Introduction to Distributed Systems
WT 19/20

Assignment 4 – Part II (programming)

Submission Deadline: Monday, 16.12.2019, 08:00

- *Submit the solution in PDF via Ilias (only one solution per group).*
- *Respect the submission guidelines (see Ilias).*

4 Distributed Debugging Algorithm [18 points]

Consider the distributed debugging algorithm presented in the lecture. Given are the following two worker processes P_1 and P_2, which can communicate by exchanging messages over reliable FIFO channels.

Listing 1: P_1

```
1 x1 := 5
2 send(P2, x1)
3 x1 := x1 * 3
4 send(P2, x1)
5 x1 := receive() - x1
```

Listing 2: P_2

```
1 x2 := receive()
2 x2 := x2 + 5
3 x2 := x2 + receive()
4 send(P1, x2)
```

Here, `send(Pn, x)` sends the value of `x` to the process P_n. The channel buffers the message until P_n is ready to receive it. The blocking call `receive()` is used to receive a message from an arbitrary communication partner. It returns the value contained in the received message. The local variables x_i of both processes are initialized to zero.

In addition, each process P_i sends a state-message to a central monitor process *every time the value of its local variable x_i changes*. A state-message contains the value of the local variable x_i and the current vector timestamp of the process. In this assignment, you will step by step implement the distributed debugging algorithm.

Note: You can download stub classes for all parts of this assignment on the course materials website (`a4q4.zip`). The process structure is set up by the provided project `DebugTest`. Use it to test your implementations for parts a), b) and c).

a) [6 points] Based on the provided source code implement the vector clocks needed for the signaling of process progress between the processes and the monitor. In the code, you will find an incomplete implementation for vector clocks `VectorClock`. Complete the class by implementing the `increment`, `get`, `update` and `checkConsistency` methods.

b) [8 points] In the provided source code, you find a stub for the monitor (i.e. `Monitor` class) and the predicates (i.e. `Predicate` class). In the `Monitor` class, complete the `buildLattice`, `findReachableStates` and `checkPredicate` methods. Additionally, complete the `predicate1` and `predicate2` methods to check if the following predicates are *possibly True* and/or *Definitely True*.

- `predicate1`: $(x_1 - x_2 = 15)$
- `predicate2`: $(x_1 + x_2 = 30)$

c) [4 points] Now, change the first process according to Listing 3 and create an additional process according to Listing 4. Complete the `predicate3` method to check the predicate $(x_1 - x_3 = 8)$.

Listing 3: P_1 *update*

```
x1 := 5
send(P2, x1)
x1 := x1 * 3
send(P2, x1)
x1 := receive() - x1
send(P3, x1)
x1 := receive() - x1
```

Listing 4: P_3

```
x3 := 4
x3 := x3 * 2
x3 := receive() - x3
x3 := x3 - 2
x3 := x3 + 11
send(P1, x3)
```