STiDC'06: Example Final Exam Questions

February 7, 2007

Problem 1

Implement a multi-valued SWMR safe register, using a *minimal* number of base multi-valued SWMR safe registers of which b can be malicious, i.e., can return arbitrary values.

Solution. The idea of the solution is the following. We use $2b + 1$ base safe registers. A write operation writes to *all* base registers the same value given as an argument. A read operation reads from all base registers, and returns the value read from a majority of them. If there is no value that is read from a majority of base registers, the read operation can return any value.

Problem 2

We can define an `UpDownCounter` object as follows. The state of the object stores an integer. The object implements 3 operations: *read* returns the state of the object without changing it, *inc* increases the state of the object by 1 and returns *ok*, and *dec* decreases the state of the object by 1 and returns *ok*.

1. Here is a proposed implementation of an `UpDownCounter` for n processes, using n atomic registers (code for process p_i):

   uses: $A[1, \ldots, n]$ – atomic registers
   initially: $A[1, \ldots, n] = 0$

   upon $read_i()$ **do**
   - $v \leftarrow 0$
   - **for** $k \leftarrow 1$ **to** n **do**
     - $v \leftarrow v + A[k].read_i()$
   - **return** v

   upon $inc_i()$ **do**
   - $v \leftarrow A[i].read_i()$
   - $A[i].write_i(v + 1)$

   upon $dec_i()$ **do**
   - $v \leftarrow A[i].read_i()$
   - $A[i].write_i(v - 1)$

   Is this a linearizable implementation of an `UpDownCounter`? If so, prove it. If not, give an execution that is not linearizable.

2. What is the consensus number of `UpDownCounter`? Show your answer is correct.

Solution for part 1. The algorithm is not a linearizable implementation of an `UpDownCounter`. To prove it, consider the following execution of the algorithm:

Step	Process p_1	Process p_2	Process p_3
1.	invokes $read_1()$		
2.	reads $A[1] = 0$		
3.	reads $A[2] = 0$		
4.		invokes $inc_2()$	
5.		writes $A[2] \leftarrow 1$	
6.		returns *ok*	
7.			invokes $dec_3()$
8.			writes $A[3] \leftarrow -1$
9.			returns *ok*
10.	reads $A[3] = -1$		
11.	returns -1		

The execution is not linearizable because the operations $inc_2()$ and $dec_3()$ are not concurrent, and so the operation $read_1()$ should have returned either 0 or 1. However, $read_1()$ returns -1.

Solution for part 2. The consensus number of `UpDownCounter` is 1. That is because `UpDownCounter` can be easily implemented from an atomic snapshot object, which can be implemented from registers (see the lecture slides). The algorithm would be the following:

uses: S – atomic snapshot (other variables are local)
initially: $c_i = 0$ at every process p_i, and the value of each element of S is 0

upon $read_i()$ **do**
  $A \leftarrow S.scan_i()$
  $v \leftarrow 0$
  for $k \leftarrow 1$ **to** n **do**
    $v \leftarrow v + A[k]$
  return v

upon $inc_i()$ **do**
  $c_i \leftarrow c_i + 1$
  $S.update_i(c_i)$

upon $dec_i()$ **do**
  $c_i \leftarrow c_i - 1$
  $S.update_i(c_i)$

Problem 3

Implement a stack using 2-consensus objects and atomic registers in a system of n processes. (Reminder: 2-consensus is an implementation of consensus for 2 processes.)

Solution. See the paper *Common2 extended to stacks and unbounded concurrency* by Y. Afek, E. Gafni and A. Morrison (PODC'06).