Consensus Number

魏恒峰

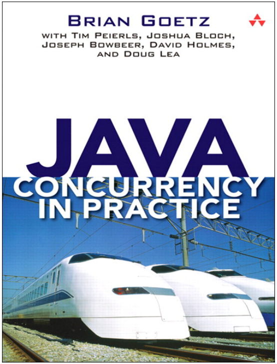
hfwei@nju.edu.cn

2017 年 12 月 14 日


Feel Free to
Ask Questions

Are you Familar with **Concurrent Programming**?

The Key: **Synchronization**!



Using the **Synchronization Primitives**
Provided by Your Favorite Languages.

`synchronized` `BlockingQueue` `Phaser`
`Semaphore` `ConcurrentMap` `Barrier`

synchronized BlockingQueue Phaser
Semaphore ConcurrentMap Barrier

```
class AtomicInteger:
  get()
  set(int newValue)

  getAndIncrement()
  getAndDecrement()
  getAndSet(int newValue)

  compareAndSet(int expectedValue, int newValue)
```

```
compareAndSet(int expectedValue, int newValue)
compareAndSwap(int expectedValue, int newValue)
```

CAS — CMPXCHG

impl.	**usage**

Tasks: Implementing Consensus using Given Primitives.

Consensus

"It looks like we have a consensus."

Propose

Decide

Propose

Decide

Definition (The Consensus Problem)

- Agreement: All (non-faulty) processes must **agreen on the same value**.
- Validity: The common decision value must be the value **proposed** by some process.
- Termination: Each (non-faulty) process must **eventually** decide on a value.

Definition (Consensus Protocol)

(Informally), a consensus protocol is a system of n processes that communicate through a set of shared objects.

Propose

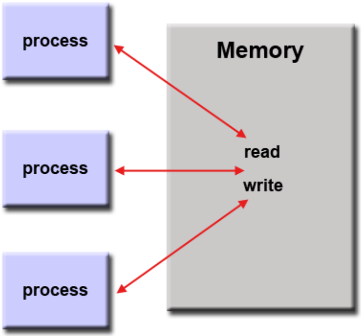
Communicate

Decide

More clarification on “termination”

termination requires wait-free implementation of the consensus object

binary consensus problem



Processes communicate through shared objects.

(redraw)

consensus object (fig here)

```
public interface Consensus<T> {
  T decide(T value);
}
```

consensus protocol

```java
public abstract class ConsensusProtocol<T>
                         implements Consensus<T> {
  protected T[] proposed = (T[]) new Object[N];

  void propose(T value) {
    proposed[ThreadID.get()] = value;
  }

  public abstract T decide(T value);
}
```

implement X using Y

```java
public class CASConsensus<T>
    extends ConsensusProtocol<T> {
  private final int FIRST = -1;
  private AtomicInteger r = new AtomicInteger(FIRST);

  @Override
  public T decide(T value) {
    propose(value);

    int i = ThreadID.get();
    if (r.compareAndSet(FIRST, i))  // I won
      return proposed[i];
    else                            // I lose
      return proposed[r.get()];
  }
}
```

Theorem (Computational Power of CAS)

A register providing compareAndSet() and get() methods can solve the consensus problem for any number of threads.

Definition (Solving Consensus)

"X solves n-process consensus" if there exists a consensus protocol

$$P_1, \cdots, P_n; W, X$$

- W is a set of (atomic) read/write registers;
- W and X may be initialized to any state.

Definition (Consensus Number)

The **consensus number** for X is the largest n for which X solves n-thread consensus.

If no largest n exists, the consensus number is said to be infinite.

Theorem (Consensus Number of CAS)

A register providing compareAndSet() and get() methods has infinite **consensus number**.

Lemma (Y Implements X)

Theorem (Consensus Number as . . .)

in the following, main results (table)

beautiful ideas and proofs


前方高能
—WARNING—


KEEP
CALM
AND
TAKE A
DEEP BREATH

Protocol in terms of set:

Protocol

$$\mathcal{P} = \{\text{All executions of this protocol}\}$$

Execution

$$e = \sigma_0 \xrightarrow{o_1} \sigma_1 \xrightarrow{o_2} \cdots \xrightarrow{o_{n-1}} \sigma_n$$

State

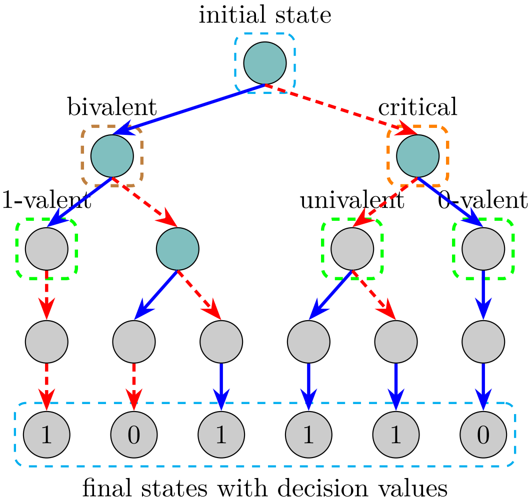
$$\sigma_i : \text{ states of individual threads} + \text{states of shared objects}$$

Operation

$$o_i : \text{ method calls to a shared object}$$

Modeling $\mathcal{P}$ as a Computation "Tree"

(Binary Consensus for 2 Threads)



Theorem (Bivalent Initial State)

Every 2-thread binary consensus protocol has a bivalent initial state.

Theorem (Bivalent Initial State)

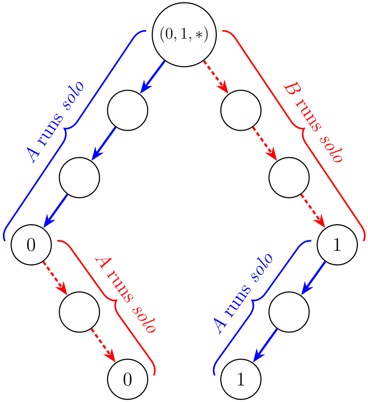
Every 2-thread binary consensus protocol has a bivalent initial state.

$$(A, B, O) : (0, 0, *) \quad (1, 1, *) \quad (0, 1, *) \quad (1, 0, *)$$

Theorem (Bivalent Initial State)

Every 2-thread binary consensus protocol has a bivalent initial state.

$$(A, B, O) : (0, 0, *) \quad (1, 1, *) \quad (0, 1, *) \quad (1, 0, *)$$



Lemma (Bivalent Initial State)

Every n-thread (binary) consensus protocol has a bivalent initial state.

Lemma (Bivalent Initial State)

Every n-thread (binary) consensus protocol has a bivalent initial state.

Theorem (Existence of Critical States)

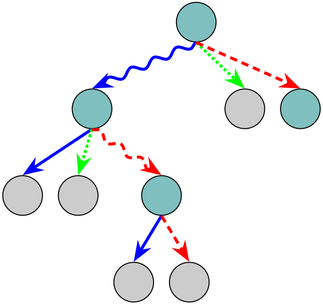
Every wait-free consensus protocol has a critical state.

Lemma (Bivalent Initial State)

Every n-thread (binary) consensus protocol has a bivalent initial state.

Theorem (Existence of Critical States)

Every wait-free consensus protocol has a critical state.




relax
breathe
smile


1

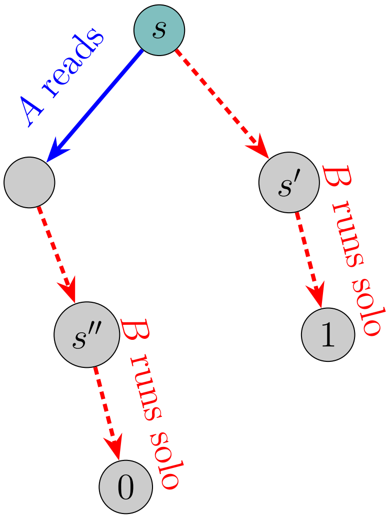
Theorem (Consensus Number of Atomic Registers)

Atomic registers have consensus number 1.

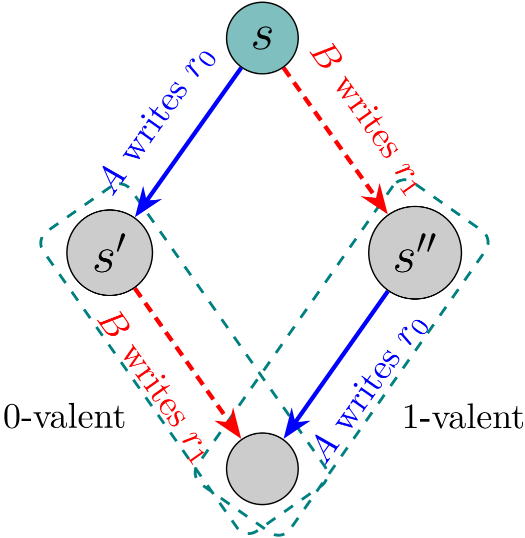
Proof.

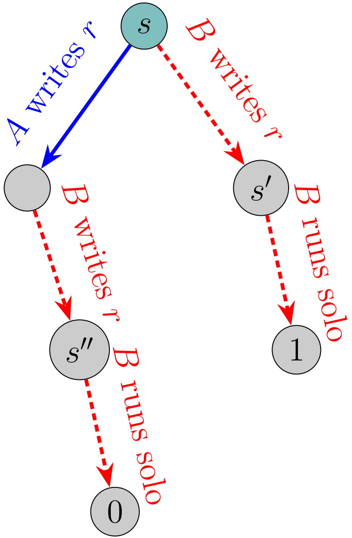
(1) Run the protocol until it reaches a critical state s.

(2) What is the next?



s
A reads
s'
B runs solo
s''
B runs solo
1
0


s
A writes r_0
B writes r_1
s'
s''
B writes r_1
A writes r_0
0-valent
1-valent



s
A writes r
B writes r
s'
B writes r
B runs solo
s''
B runs solo
1
0

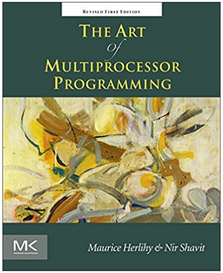
Theorem ("Weakness" of Atomic Read/Write Registers)

It is impossible to construct a wait-free implementation of any object with consensus number greater than 1 using atomic read/write registers.

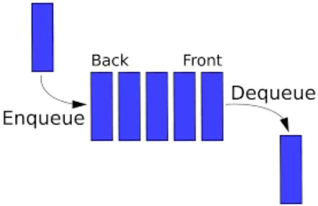
Theorem ("Weakness" of Atomic Read/Write Registers)

It is impossible to construct a wait-free implementation of any object with consensus number greater than 1 using atomic read/write registers.

"$\cdots$ is perhaps one of the most striking impossibility results in Computer Science."

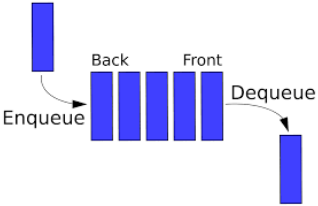



2



Theorem (Consensus Number of Queue)

FIFO queues have consensus number 2.



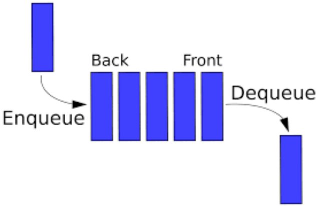
Theorem (Consensus Number of Queue)

FIFO queues have consensus number 2.

Proof.

≥ 2 < 3

$\square$



Theorem (Consensus Number of Queue)

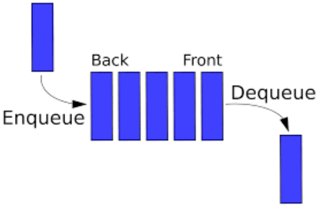
FIFO queues have consensus number 2.

Proof.

≥ 2 < 3

By protocol for 2-thread consensus.

$\square$



Theorem (Consensus Number of Queue)

FIFO queues have consensus number 2.

Proof.

≥ 2	< 3
By protocol for 2-thread consensus.	By the valency argument.

$\square$

Theorem ("Power" of FIFO Queue)

The FIFO queue class can solve 2-thread consensus.

Theorem ("Power" of FIFO Queue)

The FIFO queue class can solve 2-thread consensus.

```
public class QueueConsensus<T>
     extends ConsensusProtocol<T> {
  private static final int WIN = 0;
  private static final int LOSE = 1;

  Queue queue;

  // initialize queue with two items
  public QueueConsensus() {
    queue = new Queue();
    queue.enq(WIN);
    queue.enq(LOSE);
  }
```

```java
public class QueueConsensus<T>
     extends ConsensusProtocol<T> {

  // figure out which thread was fist
  public T decide(T val) {
    propose(val);

    int status = queue.deq();
    int i = ThreadID.get();

    if (status == WIN)
      return proposed[i];
    else
      return proposed[1-i];
  }
}
```

Theorem ("Weakness" of FIFO Queue)

The FIFO queue class cannot solve 3-thread consensus.

Proof.

By the valency argument and case analysis. □

Theorem (Consensus Number 2)

Many similar data types, such as FIFO queue, stack, set, list, and priority queue, all have consensus number 2.

Theorem (Consensus Number 2)

Many similar data types, such as FIFO queue, stack, set, list, and priority queue, all have consensus number 2.

Theorem (Consensus Number 2 (More))

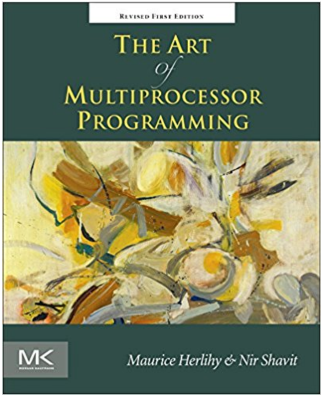
The Idea of “Consensus Number”. (Maurice Herlihy@TOPLAS’1991)

Wait-Free Synchronization

MAURICE HERLIHY
Digital Equipment Corporation

A *wait-free* implementation of a concurrent data object is one that guarantees that any process can complete any operation in a finite number of steps, regardless of the execution speeds of the other processes. The problem of constructing a wait-free implementation of one data object from another lies at the heart of much recent work in concurrent algorithms, concurrent data structures, and multiprocessor architectures. First, we introduce a simple and general technique, based on reduction to a consensus protocol, for proving statements of the form, “there is no wait-free implementation of X by Y.” We derive a hierarchy of objects such that no object at one level has a wait-free implementation in terms of objects at lower levels. In particular, we show that atomic read/write registers, which have been the focus of much recent attention, are at the bottom of the hierarchy: they cannot be used to construct wait-free implementations of many simple and familiar data types. Moreover, classical synchronization primitives such as *test&set* and *fetch&add*, while more powerful than *read* and *write*, are also computationally weak, as are the standard message-passing primitives. Second, nevertheless, we show that there do exist simple universal objects from which one can construct a wait-free implementation of any sequential object.

“The Art of Multiprocessor Programming”.
(Maurice Herlihy & Nir Shavit@2008)



The Idea of "Valency Argument". (FLP@JACM'1985)

Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.


Thank
You!