Self-Stabilization

Definition of self-stabilization

A system *S* is *self-stabilizing* with respect to predicate *P*

if it satisfies the following two properties:

- Closure: *P* is closed under the execution of *S*. That is, once *P* is established in *S*, it cannot be falsified.

- Convergence: Starting from an arbitrary global state, *S* is guaranteed to reach a global state satisfying *P* within a finite number of state transitions.

Definition of stabilization

[Arora and Gouda] We define *stabilization* for system *S* with respect to two predicates *P* and *Q*, over its set of global states.

Predicate *Q* denotes a restricted start condition. *S* satisfies $Q \rightarrow P$ (read as *Q* stabilizes to *P*) if it satisfies the following two properties:

- Closure: *P* is closed under the execution of *S*. That is, once *P* is established in *S*, it cannot be falsified.
- Convergence: If *S* starts from any global state that satisfies *Q*, then *S* is guaranteed to reach a global state satisfying *P* within a finite number of state transitions.

Randomized self-stabilization

A system is said to be *randomized self-stabilizing system,* if and only if it is self-stabilizing and the expected number of rounds needed to reach a correct state (legal state) is bounded by some constant *k.*

Probabilistic self-stabilization

A system S is said to be *probabilistically self stabilizing* with respect to a predicate P if it satisfies the following two properties:

- Closure: P is closed under the execution of S. That is, once P is established in S, it cannot be falsified.

- Convergence: There exists a function f from natural numbers to [0, 1] satisfying $\lim_{k \to \infty} f(k) = 0$, such that the probability of reaching a state satisfying P, starting from an arbitrary global state within k state transitions, is $1 - f(k)$.

Issues in design of self-stabilization algos

- Number of states in each of the individual units in a distributed system.
- Uniform and non-uniform algorithms.
- Central and distributed demons.
- Reducing the number of states in a token ring.
- Shared memory models.
- Mutual exclusion.
- Costs of self-stabilization.

Dijkstra’s self-stabilizing token ring

A legitimate state must satisfy the following constraints:

- There must be at least one privilege in the system (liveness or no deadlock).
- Every move from a legal state must again put the system into a legal state (closure).
- During an infinite execution, each machine should enjoy a privilege an infinite number of times (no starvation).
- Given any two legal states, there is a series of moves that change one legal state to the other (reachability).

Dijkstra’s self-stabilizing token ring

Dijkstra considered a legitimate (or legal) state as one in which exactly one machine enjoys the privilege.

- This corresponds to a form of mutual exclusion, because the privileged process is the only process is the only process that is allowed in its critical section.
- Once the process leaves the critical section, it passes the privilege to one of its neighbors.

First solution

For any machine, we use the symbols S, L, and R to denote its own state of the left neighbor and the state of the right neighbor on the ring, respectively.

The exceptional machine:

If $L = S$ then

$S := (S + 1) \bmod K$

End If;

The other machine:

If $L \neq S$ then

S := L

End if;

Second Solution

The second solution uses only three-state machines. The state of each machine is in {0, 1, 2}.

The bottom machine, machine 0:

If $(S + 1) \bmod 3 = R$ then

$S := (S - 1) \bmod 3$

The top machine, machine $n - 1$:

If $L = R$ and $(L + 1) \bmod 3 \neq S$ then

$S := (L + 1) \bmod 3$

Second Solution Continued

The other machines:

If $(S + 1)$ mod 3 = L then

$S := L$

If $(S + 1)$ mod 3 = R then

$S := R$

The condition $(s + 1) \bmod 3$ covers the three possible states; for $s = 0, 1, 2$, we have $(s + 1) \bmod 3 = 1, 2, 0$. These result in the following three possibilities:

1. If $s = 0$ and $r = 1$, then the state of s is changes to 2.

2. If $s = 1$ and $r = 2$, then the state of s is changes to 0.

3. If $s = 2$ and $r = 0$, then the state of s is changes to 1.

The top machine, machine $n – 1$, behaves as follows:

If $L = R$ and $(L = 1)$ mod 3 ≠ S then

$S := (L + 1)$ mod 3

The state of the top machine depends upon both its left and right neighbors (the bottom machine). The condition specifies that the left neighbor (L) and the right neighbor (R) should be in the same state and $(L + 1)$ mod 3 should not be equal to S. (Note that $(L + 1)$ and 3 is 1, 2, 0 when L is 0, 1, 2, respectively). Thus the state of the top machine is as follows:

1. 1, when its left neighbor is 0.
2. 2, when its left neighbor is 1.
3. 0, when its left neighbor is 2.

All other machines behave as follows:

If $(S + 1)$ mod 3 = L then

$S := L$

If $(S + 1)$ mod 3 = R then

$S := R$

While finding out the state of the other machines (machines 1 and 2 in the example below), we first compare the state of a machine with its left neighbor:

1. If $s = 0$ and $L = 1$, then $s = 0$.
2. If $s = 1$ and $L = 2$, then $s = 2$.
3. If $s = 2$ and $L = 0$, then $s = 1$.

Special networks

Ghosh found that there are special networks, where the number of states required by each processor is two.

Study Ghosh's solution