Assignment 4

Theorem 1. *It is impossible to solve two processor binary consensus using only one Test&Set object.*

Proof. We will prove the theorem using a valency argument. Let p_0 and p_1 be the two processors. As shown during the first lecture, there exists an initial multi-valent configuration. Suppose that C is a critical configuration. Then, without loss of generality (WLOG), p_0 is the processor whose step α_0 takes C to a 0-valent configuration, $C_0 = C\alpha_0$, and p_1 is the processor whose step α_1 takes C to a 1-valent configuration, $C_1 = C\alpha_1$. If α_0 and α_1 are both RESET then $C_0\alpha_1$ and $C_1\alpha_0$ are indistinguishable to both processors. Thus executions starting from $C_0\alpha_1$ and $C_1\alpha_0$ should have the same valency, but this is a contradiction.

Next suppose WLOG that α_0 is RESET and α_1 is TEST&SET. Observe that $C_1\alpha_0 \overset{p_0}{\sim} C_0$ since p_0 is in the same state and the test&set object has the same value for both configurations. Thus p_0 should output the same value starting in both configurations. This is again a contradiction.

Finally suppose both α_0 and α_1 are both TEST&SET. WLOG suppose p_0 received 0 and p_1 received 1 from the TEST&SET call. Then observe that $C_0\alpha_1 \overset{p_1}{\sim} C_1$. In both configurations the Test&Set object contains 1 and p_1 received 1 from the TEST&SET call. As above, p_1 should output the same value starting in both configurations, but this is a contradiction.

Thus starting from the initial multi-valent configuration it is always possible to move to a bi-valent configuration. Any supposed algorithm which solves binary consensus using only one test&set object will have an infinite execution and will contradict the termination condition. □

Theorem 2. *It is possible to solve two processor binary consensus using two Test&Set objects.*

Proof. We will present an algorithm to solve two processor binary consensus using two Test&Set objects and prove its correctness. Let the processors be denoted p_1, p_2 and let the Test&Set objects be denoted t_1, t_2. Further let the input to processor p_i be u_i.

The two Test&Set registers will play different roles in the algorithm. t_1 will be the competition object while t_2 will hold the input value of the processors. First processors "set" t_2 to its input value, then compete for t_1. A processor p_i "sets" t_2 by calling TEST&SET (t_2) if and only if $u_i = 1$. Further, if $u_i = 1$, p_i will continue to compete for t_1 if it was the first to set t_2. The winner of the competition for t_1 (received 0 from TEST&SET(t_1)) will output its own value. The loser will either check the value stored in t_2 or output 0 depending on whether its input was zero or one respectively. See Algorithm 1 for the associated pseudo-code.

Next we prove the correctness of the algorithm. It is easy to see that the algorithm terminates for every processor p_i. To see that validity holds, consider inputs $(u_1, u_2) = (0, 0)$ and $(u_1, u_2) = (1, 1)$. If $u_1 = u_2 = 0$, then both processors will compete for t_1. WLOG suppose that p_1 received 0 and p_2 received 1. p_1 will output 0. p_2 will TEST&SET(t_2), obtain 0 since t_2 was not set, and output 0 as well. Both processors, independent of whether the other processor crashed or not, will output 0. Similarly, suppose $u_1 = u_2 = 1$. Both processors will try to set t_2. WLOG suppose that p_1 received 0 and p_2 received 1. p_2 will output 1 safe in the knowledge that the other processor must have input 1 as well. p_1 will compete for t_1, receive a 0 since it is the only processor competing, and output 1. Again, both processors, independent of whether the other processor crashed or not, will output 1. Thus the validity condition holds.

We show agreement by consider the algorithm on input $(u_1, u_2) = (1, 0)$ (the case where the two inputs are equal is considered above and the case where $(u_1, u_2) = (0, 1)$ is identical by symmetry). p_1 sets t_2, receives 0, then competes for t_1. p_2 competes for t_1 directly. Two cases are possible:

p_1 gets 0: p_1 wins t_1 and outputs 0. p_2 received 1 from t_1 and outputs $\neg 1 = 0$. In essence p_2 *knows* that the other processor had input 0 since it "saw" a 0 in t_2 prior to setting t_2. Both processors were able to output a value irregardless of whether the other processor crashed or not.

p_1 gets 1: p_2 wins t_1 and outputs 1. p_1 received 1 from t_1 and performs TEST&SET(t_2). Since this case can only occur when p_1 successfully set t_2, p_1 will receive and output 1. Both processors were able to output a value irregardless of whether the other processor crashed or not.

By the above case analysis, we see that the algorithm satisfies the agreement condition. $\square$

Algorithm 1 Algorithm for two processor binary consensus using two Test&Set objects: code for processor p_i.

```
if $u_i = 0$ then
    $c \leftarrow$ TEST&SET($t_1$)
    if $c = 0$ then
        output 0
    else
        $s \leftarrow$ TEST&SET($t_2$)
        output $s$
    end if
else
    $s \leftarrow$ TEST&SET($t_2$)
    if $s = 1$ then
        output 1
    end if
    $c \leftarrow$ TEST&SET($t_1$)
    output $\neg c$
end if
```

Claim 3. *It is possible to achieve 1-resilient consensus for 3-processes using Test&Set objects and registers.*

Proof. We will presents an algorithm which achieves 1-resilient consensus for 3-processors using only Test&Set objects and registers and prove its correctness.

Let the processors be p_0, p_1, p_2 and let u_i be the input to processor p_i. In addition, processor p_2 has two registers r_2 and c_2 used to store the output and check values respectively. Since Test&Set objects have consensus number 2 (Attiya page 326), there exists an asynchronous wait-free 2-consensus algorithm, T&S-TWO-CONCENSUS, which only uses Test&Set objects and read/write registers. We will use T&S-TWO-CONCENSUS as a subroutine in our algorithm. The function T&S-TWO-CONCENSUS(p, u, p', u') takes two processors (p and p') along with their respective inputs (u and u') and outputs the consensus value $v \in \{u, u'\}$. The write function of the registers is WRITE(r, v) and it writes value v to register r. The read function of the registers is READ(r) and it outputs the value stored in register r.

The high-level overview of our algorithm is as follows: p_0 and p_1 use T&S-Two-Consensus to agree on value val. Both processors will write val to r_2 then write a 1 to the check register c_2. Finally they output val and terminate. See Algorithm 2 for the associated pseudo-code. p_2 will adopt the value from T&S-Two-Consensus by continuously checking to see if $c_2 = 1$. If so, p_2 will read and output the value of r_2. See Algorithm 3 for the associated pseudo-code.

Next we show the correctness of the algorithm. The termination condition clearly holds for p_0 and p_1. Since at most one processor can fail, one of p_0 and p_1 will successfully write val to r_2 and set c_2. Thus p_2 will eventually see a 1 in c_2 and will terminate as well. Since we assumed that T&S-Two-Concensus is an asynchronous wait-free 2-consensus algorithm, p_0 and p_1 must output the same value val and $val \in \{u_0, u_1\}$. val is then written to r_2 by p_0 or p_1 so p_2 must also output val. Thus the validity and agreement conditions hold as well.

□

Algorithm 2 Algorithm for 1-resilient consensus for 3-processors using only Test&Set objects and read/write registers: code for processors p_0 and p_1.

```
1: $val \leftarrow$ T&S-Two-Concensus$(p_0, u_0, p_1, u_1)$
2: write$(r_2, val)$
3: write$(c_2, 1)$
4: output $val$
```

Algorithm 3 Algorithm for 1-resilient consensus for 3-processors using only Test&Set objects and read/write registers: code for processor p_2.

```
1: $c \leftarrow$ read$(c_2)$
2: while $c = 0$ do
3:     $c \leftarrow$ read$(c_2)$
4: end while
5: $val \leftarrow$ read$(r_2)$
6: output $val$
```