Distributed Algorithms CPSC-561
Assignment 2

Michael McCulloch

1 LL/SC

For all of the following algorithms, they are each wait-free, as they do not contain a loop.

A) CAS from SCAS

```
public class CAS_FROM_SCAS {
    public boolean CAS(Object prev, Object succ){
        Object last = SCAS.scas(prev, succ);
        return last == prev;
    }}
```

Let the invocation of SCAS.scas() be the linearization point X. *Claim:* If a process P executes X before another process Q executes X, then the invocation of P is ordered before the invocation of Q. *Proof:* By the definition of SCAS, if it returns the same value as it's first argument, it must have succeeded. If this value is different, another process Q must have finished it's invocation before the P's invocation of SCAS and after the assignment of it's result to `last`, a contradiction.

B) SCAS from CAS

```
public class SCAS_FROM_CAS {
    public Object SCAS(Object prev, Object succ){
        boolean success = CAS.cas(prev, succ);
        if (success) {
            return prev;
        } else {
            return CAS.read();
        }}}
```

Let the invocation of CAS.cas() be the linearization point X. *Claim: see above. Proof:* If CAS is successful, then the old value passed in must be the old value. If CAS is unsuccessful, then a different value must have been the old value, which can only be the case if another process executed X first. A contradiction.

C) CAS from LL/SC

```
public class CAS_FROM_LLSC {
    public boolean CAS(Object prev, Object succ){
        Object old = LLSC.ll();
        if (old != prev){
            return false;
        } else {
            return LLSC.sc(succ);
        }}}
```

Let the invocation of `LLSC.ll()` be the linearization point X. *Claim:* If a process P executes X after another process Q, then it is ordered after Q. *Proof:* If any process P executes `LLSC.ll()` at t, then the only next successfull invocation of `LLSC.sc()` will be P's at t', unless another process Q has executed `LLSC.ll()` between t and t', in which case Q's invocation of `LLSC.sc()` will be the successful one.

2 $\mathbb{Z}_k$-Counter

Solution for consensus no. 2:

Algorithm 1 Decision procedure 2 processes

procedure DECIDE$_i$
  $Val[i] \leftarrow V_i$
  $x \leftarrow Z_k.Inc()$ mod 2
  if $x = 0$ **then**
    return $Val[i]$
  else
    return $Val[1 - i]$
  end if
end procedure

Claim: No consensus-3 solution exists using only Z_k counters and registers. *Proof by cases:*

1.

For three processes, A, B and C let X be a critical configureation and π_I be the step that process I takes to increment the counter. Then to a third process C, the branches $\pi_A \pi_B(X)$ and $\pi_B \pi_A(X)$ are indistinguishable, which contradicts X being critical.

3 SRSW

3.1 A

1. w : W.write(5)
2. w : choose c

`IF` c $\geq$ 5 `THEN`

3. r : R.read()
4. w : W.write(c)

`ELSE`

3. w : W.write(c)
4. r : R.read()

There are two ways in which r may read 5, but no ways it may read 6
Expected Value = (1/6)*1 + (1/6)*2 + (1/6)*3 + (1/6)*4 + (2/6)*5
Expected Value = (1/6) * 10 + (2/6) * 5
Expected Value = 20/6 = 3.3

3.2 B

1. r : R.read()$_{5-9}$
2. w : W.write(5)
3. w : choose c

`IF` c $\geq$ 4

`THEN:`

4. r : R.read()$_{10-13}$
5. w : W.write(c)

`ELSE:`

4. w : W.write(c)
5. r : R.read()$_{10-13}$

There are now 3 ways in which r may read 4, and no ways it may read 5 or 6:
Expected Value = (1/6) * (1+2+3) + (3/6) * 4 = 3