Example Scenarios for AKY 90 – Memory-Efficient Self Stabilizing Protocols for General Networks

Tal Kellner, Yinon Avraham

Following are several scenarios which are examples for the behaviour of the AKY90 algorithm. Each scenario comes with a file that contains the example – a network already in the state, ready for the adversary's action.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| # | Name | File Name | Description | Adversary Action | Expected Result | Reason |
| 1 | Remove Root on Grant | AKY90\_GrantRemoveRoot.nas | What happens when the root is removed just a step before it grants the last node in the chain. | 1. Remove the root (7)  or- open the file AKY90\_GrantRemoveRoot\_after.nas 2. Execute the algorithm | The last node in the chain joins the tree of the node that was removed but the tree stabilizes after a few steps with the new root. | The last node accepts the grant to join the tree of the removed node. But, all the rest of the nodes slowly identify the situation and reset themselves. Eventually, also the last node joins the new tree. |
| 2 | [BUG?] Change the "To" variable | AKY90 \_ToNotParent.nas | What happens when the “To” variable in node v is changed to a node w that is not a neighbour of v. | 1. Click on Node (1) and change the value of “To” from 5 to 3 or- open the file AKY90 \_ToNotParent\_after.nas 2. Execute the algorithm and see that it does not fix itself and gets stuck | Expected- The node resets its request variables. Actual- The algorithm gets stuck. | There is no check for the validity of the "To" variable in any of the conditions / actions |
| 3 | Reset Root | AKY90\_ResetRoot.nas | What happens if we change the value of the “Root” variable in the root node of the tree and set it to a different node. | 1. Click on Node (4) and change the “Root” variable's value from 4 to 2 or- open the file AKY90\_ResetRoot\_after.nas 2. Execute the algorithm | Node (4) set itself as root, the algorithm stabilizes. | The root node (4) identifies that its ID is higher than its root and that its root is not one of its neighbours. Due to that it resets itself to be a root. |
| 4 | [BUG?] Ask by not a Child, not an Orphan | AKY90\_AskNoChildNoOrphan.nas  AKY90\_NoChildNoOrphan2.nas | What happens when a node, which already belongs to a tree, asks one of its neighbours, which belong to a tree with a higher root, to join the tree it belongs to. | Execute the algorithm | Expected- The node joins the tree with the higher root ID. Actual- The algorithm gets stuck and does not stabilize. | Actions 4 & 5 are responsible for forwarding "Ask" requests. Action 4 forwards requests from orphan nodes, and action 5 forwards requests from nodes that are its children. The situation here is none of the above. |
| 5 | Split Trees | AKY90\_SplitTrees.nas | What happens when an edge that connects two parts of the tree is removed. | 1. Delete the edge between Node (4) and Node (1)  or- open the file AKY90\_SplitTrees\_after.nas 2. Execute the algorithm. | The algorithm will stabilize - 2 separate trees. | Both nodes (1) & (4) reset their variables – node (1) to be a root, node (4) to not being forwarding requests. From now on, the algorithm, works on the two separate networks. |
| 6 | Add a Node With a High ID | AKY90\_AddLargestIDNode.nas | What happens when after the tree is stabilized a node that has the highest ID is added to the network. | 1. Connect Node (6) to (7)  or- open the file AKY90\_AddLargestIDNode\_after.nas 2. Execute the algorithm. | Node (7) will be the root of the tree. | Node (6) asks node (7) to join its tree because its ID is higher than the root it belongs to (6). After the root of node 6 is changed, all the other nodes reset and joins the tree of node (7). |
| 7 | Add a Node With a Small ID | AKY90\_AddSmallIDNode.nas | What happens when after the tree is stabilized a node with a small ID is added to the network. | 1. Connect Node (3) to (1)  or- open the file AKY90\_AddSmallIDNode\_after.nas 2. Execute the algorithm. | Node (3) will be part of the tree. | Node (3) identifies that its neighbour (1) has a higher root. It asks to join its tree. The algorithm stabilizes. |

# Suspected Bugs in the Algorithm

We have found two scenarios that are suspected to be bugs in the algorithm:

* **Scenario 2**
  + Current:  
    It seems that the algorithm has no check for the validity of the "*To*" request variable. Condition C2 is supposed to verify that the variables that relate to the forwarding requests process are in a legal state. Condition C2', which is part of C2, verifies that node v is forwarding a request from a neighbour w. This validation checks the validity of the "*Request*" & "*From*" variables. There is no validation of the "*To*" variable.
  + Suggested fix:  
    Modify condition C2 as follows:  
    *(C2' AND v.To = w.Id AND w is neighbour of v) OR (v.Request, v.To, v.From, and v.Direction are undefined)*This way condition C2 can also check that the "*To*" variable is valid.
* **Scenario 4**
  + Current:  
    It seems that the algorithm does not handle the situation of forwarding a request from a node that is part of a different tree (has a root and parent different than its own ID). The two actions that are responsible on forwarding requests are actions 4 & 5. Both of these actions do not handle the above situation.  
    Action 4 is responsible for forwarding "*Ask*" requests from orphan nodes, as can be seen in the following part of the guard:   
    *(w is neighbour of v | (w.Direction = Ask) AND (w.To = v.ld) AND (w.Request =w.Id =w.Root =w.From)*Action 5 is responsible for forwarding "*Ask*" requests from a node's children, as can be seen in the following part of the guard:  
    *(w is neighbour of v | (w child of v) AND (w.To = v.ld) AND (w.Direction = Ask))*
  + Suggested fix:  
    Modify the guard of action 4 as follows:  
    Change *(w.Request =w.Id =w.Root =w.From)* 🡪 *(w.Request =w.Id =w.From)*  
    This way action 4 can handle both situations when the node is orphan and when it belongs to a different tree but wants to join this tree.