Casper research:
A Template for Correct-by-Construction Consensus Protocols
Featuring:
A specification of Casper the Friendly GHOST

October 26, 2017

Abstract

We detail a process for generating correct-by-construction consensus protocols while providing language and ideas for reasoning about consensus protocols. Then we share the specifications of and experimental observations from two consensus protocols that are derived according to this method—one replicating a single bit, and another replicating a blockchain (i.e. Casper the Friendly GHOST).

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[CITE (http://paxos.systems/)]. 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[CITE].

Traditional consensus protocols have (possibly “hard-coded”) consensus on the set of nodes that can influence the execution of the protocol. In contrast, blockchain consensus protocols generally do not have consensus on the set of nodes that form consensus[1].

Checking message validity is a core component in blockchain consensus protocols, but this concept rarely makes an appearance in traditional consensus protocols.

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

Blockchain consensus protocols like Bitcoin do not finalize/decide on one block one 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”). Thus, block-depth

[1] Although Bitcoin and related protocols do not have consensus on a set of influencer nodes, they do not let anyone freely influence the execution of the protocol due to the computational difficulty of producing a valid block.

serves as a proxy for finalization. In blockchain consensus protocols, each node only requires approximately one message, $\mathcal{O}(1)$, for every block(!).

Traditional consensus protocol research has focussed on producing protocols which 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, blockchain consensus protocols like the Bitcoin blockchain are 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 $n = 3f + 1$)[CITE]. On the other hand, it is less clear exactly how many faults (measured as a proportion of hashrate) blockchain consensus protocols can tolerate[CITE].

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 definiton, 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

The purpose of this section is to give an abstract consensus safety proof "at the top" of the correct-by-construction process. We have aimed to give 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 will say nothing in the consensus safety proof about the possible values of the consensus, $\mathbf{C}$.

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

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

We are going to 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 that decisions can always be implemented by this map (it's possible for $\mathcal{E}$ to be “monotonic”), but that decisions can not 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 will look only at protocols with estimators and ask whether/when *hypothetical* decisions on estimates would be consensus safe.

I.e., “*if* a node *were* to make a decisions on estimate p at protocol state σ, then they would have consensus safety with a node *hypothetically* making a decision on estimate q at state σ' (under some conditions)”, or “*if* it were to decide on the block 7 confirmations deep, my Bitcoin client would have consensus safety with your node, *if* it made similar decisions (with very high probability in a synchronous network with less than some amount of Byzantine hashrate).”

So we are going to take the “estimate safety consensus protocol” to be a tuple $(\mathbf{C}, \mathcal{L}_\mathbf{C}, \mathbf{\Sigma}, \mathcal{E})$. We will define these things (at first) using only the constraints we need to present the safety proof with a reasonable degree of rigor. Then we will continue (in this and the next part of the document) to refine[citation to $wikipedia.org/wiki/Refinement_(computing)$] the protocol definition, until anything satisfying it can plausibly be said to be an 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.

So first we present the components of the definition of “estimate safety consensus protocols” that we will use in the consensus safety proof, then we state and prove the consensus safety theorem, and then we continue with the definition's development until we feel like 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 $(\mathbf{C}, \mathcal{L}_\mathbf{C}, \mathbf{\Sigma}, \mathcal{E})$, containing:

$\mathbf{C}$, the set of possible values of the consensus

$\mathcal{L}_\mathbf{C}$, a logic with propositions, $props(\mathcal{L}_\mathbf{C})$, that are satisfied (or not) by the elements of $\mathbf{C}$

$\mathbf{\Sigma}$, the category of protocol executions, with objects $ob(\mathbf{\Sigma})$ called “protocol states”

$\mathcal{E} : ob(\mathbf{\Sigma}) \to props(\mathcal{L}_\mathbf{C})$, 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, $props(\mathcal{L}_\mathbf{C})$, which somehow implement maps $p : \mathbf{C} \to \{True, False\}$, mapping all possible consensus values to “True” or “False”
- a negation operator $\neg : props(\mathcal{L}_\mathbf{C}) \to props(\mathcal{L}_\mathbf{C})$, such that for every $c \in \mathbf{C}$, $p(c)$ 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, $p_3 \ldots$, p_n can be “compressed” into a single proposition $p_1 \wedge p_2 \wedge p_3 \ldots \wedge p_n$ which is true if and only if each of the propositions is true.

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

- $ob(\mathbf{\Sigma})$, a set of objects called “protocol states”: σ_1, σ_2, σ_3,...
- $mo(\mathbf{\Sigma})$, a set of morphisms called “protocol executions”: τ_1, τ_2, τ_3,...
- each execution maps a protocol state to a protocol state $\tau : ob(\mathbf{\Sigma}) \to ob(\mathbf{\Sigma})$

- if a morphism τ maps object σ_1 to object σ_2, we write $\sigma_1 \xrightarrow{\tau} \sigma_2$
- if one execution begins where the other ends, protocol executions compose as follows:

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

,

- $mo(\mathbf{\Sigma})$ is closed under composition: $\tau_1, \tau_2 \in mo(\mathbf{\Sigma}) \implies \tau_1 \circ \tau_2 \in mo(\mathbf{\Sigma})$
- there exists a unique identity morphism τ_0, "do nothing", such that: $\sigma_1 \xrightarrow{\tau_0} \sigma_2 \implies \sigma_2 = \sigma_1$, for all $\sigma_1 \in ob(\mathbf{\Sigma})$
- we denote $\exists \tau \in mo(\mathbf{\Sigma}) \cdot \sigma \xrightarrow{\tau} \sigma'$ simply as $\sigma \to \sigma'$

Finally, the estimator map $\mathcal{E} : ob(\mathbf{\Sigma}) \to props(\mathcal{L}_{\mathbf{C}})$ has the following property:

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

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 $\mathcal{L}_{\mathbf{C}}$ and thereby implicitly satisfy $p \in props(\mathcal{L}_{\mathbf{C}})$ and variables named with "σ" will be a objects from the protocol executions category $\mathbf{\Sigma}$ and hence will implicitly satisfy $\sigma \in ob(\mathbf{\Sigma})$

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

$$\forall \sigma' \ . \ \sigma \to \sigma' \implies \mathcal{E}(\sigma') \Rightarrow p$$

We will often refer to this simply as "safety", and we denote safety on p at σ as $S(p, \sigma)$, and the absense of safety on p at σ 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 σ that is safe on p; $S(p, \sigma)$. (We will have more discussion later about what nodes are, and on how nodes can tell when they have estimate safety.)

Before we look at the theorem we have set out to prove in this part of the document, I want mention and give notation for the theorem's assumption: That a pair of states σ_1, σ_2 have a common future protocol state. Since we will refer to this property now and again repeatedly later in the document, we will denote this relationship as $\sigma_1 \sim \sigma_2$.

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

$$\sigma_1 \sim \sigma_2 \overset{def}{\iff} \exists \sigma_3 \ . \quad \begin{tikzcd} & \sigma_3 & \\ \sigma_1 \arrow[ur] & & \sigma_2 \arrow[ul] \end{tikzcd}$$

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 will first present and prove three lemmas.

Lemma 1 (Forwards safety).

$$\forall \sigma' \ . \ \sigma \to \sigma' \wedge S(p, \sigma) \implies S(p, \sigma').$$

This lemma says that if the protocol is safe on p at state σ, it will also be safe on p for all future protocol states.

Intuitively, p holds for all futures σ' of σ, and the futures of σ' are also futures of σ, so we can see that p will also hold in the futures of σ'.

Here is a rigorous version of the argument:

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

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

Lemma 2 (Current consistency)**.**

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

The lemma says that if p is safe in σ, then its negation $\neg p$ is not safe in that same state σ. It says that we can't have contradictory propositions (i.e. p and $\neg p$) both be safe at the same state σ. This result will follow from the consistency property of $\mathcal{E}$ and the definition of S.

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

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

Lemma 3 (Backwards consistency)**.**

$$\forall \sigma' \, . \, \sigma' \to \sigma \wedge S(p, \sigma) \implies \neg S(\neg p, \sigma')$$

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

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

Proof. Assuming that $\sigma' \to \sigma$ for an arbirary σ' and $S(p, \sigma)$, we want to show that $\neg S(\neg p, \sigma')$

$S(\neg p, \sigma') \implies S(\neg p, \sigma)$	(from assumption $\sigma' \to \sigma$ and lemma 1, by modus ponens)	(1)
$S(\neg p, \sigma) \implies \neg S(p, \sigma)$	(lemma 2)	(2)
$S(\neg p, \sigma') \implies \neg S(p, \sigma)$	(follows from (1) and (2), by hypothetical syllogism)	(3)
$S(p, \sigma) \implies \neg S(\neg p, \sigma')$ ■	(contrapositive of (3))	(4)

We are now finally ready to give the proof of the distributed consensus safety theorem. But before we look at the proof, lets do a quick overview. We are going to show that $\sigma_1 \sim \sigma_2$ means that the nodes with these states are consensus safe. $\sigma_1 \sim \sigma_2$ means that there eixsts a σ_3 such that..

$$\begin{tikzcd}
& \sigma_3 & & & & S(p,\sigma_3) \arrow[dddr, Rightarrow, "\text{Lemma 3}"] & \\
& & & {} \arrow[r, Rightarrow] & {} & & \\
& & & {} \arrow[r, Rightarrow] & {} & & \\
\sigma_1 \arrow[uuur] & & \sigma_2 \arrow[uuul] & & S(p,\sigma_1) \arrow[uuur, Rightarrow, "\text{Lemma 1}"] \arrow[rr, Rightarrow, "\text{Lemma 1} \circ \text{Lemma 3}"] & & \neg S(\neg p,\sigma_2)
\end{tikzcd}$$

The safety proof (illustrated above) uses lemma 1 (Future safety) between σ_1 and σ_3, and lemma 3 (Backwards consistency) for σ_3 and σ_2, to conclude that $S(p,\sigma_1) \implies \neg S(\neg p,\sigma_2)$. This property is (it turns out) equivalent to $\neg(S(p,\sigma_1) \wedge S(\neg p,\sigma_2))$. The logic is shown in this proof:

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

$\exists \sigma_3 \,.\, \sigma_1 \to \sigma_3 \wedge \sigma_2 \to \sigma_3$ (introducing assumption $\sigma_1 \sim \sigma_2$ by its definition) (1)

$S(p,\sigma_1) \implies S(p,\sigma_3)$ (follows from $\sigma_1 \to \sigma_3$ in (1) and Lemma 1, by modus ponens) (2)

$S(p,\sigma_3) \implies \neg S(\neg p,\sigma_2)$ (follows from $\sigma_2 \to \sigma_3$ in (1) and Lemma 3, by modus ponens) (3)

$S(p,\sigma_1) \implies \neg S(\neg p,\sigma_2)$ (implied by (2) and (3), by hypothetical syllogism) (4)

$\neg S(p,\sigma_1) \vee \neg S(\neg p,\sigma_2)$ (equivalent to (4), by material implication) (5)

$\neg(S(p,\sigma_1) \wedge S(\neg p,\sigma_2))$ ■ (equivalent to (5), by De Morgan's law) (6)

Line (4) of the proof is perhaps the most useful thing to understand. Important enough to be called a lemma and named "Distributed consistency":

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

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

It's a property not unlike Lemmas 2 and 3 (current consistency and backwards consistency), however it applies to protocol states for which neither $\sigma_1 \to \sigma_2$ nor $\sigma_2 \to \sigma_1$. It means that decisions made at σ_1 are consistent with decisions made for *all* protocol states σ_2, at least if $\sigma_1 \sim \sigma_2$.

However (fortunately and unfortunately), there is a good reason that it can't be that $\sigma \sim \sigma'$ for all pairs of states.

All consensus protocols require that nodes are able to make decisions on mutually exclusive outcomes. Estimate safety consensus protocols only make decisions on safe estimates, but we haven't yet required that it's possible to have inconsistent safe estimates. So will continue our definition so we satisfy the non-triviality constraint.

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

(Non-triviality) There exist a proposition p and three protocol states σ_1, σ_2 such that $S(p,\sigma_1) \wedge S(\neg p,\sigma_2)$, and there exists some state σ_0 which has a protocol execution to each of them:

$$\begin{tikzcd}
\sigma_1 & & \sigma_2 \\
& \sigma_0 \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

This part of the definition constrains the things that satisfy the proof, and will make things interesting because of the following result:

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

$$\exists \sigma_1, \sigma_2 \,.\, \sigma_1 \not\sim \sigma_2$$

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

Proof.

$$
\begin{aligned}
&\exists \sigma_1, \sigma_2 \,.\, S(p, \sigma_1) \wedge S(\neg p, \sigma_2) && \text{(from non-triviality)} && (1)\\
&\sigma \sim \sigma' \implies \neg(S(p, \sigma) \wedge S(\neg p, \sigma')) && \text{(consensus safety theorem)} && (2)\\
&S(p, \sigma) \wedge S(\neg p, \sigma') \implies \sigma \nsim \sigma' && \text{(contrapositive of (2))} && (3)\\
&\exists \sigma_1, \sigma_2 \,.\, \sigma_1 \nsim \sigma_2 \quad \blacksquare && \text{(from (2) and (3), by modus ponens with substitutions } \sigma = \sigma_1 \text{ and } \sigma' = \sigma_2\text{)} && (4)
\end{aligned}
$$

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

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

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

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 $\nsim$).

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.

Outline of remainder of part 1:

define bivalence

talk about locking, give some results for locking (maybe w sketch proof)

show locked on locked on = locked on show that anything not locked on with a morphism into locked on also has a morphism not locked on. show that decisions on locked on are consensus safe if we share future

Bring in stuck-freeness: there is a path to S(p) or there is a path to S(not p) (or both).

show choose 1 of 3: locked on S(p), locked on S(not p), and bivalent (locked on S(p) union S(not p))

Wrap up: Talk more about the safety proof, and what it means, and what we know

Then transition to Part 2: Talk a bit about nodes and about why we need to talk about nodes (how do we really guarantee "with something in the protocol" that nodes will make the same?) Talk about how that would look like for a protocol to be bft using this construction

3 Part 2: Nodes, their interaction, and Byzantine faults

The definition of protocol executions in Part 1 does not include nodes, or their interaction. As we have just seen, it is hard to justify why nodes would make the same decision unless they are receiving messages from "the same set" of nodes.

We will therefore go for the traditional route and assume that we have (hard-coded) consensus on "the set of consensus-forming nodes". We are going to do this by insisting on the following:

Definition 3.1 (Estimate Safety Consensus Protocols, Part 4 of ?: Nodes Running in Parallel)**.** Protocol states $\sigma \in ob(\mathbf{\Sigma^1})$ satisfy one of the following:

$$\sigma = (\sigma_1, \sigma_2, \sigma_3, ..., \sigma_{n-1}, \sigma_n)$$

And from now on, σ_i will refer to the i'th entry of the tuple σ, *which is a protocol state in another category* $\mathbf{\Sigma}$.

And the protocol state transitions $mo(\mathbf{\Sigma^1})$ satisfy:

$$\sigma \to \sigma' \implies \sigma_i \to \sigma'_i \quad \forall i = 1...n$$

We will now pause the definition for further analysis again.

The objects in the category $\mathbf{\Sigma^1}$ are tuples of protocol states in another category $\mathbf{\Sigma}$. $\mathbf{\Sigma}$ still enjoys the benefits of the safety proof. $\mathbf{\Sigma^1}$'s morphisms currently allow independent protocol executions of $\mathbf{\Sigma}$. It would certainly be more convenient if the morphisms in $\mathbf{\Sigma^1}$ guaranteed that their source and domain (the states of $\mathbf{\Sigma^1}$ they morph from and to) had protocol states from $\mathbf{\Sigma}$ with consensus safety.

Definition 3.2 (Estimate Safety Consensus Protocols, Part 5 of ?: Nodes Running in Parallel with Consensus Safety)**.** We now require that the protocol state transitions $mo(\mathbf{\Sigma^1})$ satisfy an additional constraint:

For σ, σ' in $ob(\mathbf{\Sigma^1})$:

$$\begin{aligned}
\sigma \to \sigma' &\implies \sigma_i \to \sigma'_i \quad \forall i = 1...n \\
\sigma \to \sigma' &\implies \sigma_i \sim \sigma_j \quad \forall i = 1...n, j = 1...n \\
\sigma \to \sigma' &\implies \sigma'_i \sim \sigma'_j \quad \forall i = 1...n, j = 1...n
\end{aligned}$$

Now we have restricted the consensus protocol executions only to ones that have consensus safety. However, we have not yet indicated in any way that nodes are able to communicate. We just (somewhat artificially) restricted the protocol executions of nodes (who are being modelled by entries in tuples which are objects of our category $\mathbf{\Sigma}$) to guarantee they won't experience consensus failure.

Unfortunately there's nothing in our model right now for communication. But at least we do know that the category $\mathbf{\Sigma^1}$ of consensus safe protocol executions of $\mathbf{\Sigma}$ also satisfies our consensus safety proof. So, lets run nodes who use that category as a protocol in parallel and see what happens:

Definition 3.3 (Estimate Safety Consensus Protocols, Part 6 of ?: Adding Communication)**.** Protocol states for a new category of protocol states $\mathbf{\Sigma^2}$, $\sigma \in ob(\mathbf{\Sigma^2})$, will satisfy the following:

$$\sigma = (\sigma_1, \sigma_2, \sigma_3, ..., \sigma_{n-1}, \sigma_n)$$

This time with $\sigma_i \in \mathbf{\Sigma^1}$

And the protocol state transitions $mo(\mathbf{\Sigma^2})$ satisfy:

$$\sigma \to \sigma' \land \sigma \neq \emptyset \implies \sigma_i \to \sigma_i' \quad \forall i = 1...n$$

We will now pause the definition for further analysis again.

Lets review some things, quickly:

- $\mathbf{\Sigma^2}$ is the category of n parallel nodes executing protocol $\mathbf{\Sigma^1}$,
- $\mathbf{\Sigma^1}$ is the category of n parallel nodes executing protocol $\mathbf{\Sigma}$, without consensus failure

So $\mathbf{\Sigma^2}$ is the category of n nodes watching n nodes execute $\mathbf{\Sigma}$ while maintaining common futures

Do the morphisms in $\mathbf{\Sigma^2}$ evidence exactly n parallel protocol executions of $\mathbf{\Sigma}$? Not necessarily.

Specifically, it's possible given the definition we have so far for us to have an object σ such that we have:

$$\sigma_{ik} \not\leftrightarrow \sigma_{jk} \iff \neg(\sigma_{ik} \to \sigma_{jk} \lor \sigma_{jk} \to \sigma_{ik})$$

Or for it to have a morphism with two objects $\sigma \to \sigma'$:

Then we know that

$$\sigma_{ik} \to \sigma_{ik}' \land \sigma_{jk} \to \sigma_{jk}'$$

but

$$\sigma_{ik} \not\leftrightarrow \sigma_{jk} \lor \sigma_{ik}' \not\leftrightarrow \sigma_{jk}' \lor \sigma_{ik} \not\leftrightarrow \sigma_{jk}' \lor \sigma_{jk} \not\leftrightarrow \sigma_{ik}'$$

So just as we can have an object in $\mathbf{\Sigma^2}$ that has objects at indeces $*k$ in $\mathbf{\Sigma}$ which can't have appeared from a single execution in $\mathbf{\Sigma}$, we can have a mophism in $\mathbf{\Sigma^2}$ which maps one state to another which together have objects at indeces $*k$ which can't have appeared from a single execution in $\mathbf{\Sigma}$.

For example, if neither of the dashed arrows in the following diagram exist, then nodes i and j could not have been watching the same "node" k.

$$\begin{tikzcd}
\sigma_i' k \arrow[dr, dashed] & \sigma_j' k \arrow[l] \arrow[dl, dashed] \\
\sigma_i k \arrow[u] & \sigma_j k \arrow[l] \arrow[u]
\end{tikzcd}$$

Definition 3.4 (Estimate Safety Consensus Protocols, Part 6 of ?: Getting rid of Equivocation)**.** Protocol states for the category $\mathbf{\Sigma^2}$, $\sigma \in ob(\mathbf{\Sigma^2})$, will now also satisfy the following:

$$\forall i, j, k \;.\; \sigma_{ik} \leftrightarrow \sigma_{jk}$$

And the protocol state transitions $mo(\mathbf{\Sigma^2})$ additionally now need to satisfy:
For $\sigma \to \sigma'$:

$\forall k \;.\; \exists\{\tau_i \in mo(\mathbf{\Sigma})\}_{i=1}^{N}$ such that the path $\sigma_{*1} \xrightarrow{\tau_1} \sigma_{*2} \xrightarrow{\tau_3} \sigma_{*3} \xrightarrow{\tau_4} \sigma_{*4} ... \xrightarrow{\tau_N} \sigma_{*N}$
contains each term in σ or σ' of the form σ_{ik} or σ'_{ik}, respectively, with
σ_{ik} not appearing after σ'_{ik} in path $\{\tau_i \in mo(\mathbf{\Sigma})\}_{i=1}^{N}$

I.e. there exists a (single threaded) execution of protocol states $\sigma_i j$ or $\sigma'_i j$ with last index $j = k$ which passes in the direction of $\sigma_{ik} \to \sigma'_{ik}$.
The purpose of this condition is to restrict $\mathbf{\Sigma^2}$ to parallel executions of $\mathbf{\Sigma^1}$ which each observe "the same" parallel execution of Σ.

Lets do some more review:

- $\mathbf{\Sigma^2}$ is the category of n parallel nodes executing protocol $\mathbf{\Sigma^1}$, without equivocation
- $\mathbf{\Sigma^1}$ is the category of n parallel nodes executing protocol $\mathbf{\Sigma}$, without consensus failure

So, a transition in $\mathbf{\Sigma^2}$ has the following properties:

It takes

n nodes who are watching n nodes safely execute a consensus protocol...

...to...

n nodes each of which have new states of n nodes executing a consensus protocol.

Lets suppose that we achieve safety for a node at $\sigma_{ij} \in \mathbf{\Sigma}$....

Then it doesn't break consensus safety of σ_i In fact it locks every $\sigma_i k$ to that outcome [proof required]...

And because additionally every different node in σ is observing the same execution of $\mathbf{\Sigma^2}$, we know also that every $\sigma_j k$ is also locked to that outcome [proof required]

The conclusion is that $S(p, \sigma_{ij}) \implies Locked(\sigma_k l, \{\sigma' : S(p, \sigma')\})$

——

so it would be great if we could guarantee that nodes execute $\mathbf{\Sigma}$ without consensus failure and without equivocation. Well, we happen to know that by "without consensus failure" we mean $\sim$, which can be achieved by finding a possible future.

——-

maybe we can show that without equivocation there is always a shared future? maybe ¡ k faults $\implies \sim$?

——

in the end we want a protocol that:

detects equivocation and doesn't transition if there's too much

like if $\mathbf{\Sigma^2}$ allowed some equivocation, but not too much

How do the estimators relate?!?

———————————————————

Outline for part 2:

Okay lets have objects be tuples of objects

Justify why this will model nodes running the protocol, while also modelling the protocol itself

Restrict morphisms so they are the same nodes

(or so they have some amount of byzantine faults (perhaps by weight))

show this can be implemented in a correct-by-construction manner with message collection and fault detection maybe good entry point for ”we always have a common future state, but it may not be byzantine free enough for us to get there” which maybe can lead to the whole unions and deterministic functions of views thing.

Talk about initial protocol states

Talk about how we unified states of the protocol and states of nodes running the protocol

Justify that the original definition is still satisfied

Show that nodes have consensus safety Show that $S(p, \sigma_1) \implies LockedonS(p, \sigma_2)$ if validator 1 and 2 keep a common future

Show that non-triviality and stuck-freeness can still hold

Talk about the importance of a “good estimator” for non-triviality and stuck-freeness

Latest messages only Follow-the-weights

Maybe we can generally use something like:

$$\mathcal{E}(\sigma) = argmax_{p \in \mathcal{L}_{\mathbf{C}}} \sum_{c \in \mathbf{C}} \sum_{i=1}^{n} W(i) * (\mathcal{E}(\sigma_i)(c) \wedge p(c))$$

But in practice we use some more tractable things.

——————————————————— what about safety oracles?

———————————————————-

Outline for part 3:

Spec casper the friendly binary consensus Spec casper the friendly ghost

Give data

———————————————————-

TO DO:

- So why do we care about $\neg(S(p, \sigma_1) \wedge S(\neg p, \sigma_2))$ or $S(p, \sigma_1) \implies \neg S(\neg p, \sigma_2)$?
- Tell me about non-triviality and failure modes, plox
- Can we get some drawings?
- How is this a consensus protocol? Or how is it related to a consensus protocol?
- How about nodes? What is a network?
- What about stuck-free ness and locks?
- What about liveness m8?