Refinement and Verification of CBC Casper

Ryuya Nakamura*†, Takayuki Jimba†, and Dominik Harz‡
* Faculty of Engineering, The University of Tokyo
† Research and Development, LayerX
Email: {ryuya.nakamura,takayuki.jinba}@layerx.co.jp
‡ Department of Computing, Imperial College London
Email: d.harz@imperial.ac.uk

***Abstract*—Decentralised ledgers are a prime application case for consensus protocols. Changing sets of validators have to agree on a set of transactions in an asynchronous network and in the presence of Byzantine behaviour. Major research efforts focus on creating consensus protocols under such conditions, with proof-of-stake (PoS) representing a promising candidate. PoS aims to reduce the waste of energy inherent to proof-of-work (PoW) consensus protocols. However, a significant challenge is to get PoS protocols "right", i.e. ensure that they are secure *w.r.t.* safety and liveness. The "Correct-by-Construction" (CBC) Casper approach by the Ethereum project employs pen-and-paper proofs to ensure its security. CBC Casper is a framework to define consensus protocols and aims to prove safety without loss of abstractness. Each member of the CBC Casper family of protocols is defined by five *parameters*. CBC Casper models the protocol by a *state* of each validator and *messages* sent by validators. Each validator can transition its state using messages by other validators that include their current consensus value and a justification (i.e. their previous messages). We extend CBC Casper in three ways. First, we summarise the research of CBC Casper and extend the definitions of safety and liveness properties. To this end, we discuss an instance of CBC Casper called Casper The Friendly GHOST (TFG), a consensus protocol using a variant of the GHOST fork-choice rule. Second, we refine the properties of messages and states in CBC Casper and give a definition of blockchain safety for Casper TFG. Third, we formally verify the CBC Casper framework together with our refined message and state properties as well as our blockchain safety definition in the Isabelle/HOL proof assistant.**

I. Introduction

Consensus protocols are at the core of cryptocurrencies. They determine which set of transactions are considered globally valid and need to ensure to be resilient against malicious actors. Generally, a consensus protocol aims to ensure security through *safety* and *liveness*. Consensus protocols with Byzantine actors offer a trade-off of the two properties, as it is not possible to provide both safety and liveness with at least one Byzantine fault in asynchronous networks [2]–[4].

In cryptocurrencies, Nakamoto consensus introduced the principle of proof-of-work (PoW) and combined this with rewards paid to honest participants [5]. Nakamoto consensus is secure under certain conditions when a majority of nodes behaves honestly [6]–[8]. However, Nakamoto consensus and similar PoW protocols consume large amounts of energy and have limited throughput of transactions [9]. Proof-of-stake (PoS) is a proposal to replace the wasteful mining process with staking of cryptocurrencies (e.g. [1], [10], [11]).

Ethereum seeks to replace its current PoW consensus with a more efficient PoS protocol. In Ethereum, two proposals for PoS are discussed. First, Casper the Friendly Finality Gadget (FFG) is introduced initially to provide *finality* in an existing blockchain consensus protocol via PoS [12]. This proposal is modified to a full PoS blockchain later [13]. Second, "Correct-by-Construction" (CBC) Casper is a proposal by Zamfir et al. [14]. An extension to this work, the Minimal CBC Casper family of consensus protocols, is introduced in [1]. This paper builds on and extends the second strand of work, CBC Casper.

CBC Casper[1] intends to create a family of consensus protocols which provides safety in asynchronous networks with Byzantine faults. CBC Casper is a framework to create consensus protocols based on an iterative process rooted in a rigorous mathematical model. The mathematical models captures the parameters of a consensus protocol and their relationships. The iterative process allows to create consensus protocol *instances* that inherit proven properties from the abstract model. Further, CBC Casper provides a "decision function" called a *safety oracle* [15], which a validator can query to know whether a certain value (e.g. block) is agreed upon. For example, CBC Casper provides an abstract proof for asynchronous safety given any consensus values and fork-choice rule. Casper The Friendly GHOST (TFG) is an instance of a CBC Casper protocol that (i) inherits the mathematical model and its safety proof, (ii) instantiates the abstract parameters with a chain of blocks as consensus values and using a variant of GHOST [16] as *fork-choice rule*, and (iii) uses a safety oracle to decide which blocks are members of the chain of blocks.

A. Contribution

Our contributions are summarised below (cf. Fig. 1).

- **Systematisation of CBC Casper:** We summarise CBC Casper from three working papers [1], [14], [17] and give further pointers from blog articles and talks where relevant. We further add aspects including a discussion of faults, a liveness property, and instantiating protocols.
- **Mathematical foundation:** We refine basic properties of CBC Casper including ordering of justifications, ordering of messages, and state transitions. We verify the basic properties in the Isabelle/HOL proof assistant.

[1]**Note:** When we use the term CBC Casper we refer to the Minimal CBC Casper paper by Zamfir et al. [1].

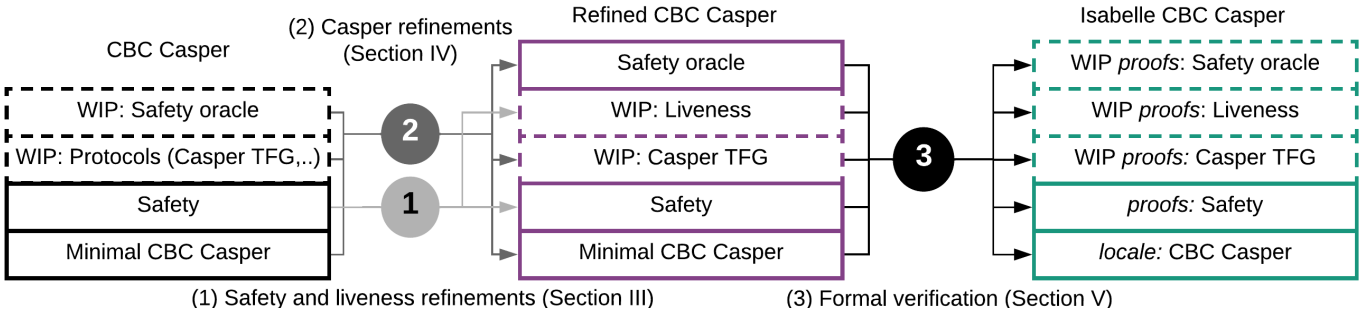

Fig. 1. Our work is summarised in three steps. (1), we update the definition of asynchronous safety and discuss how CBC Casper handles safety and liveness faults. (2), we refine the message and state properties of the Minimal CBC Casper paper [1]. (3), we formally verify CBC Casper in Isabelle/HOL. Dotted lines indicate work-in-progress (WIP).

- **Safety proofs:** We extend CBC Casper by presenting a general definition of asynchronous safety and a blockchain-specific safety definition for Casper TFG. This work is formally verified. Our proofs in Isabelle/HOL are open source[2].
- **Safety oracle:** Last, we revise the safety oracle and show that Byzantine validators must be less than $1/3$ by weight so that validators can reach decisions to finalise blocks.

B. Outline

Section II introduces our systematisation of knowledge of CBC Casper. We discuss the safety and liveness of CBC Casper in Section III. Next, we introduce refinements to CBC Casper in Section IV. This is followed by the description of our approach to formally verify CBC Casper in Isabelle/HOL in Section V. Further, we present a more realistic construction of CBC Casper the Friendly GHOST. We compare our contributions to related work in Section VI. The article is concluded and future work is presented in Section VII.

II. CBC Casper

CBC Casper introduces a family of "correct-by-construction" consensus protocols. We first describe the CBC Casper approach, its parameters, followed by an overview of the protocol.

A. Design approach

CBC Casper abstracts consensus protocols by five *parameters* and provides a *minimal specification*. This minimal specification is intentionally abstract. It is designed to share a proof of safety within the range of the parameters and without assumptions on message delay. By restricting parameters, consensus protocols can be *instantiated* to have other desirable properties without breaking the safety proof. For example, we show a protocol where we restrict one parameter, the *consensus value*, to a chain of blocks. For this, we provide a blockchain-specific safety proof in Section IV-C.

CBC Casper allows an iterative design of consensus protocols by starting from the minimal specification equipped with the proof of safety and the safety oracle. Also, we can discuss the trade-offs between metrics such as latency to finality, the number of validators, and the number of messages required for each protocol within one framework [18]. Because of this design principle, the minimal specification of CBC Casper abstracts its implementation details including the data structure of consensus values, block proposing mechanism, fork choice rule, and signature scheme[3]. Instances of CBC Casper are introduced including Casper the Friendly Binary Consensus, Casper TFG, and a sharding consensus protocol.

Note that *liveness* is not covered by the Minimal CBC Casper specification and hence is discussed specifically in each instance of a protocol. We describe this in Section III-D. An overview of CBC Casper's design approach is given in Figure 2.

B. Parameters

The parameters of CBC Casper are listed in Table I. Validators $\mathcal{V}$ are the finite set of consensus-forming nodes[4]. Weights $\mathcal{W}$ are assigned to each validator. In PoS consensus protocols, the weight can be decided by the amount of stake. The fault threshold t is the upper bound of the added weight of validators which are subject to faults described in Section III-A. Consensus values $\mathcal{C}$ are the set of all possible values which validators agree on. In a binary protocol, consensus values are a set $\{0, 1\}$. In blockchain protocols, they are the set of all possible blocks or chains. The estimator $\mathcal{E}$ is a function executed by each validator in the consensus protocol. Typically, this corresponds to the *fork-choice rule* and derives consensus values from the protocol states Σ. For example, Casper TFG uses a variant of GHOST [16] for the estimator. Validators choose among consensus values if and only if the estimator returns a set with multiple elements.

C. States and messages

In CBC Casper, validators make decisions individually based on their local *states* and a state consists of *messages*.

[2]https://github.com/LayerXcom/cbc-casper-proof

[3]Ali et al. [19] reviewed CBC Casper under the assumption of a "complete" consensus protocol. However, by definition, Minimal CBC Casper is abstract. Their comments are helpful for creating protocol instances that are based on CBC Casper.

[4]In [1], the condition that $\mathcal{V}$ must be finite is not explicitly stated. This condition is reasonable if we consider the protocol is used in real world and also necessary for the later proofs.

TABLE I
CBC CASPER PROTOCOL PARAMETERS.

Parameter	Name	Definition
$\mathcal{V}$	Validator names	$\mathcal{V} \neq \emptyset$: A finite set of validators.
$\mathcal{W}$	Validator weights	$\mathcal{W} : \mathcal{V} \to \mathbb{R}_+$: A function assigning positive weights to $\mathcal{V}$.
t	Fault threshold	$t \in \mathbb{R}_+, t < \sum_{v \in \mathcal{V}} \mathcal{W}(v)$: A positive number smaller than the total weights of all validators.
$\mathcal{C}$	Consensus values	$\lvert\mathcal{C}\rvert > 1$: A set of all consensus values.
$\mathcal{E}$	Estimator	$\mathcal{E} : \Sigma \to \mathcal{P}(\mathcal{C}) \setminus \{\emptyset\}$: A function that returns the set of consensus values from a protocol state Σ.

A *state* represents the set of messages that the validator at that state received so far. A *message* is defined as a 3-tuple consisting of an "estimate" $c \in \mathcal{C}$, together with a "sender" $v \in \mathcal{V}$, and a "justification" $\sigma \in \Sigma$.

Definition 1 (Estimate, Sender, Justification):

$$\text{Estimate}((c, v, \sigma)) := c \tag{1}$$

$$\text{Sender}((c, v, \sigma)) := v \tag{2}$$

$$\text{Justification}((c, v, \sigma)) := \sigma \tag{3}$$

Senders are other validators in the network and authenticated e.g. by a digital signature. Estimates are votes on the consensus value by a sender. For example, a certain chain of blocks within a set of alternative chains. Last, the justification contains all previous messages that the validator received and provides an argument on why the consensus value was reached[5]. Specifically, validators can only have estimates which are the result of executing the estimator function $\mathcal{E}$ for the justification. For example, in Casper TFG, estimates are the blocks selected by its fork choice rule. Justifications are integral to safety oracles described in Section III-C.

[5]In practice, messages do not need to include all the justified messages. Since justification is partially ordered in M (see Section IV), a finite set of messages form a DAG where the vertices are messages and the reachability is the relation of justification. Therefore, it is enough for a message to include the messages which it is connected to in the transitive closure of the DAG.

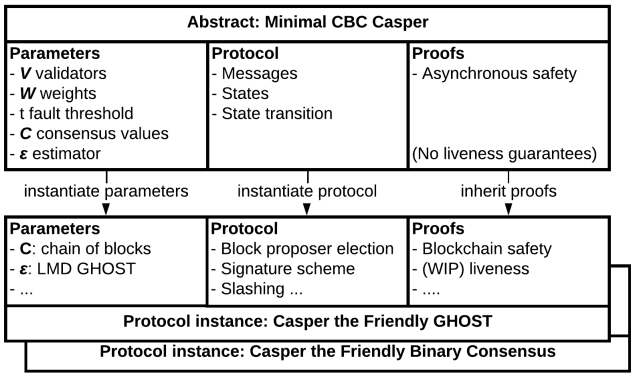

Fig. 2. Minimal CBC Casper is an abstract framework. Protocols that use Minimal CBC Casper are instances that share the proofs of Minimal CBC Casper. In this example we display how CBC Casper is used to create an instance of Casper the Friendly GHOST. First, the parameters are instantiated (as described in Section III-D). Next, the abstract protocol is made concrete by adding the required components. Further, the safety proof is added as discussed in Section IV-C (the liveness proof is WIP). Last, Casper TFG is constructed and verified using Isabelle/HOL (see Section V-E).

The sets of possible states Σ and messages M are defined below. It is an interpretation how possible validators' states and messages evolve during the execution of the protocol.

Definition 2 (States and messages):

$$\Sigma^0 := \{\emptyset\} \tag{4}$$

$$M^n := \{m \in \mathcal{C} \times \mathcal{V} \times \Sigma^n \mid \text{Estimate}(m) \in \mathcal{E}(\text{Justification}(m))\} \tag{5}$$

$$\Sigma^n := \{\sigma \in \mathcal{P}_{\text{finite}}(M^{n-1}) \mid \forall m \in \sigma.\ \text{Justification}(m) \subseteq \sigma\},\ \text{for } n > 0 \tag{6}$$

$$M := \bigcup_{i=0}^{\infty} M^i \tag{7}$$

$$\Sigma := \bigcup_{i=0}^{\infty} \Sigma^i \tag{8}$$

$\mathcal{P}_{\text{finite}}$ is a power set function which only returns finite sets. Hence all $\sigma \in \Sigma$ and $\text{Justification}(m)$ of $m \in M$ are finite. We formally verify that Σ^n and M^n are monotonically growing, i.e. for all $n \in \mathbf{N}$ [6], $\Sigma^n \subseteq \Sigma^{n+1}$ and $M^n \subseteq M^{n+1}$. Note that a set of messages is not necessarily a state, i.e. do not necessarily exists in Σ. When $m_1 \in \text{Justification}(m_2)$, m_1 is *justified* by m_2 and m_2 is a *later* message of m_1. The message sent by a validator which is not justified by any messages from the same validator is called the *latest message* of the validator.

Definition 3 (State transition): With a newly received message, a validator can transition its state to a new state σ', if and only if $\sigma' \in \Sigma$. This is called *state transition* defined as a relationship $\to$ which represents a set inclusion.

$$\sigma_1 \to \sigma_2 :\Leftrightarrow \sigma_1 \subseteq \sigma_2 \tag{9}$$

A set of states which a state σ can transit to is called the *future states* of σ.

A final decision on a new consensus value is reached after seeing "enough" valid messages from validators determined by the safety oracle. The minimal specification only covers definitions of valid messages and states, state transitions, and the safety oracle. Also, the minimal specification does not make any assumption of timing i.e. it is in an asynchronous network. Other specifications such as validators' strategies (e.g. when they should send a message) are not specified.

III. SAFETY AND LIVENESS

We discuss faults, safety and liveness properties as well as the safety oracle in CBC Casper. Last, we elaborate the current state of CBC Casper.

[6]In this paper, $\mathbf{N}$ represents a set of integers greater or equal to 0.

A. Faults

Validators are described by their local states $\{\sigma_i \mid 1 \leq i \leq n\} \subseteq \Sigma$. We assume that protocol-following validators can detect valid messages and transit to valid states. Therefore, we need to consider faults that prevent consensus under these assumptions. For example, if a validator σ_i fails to send or receive a message m, a state transition to $\sigma_i \cup \{m\}$ is not possible. This is considered a liveness fault (omission fault). However, in asynchronous networks omission faults cannot be distinguished from latency.

We can prove safety by considering *equivocation faults*. Equivocation is a Byzantine behaviour where a validator shows different protocol execution [20].

Definition 4 (Equivocation): In CBC Casper, equivocation is defined as producing a pair of messages which do not justify each other.

$$\begin{aligned} m_1 \perp m_2 :\Leftrightarrow \mathrm{Sender}(m_1) = \mathrm{Sender}(m_2) \wedge m_1 \neq m_2 \\ \wedge\, m_1 \notin \mathrm{Justification}(m_2) \wedge m_2 \notin \mathrm{Justification}(m_1) \end{aligned} \tag{10}$$

This is considered a fault because in a single execution of the protocol, every time a validator sends a message, the validator must have seen their previous messages.

Definition 5 (Equivocating validators): Equivocating validators can show inconsistent decisions to different validators who received only one of the equivocating messages.

$$\begin{aligned} E(\sigma) := \{v \in \mathcal{V} \mid \exists m_1, m_2 \in \sigma. \\ m_1 \perp m_2 \wedge\ \mathrm{Sender}(m_1) = v\} \end{aligned} \tag{11}$$

Note that equivocation faults are *accountable* [21], i.e. a deposit and slash mechanism to economically prevent equivocating can be implemented because a pair of equivocating messages is a verifiable evidence.

For convenience, we define helper functions about equivocation faults. For this, we first define a function to measure the weight of a given set of validators.

Definition 6 (Weight measure):

$$W(V) := \sum_{v \in V} \mathcal{W}(v) \tag{12}$$

Then we define a function to measure the weight of equivocating validators.

Definition 7 (Equivocation fault weight):

$$F(\sigma) := W(E(\sigma)) \tag{13}$$

We define the set of states where there are t equivocation faults (by weight) or less.

Definition 8 (States with equivocation fault threshold):

$$\Sigma_t := \{\sigma \in \Sigma \mid F(\sigma) \leq t\} \tag{14}$$

Moreover, we define the set of future states of σ where there are t equivocation faults or less.

Definition 9 (Future states with equivocation fault threshold):

$$\mathrm{Futures}_t(\sigma) := \{\sigma' \in \Sigma_t \mid \sigma \to \sigma'\} \tag{15}$$

We need to discuss more kinds of faults other than equivocation faults when we instantiate a protocol to prove liveness.

B. Safety

CBC Casper claims that its family of consensus protocols is *asynchronously safe*, i.e. it is not possible for validators to make inconsistent decisions in an asynchronous network, without assumptions on the time delay between sending and receiving messages. In this section, we introduce asynchronous safety of CBC Casper, modifying the definition in [1].

Validators in CBC Casper make decision on *properties* of consensus values[7]. For example, in binary consensus protocols where $\mathcal{C} = \{0, 1\}$, the property can be "a consensus value is 1". In blockchain consensus protocols, the property is the membership of a certain block b in a chain.

Definition 10 (Block membership): A consensus value is a block which has a certain block b as its ancestor.

$$\mathrm{P}_{\mathrm{block}}(b) := \lambda b'.\ b \downharpoonright b' \tag{16}$$

Here $b \downharpoonright b'$ defined to return true if and only if b is an ancestor block of b'. A property is called to "hold at a state" if all the consensus values the estimator returns at the state satisfy the property.

Definition 11 (Decision): Validators in CBC Casper make decisions on *safe properties* defined as properties which hold in any future state where there are t equivocation faults by weight or less. Hence, a decision of a validator at a state σ is defined as:

$$\begin{aligned} \mathrm{Decisions}(\sigma) := \{p \mid \forall \sigma' \in \mathrm{Futures}_t(\sigma). \\ \forall c \in \mathcal{E}(\sigma').\ p(c)\} \end{aligned} \tag{17}$$

Decision on $\mathrm{P}_{\mathrm{block}}(b)$ means the *finality* of the block b.

Finally, we introduce the definition of safety. Safety is discussed for n validators that have no more than t equivocation faults by weight. Because CBC Casper models validators local states, this assumption is expressed as: there are no more than t equivocation faults in the union of (local) states.

Theorem 1 (Safety): If there are t equivocation faults or less in the union of states of a finite number n of validators, a negation of a property decided by one validator is not decided by another validator.

$$\begin{aligned} \{\sigma_i \mid 1 \leq i \leq n\} \subseteq \Sigma_t.\ F(\bigcup_{i=1}^{n} \sigma_i) \leq t \\ \Longrightarrow \forall p \in \bigcup_{i=1}^{n} \mathrm{Decisions}(\sigma_i). \\ \neg\ p \notin \bigcup_{i=1}^{n} \mathrm{Decisions}(\sigma_i) \end{aligned} \tag{18}$$

Our definition of safety is different from [1]. The reason of this modification is explained in Appendix A.

We provide the proof of safety in this abstract form and elaborate on the safety of blockchain consensus protocols in Section IV-C. Also, these are formally verified in Isabelle/HOL and the representation is introduced in Section V-C.

[7]The safety of decisions on properties of *states* is also discussed in [1]. We omit it in this paper for the sake of brevity.

C. Safety oracle

To make a decision, validators must *detect* that some properties hold in any future state where there are t equivocations or less. Although we can have a detection rule specific to a certain protocol and properties, CBC Casper implements a general decision mechanisms called *safety oracle* so that the decision mechanism can be discussed inside the CBC approach. Different types of safety oracles are proposed including the *clique oracle* [18]. The draft of the updated version of [1] has a section of safety oracle which has the definition of clique oracle and the unfinished proof of the correctness of the oracle, which is in parts described formally in [15]. In the rest of this section, we extend this work of safety oracle.

First, the concept of "clique" is introduced. Clique is a set of non-equivocating validators who are "locked on" a property p, that is, validators who mutually see each other agreeing on p and will not see each other disagreeing on p in the observer's state. A validator is *agreeing* on p if and only if the *latest estimate* (i.e. the estimate of its latest message) satisfies p. A validator is *disagreeing* on p if and only if she is neither agreeing on p nor equivocating.

Definition 12 (Clique): In a state σ, a *clique* is a set of non-equivocating validators V such that for all $v \in V$, (i) in the justification of the latest message of v, p holds at the latest estimate of the other validators in V, and (ii) σ does not include any later message of the other validators in V whose estimate does not satisfy p.

For the clique oracle to work, the target property must be a *max driven* property i.e. a property which holds when the set of non-equivocating validators agreeing on the property is larger than the set of non-equivocating validators disagreeing on the property by weight. The clique oracle makes a validator decide on a property p when there is a clique larger than a certain *threshold* by weight. (Note that this threshold is different from equivocation fault threshold t.)

Definition 13 (Clique oracle threshold):

$$T(\sigma) := W(\mathcal{V})/2 + t/2 - F(\sigma) \tag{19}$$

A blockchain consensus protocol explained in Section III-D can adopt this threshold.

The validity of clique oracle is supported by this theorem.

Theorem 2 (Clique oracle detects safe properties): If there is a clique for a max driven property p larger than $T(\sigma)$, p holds in any future state of σ where there are t equivocations or less.

In Appendix B, we show this theorem by extending the proofs in Rush [15]. Concerning the Byzantine fault threshold t', we prove that $t' < W(\mathcal{V})/3$ is a necessary and sufficient condition for the protocol not to "get stuck" i.e. there are enough non-faulty validators that can form a clique in Appendix C.

The safety oracle is not yet finalised and is currently being debated. The clique oracle described above is not shown to be optimal and further research *w.r.t.* computational efficiency and latency for detection is left as future work.

D. Liveness

The minimal specification provides safety and a safety oracle but not *liveness* i.e. that validators eventually decide on a proposed block. Proving liveness in the minimal specification is impossible because it does not make assumptions on timing and we cannot prove both safety and liveness in an asynchronous network [3]. Therefore, we need to instantiate a specific protocol from CBC Casper and specify the validator strategy to prove liveness. Also, additional faults other than equivocation need to be considered. In this section, we discuss an instance of a protocol and how we can discuss liveness with it, while we leave the complete and formally verified proof of liveness as future work.

We instantiate Casper TFG [1], which is a blockchain consensus protocol where the consensus values are blocks and the fork-choice rule is a variant of GHOST [16] called *latest message driven (LMD) GHOST* that calculates the score of a block only by the latest messages of validators. In Appendix D, we prove that the block membership property (*Definition 10*) is max driven in LMD GHOST so we can use the clique oracle from Section III-C. Here we instantiate it as a "vote-by-block" style blockchain consensus protocol where all messages must propose a new block as its estimate by specifying a certain mechanism of block proposer election[8]. For this, we modify the estimator so that it only allows new blocks which extend the head decided by LMD GHOST. Then, we discuss liveness of this protocol.

Conjecture 1 (Liveness with clique oracle): Protocol-following validators eventually reach a state $\sigma \in \Sigma_t$ where there exists a clique of $\mathrm{P}_{\mathrm{block}}(b)$ which is larger than $T(\sigma)$ by weight (under appropriate assumptions on fault and network).

In blockchain consensus protocol, the proposers of descendant blocks of b are agreeing on $\mathrm{P}_{\mathrm{block}}(b)$ at the estimate of the message which proposed that block. Also, a message justifies all the messages which proposed its ancestor blocks as a part of the evidence of the fork choice. Therefore, a clique V for $\mathrm{P}_{\mathrm{block}}(b)$ forms when each of validators in V has produced a descendant block of b and *again* produced a descendant block of b justifying the first messages of all the validators in V, as illustrated in Figure 3. Moreover, a clique of $\mathrm{P}_{\mathrm{block}}(b)$ is also valid for $\mathrm{P}_{\mathrm{block}}(b')$ if b' is an ancestor block of b. From these observations, we conjecture that if a sufficient number of validators converges to the same chain, they form a clique which finalises blocks that are "deep" in the chain. We leave more detailed analysis of liveness as future work.

E. Discussions

CBC Casper is an ongoing research and has possible extensions and issues in both theory and practice. The previous paper of CBC Casper by Zamfir [14] claims that to remove in-protocol threshold by the decision-making rule based on the *subjective* equivocation fault threshold t demotivate to form

[8]We can also instantiate a "block-and-vote" protocol where the estimator also allows proposed blocks as described in V-E. This is one of the design choices that CBC Casper is designed to provide.

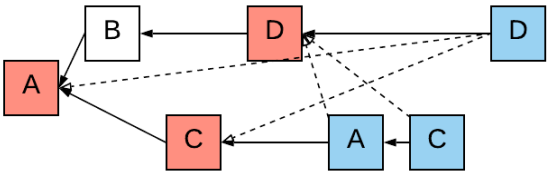

Fig. 3. A clique in a blockchain. Plain arrows represent the chain structure and dotted arrows represent justifications. (Arrows of justifications to ancestor blocks are omitted.) Validator A, C, and D form a clique for the first block from A. All the blue blocks justify all the red blocks.

a cartel. Also, when we consider using CBC Casper for a public blockchain, we should achieve a *dynamic validator set* because currently we are assuming static validator set $\mathcal{V}$. We proposed a way to rotate validators specific to blockchain consensus protocols based on modified LMD GHOST which calculate a score of a block by the weight extracted from the blockchain [22], extending the proposal in [14].

There is a large gap between the minimal specification and an implementable specification. First, CBC Casper makes no assumptions on how the sender of each message is authenticated but it is important when the protocols are used with *untrusted parties*. Second, for blockchain consensus protocols in CBC Casper the mechanism of *block proposal* and the *fork-choice rule* need to be specified. These are closely related to the liveness of the protocol as discussed in Section III-D.

IV. Refinements of CBC Casper

We capture *message justifications* and *state transitions* in CBC Casper using *binary relations* and *ordering*. Message justifications and state transitions form the foundation of the formal verification described in Section V and the proofs of safety oracle in Appendix B. Moreover, using the lemmas of message justifications and state transitions, we also provide the proof of asynchronous safety described in Section III-B. Furthermore, we instantiate *blockchain consensus safety* from this abstract safety. The lemmas and theorems introduced in this section are proved in Isabelle/HOL.

First, we introduce a lemma prerequisite for our later proofs.

Lemma 1:

$$\forall \sigma \in \Sigma.\ \forall m \in \sigma.\ \mathrm{Justification}(m) \subseteq \sigma \tag{20}$$

Proof: There exists $n \in \mathbf{N}$ s.t. $\sigma \in \Sigma^n$ by the definition of Σ. When $n = 0$, we have $\sigma = \emptyset$ by the definition of Σ^0 and hence $\forall m \in \sigma.\ P$ for any P. This implies the goal. When $n \geq 1$, we can prove the goal by the definition of Σ^n. ■

A. Message justification

Definition 14 (Message justification): We define *message justifications* as a binary relation over M.

$$m_1 \prec m_2 :\Leftrightarrow m_1 \in \mathrm{Justification}(m_2) \tag{21}$$

In this subsection we prove lemmas about ordered sets defined with this relation. Also, we define a function From_sender which returns a set of messages in $\sigma \in \Sigma$ sent by a validator v as follows.

Definition 15 (Messages from a sender):

$$\mathrm{From\_sender}(v, \sigma) = \{m \in \sigma : \mathrm{Sender}(m) = v\} \tag{22}$$

Lemma 2: $\prec$ is a *strict partial order* on M.

Proof: We prove this lemma by showing that $\prec$ on M is *irreflexive* and *transitive*. We first prove irreflexivity:

$$\forall m \in M.\ \neg\ m \prec m \tag{23}$$

We assume that there exists $m \in M$ such that $m \prec m$ towards contradiction. By the definition of M, there exists $n \in \mathbf{N}$ s.t. $m \in M^n$. By the definition of M^n, $\mathrm{Justification}(m) \in \Sigma^n$. When $n = 0$, we have $\mathrm{Justification}(m) = \emptyset$ by the definition of Σ^0 and this contradicts to the assumption $m \in \mathrm{Justification}(m)$. When $n \geq 1$, $\mathrm{Justification}(m) \in \Sigma^n$ implies $\mathrm{Justification}(m) \in \mathcal{P}_{\mathrm{finite}}(M^{n-1})$ by the definition of Σ^n. From this and $m \in \mathrm{Justification}(m)$, we have $m \in M^{n-1}$. By repeating these, $m \in M^0$ is derived and this makes the same contradiction in $n = 0$.

Next, we prove transitivity:

$$\begin{aligned} &\forall m_1, m_2, m_3 \in M.\ m_1 \prec m_2 \\ &\wedge\ m_2 \prec m_3 \implies m_1 \prec m_3 \end{aligned} \tag{24}$$

We use *monotonicity of justification* [1]:

$$\begin{aligned} &\forall m, m' \in M.\ m \prec m' \\ &\implies \mathrm{Justification}(m) \subseteq \mathrm{Justification}(m') \end{aligned} \tag{25}$$

From this and the assumption, we have $\mathrm{Justification}(m_2) \subseteq \mathrm{Justification}(m_3)$. From this and the assumption $m_1 \in \mathrm{Justification}(m_2)$, we have the goal $m_1 \in \mathrm{Justification}(m_3)$. Finally, the irreflexivity and the transitivity of $\prec$ on M implies that it is a strict partial order. ■

Lemma 3: $\prec$ is *well-founded* on M.

Proof: First, we prove *strict monotonicity of justification*:

$$\begin{aligned} &\forall m, m' \in M.\ m \prec m' \\ &\implies \mathrm{Justification}(m) \subset \mathrm{Justification}(m') \end{aligned} \tag{26}$$

By the monotonicity of justification, we have $\mathrm{Justification}(m) \subseteq \mathrm{Justification}(m')$. Also, because of the irreflexivity of justification, we have $m \notin \mathrm{Justification}(m)$ These and the assumption $m \in \mathrm{Justification}(m')$ implies the strict monotonicity of justification.

Then, we prove the goal by contradiction. If there exists a *infinite descending chain* consists of elements of M i.e., a sequence m_0, m_1, m_2 ... such that $m_{i+1} \prec m_i$ for all $i \in \mathbf{N}$, the size of $\mathrm{Justification}(m_i)$ is decreasing monotonically due to the strict monotonicity of justification. However, this leads to contradiction because the size of $\mathrm{Justification}(m_i)$ is greater or equal to 0 and justifications of messages are finite by the definition of Σ^n and M^n. ■

Lemma 4: $\prec$ is a *strict linear order* on $\mathrm{From\_sender}(v, \sigma)$ if $\sigma \in \Sigma$ and $v \in \mathcal{V}$ is non-equivocating validator.

Proof: $\mathrm{From\_sender}(v, \sigma)$ is a subset of σ by the definition. Also, we can prove $\sigma \subseteq M$ by the definition of states and messages. From these, we have

$\mathrm{From\_sender}(v, \sigma) \subseteq M$. This, *Lemma 2* and *Lemma 3* implies that $\prec$ on $\mathrm{From\_sender}(v, \sigma)$ is *partially ordered* and *well-founded*. Then, we show the goal by showing that $\prec$ on $\mathrm{From\_sender}(v, \sigma)$ is *semi-connex* i.e. any distinct pair of messages in $\mathrm{From\_sender}(v, \sigma)$ are *comparable* under $\prec$:

$$\forall m_1, m_2 \in M.\ m_1 = m_2 \vee m_1 \prec m_2 \vee m_2 \prec m_1 \quad (27)$$

This is proven by the definition of equivocation and the assumption $v \notin E(\sigma)$. ■

Lemma 5: $\prec$ is a *strict well order* on $\mathrm{From\_sender}(v, \sigma)$ if $v \in \mathcal{V}$ is non-equivocating validator.

Proof: We can prove this because by *Lemma 3* and *Lemma 4*, $\prec$ on $\mathrm{From\_sender}(v, \sigma)$ is well-founded and a strict linear order if $v \in \mathcal{V}$ is non-equivocating validator. ■

B. State transition

We define *state transition* as a binary relationship $\to$ over Σ which is introduced in Section II-C. Then, we prove lemmas about state transitions.

Lemma 6: $\to$ is a *partial order* on Σ.

Proof: We can prove this lemma by showing $\to$ has *reflexivity*, *transitivity* and *antisymmetry* on Σ. These properties are obvious because $\to$ is a relation of subset. ■

Lemma 7: Receiving a single message m at a state σ makes state transition if and only if $\mathrm{Justification}(m) \subseteq \sigma$.

$$\begin{aligned} &\forall \sigma \in \Sigma.\ \forall m \in M.\ \sigma \cup \{m\} \in \Sigma \\ &\iff \mathrm{Justification}(m) \subseteq \sigma \end{aligned} \quad (28)$$

Proof: First, we prove the $\implies$ direction. By the assumption and *Lemma 1*, we have $\mathrm{Justification}(m) \subseteq \sigma \cup \{m\}$. This and the irreflexivity of justification implies the goal.

Next, we prove the $\impliedby$ direction. There exists $n_1, n_2 \in \mathbf{N}$ s.t. $\sigma \in \Sigma^{n_1} \wedge m \in M^{n_2}$. When $n_1 = 0$, we have $\sigma = \emptyset$ by the definition of Σ^0. From this and $\mathrm{Justification}(m) \subseteq \sigma$, we have $\mathrm{Justification}(m) = \emptyset$ and hence $\mathrm{Justification}(m) = \Sigma^0$. From this and $m \in M$, we have $m \in M^0$. Hence, $\{m\} \in \mathcal{P}_{\mathrm{finite}}(M^0)$. From this, $\sigma \cup \{m\} = \{m\}$ and $\mathrm{Justification}(m) \subseteq \sigma$, we can show $\sigma \cup \{m\} \in \Sigma^1$ and hence $\sigma \cup \{m\} \in \Sigma$.

When $n_1 \geq 1$, because Σ^n and M^n are *monotonic*[9] i.e., $\Sigma^n \subseteq \Sigma^{n+1} \wedge M^n \subseteq M^{n+1}$, there exists $n' \in \mathbf{N}$ s.t. $n' \geq 1 \wedge n' \geq n_1 \wedge n' \geq n_2 \wedge \sigma \in \Sigma^{n'} \wedge m \in M^{n'}$. By the definition of Σ^n, we have $\sigma \in \mathcal{P}_{\mathrm{finite}}(M^{n'-1})$. From this and $M^{n'-1} \subseteq M^{n'}$, we have $\sigma \in \mathcal{P}_{\mathrm{finite}}(M^{n'})$. From this and $m \in M^{n'}$, we have $\sigma \cup \{m\} \in \mathcal{P}_{\mathrm{finite}}(M^{n'})$. Also, by *Lemma 1* and $\sigma \in \Sigma$, we have $\forall m \in \sigma.\ \mathrm{Justification}(m) \subseteq \sigma$ and hence $\forall m \in \sigma \cup \{m\}.\ \mathrm{Justification}(m) \subseteq \sigma \cup \{m\}$. From this and $\sigma \cup \{m\} \in \mathcal{P}_{\mathrm{finite}}(M^{n'})$, $\sigma \cup \{m\} \in \Sigma^{n'+1}$ is derived. Therefore, we can show $\sigma \cup \{m\} \in \Sigma$ by the definition of Σ. ■

[9]This is proved by induction. We omit the proof of it.

Lemma 8: About a strict subset σ' of a state σ, the difference has a message m such that $\mathrm{Justification}(m) \subseteq \sigma'$. (Note that σ' is a set of message but not necessarily a state.)

$$\begin{aligned} &\forall \sigma \in \Sigma.\ \forall \sigma'.\ \sigma' \subset \sigma \\ &\implies \exists m \in \sigma - \sigma'.\ \mathrm{Justification}(m) \subseteq \sigma' \end{aligned} \quad (29)$$

Proof: By *Lemma 1*, we have $\mathrm{Justification}(m) \subseteq \sigma$. We assume $\forall m \in \sigma - \sigma'.\ \mathrm{Justification}(m) \nsubseteq \sigma'$ towards contradiction. From this, $\sigma' \subset \sigma$ and $\mathrm{Justification}(m) \subseteq \sigma$, we have $\forall m \in \sigma - \sigma'.\ \exists m' \in \mathrm{Justification}(m).\ m' \in \sigma - \sigma'$. This implies $\forall m \in \sigma - \sigma'.\ \exists m' \in \sigma - \sigma'.\ m' \prec m$. However, this leads contradiction because $\sigma - \sigma'$ is a subset of M and hence we have $\prec$ have a minimal element on $\sigma - \sigma'$ by *Lemma 3*. ■

Lemma 9: The union of a finite set of states is a state.

Proof: First, we prove a lemma that the union of *two* states is a state.

$$\forall \sigma_1,\ \sigma_2 \in \Sigma.\ \sigma_1 \cup \sigma_2 \in \Sigma \quad (30)$$

When $\sigma_1 \subseteq \sigma_2$, this is obvious because $\sigma_1 \cup \sigma_2 = \sigma_2$. We prove the case when $\sigma_1 \nsubseteq \sigma_2$. Using *Lemma 7* and *Lemma 8*, we can derive as follows.

$$\begin{aligned} &\forall \sigma, \sigma' \in \Sigma.\ \sigma \nsubseteq \sigma' \\ &\implies \sigma \cap \sigma' \subset \sigma \\ &\implies \exists m \in \sigma - \sigma \cap \sigma'.\ \mathrm{Justification}(m) \subseteq \sigma \cap \sigma' \\ &\implies \exists m \in \sigma - \sigma'.\ \mathrm{Justification}(m) \subseteq \sigma' \\ &\implies \exists m \in \sigma - \sigma'.\ \sigma' \cup \{m\} \in \Sigma \end{aligned} \quad (31)$$

This means that when we have a pair of states σ and σ' such that $\sigma' \nsubseteq \sigma$, we can pick up a message m from the difference $\sigma - \sigma'$ and get a new pair of states σ and $\sigma' \cup \{m\}$ such that $\sigma' \cup \{m\} \nsubseteq \sigma$. By repeating this n times where n is the number of messages in the $\sigma - \sigma'$, we eventually get a state $\sigma' \cup \{\sigma - \sigma'\} = \sigma \cup \sigma'$. This complete the proof of the case when $\sigma_1 \nsubseteq \sigma_2$.

Now we proved the lemma that the union of two states is always a state. About a finite set of states, we can prove its union is a state by using this lemma repeatedly. ■

Lemma 9 also implies that there is no *race condition* i.e. when $\sigma \cup \{m_1\} \in \Sigma$ and $\sigma \cup \{m_2\} \in \Sigma$, the order of receiving m_1 and m_2 does not matter because $\sigma \cup \{m_1, m_2\} \in \Sigma$.

C. Consensus Safety

In this subsection, we provide the proof sketch of *Theorem 1* and introduce blockchain-specific safety.

Proof: We prove *Theorem 1*. We use two lemmas regarding decisions[10]. First, *forward consistency*:

$$\begin{aligned} &\forall \sigma \in \Sigma_t.\ \forall \sigma' \in \mathrm{Futures}_t(\sigma).\ \forall p.\ p \in \mathrm{Decisions}(\sigma) \\ &\implies p \in \mathrm{Decisions}(\sigma') \end{aligned} \quad (32)$$

[10]These are extensions of *Lemma 2* and *Lemma 3* of [1] for properties of consensus values.

Second, *backward consistency*:

$$\forall \sigma \in \Sigma_t.\ \forall \sigma' \in \mathrm{Futures}_t(\sigma).\ \forall p.\ p \in \mathrm{Decisions}(\sigma') \implies \neg\ p \notin \mathrm{Decisions}(\sigma) \tag{33}$$

By *Lemma 9*, we have $\bigcup_{i=1}^{n} \sigma_i \in \Sigma$. From this and the assumption $F(\bigcup_{i=1}^{n} \sigma_i) \leq t$, we have $\bigcup_{i=1}^{n} \sigma_i \in \Sigma_t$. Let σ' be $\bigcup_{i=1}^{n} \sigma_i$. Because $\forall i \in \mathbf{N}.\ i \leq n \implies \sigma_i \subseteq \sigma'$, we have $\sigma' \in \mathrm{Futures}_t(\sigma_i)$. (i.e. σ' is a *common future* of the validators). By forward consistency, $\forall p \in \bigcup_{i=1}^{n} \mathrm{Decisions}(\sigma_i).\ p \in \mathrm{Decisions}(\sigma')$. Therefore, by backward consistency, $\neg\ p \notin \bigcup_{i=1}^{n} \mathrm{Decisions}(\sigma_i)$. ∎

Next, we explain this abstract safety which depends neither on parameters nor properties and implies the safety in blockchain, i.e. validators do not decide on conflicting blocks.

Definition 16 (Block conflict): Two blocks are *conflicting* if and only if neither block is the ancestor of the other block.

$$\mathrm{Conflicting}(b_1, b_2) := \neg\ (b_1 \lfloor b_2 \vee b_1 \lfloor b_2) \tag{34}$$

Then, because we consider blockchain consensus protocols, we restrict $\mathcal{C}$ to satisfy this condition, i.e. ancestor blocks do not conflict.

$$\forall b,\ b',\ b'' \in \mathcal{C}.\ b \lfloor b' \wedge b'' \lfloor b' \implies b \lfloor b'' \vee b'' \lfloor b \tag{35}$$

Finally, we introduce blockchain safety defined as:

Theorem 3 (No decision on conflicting blocks): If a validator at a state σ_1 considers the block b_1 finalised, a validator at a state σ_2 does not considers a block b_2 finalised if b_1 and b_2 are conflicting when there are t equivocations or less at the union of their state.

$$\begin{aligned} \forall b_1, b_2 \in \mathcal{C}.\ \forall \sigma_1, \sigma_2\ \in \Sigma_t.\ & \\ F(\sigma_1 \cup \sigma_2) \leq t \wedge \mathrm{Conflicting}(b_1, b_2) & \\ \implies \mathrm{P}_{\mathrm{block}}(b_1) \in \mathrm{Decisions}(\sigma_1) & \\ \implies \mathrm{P}_{\mathrm{block}}(b_2) \notin \mathrm{Decisions}(\sigma_2) & \end{aligned} \tag{36}$$

Proof: Using the $n = 2$ case of safety *Theorem 1*, $\neg\ \mathrm{P}_{\mathrm{block}}(b_1) \notin \mathrm{Decisions}(\sigma_2)$ is derived. By this and the definition of Decisions, we have $\exists \sigma' \in \mathrm{Futures}_t(\sigma_2).\ \forall b \in \mathcal{E}(\sigma').\ b_1 \lfloor b$. Assume $\mathrm{P}_{\mathrm{block}}(b_2) \in \mathrm{Decisions}(\sigma_2)$ towards contradiction. Unfolding the definition of Decisions, we have $\forall \sigma' \in \mathrm{Futures}_t(\sigma_2).\ \forall b \in \mathcal{E}(\sigma').\ b_2 \lfloor b$. From this and $\exists \sigma' \in \mathrm{Futures}_t(\sigma_2).\ \forall b \in \mathcal{E}(\sigma').\ b_1 \lfloor b$, we have $\exists \sigma' \in \mathrm{Futures}_t(\sigma_2).\ \forall b \in \mathcal{E}(\sigma').\ b_1 \lfloor b \wedge b_2 \lfloor b$. From this and the assumption that ancestor blocks do not conflict, we have $b_1 \lfloor b_2 \vee b_2 \lfloor b_1$. This makes contradiction to the assumption $\mathrm{Conflicting}(b_1, b_2)$. ∎

V. Formal verification of CBC Casper

We introduce our work on formal verification of CBC Casper[11]. Isabelle/HOL is an interactive proof assistant that utilises functional programming and higher-order logic [23].

[11]Names of theorems and definitions in Isabelle/HOL are shortened from the public repository.

A. Protocol definition

We instantiate CBC Casper with the parameters listed in Section II-B using Isabelle's *locale* feature [24] to parameterize theories as displayed in Figure 4.

```
locale Protocol =
  fixes V :: validator set
  and W :: validator ⇒ real
  and t :: real
  and C :: consensus-value set
  and ε :: message set ⇒ consensus-value set
  assumes V-type: V ≠ ∅ ∧ finite V
  and W-type: ∀ v ∈ V. W v > 0
  and t-type: 0 ≤ t t < sum W V
  and C-type: card C > 1
  and ε-type: ∀ σ ∈ Σ. ε σ ∈ Pow C − {∅}
```

Fig. 4. Excerpt of datatypes and locale definition of CBC Casper in Isabelle/HOL.

B. Types of functions

Every function in CBC Casper is defined with their types, using sets such as validators $\mathcal{V}$, consensus values $\mathcal{C}$, states Σ and messages M. As Isabelle does not support dependent types, we define them as *set* types and prove that the every function we define matches with its original types[12]. We show as an example the Observed function defined in [1] as follows:

$$\mathrm{Observed}:\ \mathcal{P}(M) \to \mathcal{P}(\mathcal{V}) \tag{37}$$

$$\mathrm{Observed}(\sigma) := \{\mathrm{Sender}(m) : m \in \sigma\} \tag{38}$$

Our definition and a lemma about the type of this function in Isabelle/HOL are shown in Figure 5. Although the type validator set does not necessary force the function *observed* to return a set of validators in $\mathcal{V}$, the lemma *observed-type* proves the type of the output is $\mathcal{P}(\mathcal{V})$ if the type of the input is $\mathcal{P}(M)$. We proved lemmas like this for all functions to make sure that their types are correct.

```
definition observed :: message set ⇒ validator set
  where
    observed σ = {sender m | m. m ∈ σ}

lemma (in Protocol) observed-type :
  ∀ σ ∈ Pow M. observed σ ∈ Pow V
```

Fig. 5. Definition of *observed* and a lemma of it.

C. Asynchronous safety

We prove *Theorem 1* and *Theorem 3* in Isabelle/HOL. The theorem displayed in Figure 6 is the definition of *Theorem 1* in Isabelle/HOL.

σ-set represents the set of the validators' states. *finite σ-set* means that the number of the validators is finite. *faults-lt-threshold*($\bigcup$ *σ-set*) states that in the union of the state of validators there are not more than t equivocation faults. *decisions* represents *Definition 11*.

[12]The types *validator*, *consensus_value*, *message* and *state* in Figure 4 are *supertypes* of them.

```
theorem (in Protocol) n-party-consensus-safety :
 ∀ σ-set. σ-set ⊆ Σt
 ⟶ finite σ-set
 ⟶ faults-lt-threshold (⋃ σ-set)
 ⟶ (∀ p ∈ ⋃ {decisions σ' | σ'. σ' ∈ σ-set}.
       (λc. (¬ p c)) ∉ ⋃ {decisions σ' | σ'. σ' ∈ σ-set})
```

Fig. 6. Theorem n-party-consensus-safety in Isabelle/HOL.

The theorem displayed in Figure 7 is the definition of *Theorem 3* in Isabelle/HOL. *conflicting* and *membership* represents *Definition 16* and *Definition 10* respectively.

```
theorem (in Blockchain) no-decision-on-conflicting-blocks :
 ∀ σ1 σ2. {σ1, σ2} ⊆ Σt
 ⟶ faults-lt-threshold (σ1 ∪ σ2)
 ⟶ (∀ b1 b2. {b1, b2} ⊆ C ∧ conflicting (b1, b2)
    ⟶ membership b1 ∈ decisions σ1
    ⟶ membership b2 ∉ decisions σ2)
```

Fig. 7. Theorem no-decision-on-conflicting-blocks in Isabelle/HOL.

D. Properties of message justification and state transition

We prove the properties of message justifications and state transitions described in Section IV using theories of *HOL Session* [25] and *Restricted_Predicates* in AFP [26]. With the message justification, we prove the below lemma from [1].

Lemma 10: Non-equivocating validators observed in a state σ have a single latest message in σ.

Here a validator is called *observed* in a state if and only if there exists a message from the validator in the state. This lemma is important for clique orcle because the proof of its correctness depends on it [15] and also for LMD GHOST. However, as it is already pointed out [27], the proof in [1] based on the relation of comparison of the size of justifications of message is not correct.

We formally verify this lemma in Isabelle/HOL as follows. First, we prove that a latest message of a validator v in a state σ is a *maximal element* about $\prec$ on $\mathrm{From\_sender}(v, \sigma)$. Next, as an independent contribution, we prove that a linear order on non-empty finite set has one maximum by proving that:

- A strict partial order on non-empty finite set has at least one maximal element.
- A strict partial order has at most one maximum.
- A maximal element is a maximum in linearly ordered set.

From this theorem, we prove *Lemma 10* because $\prec$ is a strict linear order on $\mathrm{From\_sender}(v, \sigma)$ if v is an non-equivocating validator as we described in IV-A.

E. Casper the Friendly GHOST

Casper TFG is an instance of CBC Casper as introduced in Section III-D. In Casper TFG, consensus values are blocks and the estimator is based on LMD GHOST. We construct Casper TFG in Isabelle/HOL, modifying the construction in the CBC Casper paper. First, we defined LMD GHOST as *GHOST* function.

Next, we defined the estimator of Casper TFG. The original estimator function [1] only allows blocks in $\mathrm{GHOST}(\{g\}, \sigma)$.

```
function (in Ghost) GHOST :: (block set * state) => block set
 where
  GHOST (b-set, σ) =
    (⋃ b ∈ {b ∈ b-set. children (b, σ) ≠ ∅}.
GHOST (best-children (b, σ), σ))
     ∪ {b ∈ b-set. children (b, σ) = ∅}
```

Fig. 8. GHOST in Isabelle/HOL

This results in a construct where no other block than the genesis block g can be proposed because $\mathrm{GHOST}(\{g\}, \sigma)$ returns blocks which is appeared in σ. Therefore, we modify the definition to include not only the result of LMD GHOST but also the child blocks of them.

$$\mathcal{E}(\sigma) = \mathrm{GHOST}(\{g\}, \sigma) \cup \bigcup_{b \in \mathrm{GHOST}(\{g\}, \sigma)} \mathrm{Children}(b, \sigma) \tag{39}$$

This estimator can be used in "block-and-vote" protocol, which has two types of messages i.e. messages which propose a new block and messages which *vote* for a proposed block.

F. Improving the CBC Casper paper

We found several issues in [1] through the formal verification that are reported to the authors. We list these in Appendix E.

VI. Related work

We compare our verification efforts to related work. *Toychain* is a formalisation of a consensus protocol in the Coq proof assistant [28]. Toychain provides an explicit data structure, i.e. block forest, and parameters to instantiate the protocol. It offers a small-step semantics for message passing and is proven to reach consensus after quiescent network under an assumption that every nodes know each other. In our CBC Casper proofs, the data structure of the consensus values are abstracted. Hence, our model can be applied to blockchains, DAGs, or other data constructs. Further, Toychain proves eventual consistency without considering Byzantine faults and leaves liveness properties as future work.

Safety of a simplified version of *CBC Casper* is verified in Isabelle/HOL by Hirai [29]. It covers a binary consensus where non-equivocating two-party validators, "max weight" estimator and "tie braking" weights are allowed along with simpler definitions of messages and states.

Casper FFG is initially verified in Isabelle/HOL by Hirai [30]. This work is extended by verifying Casper FFG in the Coq proof assistant [31]. Notably, this work builds on Toychain. The proofs cover the two properties accountable safety and plausible liveness with a static finite set of validators. Both proofs rely on the assumption that at least $2/3$ of validators behave honestly.

The *Nakamoto* consensus protocol is subject to extensive analysis by Garay et al. [32] and Pass et al. [33] using a synchronous and semi-synchronous setting respectively. They define three general properties of blockchain consensus protocols: common prefix, chain growth, and chain quality. This

work is extended by using the UC model [34]. The UC model for Bitcoin includes an adversary that can selectively insert or delay messages as well as adaptive corruption of validators. Also, the ledger functionality is split into a lottery aspect and the remainder of the protocol. A UC treatment or other simulation-based proof of CBC Casper including exact definitions of ideal functionalities, protocols, and a simulator is a promising direction for future work.

The family of *Ouroboros* consensus protocol is introduced in [35]. Further, Ouroboros is extended for a semi-synchronous network with Praos [36]. Unlike CBC Casper, Ouroboros and Praos are probabilistic consensus protocol. From the initial simulation-based analysis, Ouroboros moved to a UC model with Genesis [11]. This work introduces ideal functionalities, protocols, and a simulator that can be applied to other PoS protocols, like CBC Casper, as well. Moreover, there is a repository including formal verification of Ouroboros Praos [37] in Isabelle/HOL and an implementation in Haskell.

Algorand [38] is also proposals for PoS consensus protocols. The protocol includes an assumption that $2/3$ of the weighted participants are honest. This work is relevant for its approach to validator election as this is not covered in CBC Casper.

HotStuff is a consensus protocol using a semi-synchronous model with a leader requiring linear communication [39]. It contributes a property termed optimistic responsiveness where a leader needs to wait only for a fraction of responses from other nodes (the total nodes minus the faulty ones) to guarantee progress. Further, HotStuff offers a framework to analyse other protocols under which a version of Casper FFG is discussed.

Other work extends consensus protocols by introducing DAGs with an adoption of existing fork-choice rules, e.g. *SPECTRE* [40], *PHANTOM* [41]. The work on CBC Casper is orthogonal to these proposals as the fork-choice rule as well as the data structure are parameters in CBC Casper.

Apart from existing protocols, related work is also concerned with attacks specific to PoS protocols. Nothing at stake attacks describes the notion that it is cost-free to produce alternative versions of a chain of blocks (or other consensus data structure) [42]. Long-range attacks rewrite blockchain history and are related to the fact that creating an alternative chain is practically cost-free. Long-range attacks also include the idea of "posterior corruption" [43]. Stake bleeding attacks are a form of long-range attacks where a coalition of attackers can rewrite the blockchain history by including transactions from the current agreed-upon chain into an attacking chain [44]. The idea is to include all transactions of the coalition into blocks that are created by the coalition and thereby increasing its relative stake. This attack works on PoS blockchains without check-pointing. Known leader schedules describe attacks where specific validators are targeted, since (parts of) the future validator leaders are public [42].

VII. Conclusion

We present a summary of CBC Casper and contribute refinements to the CBC Casper protocol family. Our refinements include a new definition of asynchronous safety for the CBC Casper framework. Further, we define and prove blockchain safety and discuss liveness for a protocol instance called Casper TFG, which builds on a variant of GHOST. Next, we revise properties about messages and state transition and propose a revised clique oracle. We show that validators can reach decisions with a clique oracle where $2/3$ of validators by weight are honest. Our work is verified using Isabelle/HOL. We are currently working on a complete specification of Casper TFG including validator rotation and a more efficient construction to verify the DAG of messages. Future work also includes revising our liveness proofs with a formal definition of an adversary as well as verifying the safety oracle in Isabelle/HOL.

Acknowledgment

We would like to thank Yusuke Ishibashi for supporting us with our questions concerning proofs in Isabelle. Further, we like to thank Bernardo David for comments on an earlier version of this paper.

References

[1] V. Zamfir, N. Rush, A. Asgaonkar, and G. Piliouras, "Introducing the "Minimal CBC Casper" Family of Consensus Protocols," 2018.

[2] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," vol. 4, no. 3. ACM, 1982, pp. 382–401. [Online]. Available: http://people.cs.uchicago.edu/~shanlu/teaching/33100_wi15/papers/byz.pdf

[3] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process," vol. 32, no. 2. ACM, 1985, pp. 374–382. [Online]. Available: http://macs.citadel.edu/rudolphg/csci604/ImpossibilityofConsensus.pdf

[4] S. Gilbert and N. Lynch, "Perspectives on the CAP Theorem," *Computer*, vol. 45, no. 2, pp. 30–36, 2 2012. [Online]. Available: http://ieeexplore.ieee.org/document/6122006/

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf, Dec 2008, accessed: 2015-07-01. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[6] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *Financial Cryptography and Data Security*. Springer, 2014, pp. 436–454. [Online]. Available: http://arxiv.org/pdf/1311.0243

[7] K. Nayak, S. Kumar, A. Miller, and E. Shi, "Stubborn mining: Generalizing selfish mining and combining with an eclipse attack," in *1st IEEE European Symposium on Security and Privacy, 2016*. IEEE, 2016. [Online]. Available: http://eprint.iacr.org/2015/796.pdf

[8] L. Luu, J. Teutsch, R. Kulkarni, and P. Saxena, "Demystifying incentives in the consensus computer," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2015, pp. 706–719. [Online]. Available: http://www.comp.nus.edu.sg/~prateeks/papers/VeriEther.pdf

[9] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. Gün, "On scaling decentralized blockchains," in *3rd Workshop on Bitcoin and Blockchain Research, Financial Cryptography 16*, 2016. [Online]. Available: http://www.tik.ee.ethz.ch/file/74bc987e6ab4a8478c04950616612f69/main.pdf

[10] I. Bentov, A. Gabizon, and A. Mizrahi, "Cryptocurrencies without proof of work," http://arxiv.org/pdf/1406.5694.pdf, 2014, accessed: 2016-03-09. [Online]. Available: http://arxiv.org/pdf/1406.5694.pdf

[11] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros Genesis: Composable Proof-of-Stake Blockchains with Dynamic Availability," *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security - CCS '18*, no. 780477, pp. 913–930, 2018.

[12] V. Buterin and V. Griffith, "Casper the friendly finality gadget," arXiv:1710.09437, 2017, accessed:2017-11-06. [Online]. Available: https://arxiv.org/pdf/1710.09437.pdf

[13] Ethereum, "Ethereum 2.0 Specifications," 2019. [Online]. Available: https://github.com/ethereum/eth2.0-specs
[14] V. Zamfir, "Casper the friendly ghost: A correct by construction blockchain consensus protocol," https://github.com/ethereum/research/blob/master/papers/CasperTFG/CasperTFG.pdf, accessed 2018-06-27.
[15] N. Rush and R. Nakamura, "CBC Casper Safety Oracle," 2019. [Online]. Available: https://github.com/cbc-casper/cbc-casper-paper/pull/13
[16] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527. [Online]. Available: http://www.cs.huji.ac.il/~avivz/pubs/15/btc_ghost_full.pdf
[17] V. Zamfir, "A Template for Correct-by-Construction Consensus Protocols," 2017. [Online]. Available: https://github.com/ethereum/research/blob/master/papers/cbc-consensus/AbstractCBC.pdf
[18] Ethereum, "Correct-by-construction Casper Wiki," 2019. [Online]. Available: https://github.com/ethereum/cbc-casper/wiki
[19] M. Ali, J. Nelson, and A. Blankstein, "Peer Review: CBC Casper," 2018. [Online]. Available: https://medium.com/@muneeb/peer-review-cbc-casper-30840a98c89a
[20] A. Clement, F. Junqueira, A. Kate, and R. Rodrigues, "On the (limited) power of non-equivocation," in *Proceedings of the 2012 ACM symposium on Principles of distributed computing - PODC '12*. New York, New York, USA: ACM Press, 2012, p. 301. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2332432.2332490
[21] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," http://atrium.lib.uoguelph.ca/xmlui/bitstream/handle/10214/9769/Buchman_Ethan_201606_MAsc.pdf, Jun 2016, accessed: 2017-02-06.
[22] R. Nakamura, "Validator rotation in CBC Casper," 2019. [Online]. Available: https://ethresear.ch/t/validator-rotation-in-cbc-casper/5200
[23] T. Nipkow and K. Gerwin, *Concrete Semantics: with Isabelle/HOL*. Springer Verlag, 2017. [Online]. Available: http://www.concrete-semantics.org/concrete-semantics.pdf
[24] C. Ballarin, "Tutorial to Locales and Locale Interpretation," 2012. [Online]. Available: https://isabelle.in.tum.de/doc/locales.pdf
[25] Isabelle, "Session HOL," 2019. [Online]. Available: https://isabelle.in.tum.de/dist/library/HOL/HOL/index.html
[26] M. Ogawa and C. Sternagel, "Theory Restricted_Predicates," 2018. [Online]. Available: https://www.isa-afp.org/browser_info/current/AFP/Open_Induction/Restricted_Predicates.html
[27] A. Fackler, "CBC Casper Lemma 6 and 9," 2018. [Online]. Available: https://github.com/cbc-casper/cbc-casper-paper/issues/14
[28] G. Pîrlea and I. Sergey, "Mechanising blockchain consensus," *Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs - CPP 2018*, pp. 78–90, 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3176245.3167086
[29] Y. Hirai, "Formal methods on another Casper," 2017. [Online]. Available: https://medium.com/@pirapira/formal-methods-on-another-casper-8a75f6e02073
[30] ——, "A repository for PoS related formal methods," 2018. [Online]. Available: https://github.com/palmskog/pos
[31] K. Palmskog, M. Gligoric, L. Pena, B. Moore, and G. Rosu, "Verification of Casper in the Coq Proof Assistant," Tech. Rep., 2018.
[32] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol with chains of variable difficulty," http://eprint.iacr.org/2016/1048.pdf, 2016, accessed: 2017-02-06. [Online]. Available: http://eprint.iacr.org/2016/1048.pdf
[33] R. Pass, L. Seeman, and a. shelat, "Analysis of the blockchain protocol in asynchronous networks," http://eprint.iacr.org/2016/454.pdf, 2016, accessed: 2016-08-01. [Online]. Available: http://eprint.iacr.org/2016/454.pdf
[34] C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas, "Bitcoin as a transaction ledger: A composable treatment," Cryptology ePrint Archive, Report 2017/149, 2017, accessed:2017-09-26. [Online]. Available: https://eprint.iacr.org/2017/149.pdf
[35] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," Cryptology ePrint Archive, Report 2016/889, 2016, https://eprint.iacr.org/2016/889.
[36] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake protocol," Cryptology ePrint Archive, Report 2017/573, 2017, accessed: 2017-06-29. [Online]. Available: http://eprint.iacr.org/2017/573.pdf
[37] W. Jeltsch, L. Brünjes, J. Diaz, E. d. Vries, D. Coutts, and A. Löh, "High-assurance implementation of (a variant of) the Ouroboros protocol," 2019. [Online]. Available: https://github.com/input-output-hk/fm-ouroboros
[38] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," Cryptology ePrint Archive, Report 2017/454, 2017, accessed: 2017-06-29. [Online]. Available: http://eprint.iacr.org/2017/454.pdf
[39] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT Consensus in the Lens of Blockchain," 2018. [Online]. Available: http://arxiv.org/abs/1803.05069
[40] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "Spectre: A fast and scalable cryptocurrency protocol," Cryptology ePrint Archive, Report 2016/1159, 2016, accessed: 2017-02-20. [Online]. Available: http://eprint.iacr.org/2016/1159.pdf
[41] Yonatan Sompolinsky and Aviv Zohar, "Phantom: A scalable blockdag protocol," Cryptology ePrint Archive, Report 2018/104, 2018, accessed:2018-01-31. [Online]. Available: https://eprint.iacr.org/2018/104.pdf
[42] E. Blum, A. Kiayias, C. Moore, S. Quader, and A. Russell, "Linear Consistency for Proof-of-Stake Blockchains," 2018. [Online]. Available: https://eprint.iacr.org/2017/241
[43] I. Bentov, R. Pass, and E. Shi, "Snow white: Provably secure proofs of stake," https://eprint.iacr.org/2016/919.pdf, 2016, accessed: 2016-11-08. [Online]. Available: https://eprint.iacr.org/2016/919.pdf
[44] P. Gaži, A. Kiayias, and A. Russell, "Stake-Bleeding Attacks on Proof-of-Stake Blockchains," *2018 Crypto Valley Conference on Blockchain Technology (CVCBT)*, pp. 85–92, 2018.

APPENDIX A
DEFINITION OF SAFETY

Theorem 1 is defined differently from [1][13]. The original definition of safety[14] refers to "all properties hold at the same time" instead of "the negation of any property is not decided by another validator" like ours. Specifically, it is defined as the existence of *any* consensus value c in $\mathcal{C}$ that satisfies all properties in the union of all validators' decisions. We claim this is different from general definitions of safety in a consensus protocol because there is no limitation on the consensus value c i.e. c is allowed even if no validators can actually have c as the estimate in the current state or future states.

The motivation to define safety as "all properties hold at the same time" is to remove a tricky case from the safety. That is, the authors consider not only properties decided by a single validator but also properties decided *across* multiple validators. For example, four validators might decide on p_1, p_2, p_3, and p_4 respectively where $p_1 \wedge p_2 \implies \neg\,(p_3 \wedge p_4)$ but none of them decide on $\neg\, p_i$ $(1 \leq i \leq 4)$. This case satisfies our definition of safety, but might lead to problems in reaching consensus.

Although this issue is worth considering, we omit it as it does not affect the safety of blockchain consensus protocols as elaborated in in Section IV-C. From *Theorem 3*, we can derive that when two validators decide on p_1 and p_2 such that $p_1 = \mathrm{P}_{\mathrm{block}}(\mathrm{b}_1)$ and $p_2 = \mathrm{P}_{\mathrm{block}}(\mathrm{b}_2)$ respectively, b_1 and b_2 are not conflicting i.e. either block is the ancestor of the other block. This means that $p_1 \implies p_1 \wedge p_2$ or $p_2 \implies p_1 \wedge p_2$ i.e. a decision across multiple validators is a decision of a single validator.

[13] In $n = 2$ case, our definition is essentially the same with "Theorem 3 Two-party consensus safety" of [1], which is defined for properties of *states* unlike our definition defined for properties of *consensus values*.

[14] "Theorem 5 n-party consensus safety for properties of the consensus".

APPENDIX B
CORRECTNESS OF CLIQUE ORACLE

In this section, we prove *Theorem 2* extending the in-progress proof and reusing lemmas in [15]. First, we introduce the formal definition of a max driven property, modifying the original definition so that it can be applicable to a block membership property (see also Appendix D).

Definition 17 (Max driven property):

$$\begin{aligned}\mathrm{Max\_Driven}(p,\sigma) :\Leftrightarrow \forall \sigma \in \Sigma.\ \sigma \rightarrow \sigma' \\ W(\mathrm{Agreeing}(p,\sigma')) > W(\mathrm{Disagreeing}(p,\sigma')) \\ \implies \forall c \in \mathcal{E}(\sigma').\ p(c)\end{aligned} \tag{40}$$

Here $\mathrm{Agreeing}(p,\sigma)$ is defined as a set of non-equivocating validators observed in σ whose latest estimate satisfies p and $\mathrm{Disagreeing}(p,\sigma)$ is defined as $\mathcal{V} \setminus \mathrm{Agreeing}(p,\sigma) \setminus E(\sigma)$ i.e. a set of non-equivocating validators who are not observed in σ or else have the latest estimate in σ that does not satisfy p [15]. Note that for all p, $\mathrm{Agreeing}(p,\sigma)$, $\mathrm{Disagreeing}(p,\sigma)$ and $E(\sigma)$ are subsets of $\mathcal{V}$ and mutually disjoint.

Then, we show the validity of the threshold.

Lemma 11: If there is a clique for a property p larger than $T(\sigma)$ and $\mathrm{Max\_Driven}(p,\sigma)$, p holds at σ if $\sigma \in \Sigma_t$.

Proof: From the *Lemma 37* in [15], the clique is a subset of $\mathrm{Agreeing}(p,\sigma)$. Therefore, we have $\mathrm{Agreeing}(p,\sigma) > T(\sigma)$. From this and $t - F(\sigma) > 0$, we have:

$$\begin{aligned}&W(\mathrm{Agreeing}(p,\sigma)) > W(\mathcal{V})/2 + t/2 - F(\sigma) \\ &\quad\Longleftrightarrow W(\mathrm{Agreeing}(p,\sigma)) > \\ &\qquad \{W(\mathcal{V}) - F(\sigma)\}/2 + \{t - F(\sigma)\}/2 \\ &\quad\implies W(\mathrm{Agreeing}(p,\sigma)) > \{W(\mathcal{V}) - F(\sigma)\}/2 \\ &\quad\Longleftrightarrow W(\mathrm{Agreeing}(p,\sigma))/2 > \\ &\qquad \{W(\mathcal{V}) - W(\mathrm{Agreeing}(p,\sigma)) - F(\sigma)\}/2 \\ &\quad\Longleftrightarrow W(\mathrm{Agreeing}(p,\sigma)) > \\ &\qquad W(\mathcal{V} \setminus \mathrm{Agreeing}(p,\sigma) \setminus E(\sigma)) \\ &\quad\Longleftrightarrow W(\mathrm{Agreeing}(p,\sigma)) > W(\mathrm{Disagreeing}(p,\sigma))\end{aligned} \tag{41}$$

Hence, by the definition of a max driven property, we have $\forall c \in \mathcal{E}(\sigma).\ p(c)$. ■

Next, we prove that a clique larger than the threshold exists for any future state where there are t equivocations or less.

Lemma 12: If there is a clique for a max driven property p larger than $T(\sigma)$, there always exists a clique for p in $\sigma' \in \mathrm{Futures}_t(\sigma)$.

Proof: First, we prove that a clique larger than the threshold in σ_a exists over a *minimum state transition*, i.e. a state transition made by receiving a single message m if there are t equivocations or less in the next state $\sigma_a \cup \{m\}$. We show this by the case analysis about m. Let v be $\mathrm{Sender}(m)$ and σ_b be $\sigma_a \cup \{m\}$. (a) If v is not a member of the clique, the clique exists in the next state from the *Lemma 16* in [15]. (b) If v is a member of the clique and m does not show the equivocation of v, the clique exists in the next state from the *Lemma 31* in [15]. (c) If v is a member of the clique and m shows the equivocation of v, v is removed by the clique and the weight of the clique decreases by $\mathcal{W}(v)$. About the difference of the threshold, we have:

$$\begin{aligned}T(\sigma_b) - T(\sigma_a) &= \{W(\mathcal{V})/2 + t/2 - F(\sigma_b)\} \\ &\quad - \{W(\mathcal{V})/2 + t/2 - F(\sigma_a)\} \\ &= -F(\sigma_b) + F(\sigma_a) \\ &= -\mathcal{W}(v)\end{aligned} \tag{42}$$

This means that the threshold also decreases by $\mathcal{W}(v)$. Hence, the clique is larger than the threshold in the next state even though v is removed from the clique. From (a), (b) and (c), we can show that a clique larger than the threshold exists over a minimum state transition if there are t equivocations or less in the next state.

From this result, we can prove the target lemma. When $\sigma = \sigma'$, the lemma obviously holds. When $\sigma \subset \sigma'$, from *Lemma 7* and *Lemma 8*, we have $\exists m \in \sigma' - \sigma.\ \sigma \cup \{m\} \in \Sigma$. From this, we can derive $\exists m \in \sigma' - \sigma.\ \sigma \cup \{m\} \in \Sigma_t$ because $\sigma' \in \Sigma_t$ and hence $\forall \sigma'' \in \Sigma.\ \sigma'' \rightarrow \sigma' \implies \sigma'' \in \Sigma_t$ [16]. By using this repeatedly, we have a set of minimum state transitions from σ to σ' and in all the intermediate states, there are t equivocations or less. Because a clique larger than the threshold exists over a minimum state transition if there are t equivocations or less in the next state, we can inductively show that a clique larger than the threshold exits in σ'. ■

Finally, we prove *Theorem 2*.

Proof: If there is a clique for a property p larger than $T(\sigma)$ and $\mathrm{Max\_Driven}(p,\sigma)$ holds, there always exists a clique for p in $\sigma' \in \mathrm{Futures}_t(\sigma)$ because of *Lemma 12*. Also, $\mathrm{Max\_Driven}(p,\sigma')$ holds by the definition of Max_Driven. From this and from *Lemma 11*, p holds at σ'. ■

APPENDIX C
BYZANTINE FAULT THRESHOLD FOR CLIQUE ORACLE

We prove the lemma about the Byzantine fault threshold for clique oracle.

Lemma 13: Let the Byzantine fault threshold t', $t' < W(\mathcal{V})/3$ is a necessary and sufficient condition for the protocol not to get stuck i.e. there are enough non-faulty validators that can form a clique.

Proof: Because equivocation faults are Byzantine faults, $0 \leq t \leq t'$. The net weight of non-faulty validators is $W(\mathcal{V}) - t'$. By the clique oracle threshold,

$$\begin{aligned}&W(\mathcal{V}) - t' > W(\mathcal{V})/2 + t/2 - F(\sigma) \\ &\Longleftrightarrow -W(\mathcal{V})/2 + t/2 + t' < F(\sigma)\end{aligned} \tag{43}$$

[15] In [15], $\mathrm{Agreeing}(p,\sigma)$ is defined to include equivocating validators and validators who do not have any message in σ but we assume this is a mistake. Our modification of the definition does not break the proofs because $\mathrm{Agreeing}(p,\sigma)$ is used in the definition of clique (in this case equivocating validators and validators who do not have any message in σ are precluded) or the right-hand side of lemmas (in this case the lemma still holds since our definition of $\mathrm{Agreeing}(p,\sigma)$ is a subset of the original). Also, $\mathrm{Disagreeing}(p,\sigma)$ is defined differently but it is not used in the original proofs.

[16] This is proved by the monotonicity of $F(\sigma)$ i.e. all equivocations in a state exists in all future states of the state.

Since this hold for any $F(\sigma)$ such that $0 \leq F(\sigma) \leq t$,

$$\begin{aligned} & -W(\mathcal{V})/2 + t/2 + t' < 0 \\ \iff & t < W(\mathcal{V}) - 2t' \end{aligned} \tag{44}$$

Since this hold for any t such that $0 \leq t \leq t'$,

$$\begin{aligned} & t' < W(\mathcal{V}) - 2t' \\ \iff & t' < W(\mathcal{V})/3 \end{aligned} \tag{45}$$

■

Appendix D
Max driven block membership property in LMD GHOST

In this section, we prove that block membership properties are max driven if the estimator is based on LMD GHOST.

Lemma 14: A block membership property is max driven at a state σ i.e. $\mathrm{Max\_Driven}(\mathrm{P}_{\mathrm{block}}(b), \sigma)$ when b is observed at σ i.e. $\exists m \in \sigma.\ b = \mathrm{Estimate}(m)$ if the estimator does not allow blocks other than the head blocks decided by LMD GHOST or the children of them.

Proof: About a state σ' such that $\sigma \to \sigma'$, b is also observed at σ'. By the definition of $\mathrm{Max\_Driven}$, we assume:

$$\begin{aligned} & W(\mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b), \sigma')) > \\ & \qquad W(\mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b), \sigma')) \end{aligned} \tag{46}$$

Validators agreeing on $\mathrm{P}_{\mathrm{block}}(b)$ are also agreeing on $\mathrm{P}_{\mathrm{block}}(b')$ if b' is the ancestor of b. Hence we have:

$$\begin{aligned} & \mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b), \sigma') \subseteq \mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma') \\ & \quad \Longrightarrow W(\mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma')) > \\ & \qquad W(\mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b), \sigma')) \end{aligned} \tag{47}$$

From this and the assumption, we have:

$$\begin{aligned} & W(\mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma')) > \\ & \qquad W(\mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b), \sigma')) \end{aligned} \tag{48}$$

Also, we can derive:

$$\begin{aligned} & W(\mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma')) > \\ & \qquad W(\mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b), \sigma')) \\ & \quad \Longrightarrow W(\mathcal{V} \setminus \mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma') \setminus E(\sigma')) > \\ & \qquad W(\mathcal{V} \setminus \mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b), \sigma') \setminus E(\sigma')) \\ & \quad \Longrightarrow W(\mathcal{V}) - W(\mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma')) \\ & \qquad - F(\sigma') > W(\mathcal{V}) \\ & \qquad - W(\mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b), \sigma')) - F(\sigma') \\ & \quad \Longrightarrow W(\mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b), \sigma')) > \\ & \qquad W(\mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma')) \end{aligned} \tag{49}$$

From these, we can derive:

$$\begin{aligned} & W(\mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma')) > \\ & \qquad W(\mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b), \sigma')) \\ & \quad \Longrightarrow W(\mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma')) > \\ & \qquad W(\mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma')) \end{aligned} \tag{50}$$

In LMD GHOST, the score of b' is $W(\mathrm{Agreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma'))$ and the score of the other block at the same height is $W(\mathrm{Disagreeing}(\mathrm{P}_{\mathrm{block}}(b'), \sigma'))$ or less. From this and b' is observed in σ', b' is the best child of its parent block[17]. From this, all the ancestor blocks of b are the best child of their parent block and hence the head blocks decided by LMD GHOST are b or the descendant blocks of b. Because the estimator does not allow blocks other than the head blocks decided by LMD GHOST or the children of them, we can derive $\forall c \in \mathcal{E}(\sigma').\ \mathrm{P}_{\mathrm{block}}(b)(c)$. ■

Appendix E
Improvements of the Minimal CBC Casper paper

We list the improvements and errors we found in the Minimal CBC Casper paper [1].

1) *Subset membership of messages* The type of M is defined as the strict subset of $\mathcal{C} \times \mathcal{V} \times \Sigma$ but we proved that when the estimator is constant function which returns C for any input, $M = \mathcal{C} \times \mathcal{V} \times \Sigma$. Therefore, the type should be states as $M \subseteq \mathcal{C} \times \mathcal{V} \times \Sigma$.
2) *Types of functions* We modified the type of $\mathrm{Later\_From}$ from $M \times \mathcal{V} \times \mathcal{P}(M)$ to $M \times \mathcal{V} \times \mathcal{P}(M) \to \mathcal{P}(M)$. Also, we proposed the modification of the type of $\mathcal{E}$ from $\Sigma \to \mathcal{P}(\mathcal{C}) \setminus \emptyset$ to $\Sigma \to \mathcal{P}(\mathcal{C}) \setminus \{\emptyset\}$.
3) $\mathrm{Blocks\_In}$ *is not defined* We pointed out that the function $\mathrm{Blocks\_In}$ used in the instantiation of *Casper the Friendly CBC Finality Gadget* is not defined.
4) *Definition of epoch score function* We modified the definition of $Epoch\_Score(e, \sigma)$ by replacing an unknown variable e with b.

[17] We formally verified this in Isabelle/HOL.