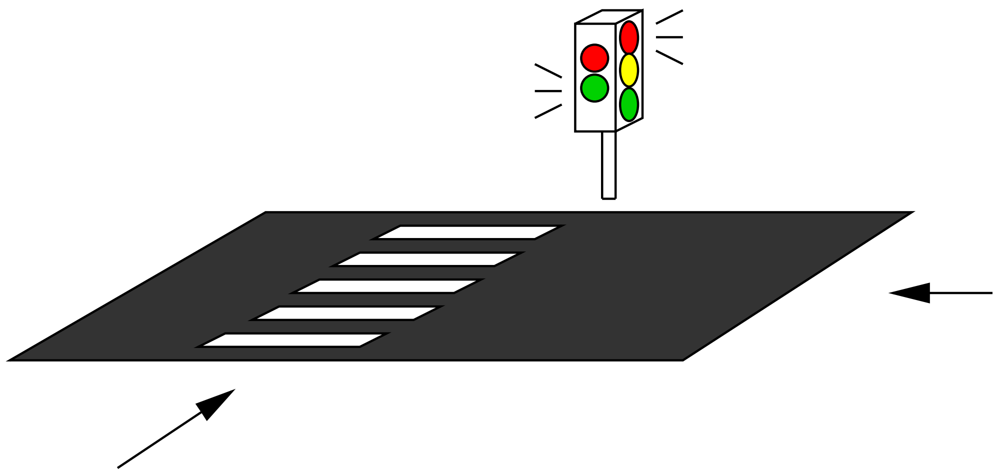
B What are the limitations of self-stabilization (with respect to fault tolerance)?

C What exciting fault tolerance abilities does self-stabilization offer?

Correctness regarding a specification

- Specification is *safety* and *liveness*
  [Lamport 1977; Alpern and Schneider 1985]
- *safety*: "something bad will never happen"
  (system never leaves a set of legal states)
- *liveness*: "something good will eventually happen"
  (good things are kept possible)

Example: traffic lights



Faults and how they are modeled

- Faulty behavior is just another kind of (programmable) behavior
  [Cristian 1985; Arora and Gouda 1993]

- Specifying faults is like adding code

- Example "memory perturbation": global action

  $$\mathbf{true} \quad \longrightarrow \quad \mathit{state} := \langle \text{random} \rangle$$

- Example "crash": boolean flag *up*

  $$\begin{array}{lcl} \mathit{up} & \longrightarrow & \mathit{up} := \mathbf{false} \\ \mathit{up} \wedge \langle \text{other guards} \rangle & \longrightarrow & \langle \text{other commands} \rangle \end{array}$$

- "Fault = program action"

A taxonomy of faults

Name	Definition	Example
transient	assignment to regular variables	memory perturbation
Δ-*permanent*	assignment to error variables with possibility of further change	repairable crash
permanent	irrevertive assignment to error variables	perpetual crash

- Normal variables $\times$ error variables = extended state space
- Transformation of a program A into (virtual) program A'

Fault tolerance properties of A

- Program A is safe and live.
- $A' = A +$ "faults"

What can be said about the safety and liveness properties of A'?

Four forms of fault tolerance [Arora and Kulkarni 1998]

A'	still live	not live
still safe	A is *masking* fault tolerant	A is *fail safe* fault tolerant
not safe	A is *non-masking* fault tolerant	A is not fault tolerant at all

What type is self-stabilization?

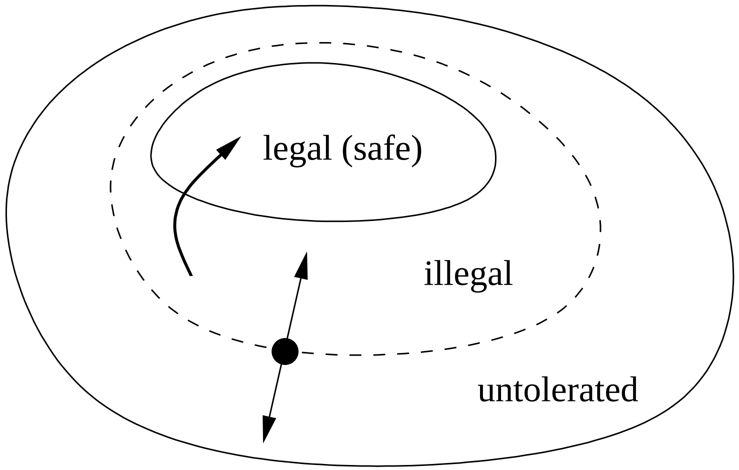
- Arbitrary state $\rightarrow$ legal state
- safe and live when in legal states

  $\Rightarrow$ *liveness* is ensured
- Arbitrary state is "unsafe" state

  $\Rightarrow$ *safety* is violated

Self-stabilization is special form of non-masking fault tolerance.

How does self-stabilization relate to other forms of non-masking fault tolerance?



“pushing non-masking fault tolerance to the rim”

B Problems of self-stabilization

1. What use is a violation of safety in practical systems?
2. Tolerating only transient faults is not enough.
3. How to design, implement and verify self-stabilization property?
4. Real-time issues (how timely is recovery)?

What use is a violation of safety in practical systems?

- No use, if safety is mission critical (flight control, etc.)
- Useful where performance, not correctness is influenced.

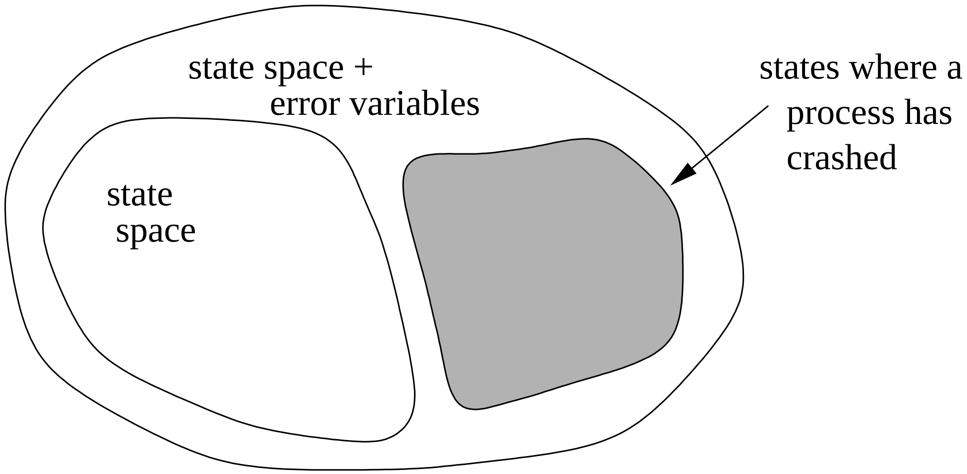
Example: SunSCALR Framework for Web Servers
[Singhai et al. 1998]

- Consistency predicate and convergence actions
- Users may experience disrupted service
- "Not fool proof, but it works"

What types of faults are tolerated?

- By nature: transient faults on *all* nodes (*this is a lot!*)
- Permanent faults?

Transferring the definition to the extended state space



Example: Failure detection protocol between two nodes A and B

process A
  var *alive* : **boolean**
  err *up* **init true**
  begin

$$\begin{array}{llll} & up & \longrightarrow & alive := B.up \\ \| & up & \longrightarrow & up := \textbf{false} \end{array}$$

  end

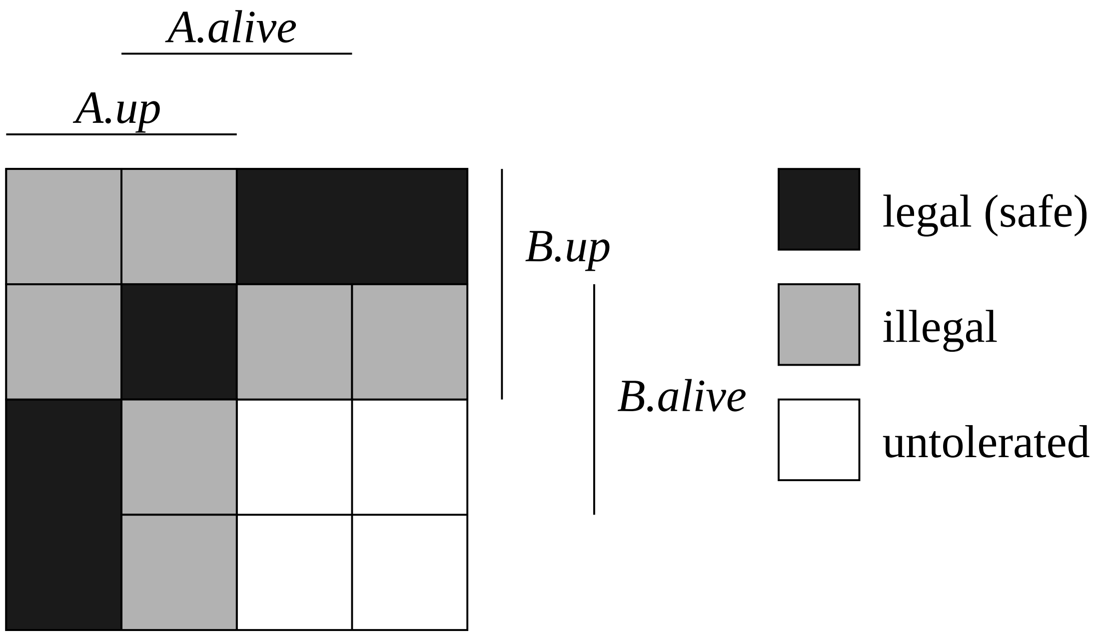
process B
  var *alive* : **boolean**
  err *up* **init true**
  begin

$$\begin{array}{llll} & up & \longrightarrow & alive := A.up \\ \| & up & \longrightarrow & up := \textbf{false} \end{array}$$

  end

State diagram

- *Legal states:* both are up and know that the other is up too, or one is up and knows that the other is down
- *Illegal states: A* falsely suspects *B* or vice versa
- *Untolerated states:* both are down



Characterization of the permanent faults tolerable with self-stabilization

- Permanent faults can be tolerated as long as
  - they are eventually detectable [Anagnostou and Hadzilacos 1993] and
  - enough redundancy is available. [Gopal and Perry 1993]
- Δ-permanent faults must additionally be "stable" long enough.

C Prospects of self-stabilization

- Large fault class can be tolerated by a single mechanism
- Can focus on detection and subsequent repair of permanent faults (no necessity to initialize the system again)

Conclusions

- Self-stabilization is an "extreme" form of non-masking fault tolerance
- Self-stabilizing systems are in general "unsafe"
- Can be used wherever safety merely influences performance
- Use to combat transient faults on all nodes *and* a large class of permanent faults on a subset of nodes

References

ALPERN, B. AND SCHNEIDER, F. B. 1985. Defining liveness. *Information Processing Letters 21*, 181–185.

ANAGNOSTOU, E. AND HADZILACOS, V. 1993. Tolerating transient and permanent failures. In *WDAG93 Distributed Algorithms 7th International Workshop Proceedings, Springer-Verlag LNCS:725* (1993), pp. 174–188.

ARORA, A. AND GOUDA, M. 1993. Closure and convergence: a foundation of fault-tolerant computing. *IEEE Transactions on Software Engineering 19*, 11, 1015–1027.

ARORA, A. AND KULKARNI, S. S. 1998. Component based design of multitolerance. *IEEE Transactions on Software Engineering 24*, 1, 63–78.

CRISTIAN, F. 1985. A rigorous approach to fault-tolerant programming. *IEEE Transactions on Software Engineering 11*, 1 (Jan.), 23–31.

GOPAL, A. S. AND PERRY, K. J. 1993. Unifying self-stabilization and fault-tolerance. In *PODC93 Proceedings of the Twelfth Annual ACM Symposium on Principles of Distributed Computing* (1993), pp. 195–206. ACM Press.

LAMPORT, L. 1977. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering 3*, 2 (March), 125–143.

SINGHAI, A., LIM, S.-B., AND RADIA, S. R. 1998. The SunSCALR framework for internet servers. In *Proceedings of the 28th Symposium on Fault Tolerant Computing Systems (FTCS-28)* (June 1998), pp. 108–117.