

---

# A Template for Correct-by-Construction Consensus Protocols

DRAFT v0.01

---

Vlad Zamfir  
Ethereum Foundation

## Abstract

We detail a process for generating correct-by-construction consensus protocols while providing language and ideas for reasoning about consensus protocols.

## 1. Part 0: Introduction

Consensus protocols are used by nodes in distributed systems to decide on the same values, or on the same list of inputs to a replicated state machine (with irrevocable finality). There are, roughly speaking, two classes of consensus protocols known today. One we refer to as “traditional consensus”. This class has its “genetic roots” in Paxos and multi-Paxos[CITE], and in the “traditional” consensus protocol research from the 80s and 90s[CITE]. The other we refer to as “blockchain consensus”. These are protocols that have their roots in the Bitcoin blockchain and Satoshi Nakamoto’s whitepaper[CITE]. We now discuss the differences between these classes of protocols, before giving an overview of the correct-by-construction process.

### 1.1. Comparing Traditional Consensus To Blockchain Consensus

Traditional consensus protocols (such as multi-Paxos and pbft) are notoriously difficult to understand[?]. Blockchain consensus protocols, on the other hand, are much more accessible. This difference comes at least in part from the relative simplicity of Bitcoin’s specification.

In the context of state machine replication, traditional protocols decide (with irrevocable finality) on one “block” of state transitions/transactions to add to the shared operation log at a time. To decide on a block, a node must receive  $\mathcal{O}(N)$  messages, where  $N$  is the number of consensus-forming nodes.

Blockchain consensus protocols like Bitcoin do not finalize/decide on one block at a time. In fact, the Bitcoin blockchain in particular does not make “finalized decisions” at all; blocks are “orphaned” if/when they are not in the highest total difficulty chain. However, if the miners are able to mine on the same blockchain, then the blocks that get deep enough into the blockchain won’t be reverted (“orphaned”). A block’s depth in the blockchain therefore serves as a proxy for finalization. In the average case for blockchain consensus protocols, each node only requires approximately one message,  $\mathcal{O}(1)$ , for every block.

Traditional consensus protocol research has focused on producing protocols that are asynchronously safe (i.e. blocks won’t be reverted due to arbitrary timing of future events) and live in asynchrony (or partial synchrony) (i.e. nodes eventually decide on new blocks). On the other hand, the Bitcoin blockchain is not safe in an asynchronous network but is safe and live (for unknown block-depth or “confirmation count”) in a “partially synchronous network.”

Traditional Byzantine fault tolerant consensus protocols have precisely stated Byzantine fault tolerance numbers (often can tolerate less than a third Byzantine faults, or up to  $t$  faults when there are  $3t + 1$

nodes)[CITE]. On the other hand, it is less clear exactly how many faults (measured as a proportion of hashrate) the Bitcoin blockchain protocol can tolerate.

## 1.2. Overview of the Correct-By-Construction Process

We now give a process for generating consensus protocols with consensus safety (in some conditions). The “correct-by-construction” process has two parts: 1) specifying data types and definitions that the protocol must satisfy to benefit from the implied results/theorems, and 2) “filling in the blanks”—defining implementations of data structures that satisfy the types/definitions required by the proof. The more abstractly part (1) is specified, the more protocols there exist that satisfy the proof, but the more blanks there are to fill in part (2).

We first present the most abstract version of the core result that we rely on, using the most abstract definition of consensus protocols for which the theorem holds that we could come up with. Then we “fill in the blanks” by making assumptions about the objects involved in our initial definition, thereby refining the abstract definition of consensus protocol, making it more concrete.

Through this process we introduce:

- the non-triviality property of consensus protocols
- a fixed known set of  $n$  consensus-forming nodes (with protocol states which are themselves optionally  $n$ -tuples of protocol states)
- asynchronous fault tolerance for a given number of Byzantine faults (if it’s less than the total number of nodes)

At every step the original proof is still satisfied, while we have more information about the consensus protocols which can be generated to satisfy the growing definition given in this process. Once these proofs are in place, generating consensus protocols that satisfy them is relatively easy. Additionally, the proofs themselves happen to be quite elementary.

Following the description of the correct-by-construction process, specifications and experimental observations of a couple of protocols that were generated by this process are given. The second of these protocols, “Casper the Friendly GHOST” is a blockchain consensus protocol that can achieve asynchronous Byzantine fault tolerant consensus safety on each block with the network overhead of the Bitcoin blockchain— $\mathcal{O}(1)$  messages/block/node.

## 2. Part 1: Abstract Consensus Safety Proof

We now give an abstract consensus safety proof as the foundation of the correct-by-construction process. We provide the safety theorem in the most generic terms possible, so that we can generate as wide of an array of protocols as possible. We therefore say nothing in the consensus safety proof about the possible values of the consensus,  $C$ .

We instead use propositions from some logic about values of the consensus,  $\mathcal{L}_C$ . The propositions are either “True” or “False” of/for every possible value of the consensus. The logic has negation and boolean operators/transformations.

We have a notion of “protocol states” and “protocol executions”, which is implemented by a category  $\Sigma$ . Finally, the consensus protocol requires a map  $\mathcal{E}$  called the “estimator” which maps protocol states to propositions in the logic about values of the consensus. These propositions, called “estimates”, are understood to be “guesses” about the value of the consensus.

We use the “looseness” of the estimator to capture the “forking” behaviour of a blockchain consensus protocol, where, for example, a block could be reverted. Note, decisions can always be implemented by this map (it’s possible for  $\mathcal{E}$  to be “monotonic”), but decisions cannot implement forking/non-monotonic estimators (because the set of decisions is monotonic by definition, but forking is not).

Therefore, in order to capture both potentially-forking estimates and certainly-final decisions we only consider protocols with estimators and ask whether/when *hypothetical* decisions on estimates are consensus safe. I.e.,

“if a node *were* to make a decision on estimate  $p$  at protocol state  $\sigma$ , then that node would have consensus safety with any other node *hypothetically* making a decision on estimate  $q$  at state  $\sigma'$  (under some conditions)”

or in the context of Bitcoin,

“if my Bitcoin client were to decide on the block 7 confirmations deep, it would have consensus safety with your client, *if* our clients used similar criteria to make decisions (with very high probability in a synchronous network with less than some amount of Byzantine hashrate).”

So we define the “estimate safety consensus protocol” to be a tuple  $(\mathbf{C}, \mathcal{L}_{\mathbf{C}}, \Sigma, \mathcal{E})$ . We first define these components using only the minimum constraints needed to present the safety proof with a reasonable degree of rigor. In this and the subsequent section, we then continue to refine[CITE [wikipedia.org/wiki/Refinement\\_\(computing\)](https://en.wikipedia.org/wiki/Refinement_(computing))] the protocol definition, until anything satisfying it can plausibly be said to be a Byzantine fault tolerant, asynchronously safe consensus protocol. Finally, the example consensus protocols given in Part 3 are constructed to satisfy the definitions and, therefore, inherit the safety proof.

Now we present the components of the definition of “estimate safety consensus protocols”, then we state and prove the consensus safety theorem, and then we refine the definition until anything satisfying the definition is reasonably a consensus protocol.

**Definition 2.1** (Estimate Safety Consensus Protocols: Part 1 of ?). An *estimate safety consensus protocol* consists of a tuple  $(\mathbf{C}, \mathcal{L}_{\mathbf{C}}, \Sigma, \mathcal{E})$ , containing:

- $\mathbf{C}$ , the set of possible values of the consensus
- $\mathcal{L}_{\mathbf{C}}$ , a logic with propositions,  $\text{props}(\mathcal{L}_{\mathbf{C}})$ , that are satisfied (or not) by the elements of  $\mathbf{C}$
- $\Sigma$ , the category of protocol executions, with objects  $\text{ob}(\Sigma)$  called “protocol states”
- $\mathcal{E} : \text{ob}(\Sigma) \rightarrow \text{props}(\mathcal{L}_{\mathbf{C}})$ , a function called the “estimator”

For now, we say nothing about the set of possible values of the consensus,  $\mathbf{C}$ .

The logic  $\mathcal{L}_{\mathbf{C}}$ , however, must have:

- a set of propositions,  $\text{props}(\mathcal{L}_{\mathbf{C}})$ , which somehow implement maps  $p : \mathbf{C} \rightarrow \{\text{True}, \text{False}\}$ , mapping all possible consensus values to “True” or “False”
- a negation operator  $\neg : \text{props}(\mathcal{L}_{\mathbf{C}}) \rightarrow \text{props}(\mathcal{L}_{\mathbf{C}})$ , such that for every  $c \in \mathbf{C}$ ,  $p(c) \text{ xor } \neg p(c)$
- normally defined boolean algebra operators “and”  $\wedge$ , “or”  $\vee$ , and “implies”  $\Rightarrow$

Using the  $\wedge$  operator, any number of propositions  $p_1, p_2, p_3, \dots, p_n$  can be “compressed” into a single proposition  $p_1 \wedge p_2 \wedge p_3 \dots \wedge p_n$  which is true if and only if each of the propositions is true.

The category of protocol states  $\Sigma$  is a category. No additional information about  $\Sigma$  is required for Step 1, but we define some properties and notation that are subsequently relevant:

- $\text{ob}(\Sigma)$ , a set of objects called “protocol states”:  $\sigma_1, \sigma_2, \sigma_3, \dots$
- $\text{hom}(\Sigma)$ , a set of morphisms called “protocol executions”:  $\tau_1, \tau_2, \tau_3, \dots$
- each execution maps a protocol state to a protocol state  $\tau : \text{ob}(\Sigma) \rightarrow \text{ob}(\Sigma)$
- if a morphism  $\tau$  maps object  $\sigma_1$  to object  $\sigma_2$ , we write  $\sigma_1 \xrightarrow{\tau} \sigma_2$

- if one execution begins where the other ends, protocol executions compose as follows:

$$\sigma_1 \xrightarrow{\tau_1} \sigma_2 \wedge \sigma_2 \xrightarrow{\tau_2} \sigma_3 \implies \sigma_1 \xrightarrow{\tau_2 \circ \tau_1} \sigma_3$$

- $hom(\Sigma)$  is closed under composition:  $\tau_1, \tau_2 \in hom(\Sigma) \implies \tau_2 \circ \tau_1 \in hom(\Sigma)$
- there exists a unique identity morphism  $\tau_0$ , “do nothing”, such that:  $\sigma_1 \xrightarrow{\tau_0} \sigma_2 \implies \sigma_2 = \sigma_1$ , for all  $\sigma_1 \in ob(\Sigma)$
- we denote  $\exists \tau \in hom(\Sigma) . \sigma \xrightarrow{\tau} \sigma'$  simply as  $\sigma \rightarrow \sigma'$

Finally, the estimator map  $\mathcal{E} : ob(\Sigma) \rightarrow props(\mathcal{L}_C)$  has the following “consistency” property:

- if  $\mathcal{E}(\sigma) \Rightarrow p$ , then  $\neg(\mathcal{E}(\sigma) \Rightarrow \neg p)$  for all  $p \in props(\mathcal{L}_C)$  and  $\sigma \in ob(\Sigma)$ .

We now pause the (still incomplete) process of giving the definition of estimate safety consensus protocols.

From now on, any variable named with lower case letter “ $p$ ” will be a proposition of  $\mathcal{L}_C$  and thereby implicitly satisfy  $p \in props(\mathcal{L}_C)$  and variables named with “ $\sigma$ ” will be a objects from the protocol executions category  $\Sigma$  and hence will implicitly satisfy  $\sigma \in ob(\Sigma)$

**Definition 2.2.** A proposition  $p$  is said to have *estimate safety* in a protocol state  $\sigma$  if

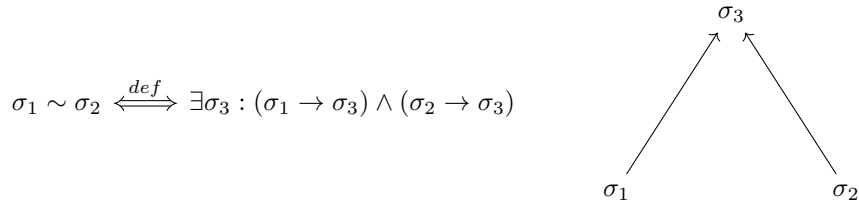
$$\forall \sigma' : \sigma \rightarrow \sigma', \mathcal{E}(\sigma') \Rightarrow p$$

We refer to this simply as “safety”, and we denote safety on  $p$  at  $\sigma$  as  $S(p, \sigma)$  and the absense of safety on  $p$  at  $\sigma$  as  $\neg S(p, \sigma)$ .

Estimate safety consensus protocols have the property that “nodes” only make “decisions” on a proposition  $p$  when they are in a protocol state  $\sigma$  that is safe on  $p$ ;  $S(p, \sigma)$ .<sup>1</sup>

Before we present the consensus safety theorem, we provide some useful notation for the theorem’s assumption: ‘that a pair of states  $\sigma_1, \sigma_2$  have a common future protocol state.’ To more easily refer to this property, we denote this relationship as  $\sigma_1 \sim \sigma_2$ .

**Definition 2.3** ( $\sigma_1 \sim \sigma_2$ ).



**Theorem 1** (Consensus safety).

$$\sigma_1 \sim \sigma_2 \implies \neg(S(p, \sigma_1) \wedge S(\neg p, \sigma_2))$$

Before we prove the theorem or discuss its utility and significance, we first present and prove three lemmas.

**Lemma 1** (Forwards safety).

$$\forall \sigma' : \sigma \rightarrow \sigma', S(p, \sigma) \implies S(p, \sigma')$$

<sup>1</sup>Further discussion on what nodes are and how nodes can tell they have estimate safety follows in subsequent sections.

This lemma asserts that if the protocol is safe on  $p$  at state  $\sigma$ , it will also be safe on  $p$  for all future protocol states. Intuitively,  $p$  holds for all futures  $\sigma'$  of  $\sigma$ . The futures of  $\sigma'$  are also futures of  $\sigma$ , so we can see that  $p$  will also hold in the futures of  $\sigma'$ .

**Proof.** Assuming that  $\sigma \rightarrow \sigma'$  for arbitrary  $\sigma'$  and  $S(p, \sigma)$ , we want to show that  $S(p, \sigma')$ .

$$\begin{aligned}
\sigma &\rightarrow \sigma' && \text{(introducing assumption)} && (1) \\
\forall \sigma'' . \sigma \rightarrow \sigma' \wedge \sigma' \rightarrow \sigma'' \implies \sigma &\rightarrow \sigma'' && \text{(by morphism composition)} && (2) \\
\forall \sigma'' . \sigma' \rightarrow \sigma'' \implies \sigma &\rightarrow \sigma'' && \text{(follows from (1) and (2), by modus ponens (with exportation))} && (3) \\
\forall \sigma^* . \sigma \rightarrow \sigma^* \implies \mathcal{E}(\sigma^*) \implies p &&& \text{(introducing assumption } S(p, \sigma) \text{ by its definition)} && (4) \\
\forall \sigma'' . \sigma' \rightarrow \sigma'' \implies \mathcal{E}(\sigma'') \implies p &&& \text{(follows from (3) and (4), by hypothetical syllogism with substitution } \sigma^* = \sigma'') && (5) \\
S(p, \sigma') &\blacksquare && \text{(equivalent to (5), by applying definition of } S) && (6)
\end{aligned}$$

**Lemma 2** (Current consistency).

$$S(p, \sigma) \implies \neg S(\neg p, \sigma).$$

This lemma asserts that if  $p$  is safe in  $\sigma$ , then its negation  $\neg p$  is not safe in that same state  $\sigma$ . It says that we cannot have contradictory propositions (i.e.  $p$  and  $\neg p$ ) both be safe in the same state  $\sigma$ . This result follows from the consistency property of  $\mathcal{E}$  and the definition of  $S$ .

**Proof.** Assuming that  $S(p, \sigma)$ , we show that  $\neg S(\neg p, \sigma)$

$$\begin{aligned}
\sigma &\rightarrow \sigma && \text{(existence of the identity morphism } \sigma \xrightarrow{\tau_0} \sigma) && (1) \\
\forall \sigma' . \sigma \rightarrow \sigma' \implies \mathcal{E}(\sigma') \implies p &&& \text{(introducing assumption } S(p, \sigma) \text{ by its definition)} && (2) \\
\mathcal{E}(\sigma) \implies p &&& \text{(follows from (1) and (2), by modus ponens with substitution } \sigma' = \sigma) && (3) \\
\mathcal{E}(\sigma) \implies p \implies \neg(\mathcal{E}(\sigma) \implies \neg p) &&& \text{(a property of } \mathcal{E} \text{ for any arguments, by definition)} && (4) \\
\neg(\mathcal{E}(\sigma) \implies \neg p) &&& \text{(follows from (3) and (4), by modus ponens)} && (5) \\
S(\neg p, \sigma) \implies \mathcal{E}(\sigma) \implies \neg p &&& \text{(as in steps (1-3) but applied to } \neg p) && (6) \\
\neg(\mathcal{E}(\sigma) \implies \neg p) \implies \neg S(\neg p, \sigma) &&& \text{(the contrapositive of (6))} && (7) \\
\neg S(\neg p, \sigma) &\blacksquare && \text{(follows from (5) and (7), by modus ponens)} && (8)
\end{aligned}$$

**Lemma 3** (Backwards consistency).

$$\forall \sigma : \sigma \rightarrow \sigma', S(p, \sigma') \implies \neg S(\neg p, \sigma)$$

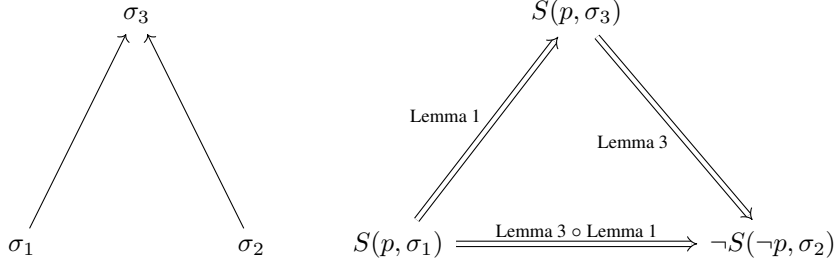
This lemma asserts that if some  $p$  is safe at state  $\sigma'$ , then its negation is not safe at any *previous* state  $\sigma$ .

The intuitive approach is to prove the lemma by contradiction: If we assume for contradiction that a previous state,  $\sigma$ , was safe on  $\neg p$ , then it follows (using forward safety)  $\neg p$  is still safe in future state,  $\sigma'$ . This contradicts the assumption that  $p$  is safe in state  $\sigma'$ , so we can conclude that  $\neg p$  is not safe:  $\neg S(\neg p, \sigma)$ .

**Proof.** Assuming that  $\sigma \rightarrow \sigma'$  for an arbitrary  $\sigma$  and  $S(p, \sigma')$ , we show that  $\neg S(\neg p, \sigma)$

$$\begin{aligned}
S(\neg p, \sigma) &\implies S(\neg p, \sigma') && \text{(from assumption } \sigma \rightarrow \sigma' \text{ and lemma 1)} && (1) \\
S(\neg p, \sigma') &\implies \neg S(p, \sigma') && \text{(lemma 2 applied to } \sigma') && (2) \\
S(\neg p, \sigma) &\implies \neg S(p, \sigma') && \text{(follows from (1) and (2), by hypothetical syllogism)} && (3) \\
S(p, \sigma') &\implies \neg S(\neg p, \sigma) &\blacksquare && \text{(contrapositive of (3))} && (4)
\end{aligned}$$

Equipped with Lemmas 1, 2, and 3, we now present the proof of the distributed consensus safety theorem. We show that  $\sigma_1 \sim \sigma_2$  implies that the nodes with these states are consensus safe.  $\sigma_1 \sim \sigma_2$  means that there exists a  $\sigma_3$  such that...



The safety proof (illustrated above) uses Lemma 1 (Future safety) between  $\sigma_1$  and  $\sigma_3$ , and Lemma 3 (Backwards consistency) for  $\sigma_3$  and  $\sigma_2$ , to conclude that  $S(p, \sigma_1) \implies \neg S(\neg p, \sigma_2)$ . As it turns out, this property is equivalent to  $\neg(S(p, \sigma_1) \wedge S(\neg p, \sigma_2))$ .

**Proof.** Theorem 1 (Consensus safety:  $\sigma_1 \sim \sigma_2 \implies \neg(S(p, \sigma_1) \wedge S(\neg p, \sigma_2))$ )

$$\begin{array}{ll}
\exists \sigma_3 : (\sigma_1 \rightarrow \sigma_3) \wedge (\sigma_2 \rightarrow \sigma_3) & \text{(introducing assumption } \sigma_1 \sim \sigma_2 \text{ by its definition)} \quad (1) \\
S(p, \sigma_1) \implies S(p, \sigma_3) & \text{(follows from } \sigma_1 \rightarrow \sigma_3 \text{ in (1) and Lemma 1)} \quad (2) \\
S(p, \sigma_3) \implies \neg S(\neg p, \sigma_2) & \text{(follows from } \sigma_2 \rightarrow \sigma_3 \text{ in (1) and Lemma 3)} \quad (3) \\
S(p, \sigma_1) \implies \neg S(\neg p, \sigma_2) & \text{(implied by (2) and (3), by hypothetical syllogism)} \quad (4) \\
\neg S(p, \sigma_1) \vee \neg S(\neg p, \sigma_2) & \text{(equivalent to (4), by material implication)} \quad (5) \\
\neg(S(p, \sigma_1) \wedge S(\neg p, \sigma_2)) & \blacksquare \quad \text{(equivalent to (5), by De Morgan's law)} \quad (6)
\end{array}$$

Line (4) of the proof provides a key insight and is important enough to be called a lemma, named “Distributed consistency”:

**Lemma 4** ( $\sigma_1 \sim \sigma_2 \implies$  Distributed consistency).

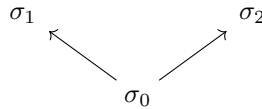
$$\sigma_1 \sim \sigma_2 \implies (S(p, \sigma_1) \implies \neg S(\neg p, \sigma_2))$$

This property is not unlike Lemmas 2 and 3 (current consistency and backwards consistency), however it applies to protocol states for which neither  $\sigma_1 \rightarrow \sigma_2$  nor  $\sigma_2 \rightarrow \sigma_1$ . It means that decisions made at  $\sigma_1$  are consistent with decisions made for *all* protocol states  $\sigma_2$ , at least if  $\sigma_1 \sim \sigma_2$ .

However (fortunately and unfortunately),  $\sigma_1 \sim \sigma_2$  cannot hold for all pairs of states—a property called “non-triviality.” All consensus protocols require that nodes be able to make decisions on mutually exclusive outcomes. Estimate safety consensus protocols only make decisions on safe estimates, but we have not yet introduced the possibility of inconsistent safe estimates. We continue our definition to satisfy the non-triviality constraint.

**Definition 2.4** (Estimate Safety Consensus Protocols: Part 2 of ?). In addition to the earlier properties, estimate safety consensus protocols also have the following:

**(Non-triviality)** There exist a proposition  $p$  and two protocol states  $\sigma_1, \sigma_2$  such that  $S(p, \sigma_1) \wedge S(\neg p, \sigma_2)$ , and there exists some state  $\sigma_0$  which has a protocol execution to each of them:



This part of the definition constrains the domain of the proof, and makes things interesting due to the following result:

**Lemma 5** (Maintaining a shared future is non-trivial).

$$\exists \sigma_1, \sigma_2 . \sigma_1 \approx \sigma_2$$

Where  $\sigma_1 \approx \sigma_2$  denotes  $\neg(\sigma_1 \sim \sigma_2)$

**Proof.**

$$\exists \sigma_1, \sigma_2 . S(p, \sigma_1) \wedge S(\neg p, \sigma_2) \quad (\text{from non-triviality}) \quad (1)$$

$$\sigma \sim \sigma' \implies \neg(S(p, \sigma) \wedge S(\neg p, \sigma')) \quad (\text{consensus safety theorem}) \quad (2)$$

$$S(p, \sigma) \wedge S(\neg p, \sigma') \implies \sigma \approx \sigma' \quad (\text{contrapositive of (2)}) \quad (3)$$

$$\exists \sigma_1, \sigma_2 . \sigma_1 \approx \sigma_2 \quad \blacksquare \quad (\text{from (1) and (3), by modus ponens with substitutions } \sigma = \sigma_1 \text{ and } \sigma' = \sigma_2) \quad (4)$$

The contrapositive of the safety theorem made an appearance on line (3) of this proof, and provides useful intuition.

**Lemma 6** (Consensus failure  $\implies \sigma_1 \approx \sigma_2$ ).

$$S(p, \sigma_1) \wedge S(\neg p, \sigma_2) \implies \sigma_1 \approx \sigma_2$$

So now with non-triviality we start to get a picture of why consensus is not going to be trivial; consensus protocols necessarily have consensus safety failure modes (even when nodes only decide on locally-safe estimates.)

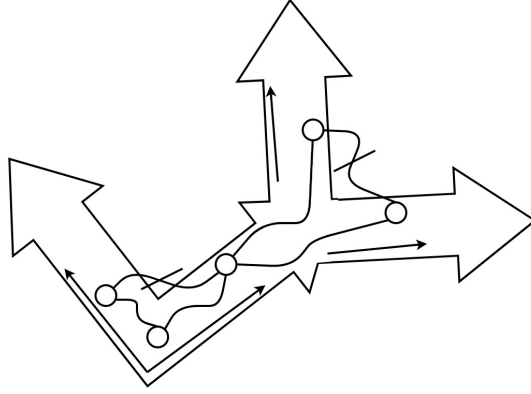


Figure 1: Displaying the bifurcations ( $n$ -furcation) inheret in the statespace of estimate safety consensus protocols (with non-triviality). States with common futures (marked with  $\sim$ ) still have the opportunity to make consistent decisions. States who don't have this opportunity don't share futures (marked with  $\approx$ ).

Figure 1 provides intuition about the nature of the consensus safety theorem and its contrapositive: If two nodes at two protocol states have a shared future, then they haven't made irreconcilable decisions in the  $n$ -furcation in protocol space. If they have made inconsistent decisions in the protocol, then they won't share a future.

The problem of consensus for us, in some way, then, is to help nodes make it to any one of any number of states with mutually exclusive safe estimates, but without losing their shared futures with other nodes in similar positions. Let us develop more language, therefore, for talking about protocol states, their safe estimates, and to their ability to evolve to states with more safe estimates.

[To be continued]