Piacente Cristian 866020

Homework H6 – Generalized symbolic execution

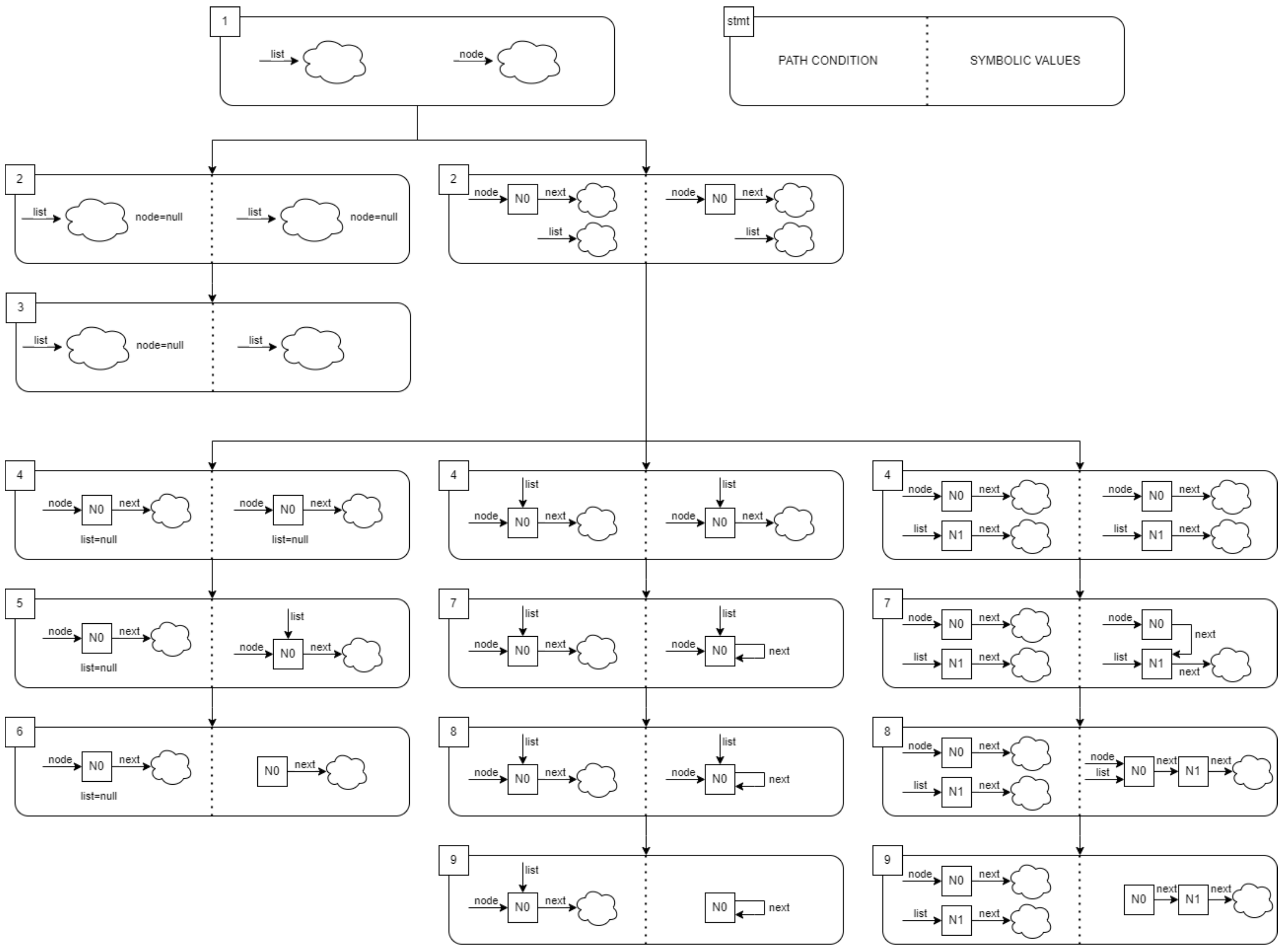
Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Replicate the analysis in Figure 1 and Figure 4 of the seminal paper on Generalized Symbolic execution with respect to the following code that returns a list after adding a node as new head (the code refers to the class Node reported in the figures of the paper):

```
1  Node addNodeToList(Node list, Node node) {
2    if(node == null) {
3      return list;
     }
4    if (list == null) {
5      list = node;
6      return list;
     }
7    node.next = list;
8    list = node;
9    return list;
   }
```

Answer the following question by using the result of the analysis to justify your answer: If we assume that there is no cycle between the references in the "list" before executing program addNodeToList, is it still guaranteed to have no cycle after completing the execution?

Here is the figure for analyzing the method addNodeToList:



Piacente Cristian 866020

Homework H6 – Generalized symbolic execution

Software Quality, Academic Year 2023-2024, University of Milan – Bicocca

Assuming there is no cycle between the references in "list" before executing program addNodeToList, we can analyze the figure to find out if it is guaranteed to have no cycle after the execution.

With the precondition of "list" being acyclic, and no condition on “node”, we have these possible scenarios:

- the parameter **"node" is null** (top left branch): in this case the "list" is returned without any modifications, **keeping its acyclicity**
- the parameter **"node" is a reference to an object N0 and "list" is null** (bottom left branch): in this case the returned "list" aliases "node", but since "node" can also be cyclic (there is no assumption on "node", e.g. node.next could reference the node object itself), then **we can have a cycle** on the returned "list"
- the parameters **"node" and "list" reference the same object** N0 (bottom center branch): here we have a self-loop, since node.next becomes "list" which references the node object itself, so **the returned "list" is NOT acyclic**
- the parameters **"node" and "list" reference two different objects** N0 and N1 (bottom right branch): it doesn't matter if "node" is cyclic or not, because node.next becomes a reference to "list" and "list" is acyclic, then "list" becomes a reference to this new object and it is returned; we **always have acyclicity maintained**.

In conclusion, with just the precondition of "list" being acyclic **it is not guaranteed** to have no cycle after completing the execution. Acyclicity is guaranteed to be maintained only in two cases: when "node" is null and when "node" and "list" reference two different objects.