Indian Institute of Technology, Bombay
2022

CS 766: Analysis of Concurrent Programs
Programming Assignment 1

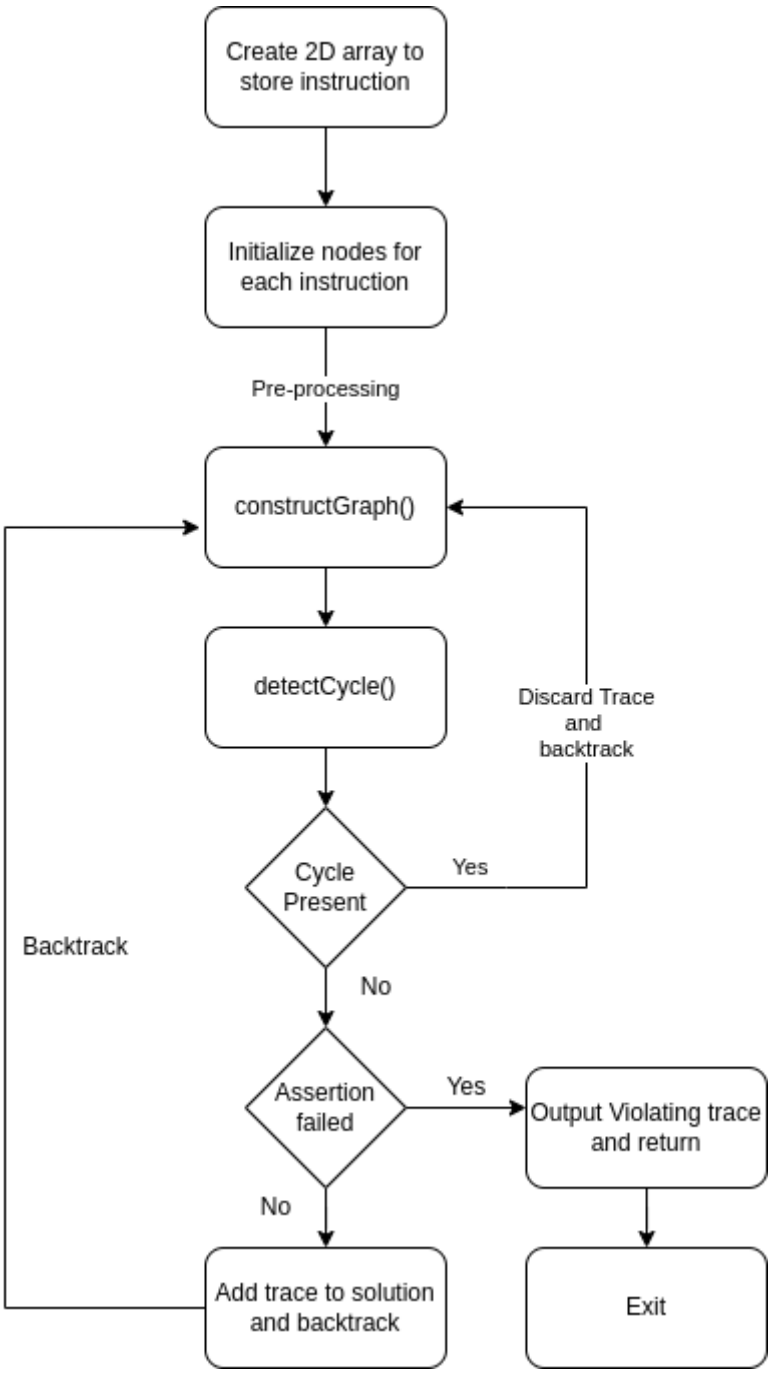
Submitted to:

Prof. Ashutosh Gupta

Submitted by: **Date: -02-2022**
Sachin Singh Lodhi (213050054)
Subodh Latkar (213050047)

Algorithm -

1. Create a two dimensional array programs where programs[i][j] denote the jth instruction present in ith program.
2. Define a class Node which is used to build to graph for each trace. This node class contains pointers for **po**, **ws**, **rf** and **fr** edges.
3. Pre-processing :
    a. For each instruction in the program array, make corresponding Nodes and assign Program Order(**po**) edges to them.
    b. For each node obtained in previous step, identify if it is read or write instruction and store the corresponding global variable along with it.
4. Call constructGraph method to create traces according to all the combinations of write synchronisation(**ws**). For this, we use backtracking. Here, first the current node is marked visited and from that node, we explore all the unvisited nodes. This helps us create all the possible combinations of ws.
5. Assign global clock values to each node which denotes the time at which the instruction corresponding to given node is executed. This will ensure that each process executes the same sequence of writes.
6. Pass the trace obtained in previous step to another method called detect cycle. Here first of all all the rf edges and fr edges are identified. Now, the graph is checked if it contains cycle. If cycle is present then we discard the trace and return.
7. Now, we have only valid traces. We store these traces in Solution list. Now we have to check assertion condition (if present)
8. For checking assertion, we use assertCheck function. For this, we have taken precedence of **and** greater than precedence of **or**. This function uses evaluate function as a utility function. In this function, we evaluate all the expression containing and and then evaluate or expressions.
9. For evaluation of condition, we propagate values of variables by executing the instruction ordered by clock values and then storing them in values array.
10. Using this values array, we check if the assert condition holds. If there is an assertion violation, then we output that violating trace and stop exploring any further.
11. If there is no assertion violation then we output all the valid traces of the program.



Steps to run program via terminal -

1) cd to the directory containing the Main.java file
2) Run command for compiling java source code - **javac Main.java**
3) Run file by typing - **java Main**
4) Give input and see output

Approach 1 -

The first approach which we thought was a simple brute force approach in which, we used the first instruction of each program as root and constructed all the possible traces. After construction of each trace, we started making **ws**, **rf** and **fr** edges between the nodes.
The drawback of this approach was that, if only read instructions are remaining then also they were arranged in all possible ways and their traces were evaluated. Since only reads are remaining so, there is no need for re-arranging them as they would always produce duplicate trace.
Also in approach 1, there was no short-circuiting implemented while checking assertion condition. To rectify these things, we used approach 2.

Approach 2 -

In this approach, we used backtracking to create all the ws combinations. Here the **ws** instructions are ordered on the basis of global clock. While creating the traces for these **ws** combinations, for each read instruction, we added the **rf** and **fr** edge at that particular instant. If we encounter any cycle during the current (partially constructed) graph then we discard it and don't go any further. This helped to eliminate the problem of duplicate traces and reduced the number of traces greatly.

Features & Optimizations -

- **Short-circuiting condition** : While checking assertion condition, we used short circuiting. If the conditions are joined by or operator and any one becomes true then the result becomes true. Similarly for and operator if any one is false then the overall result becomes false.
- **Comparison operators** : We have implemented six comparison operators which can be used in assert condition, which are - **!=, ==, <=, >=, <, >.**
- **Trace map :** The trace map is used to store all the valid traces. This helps us to eliminate any duplicate traces.