Concurrent Algorithms December 21, 2020

Solutions to Exercise 11

Problem 1. Let P be the problem of implementing C&S using base C&S objects, one of which can be non-responsive, and registers (non-faulty). Let Q be the problem of implementing consensus using registers in a system of $n > 1$ processes, one of which can crash (we know this problem to be impossible). We perform our proof by contradiction: assume there exists an algorithm A that solves P using k C&S objects, in a system of n processes (one of which can crash). If we find an algorithm B that solves problem Q, using A we have reached a contradiction.

From non-faulty C&S to consensus: We implement consensus in a system of $N = \max(k, n)$ processes, one of which can crash. A process p_i that proposes a value, writes the value in a register $R[i]$ and waits until a decided value is written in register D:

initially: $D = \perp$, $R[1, \ldots, N] = \perp$

upon $propose_i(v)$ **do**
- $R[i] \leftarrow v$
- wait until $D \neq \perp$
- **return** D

Each of the n processes then runs the following task in parallel and uses the hypothetical correct C&S object implemented using algorithm A.

parallel task $Cons_i$
- wait until some value $v \neq \perp$ is written in some register $R[j]$
- use algorithm A to call $CAS(\perp, v)$ on the non-faulty C&S object
- $D \leftarrow$ value returned by the CAS

From registers to non-responsive C&S: Each of n processes emulates one base C&S object. The processes share a 2-dimensional array CS of registers. When process i wants to invoke the CAS operation of C&S object x it invokes the following:

upon $CAS_x(oldval,newval)_i$ **do**
- $CS[x][i] \leftarrow (\texttt{invocation}, oldval, newval)$
- wait until $CS[x][i] = (\texttt{response}, retval)$
- **return** $retval$

Since one of the processes can fail, its corresponding C&S object becomes non-responsive. Each process i reads invocations from locations $CS[i][*]$ and applies them:

```
parallel task C_i
    initially: q = ⊥ (local variable)
    while true do
        for j ← 1 to n do
            (type, oldval, newval) ← CS[i][j]
            if type = invocation then
                if q = oldval then q ← newval
                CS[i][j] ← (response, q)
```

Problem 2. We use $2t+1$ base registers, so that always majority is correct. Read/write from/to majority of registers.

uses: $R[1, \ldots, 2t+1]$ – SWMR registers t of which can be non-responsive

upon $write_1(v)$ **do**
- $ts \leftarrow ts + 1$
- invoke $write_1(ts, v)$ on all $R[1, \ldots, 2t+1]$
- wait for $t+1$ responses

upon $read_i$ **do**
- invoke $read_i(v)$ on all $R[1, \ldots, 2t+1]$
- wait for $t+1$ responses
- **return** *the value v with the highest timestamp ts*

The presented algorithm implements a regular SWMR register. However, a regular register can be transformed into an atomic one (see the lecture slides about register transformations).