SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

CONCURRENT SYSTEMS
LECTURE 10

Prof. Daniele Gorla

Consensus Number

Which objects allow for a wait free impementation of (binary) consensus?

→ the answer depends on the number of participants

The **consensus number** of an object of type T is the greatest number n such that it is possible to wait free implement a consensus object in a system of n processes by only using objects of type T and atomic R/W registers.

For all T, $CN(T) > 0$; if there is no sup, we let $CN(T) := +\infty$

Thm: let $CN(T1) < CN(T2)$, then there exists no wait free implementation of objects of type T2 in a system of n processes that only uses objects of type T1 and atomic RW reg.s, for all n s.t. $CN(T1) < n \leq CN(T2)$.

Proof

- Fix such an n; by contr., there exists a wait free implementation of objects of type T2 in a system of n processes that only uses objects of type T1 and atomic RW reg.s.
- Since $n \leq CN(T2)$, by def. of CN, there exists a wait free implementation of consensus in a system of n processes that only uses objects of type T2 and atomic RW reg.s.
- Hence, there exists a wait free implementation of consensus in a system of n processes that only uses objects of type T1 and atomic RW reg.s.

→ contraddiction with $CN(T1) < n$ Q.E.D.

Schedules and Configurations

Schedule = sequence of operation invocations issued by processes

Configuration = the global state of a system at a given execution time (values of the shared memory + local state of every process)

Given a configuration C and a schedule S, we denote with S(C) the configuration obtained starting from C and applying S

Let us consider binary consensus implemented by an algorithm A by using base objects and atomic R/W registers; let us call S_A a schedule induced by A.

A configuration C obtained during the execution of A is called

- **v-valent** if $S_A(C)$ decides v, for every S_A;
- **monovalent**, if there exists $v \in \{0,1\}$ s.t. C is v-valent;
- **bivalent**, otherwise.

Fundamental theorem

Thm: If A wait-free implements binary consensus for n processes, then there exists a bivalent initial configuration.

Proof:

1. Let C_i be the initial config. where all p_j (for $j \leq i$) propose 1 and all the others propose 0
2. By validity, C_0 is 0-valent and C_n is 1-valent
3. By contradiction, assume all C_i to be monovalent
4. By (2), there exists an i such that C_{i-1} is 0-valent and C_i is 1-valent
5. By definition, C_{i-1} and C_i only differ in the value proposed by p_i (0 and 1, resp.)
6. Consider an execution of A where p_i is blocked for a very long period
   - by wait freedom, all other processes eventually decide
   - call S the scheduling from the beginning to the point in which all processes but p_i have decided
   - since C_{i-1} is 0-valent, all other processes decide 0
   - By (5) and because p_i is sleeping in S, also in $S(C_i)$ all other processes decide 0
   - If in $S(C_i)$ we resume p_i and it decides 1, we contradict agreement
   - If p_i decides 0, we contradict 1-valence of C_i.

Q.E.D.

CN(Atomic R/W registers) = 1

Thm: There exists no wait-free implementation of binary consensus for 2 processes that uses atomic R/W registers.

Proof:

By contradiction, assume A wait-free, with processes p and q.

By ther previous result, it has an initial bivalent configuration C.

→ let S be a sequence of operations s.t. C' = S(C) is maximally bivalent (i.e., p(S(C)) is 0-valent and q(S(C)) is 1-valent, or viceversa)

p(C') can be R1.read() or R1.write(v) and q(C') can be R2.read() or R2.write(v')

1. $R1 \neq R2$

   Whatever operations p and q issue, we have that q(p(C')) = p(q(C'))

   But q(p(C')) is 0-val (because p(C') is) whereas p(q(C')) is 1-val

2. R1 = R2 and both operations are a read

   Like point (1)

CN(Atomic R/W registers) = 1

3. R1 = R2, with p that reads and q that writes (or viceversa) ⚡

Remark: only p can distinguish C’ from p(C’) (reads put the value read in a local variable, visible only by the process that performed the read)

Let S’ be the scheduling from C’ where p stops and q decides:

→ S’ starts with the write of q

→ S’ leads q to decide 1, since q(C’) is 1-val

Consider p(C’) and apply S’

→ because of the initial remark, q decides 1 also here

Reactivate p

→ if p decides 0, then we would violate agreement

→ if p decides 1, we contradict 0-valence of p(C’)

4. R1 = R2 and both operations are a write ⚡

Remark: q(p(C)) = q(C) cannot be distinguished by q since the value written by p is lost after the write of q

Then, work like in case (3).

Q.E.D.

CN(Test&set) = 2

```
TS a test&set object init at 0
PROP array of proposals, init at whatever

propose(i, v) :=
      PROP[i] ← v
      if TS.test&set() = 0 then return PROP[i] else return PROP[1-i]
```

Wait-freedom, Validity and Integrity hold by construction.

Agreement: the first that performs test&set receives 0 and decides his proposal; the other one receives 1 and decides the other proposal.

Thm.: there exists no A wait free that implements binary consensus for atomic R/W registers and test&set objects for 3 processes.

Proof:

The structure is the same as the previous proof. Consider 3 proc.'s p, q and r.

Let C be bivalent and S maximal s.t. S(C) (call it C') is bivalent:

→ p(C') is 0-val, q(C') is 1-val and r(C') is monoval (for example)

Let's assume that

- at C' r stops for a long time
- op_p and op_q are the next operations that p and q issue from C' by following A

1. op_p and op_q are both R/W operations on atomic registers → like in the previous proof

2. One is an operation on an atomic register and the other one is a test&set, or both are test&set but on different objects

   → like the first case of the previous proof, since p(q(C')) = q(p(C'))

3. They are both test&set on the same object

   → p(q(C')) is 1-val whereas q(p(C')) is 0-val

   Let us now stop both p and q and resume r → r cannot see any difference between p(q(C')) and q(p(C'))
   (the only diff.'s are the values locally stored by p and q as result of T&S)

   Let S' be a schedule of operations only from r that leads p(q(C')) to a decision (that must be 1)

   Since r cannot see any difference between p(q(C')) and q(p(C')), if we run S' from q(p(C')) we must decide 1 as well

   → in contradiction with q(p(C')) be 0-val

CN(Swap) = CN(Fetch&add) = 2

SAPIENZA
UNIVERSITÀ DI ROMA
DIPARTIMENTO DI INFORMATICA

```
S a swap object init at ⊥
PROP array of proposals, init at whatever

propose(i, v) :=
     PROP[i] ← v
     if S.swap(i) = ⊥ then return PROP[i] else return PROP[1-i]
```

```
FA a fetch&add object init at 0
PROP array of proposals, init at whatever

propose(i, v) :=
     PROP[i] ← v
     if FA.fetch&add(1) = 0 then return PROP[i] else return PROP[1-i]
```

REMARK: Similarly to Test&set, we can prove that no consensus is possible with 3 processes.

CN(Compare&swap) = ∞

Let us consider a verison of the compare&swap where, instead of returning a boolean, it always returns the previous value of the object, i.e.:

```
X.compare&swap(old,new) :=
        ┌ tmp ← X
atomic ─┤ if tmp = old then X ← new
        └ return tmp
```

```
CS a compare&swap object init at ⊥

propose(v) :=
     tmp ← CS.compare&swap(⊥,v)
     if tmp = ⊥ then return v else return tmp
```

Exercise: devise a consensus object with CN = ∞ by using the compare&swap that returns booleans.