Several months ago I decided to read theFLP paper that proves impossibility of deterministic binary consensus with even one faulty process and try reformulating it in an understandable way which distills the essence of the proof. Below is the result, comments or improvement suggestions are welcome

1. There are N

nodes. Each of them initially selects 1

or 0

.

1. The goal is for all of them to output 0

or 1

at the end (reach consensus), except for one of them that will become flawed during operation, as described later.

1. Nodes can exchange messages, where each message is guaranteed to get delivered, but it may take an arbitrary long time for a message to get delivered.
2. Time is split in discrete steps. At each step each node can receive messages, and send messages, according to its state. Since the size of the time step can be selected arbitrarily small, the proof works for continuous time.
3. At some point, one of the nodes becomes faulty and stops sending messages. Note, that the node does not become fully Byzantine, it simply halts.
4. Node operation, is, therefore, fully deterministic, and the only force which leads to different consensus results is timings of message delivery, as well as selection of the Byzantine node.
5. An additional requirement on consensus is that if initially all nodes propose 1

the result should be 1

. If all nodes propose 0

the result should be 0

. Note: this requirement is important, since otherwise consensus would be trivially solved by all nodes always deciding 1

, no matter the initial configuration.

1. Each initial configuration C

is described by a string of N

bits, composed of proposals of each node. C

can also be viewed as a number ranging from 0

to 2^N -1

1. A configuration is called monovalent if the resulting consensus value does not depend on timings and selection of the faulty node. By definition, C = 0

(all zeroes) and C = 2^N -1

are monovalent initial configurations, since they always result in 0

and 1

correspondingly.

1. A bivalent configuration is a much more interesting beast. It is a configuration results in 0

or 1

depending on timings and selection of the faulty node. So you can not tell in advance to which value will the consensus converge.

1. The question is whether a bivalent configuration event exists. It turns out it always exists, as is proven below.
2. Indeed, suppose a bivalent configuration does not exist. Then all configurations are either

0-monovalent or 1-monovalent. Then, since C=0

is 0-monovalent, and 2^N -1

is 1

-monovalent, and since you can move from one to another by doing a sequence of single bit flips, there should be a configuration X

, which is 0-monovalent, but by flipping one bit you can turn this configuration into a 1-monovalent configuration.

In other words, for X

consensus always converges to 0

nomatter what, but by flipping one bit in it, you turn it into a configuration that always converges to 1

, nomatter what.

This bit that we are flipping corresponds to a particular node n

. Now consider the situation where this node becomes dead immediately after algorithm starts. In this case since the node never outputs any information, the consensus value should be identical nomatter whether the node initially selects 0

or 1

. So the bit flip should not lead to a change of the final consensus value from 0

to 1

. We reached a contradiction.

Since the assumption was that a bivalent initial configuration does not exist, and we reached a contradiction based on this assumption, we have proven that a bivalent initial configuration does actually exist.

1. Now what does it mean? It means that there is always a configuration, where the result is not pre-determine. Timing and selection of the faulty node are important for the end result.

2. Now, the rest of the argument is to show that if we are at time t

and the current configuration is bivalent, there is a way to select timings to transition to t+1

and stay bi-valent. If we prove this, we will prove that there is a run, where the algorithm never ends. Or, in other words, we will prove that no algorithm that runs in a finite sequence of steps guarantees consensus.

1. Lets assume we are at time t

and the current configuration is a bi-valent configuration c

. Depending on timings, there will be different evolutions of the system, which will result in the set S

of all configurations potentially reachable from the current state.

1. Now lets follow the path of a particular message m

, which was sent a time 0 ... t

and has not yet arrived. Due to asynchronicity, m

can arrive at any time, starting from t+1

. At the point in time after m

arrives, the system may be a mono-valent state, where the evolution is on a pre-determined path to either 0

or 1

, or the may still be in a bi-valent state, where the future is undecided.

1. What we need to prove is that there is a particular timing x

of m

arrival where the system is bivalent after arrival of m

. We prove it by contradiction.

1. Since the initial configuration at time t

was bi-valent (undecided), and since after m

arrives, no-matter timing, the configuration is mono-valent (decided) to either 0

or 1

, there should be time T

such that, everything always converges to 0

if m

arrives at T

and everything always converges to 1

if m

arrives at t+1

.

1. Message m

is destined to a particular node n

. Now consider the case where this node becomes faulty exactly at time T

and stops sending any messages. Since the node provides to the outside world no information about when the message m

arrived, it is impossible for deterministic algorithms running on other nodes to differentiate between the cases of m

arriving at T

and T+1

. Therefore, the system can not converge to both 0

and 1

at the same time, and we reached a contradiction.

20 The initial assumption, that led to contradiction, was that the system was always in a monovalent state after m

arrived.

1. Therefore, we have proven that there is a run where the system is in a bivalent state after m

arrives. And since m

arrives at t+1

at earliest, we have proven that there is a run where the system is in a bivalent state at time t+1

1. Now by induction, for any arbitrary long time T

there is a run where the system is in a bivalent (undecided) state after this time. Therefore, a deterministic algorithm, which works in a finite number of steps is impossible, which concludes the proof of the FLP conjecture.

1. Now looking at step 17, the question is, how are we sure that we can make the node n

faulty at the time we need it to be faulty. May be there was already a node that became faulty earlier, so we sort of ran out of amunition? The answer is that whatever node became faulty earlier we can replace by a non-faulty node with messages delayed arbitrarily long (and, in particular, longer than t

). So the full asynchronicity is critically important for the proof.

Thats is :-)) Comments , improvements are welcome