Self-Stabilizing spanning tree

1 Algorithm

Consider a graph where the nodes $\{p_0, \ldots, p_{n-1}\}$ represents a set of n processes. Two nodes are connected if the corresponding processes can communicate. The node p_0 is a distinguished process, and is referred to as the root. Process p_i communicates with its neighbour p_j by writing to a shared register r_{ij} and reading from r_{ji}. We say that the process p_i owns the registers in which p_i writes. That is, p_i owns r_{ij} for all neighbours p_j of p_i. Each register r_{ij} comprises two fields. The field $r_{ij}.dis$ holds an integer value representing the distance from the root p_0 to p_i. This integer value is bounded by some large constant K, and any assignment of a larger value to the distance field results in the assignment of K. The field $r_{ij}.parent$ holds a binary value: if p_i considers that p_j is a parent, then $r_{ij} = 1$; otherwise $r_{ij} = 0$.

Finally, each process p_i holds the following local variables: for each neighbour m, the variable lr_{mi} is of the same type as r_{mi}, and is used to store values read from that register; an integer-valued variable $dist$; a boolean variable F.

It is assumed that each process p_i knows the list N_i of its neighbours, and that this list is ordered once and for all. Alg. 1 shows the pseudo-code of the spanning tree construction algorithm. Note that, the lines "for each $j \in N_i$" mean that we take the successive neighbours of p_i in the predefined order.

Algorithm 1: Self-stabilizing spanning tree construction

```
case Node p_0
    while true do
        foreach j ∈ N_0 do
            r_ij.(parent, dis) ← (0, 0)
        end
    end
case Node p_i ≠ p_0
    while true do
        foreach j ∈ N_i do
            lr_mi ← read(r_mi)
        end
        F ← false
        dist ← 1 + min{lr_mi.dis, m ∈ N_i}
        foreach j ∈ N_i do
            if F = false and lr_ji.dis = dist − 1 then
                r_ij.(parent, dis) ← (1, dist)
                F ← true
            else
                r_ij.(parent, dis) ← (0, dist)
        end
    end
```

Question 1. Consider a ring of $n = 4$ processes. Assume that, initially, all processes are in the same state, and the registers contain the same values. Show that, if p_0 executes the same code as the non-root processes, then it is possible to define an ordering of neighbours for each process, and an execution such that: (i) every process takes steps infinitely many times, (ii) the system does not converge. (*Hint: use the fact that opposite nodes "sees the same thing"*)

2 Solution

2.1 Answer 1

An easy answer (that I have not planned) stems from the fact that we consider a particular algorithm. Indeed, if the root executes the same code as the other processes, then the distance fields in the registers will keep increasing until it reaches the maximum value (staying there hereafter).

2.2 Answer 2

Another answer allows to show that there is no deterministic self-stabilizing algorithm which computes a spanning tree if every process executes the same code. Indeed, start from a configuration C_0 in which all processes are in the same state, and all registers contains the same value. Let a, b, c, d be the processes in the ring (say counterclockwise). Consider the schedule $S = (acbd)^\omega$, that is, each time a process takes a step, the diametrically opposite process then takes a step.

Focus for example on the beginning of the execution. In the configuration C_0, every process sees the same thing (the registers around him contains the same values) as the diametrically opposite process. Thus, if activating the process a puts it in some state, then activating the process c right after puts c in exactly the same state. By continuing this strategy, every process takes infinitely many steps, and infinitely often we reach a configuration in which every process sees exactly the same thing as the opposite process (a symmetric configuration). This prevents the computation of a spanning tree since otherwise, we would get infinitely often a spanning tree with two roots. This argument can be generalized to show the impossibility of spanning tree with uniform algorithm in case of ring of processes of size not a prime number.

This kind of symmetry argument is ubiquitious in distributed computing. It says that, if the environment (the scheduler here) can maintain a form of symmetry in the system infinitely often, then one cannot solve any problem requiring a form of symmetry breaking (e.g. leader election, spanning tree, mutual exclusion, enumeration, etc.). Also, it is not limited to impossibility results in self-stabilization.

It is also interesting to find how we can circumvent this argument. The idea is that we must assume something on the environment which allows the possibility of symmetry breaking. For example, restricting to the case where the number of processes in the ring is a prime number, or considering stochastic scheduler (selecting the next process to activate in a random manner), and so on.