Self-Stabilization

Reading: Relevant sections from Ghosh’s textbook
Motivation

• As the number of computing elements increase in distributed systems failures become more common
• We desire that fault-tolerance should be automatic, without external intervention
• Two kinds of fault tolerance
  – masking: application layer does not see faults, e.g., redundancy and replication
  – non-masking: system deviates, deviation is detected and then corrected: e.g., roll back and recovery
• Self-stabilization is a general technique for non-masking distributed systems
• We deal only with transient failures which corrupt data, but not crash-stop failures
Self-stabilization

- Technique for *spontaneous healing*
- Guarantees *eventual safety* following failures

*Feasibility demonstrated by Dijkstra (CACM `74)*

E. Dijkstra
Self-stabilizing systems

- Recover from any initial configuration to a legitimate configuration in a bounded number of steps, as long as the processes are not further corrupted.

- Assumption:
  Failures affect the state (and data) but not the program code.
Self-stabilizing systems

• The ability to spontaneously recover from any initial state implies that no initialization is ever required.

• Such systems can be deployed ad hoc, and are guaranteed to function properly within bounded number of steps

• Guarantees-fault tolerance when the mean time between failures (MTBF) $\gg$ mean time to recovery (MTTR)
Self-stabilizing systems

• Self-stabilizing systems exhibit non-masking fault-tolerance

• They satisfy the following two criteria
  – Convergence
  – Closure
Example 1: Stabilizing mutual exclusion in unidirectional ring

Consider a unidirectional ring of processes. Counter-clockwise ring. One special process (yellow above) is process with id=0. Legal configuration = exactly one token in the ring (Safety). Desired “normal” behavior: single token circulates in the ring.
Dijkstra’s stabilizing mutual exclusion

N processes: 0, 1, …, N-1
state of process j is $x[j] \in \{0, 1, 2, K-1\}$, where $K > N$

- $p_0$ if $x[0] = x[N-1]$ then $x[0] := x[0] + 1$
- $p_j$ $j > 0$ if $x[j] \neq x[j-1]$ then $x[j] := x[j-1]$

Wrap-around after $K-1$

TOKEN is @ a process $p = \text{“if” condition is true @ process } p$

Legal configuration: only one process has token
Can start the system from an arbitrary initial configuration
Example execution

\[ p_0 \text{ if } x[0] = x[N-1] \text{ then } x[0] := x[0] + 1 \]

\[ p_j \text{ j > 0 if } x[j] \neq x[j -1] \text{ then } x[j] := x[j-1] \]
Stabilizing execution

\[ p_0 \quad \text{if } x[0] = x[N-1] \text{ then } x[0] := x[0] + 1 \]

\[ p_j \quad j > 0 \quad \text{if } x[j] \neq x[j-1] \text{ then } x[j] := x[j-1] \]
What Happens

- Legal configuration = a configuration with a single token
- Perturbations or failures take the system to configurations with multiple tokens
  - e.g. mutual exclusion property may be violated
- Within finite number of steps, if no further failures occur, then the system returns to a legal configuration
Why does it work?

1. At any configuration, at least one process can make a move (has token)
2. Set of legal configurations is closed under all moves
3. Total number of possible moves from (successive configurations) never increases
4. Any illegal configuration $C$ converges to a legal configuration in a finite number of moves
Why does it work?

1. At any configuration, at least one process can make a move (has token), i.e., if condition is false at all processes
   - Proof by contradiction: suppose no one can make a move
   - Then $p_1, \ldots, p_{N-1}$ cannot make a move
   - Then $x[N-1] = x[N-2] = \ldots x[0]$
   - But this means that $p_0$ can make a move $\Rightarrow$ contradiction

   $p_0 \quad \text{if } x[0] = x[N-1] \text{ then } x[0] := x[0] + 1$

   $p_j \quad j > 0 \quad \text{if } x[j] \neq x[j-1] \text{ then } x[j] := x[j-1]$
1. At any configuration, at least one process can make a move (has token)

2. Set of legal configurations is closed under all moves

   - If only \( p_0 \) can make a move, then for all \( i,j \): \( x[i] = x[j] \). After \( p_0 \)’s move, only \( p_1 \) can make a move
   - If only \( p_i \) (\( i \neq 0 \)) can make a move
     - for all \( j < i \), \( x[j] = x[i-1] \)
     - for all \( k \geq i \), \( x[k] = x[i] \), and
     - \( x[i-1] \neq x[i] \)
     - \( x[0] \neq x[N-1] \)

   in this case, after \( p_i \)’s move only \( p_{i+1} \) can move

\[
\begin{align*}
p_0 & \quad \text{if } x[0] = x[N-1] \text{ then } x[0] := x[0] + 1 \\
p_j & \quad j > 0 \text{ if } x[j] \neq x[j-1] \text{ then } x[j] := x[j-1]
\end{align*}
\]
Why does it work?

1. At any configuration, at least one process can make a move (has token)
2. Set of legal configurations is closed under all moves
3. Total number of possible moves from (successive configurations) never increases
   - any move by $p_i$ either enables a move for $p_{i+1}$ or none at all

\[ p_0 \quad \text{if } x[0] = x[N-1] \text{ then } x[0] := x[0] + 1 \]

\[ p_j \quad j > 0 \quad \text{if } x[j] \neq x[j-1] \text{ then } x[j] := x[j-1] \]
Why does it work?

1. At any configuration, at least one process can make a move (has token)
2. Set of legal configurations is closed under all moves
3. Total number of possible moves from (successive configurations) never increases
4. Any illegal configuration C converges to a legal configuration in a finite number of moves
   - There must be a value, say v, that does not appear in C (since K > N)
   - Except for \( p_0 \), none of the processes create new values (since they only copy values)
   - Thus \( p_0 \) takes infinitely many steps, and since it only self-increments, it eventually sets \( x[0] = v \) (within K steps)
   - Soon after, all other processes copy value v and a legal configuration is reached in N-1 steps

\[
\begin{align*}
p_0 & \quad \text{if } x[0] = x[N-1] \text{ then } x[0] := x[0] + 1 \\
p_j & \quad j > 0 \quad \text{if } x[j] \neq x[j-1] \text{ then } x[j] := x[j-1]
\end{align*}
\]
Putting it All Together

- Legal configuration = a configuration with a single token
- Perturbations or failures take the system to configurations with multiple tokens
  - e.g. mutual exclusion property may be violated
- Within finite number of steps, if no further failures occur, then the system returns to a legal configuration
Summary

• Many more self-stabilizing algorithms
  – Self-stabilizing distributed spanning tree
  – Self-stabilizing distributed graph coloring
  – Not covered in the course – look them up on the web!
Reminders

• MP2, HW4 due soon after break
  – I hope you’ve already started. If not, start now! Don’t start after break; it’s too late then.

• Only 3 lectures left!

• Have a good Thanksgiving break!

• (No lectures or office hours next week)