



Self Stabilizing Algorithms

Andrew March, Sinclair Fuh, and Matthew Manoly



Overview

- Concept in distributed computing: a model in which multiple computers work together to efficiently run a software system
- A self-stabilizing system will **always** reach a legitimate state within a certain time frame
- A non self-stabilizing system can't guarantee a permanent legitimate state
- Traditional fault-tolerant systems don't always work, such as when starting in an illegitimate state

History

- First presented by Edsger W. Dijkstra in 1974
- *Self-Stabilizing Systems in Spite of Distributed Control*
- Previous systems can stabilize only if a global clock and a limit to how long each transition within the system takes existed



Source: dijkstrascry.com

Distributed Computing

- Components of a software system are split among several communicating computers running together as a single system
- Conceptually similar to parallel processing
- Networks can be local (LAN) or over the internet (WAN)
- Computers can communicate with all or some members of the network
- Commonly seen in client/server communications models
- Typically, more computers = more efficiency

Self Stabilization

- By definition, a self-stabilizing algorithm must abide by 2 rules:
 - The system will reach a valid state, no matter the state it starts in
 - The system will remain in the valid state provided a fault does not occur
- A *weak stabilizing* system can be used instead if sufficient, which only guarantees a chance of stabilizing from each state
- Time complexity is measured differently: measured in *rounds* and *cycles*
 - A *round* is the shortest time in which each processor executes at least one command
 - A *cycle* is the shortest time in which each processor executes its full set of commands

Dijkstra's token ring model

- Earliest concept of a self-stabilizing algorithm, proposed by Dijkstra himself
- Unidirectional ring of networks
- System is only valid when a single token exists in the network
- Must self stabilize from a state where there are too many, or no tokens

EXTENDS *Integers*
 CONSTANT N, M
 ASSUME $N \in \text{Nat} \setminus \{0\}$
 $\text{Procs} \triangleq 1 \dots N$

Dijkstra's stabilizing token ring with processes

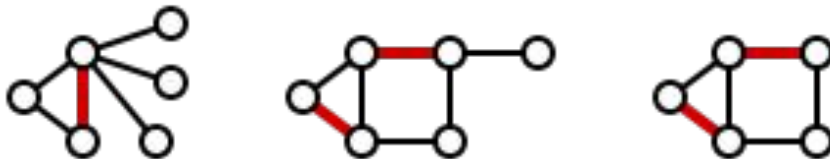
```
--algorithm TokenRing{
  variable  $\text{token} = [k \in 0 \dots N \mapsto (k \% M)]$ ;

  fair process (  $j \in \text{Procs}$  )
  {  $J0$ : while ( TRUE )
    { await  $\text{token}[\text{self}] \neq \text{token}[(\text{self} - 1)]$ ;
       $\text{token}[\text{self}] := \text{token}[(\text{self} - 1)]$ ;
    }
  }

  fair process (  $i \in \{0\}$  )
  {  $I0$ : while ( TRUE )
    { await  $(\text{token}[\text{self}] = \text{token}[N])$ ;
       $\text{token}[\text{self}] := (\text{token}[N] + 1) \% M$ ;
    }
  }
}
```

Maximal Matching

- Set of edges on a graph
- No 2 edges share vertices
- No edges can be added to the set
- Will find maximal matching set regardless of starting state



[https://en.wikipedia.org/wiki/Matching_\(graph_theory\)](https://en.wikipedia.org/wiki/Matching_(graph_theory))

| | |
|---|--|
| $\begin{array}{l} \text{marriage:} \\ \\ \text{seduction:} \\ \\ \text{abandonment:} \end{array}$ | $\begin{array}{l} \text{matched}.i = PRmarried(i) \wedge points_to.i = \mathbf{null} \wedge points_to.r = i \longrightarrow \\ points_to.i := r \\ \\ \text{matched}.i = PRmarried(i) \wedge points_to.i = \mathbf{null} \wedge \forall k \in N : points_to.k \neq i \\ \wedge (points_to.r = \mathbf{null} \wedge r > i \wedge \neg matched.r) \longrightarrow \\ points_to.i := Max\{j \in N : (points_to.j = \mathbf{null} \wedge j > i \wedge \neg matched.j)\} \\ \\ \text{matched}.i = PRmarried(i) \wedge points_to.i = j \wedge points_to.j \neq i \\ \wedge (matched.j \vee j \leq i) \longrightarrow \\ points_to.i := \mathbf{null} \end{array}$ |
|---|--|

<http://hal.upmc.fr/hal-00569219/document>

$points_to.i$: the node that i points to. Null if pointing nowhere
 $PRmarried(i)$: True if $points_to.i = j$ && $points_to.j = i$ (matched)
 $matched.j$: True if neighbor j is matched

Now do it yourself

if node i isn't pointing anywhere and a neighbor node j is pointing towards it:
point towards j

if node i isn't pointing anywhere and no neighbors are pointing towards it:
point towards the highest value neighbor j , that is also not pointing anywhere and $j > i$

if node i is pointing to a neighbor j , but j isn't pointing back and j is either $\leq i$, or matched:
stop pointing towards j

Questions?

Works Cited

<http://hal.upmc.fr/hal-00569219/document>

<https://www.cs.utexas.edu/users/EWD/ewd04xx/EWD426.PDF>

<https://whatis.techtarget.com/definition/distributed-computing>