Institute of Computer Science, University of Bern
Distributed Algorithms, Spring 2020

Prof. Christian Cachin
Jovana Mićić

Solution for exercise 8

8.1 Total-order broadcast using consensus (4pt)

Consider Algorithm 6.1 (Consensus-Based Total-Order Broadcast) and observe that the payload messages proposed to consensus are represented as a set.

a) What happens if the messages decided by consensus are not sorted deterministically before *tob-delivering* them?

b) Suppose one leaves out the sorting of decided payload messages. How can one modify the algorithm so that every process nevertheless outputs the same ordered sequence of messages?

Solution.

a) If the deterministic sorting is done prior to proposing the set for consensus, instead of a posteriori upon deciding, the processes would not agree on a set but on a sequence of messages. But if they to-deliver the messages in decided order, the algorithm still ensures the *total order* property. If the messages, on which the algorithm agrees in consensus, are never sorted deterministically within every batch (neither a priori nor a posteriori), then the *total order* property does not hold. Even if the processes decide on the same batch of messages, they might to-deliver the messages within this batch in a different order.

b) We could avoid using the deterministic sort function at the cost of proposing a single message at a time in the consensus abstraction. This means that we would need exactly as many consensus instances as there are messages exchanged between the processes. If messages are generated very slowly by processes, the algorithm ends up using one consensus instance per message anyway. If the messages are generated rapidly then it is beneficial to use several messages per instance: within one instance of consensus, several messages would be gathered, i.e., every message of the consensus algorithm would concern several messages to to-deliver. Agreeing on large batches with many messages at once is important for performance in practice, because it considerably reduces the number of times that the consensus algorithm is invoked.

8.2 Atomic register as a replicated state machine (4pt)

One of the simplest state machines is the register abstraction discussed earlier. Implement an (N, N)-atomic register using a primitive for total-order broadcast.

The replicated state is the current value of the register and the relevant commands are *write*(v) and *read*$() \to v$. Use Module 4.3 [CGR11] and show that the implemented register satisfies the respective properties.

Solution. Algorithm 1 implements an (N, N)-atomic register using a primitive for total-order broadcast. The *termination* property follows from the *validity* property of the total-order broadcast. To show that register instance *nnar* is *atomic*, we demonstrate that the *nnar*-read and *nnar*-write operations are linearizable, i.e., there exists a hypothetical serial execution with all complete operations of the actual execution, such that (1) every read returns the last value written and (2) for any two operations o_1 and o_2, if o_1 precedes o_2 in the actual execution then o_1 also appears before o_2 in the linearization. This is satisfied because of the *total order* property of the total-order broadcast which guarantees that every operation (read and write) is totally ordered among processes. This means that every read will return the last written value (1). Now, consider two operation o and o' where o occurs before o' in the actual execution. It follows from *total order* property that they are linearizable regardless of the type of o and o' (2).

Algorithm 1 (N, N)-atomic register using total-order broadcast.

```
Implements:
    (N, N)-AtomicRegister, instance nnar.
Uses:
    TotalOrderBroadcast, instance tob.

upon event ⟨ nnar-Init ⟩ do
    $val := \bot$;

upon event ⟨ nnar-Write | $v$ ⟩ do
    trigger ⟨ tob-Broadcast | [WRITE, $v$] ⟩;

upon event ⟨ tob-Deliver | $p$, [WRITE, $v$] ⟩do
    $val := v$;
    if $p = self$ then
        trigger ⟨ nnar-WriteReturn ⟩;

upon event ⟨ nnar-Read ⟩ do
    trigger ⟨ tob-Broadcast | [READ] ⟩;

upon event ⟨ tob-Deliver | $p$, [READ] ⟩ such that $p = self$ do
    trigger ⟨ nnar-ReadReturn | $val$ ⟩;
```

8.3 Replicated register with local read (2pt)

The atomic-register implementation of Ex. 8.2 sends every operation through total-order broadcast. Describe a modification of the protocol, where *reads* are executed locally, without being *tob-broadcast* to all processes. One often finds this optimization in practical systems. Does the resulting register still satisfy *atomicity*? Either argue why the property holds or demonstrate an execution that violates atomicity.

Solution. Algorithm 2 shows the modification of Algorithm 1 in which reads are executed locally, without using total-order broadcast. The resulting register satisfies *termination* property because a process simply returns its local value, but it does not satisfy *atomicity*. When the total-order broadcast is not used, operations among processes are not ordered which is the violation of *atomicity* property.

Algorithm 2 Modification of (N, N)-atomic register.

```
upon event ⟨ nnar-Read ⟩ do
    trigger ⟨ nnar-ReadReturn | $val$ ⟩;
```

Figure 1 shows an example of the modified algorithm and how it violates atomicity. The second write of process q arrives late to process r so this process will return x because it reads this value locally. The violation of atomicity occurs when the second write terminates before r read starts.

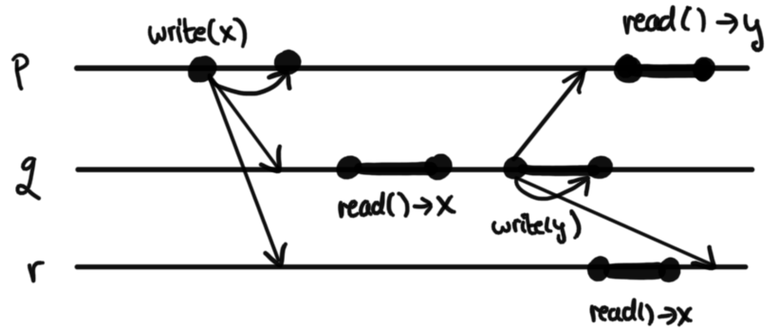

Figure 1. Example of an execution of (N, N)-atomic register when reads are executed locally.

References

[CGR11] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming (Second Edition)*, Springer, 2011.