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

```java
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 have been cause if another process executed X first. A contradiction.

C) CAS from LL/SC

```java
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 its own at t', unless another process Q has executed `LLSC.ll()` between t and t', in which case Q will be the successfull one.

2 $\mathbb{Z}_k$-Counter

Claim: There is no Consensus-3 solution using $\mathbb{Z}_k$-counters. Proof by cases for 3 processes: A, B and C.

2.1 Case: Different $\mathbb{Z}_k$-counters

w.l.o.g. assume A and B increment different $\mathbb{Z}_k$-counters. Let K be a bivalent configuration, π_A and π_b be the steps that A and B take, respectively, to increment their own counters ...FILL IN THE BLANK... then $\pi_A(K)$ and $\pi_A\pi_B(K)$ are indistinguishable to A

2.2 Case: Different/Same Registers

See proof in lecture notes.

2.3 Case: Same $\mathbb{Z}_k$-counters

A and B increment A $\mathbb{Z}_k$-counters. Let K be a bivalent configuration, π_A and π_B be the steps that A and B take, respectively, to increment the counter. Assume for contradiction that π_R causes the configuration to be R-univalent. If the counter's initial value is c, then $\pi_A(K)$ causes it to be $c + 1 \bmod k$. the same is true for $\pi_B(K)$. Hence, after $\pi_A\pi_B(K)$, the value of the counter is $c + 2 \bmod k$ and the same is true for $\pi_B\pi_A(K)$. Therefor, to a third process C, these two states are identical, and therefor are not univalent, yielding a contradiction.

3 SRSW

3.1 A

Run W_1, W_2: if c $\geq$ 5 then run R_1, W_3 else run W_3, R_1
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