Declarative distributed broadcast using three-valued modal logic and semitopologies

Murdoch J. Gabbay

Contents

Abstract

We demonstrate how to formally specify distributed algorithms as declarative axiomatic theories in a modal logic. We exhibit the method on a simple voting protocol, a simple broadcast protocol, and a simple agreement protocol. The methods scale well and have been used to find errors in a proposed industrial protocol.
The key novelty is to use modal logic to capture a declarative, high-level representation of essential system properties – the logical essence of the algorithm – while abstracting away from transitions of an abstract machine that implements it. It is like the difference between specifying code in a functional or logic programming language, versus specifying code in an imperative one.
A logical axiomatisation in the style we propose provides a precise, compact, human-readable specification that abstractly captures essential system properties, while eliding low-level implementation details; it is more precise than a natural language description, yet more abstract than source code or a logical specification thereof. This creates new opportunities for reasoning about correctness, resilience, and failure, and could serve as a foundation for human- and machine verification efforts, design improvements, and even alternative protocol implementations.
Draft of 18 January 2026.

Keywords: Distributed algorithms, three-valued modal logic, semitopology

1 Introduction

1.1 Introducing an axiomatic approach

We offer a new approach to analysing distributed algorithms, by exhibiting them as axiomatic theories in a purpose-built three-valued modal logic (one with a 'middle' truth-value **b** in-between **t** and **f**) over a semitopology (like topology but without the condition that the intersection of two open sets is necessarily open) [Gab24,Gab25]. This captures a declarative essence of the essential system properties.

Nonempty open sets of a semitopology represent quorums. If a predicate returns **t** (true) or **f** (false) this reflects correct behaviour, and if it returns **b** (both / byzantine) this reflects faulty behaviour.

The effect is similar to the difference between an explicit for-loop to apply a function `f` to each element of a list `l`, and just writing `map f l`: we abstract declaratively to the essential logic of the algorithm.

This paper is designed to introduce the basic ideas of this new declarative / axiomatic approach. These techniques can be applied to larger examples, including modern industrial-scale ones, but here we consider simpler algorithms so that we get a clearer view of the basic idea. We consider Bracha Broadcast and Crusader Agreement: these are short but non-trivial examples of the genre; they illustrate our techniques; and this provides an accessible introduction to and exposition on this new way to apply logic to study distributed algorithms.

The techniques presented in this paper scale well. In fact, more complex algorithms create *more* scope to benefit from the abstractions in this paper by eliding implementational detail. A longer draft (currently unpublished) applies similar ideas to Paxos [GZ25a].[1] In ongoing work, we have used the techniques of this paper to find errors in a proposed industrial protocol (*Heterogeneous Paxos* [SWvRM20,SWvRM21]), and we are using them to help design its replacement.

A note on terminology: we will use 'distributed algorithm' and 'distributed protocol' synonymously. The difference is in emphasis: if we write 'algorithm' we imply an omniscient view of the system from the outside, whereas if we write 'protocol' we imply a view of the system from a participant communicating with other participants according to the rules of the algorithm. However, this difference in emphasis just reflects a switch in point of view. The reader can safely identify both words with the same meaning throughout.

1.2 Motivation: distributed algorithms

Distributed algorithms are algorithms that run across multiple participants, such that the overall behaviour of the system emerges from their cooperation. Contrast with a *parallel* algorithm, which also runs across multiple participants but which would still make sense running (albeit more slowly) on a single

[1] ...but not identical; the logic and models are different, reflecting that Paxos is a different algorithm.

participant.[2] Distribution can provide benefits in resilience and scalability which could not be obtained any other way.

A fundamental challenge of distribution is that we cannot assume that all participants in the algorithm are correct: we need to be resilient against failure, and against hostile (also called *byzantine*) behaviour. Distributed algorithms are notoriously difficult to design, specify, reason about, and validate, because of the inherent difficulty of coordinating multiple participants who may be faulty, hostile, drifting in and out of network connectivity [PS17,SW89], and have different local clocks, states, agendas, etc.

This gives the field of distributed algorithms a very particular flavour. Everything revolves around this central question: how does our algorithm guarantee prompt and reliable results for correct participants, when parts of the system may be faulty?

Blockchain consensus algorithms are a good source of examples, both for distributed algorithms and for how things can go wrong.[3] Solana (`https://solana.com`) has experienced instances where its consensus mechanism contributed to temporary network halts [Yak18,Sho22]; Ripple's (`https://ripple.com`) XRP consensus protocol has been analysed for conditions in which safety and liveness may not be guaranteed [ACM20]; and Avalanche's (`https://www.avax.network`) consensus protocol has faced scrutiny regarding its assumptions and performance in adversarial scenarios [ACT22]. Even mature blockchains like Ethereum are prone to this: e.g. *balancing attacks* [NTT21] and *ex-ante reorganisation* (*reorg*) attacks [SNM$^+$22] exploit vulnerabilities in Ethereum's current protocol. Even as this paper was prepared, XRP consensus experienced a slowdown and stoppage, which was addressed in a GitHub commit.[4]

Thus, subtle flaws and/or ambiguities in protocol design have significant consequences, emphasising the need for rigorous analysis and well-defined specifications in this complex and safety-critical field.

Meanwhile in the real world, such analysis and specification may be abbreviated or skipped altogether: that is, in industrial practice a protocol may go from an initial idea written in English or pseudocode to an implementation, without undergoing a rigorous analysis. This is because such analysis is would be just too expensive. Part of the motivation for this work is, via logic, to provide a precise, compact, nontrivial, yet (relatively) lightweight methodology for specifying and reasoning about distributed protocols. Returning to the example above of an explicit for-loop versus a map function, the idea is to attain compact proofs by being as declarative and static as possible, while preserving the essence of the algorithm.

1.3 Map of the paper

- We give a simple voting algorithm in Section 2. This gives us an opportunity to introduce semitopologies and three-valued logic.
- We define the syntax of a modal logic in Section 3: this is the formal language which we will use for our axioms and correctness properties.
- We study Bracha Broadcast in Section 4.
- We study Crusader Agreement in Section 5.
- Finally, we conclude with Section 6.

2 Motivation and an illustrative example: voting

We will warm up by studying a voting algorithm. This is simpler than Bracha Broadcast but is still enough to invoke much of our machinery, including the three-valued logic and semitopologies.[5] The axiomatisation is in Figure 2. In Example 2.1.1 we describe the protocol informally:

[2] A GPU rendering algorithm is a good example of a typical parallel algorithm. It could run on a single-threaded CPU, and that would make perfectly good sense; it would just be too slow. In contrast, a blockchain is a good example of an algorithm that is fundamentally distributed and not parallel: it could still run on just one single participant, but that would completely defeat the point of the blockchain, which is precisely to *be* distributed across many participants.

[3] This work was motivated by analysis of blockchain systems.

[4] cryptodamus.io/en/articles/news/xrpl-meltdown-64-minute-outage-xrp-ledger-s-february-4th-2025-halt-decoded (permalink) and github.com/XRPLF/rippled/pull/5277 (permalink).

[5] We defer a treatment of quantifiers and formal logical syntax to the case of Bracha Broadcast.

2.1 Protocol description

Example 2.1.1 Fix $f \geq 1$ and consider a set P of $3f{+}1$ **participants**. Call a subset $Q \subseteq \mathsf{P}$ a **quorum** when $\#Q \geq 2f{+}1$; call it a **coquorum** when $\#Q \leq f$; and call it a **contraquorum** or **blocking set** when $\#Q \geq f{+}1$. A protocol to hold a ballot to choose between *true* $\mathbf{t}$ and *false* $\mathbf{f}$, proceeds as follows:

(1) Each participant p **broadcasts** to all participants a *vote* message carrying a payload $\mathbf{f}$ or $\mathbf{t}$.
(2) If p receives vote-$\mathbf{t}$ messages from a quorum of participants, then p deems that the ballot has succeeded and p **observes** $\mathbf{t}$. Similarly for $\mathbf{f}$.
    If no such majority is observed, then the ballot is deemed to have failed and p observes no value.

Remark 2.1.2 We need to choose our failure assumptions:

- do messages always arrive;
- do participants crash;
- do they follow the algorithm, or could some be dishonest/faulty/byzantine/hostile (= not follow the algorithm)?

There is no right or wrong here: we just need to be clear about our assumptions. [6]

For this exercise we assume that: message-passing is reliable (messages do not get lost or delayed); participants never crash; participants always send messages when they should; however, some participants may be dishonest and send vote-$\mathbf{t}$ messages to some participants and vote-$\mathbf{f}$ messages to others, contrary to the protocol which specifies they should broadcast the *same* vote value to *all* participants.

This might lead to a situation in which honest participants disagree about which vote won the ballot. We want to make sure that this is impossible; or to be more precise, we want to know what conditions are sufficient to ensure Agreement (this property is also called Consistency): *all honest participants observe the same value, if they observe any value at all.*

We can prove Agreement under an assumption that there is a quorum Q of honest participants, i.e. that there are at most f dishonest participants: Observe by a counting argument that any two quorums $Q_1, Q_2 \subseteq \mathsf{P}$ of votes (each of which contains at least $2f{+}1$ participants) must intersect Q on at least one participant $q \in Q \cap Q_1 \cap Q_2$. Since $q \in Q$, it is honest. This means it is impossible for a participant p to observe $\#Q_1 \geq 2f{+}1$ vote messages for $\mathbf{t}$, while at the same time *another* participant p' observes $\#Q_2 \geq 2f{+}1$ vote messages for $\mathbf{f}$, because one participant is honest and votes either for $\mathbf{t}$ or $\mathbf{f}$.

We now show how to declaratively model the protocol and the proof of Agreement. First, we will need two ingredients: a notion of *truth*, which we define in Subsection 2.2; and a notion of *quorum*, which we define in Subsection 2.3.

2.2 Three-valued logic

Our notion of truth will be three-valued:

Definition 2.2.1

(1) Let $\mathbf{3} = (\mathbf{f} < \mathbf{b} < \mathbf{t})$ be a lattice of **truth-values** with elements
- $\mathbf{f}$ (false), which is less than
- $\mathbf{b}$ (Both/Byzantine/Broken/amBivalent), which is less than
- $\mathbf{t}$ (true).

[6] For convenient reference, I include here a brief survey of some of the terminology used in the distributed systems and blockchain literature.
Call a participant in a distributed algorithm *correct* or *honest* when they follow the protocol of the algorithm. These are synonymous, but the former suggests that we view the participant as a machine, whereas the latter suggests a participant with agency to choose whether to misbehave (e.g. in a blockchain system). Call a participant that does not follow the protocol *faulty* (for machines) or *byzantine* or *adversarial* or *hostile* (for participants with agency).
Types of fault include *crash fault* (stops sending messages), *byzantine fault* (may deviate arbitrarily from the protocol), or *omission fault* (observes the protocol when live, but might choose to fake a crash).
In real life, it is also possible to play timing games (e.g. acting at the last possible moment). These are all 'honest' in a technical sense, though not necessarily in a social or legal sense; e.g. *front running* in trading is illegal. That is out of scope for this paper.

$p\wedge q$	**t**	**b**	**f**
t	**t**	**b**	**f**
b	**b**	**b**	**f**
f	**f**	**f**	**f**

$p\vee q$	**t**	**b**	**f**
t	**t**	**t**	**t**
b	**t**	**b**	**b**
f	**t**	**b**	**f**

$p\rightarrow q$	**t**	**b**	**f**
t	**t**	**b**	**f**
b	**t**	**b**	**b**
f	**t**	**t**	**t**

$p\rightharpoonup q$	**t**	**b**	**f**
t	**t**	**f**	**f**
b	**t**	**b**	**b**
f	**t**	**t**	**t**

$p\equiv q$	**t**	**b**	**f**
t	**t**	**f**	**f**
b	**f**	**t**	**f**
f	**f**	**f**	**t**

$\neg p$		$\mathsf{T}\,p$		$\mathsf{B}\,p$		$\mathsf{F}\,p$		$\mathbb{TB}\,p$		$\mathbb{TF}\,p$	
t	**f**	**t**	**t**	**t**	**f**	**t**	**f**	**t**	**t**	**t**	**t**
b	**b**	**b**	**f**	**b**	**t**	**b**	**f**	**b**	**t**	**b**	**f**
f	**t**	**f**	**f**	**f**	**f**	**f**	**t**	**f**	**f**	**f**	**t**

$p\oplus q$	**t**	**b**	**f**
t	**f**	**b**	**t**
b	**b**	**b**	**b**
f	**t**	**b**	**f**

Vertical axis above indicates values of p; horizontal axis (if nontrivial) indicates values of q.

Fig. 1. Truth-tables for some operations on $\mathbf{3}=\{\mathbf{t},\mathbf{b},\mathbf{f}\}$ (Definition 2.2.1)

Define a unary connective $\neg$ (**negation**), and binary connectives $\wedge$ (**conjunction**), $\vee$ (**disjunction**), $\rightarrow$ (**weak implication**), $\rightharpoonup$ (**strong implication**), $\equiv$ (**identity**), and $\oplus$ (**exclusive-or**), and define modalities T, B, F, $\mathbb{TB}$, and $\mathbb{TF}$, as in Figure 1.

Note that these connectives are not minimal, e.g. $tv\vee tv'$ is equal to $\neg(\neg tv\wedge\neg tv')$, and $\mathbb{TF}\,tv=\mathsf{T}\,tv\vee\mathsf{F}\,tv'$.

(2) Suppose X is a set. We may lift connectives from $\mathbf{3}$ to the function-space $X\rightarrow\mathbf{3}$ in the natural way. Thus: if $f,f':X\rightarrow\mathbf{3}$ then we may write

$$\neg f=\lambda tv.\neg f(tv)\quad\text{and}\quad f\wedge f'=\lambda tv.f(tv)\wedge f'(tv)\quad\text{and}\quad f\vee f'=\lambda tv.f(tv)\vee f'(tv).$$

(3) If $tv\in\mathbf{3}$ then write $\vDash tv$ for the judgement '$tv\in\{\mathbf{t},\mathbf{b}\}$' and read this as tv is **valid**.[7] Write $\nvDash tv$ for '$tv=\mathbf{f}$' (this happens precisely when $\neg(\vDash tv)$) and read this as tv is **invalid** or is **false**.

(4) We may write $\equiv$ for equality on truth-values, to distinguish this (where this might be helpful for clarity) from a later notion of equality on values.[8]

The implicational structure of $\mathbf{3}$ is rich (it supports sixteen implications [AAZ11, note 5, page 22]). For us, the weak implication $\rightarrow$ and strong implication $\rightharpoonup$ will suffice, and we have:

Proposition 2.2.2 *For truth-values $tv,tv'\in\mathbf{3}$, we have:*

(1) $\vDash\mathbf{t}$ *and* $\vDash\mathbf{b}$ *and* $\nvDash\mathbf{f}$.
(2) $\vDash tv\vee tv'$ *if and only if* $\vDash tv\lor\vDash tv'$, *and* $\vDash tv\wedge tv'$ *if and only if* $\vDash tv\land\vDash tv'$.
(3) $(tv\rightarrow tv')\equiv(\neg tv\vee tv')$ *and* $(tv\rightharpoonup tv')\equiv(tv\rightarrow\mathsf{T}\,tv')$.
(4) $\vDash tv\rightarrow tv'$ *if and only if* $tv\equiv\mathbf{t}$ *implies* $tv'\in\{\mathbf{t},\mathbf{b}\}$. *We call this* ***weak modus ponens***.
(5) $\vDash tv\rightharpoonup tv'$ *if and only if* $tv\equiv\mathbf{t}$ *implies* $tv'\equiv\mathbf{t}$. *We call this* ***strong modus ponens***.
(6) $\vDash tv\vee\neg tv$. *In jargon, validity satisfies* ***excluded middle*** *(p-or-not-p is valid).*
(7) $\vDash tv\wedge\neg tv$ *if and only if* $tv=\mathbf{b}$ *if and only if* $\vDash\mathsf{B}\,tv$.[9]
(8) $\vDash tv$ *if and only if* $\mathbb{TF}\,tv\equiv\mathsf{T}\,tv$.
(9) $tv\leq tv'$ *if and only if* $\neg tv'\leq\neg tv$.

[7] There is a standard distinction in logic between being a truth-value being *true* (which means: it is equal to **t**) and a logical judgement being *valid* (which may mean 'a true assertion about a model' or 'it is derivable in some derivation system' or 'it evaluates to true in a given context'). In two-valued logic this distinction is less prominent, because a predicate may be judged to be valid when it evaluates to **t** (possibly in the context of some valuation and/or interpretation). But here, we have three truth-values, and a predicate will be judged to be valid when it evaluates to **t** *or* **b**.

[8] A binary connective $\equiv:\mathbf{3}\rightarrow\mathbf{3}\rightarrow\mathbf{3}$ is presented as a truth-table in Figure 1 – meaning that we overload $\equiv$-the-identity-judgement with $\equiv$-the-function-on-truth-values. This is defensible because it is true that tv and tv' are identical, precisely when $tv\equiv tv'$ returns $\mathbf{t}$, and it is false when $\equiv$ returns $\mathbf{f}$. In symbols: $((tv\equiv tv')\equiv\mathbf{t})\equiv(tv\equiv tv')$.

[9] This means that our notion of validity is **paraconsistent**, so ϕ and not-ϕ may both be valid ([AAZ11], or see the nice survey in [Pri02]). Note that in our setting ϕ and not-ϕ cannot not be simultaneously *true*, since $\neg\mathbf{t}=\mathbf{f}$.

$$\begin{array}{rlrl}
(\mathsf{Observe?}) & \mathit{observe}(p) \to \boxdot \mathit{vote} & (\mathsf{Observe}\neg?) & \neg\mathit{observe}(p) \to \boxdot\neg\mathit{vote} \\
(\mathsf{Observe!}) & \boxdot \mathit{vote} \rightharpoonup \mathit{observe}(p) & (\mathsf{Observe}\neg!) & \boxdot\neg\mathit{vote} \rightharpoonup \neg\mathit{observe}(p) \\
(\mathsf{Correct}) & \boxdot(\mathsf{TF} \circ \mathit{vote}) & (\mathsf{3twined}) & (\boxdot f \land \boxdot f') \leq \diamonddot(f \land f')
\end{array}$$

Above: $\circ$ denotes function composition; f and f' range over arbitrary functions in $\mathsf{P} \to \mathbf{3}$ and p ranges over elements of P. $\leq$ refers to the lattice ordering on $\mathbf{3}$, namely $\mathbf{f} < \mathbf{b} < \mathbf{t}$. $\boxdot$ and $\diamonddot$ have type $(\mathsf{P} \to \mathbf{3}) \to \mathbf{3}$ here, which is why *oberve* is applied to p above and *vote* is not.
$\mathit{vote}(p) = \mathbf{t}$ means 'p voted for $\mathbf{t}$'; $\mathit{vote}(p) = \mathbf{f}$ means 'p voted for $\mathbf{f}$' and $\mathit{vote}(p) = \mathbf{b}$ means 'p sent conficted messages about its vote'. Similarly for *observe*.

Fig. 2. A simple voting protocol (Definition 2.5.1)

Proof. By inspection of the truth-tables in Figure 1. $\square$

2.3 Semitopologies

We define our universe of participants. This has a topological flavour:

Definition 2.3.1 A **semitopology** [Gab24,Gab25] is a pair $(\mathsf{P}, \mathsf{Open})$ of

- a nonempty [10] set of **points** P – which we may call **participants**, because we may think of them as participants running a distributed algorithm – and
- **open sets** $\mathsf{Open} \subseteq \mathit{powerset}(\mathsf{P})$, such that $\mathsf{P} \in \mathsf{Open}$ and Open is closed under arbitrary (possibly empty) unions.

Write $\mathsf{Open}_{\neq\varnothing} = \{O \in \mathsf{Open} \mid O \neq \varnothing\}$. We may call $O \in \mathsf{Open}_{\neq\varnothing}$ a **quorum**. Note that we assumed $\mathsf{P} \neq \varnothing$ so that also $\mathsf{Open}_{\neq\varnothing} \neq \varnothing$ (i.e. some quorum exists).

A semitopology naturally gives rise to a modal logical structure, as follows:

Definition 2.3.2 Suppose $(\mathsf{P}, \mathsf{Open})$ is a semitopology and $f : \mathsf{P} \to \mathbf{3}$ is a function, which we can think of as a unary predicate (a three-valued one, taking truth-values in $\mathbf{3}$). Define functions

$$\square, \lozenge, \boxdot, \diamonddot : (\mathsf{P} \to \mathbf{3}) \to \mathbf{3}$$

— where we read $\square f$ as '**everywhere-f**', $\lozenge f$ as '**somewhere-f**', $\boxdot f$ as '**quorum-f**', and $\diamonddot f$ as '**contraquorum-f**' — by

$$\begin{array}{ll}
\square f \equiv \bigwedge_{p \in \mathsf{P}} f(p) & \lozenge f \equiv \bigvee_{p \in \mathsf{P}} f(p) \\
\boxdot f \equiv \bigvee_{O \in \mathsf{Open}_{\neq\varnothing}} \bigwedge_{p \in O} f(p) & \diamonddot f \equiv \bigwedge_{O \in \mathsf{Open}_{\neq\varnothing}} \bigvee_{p \in O} f(p).
\end{array}$$

Lemma 2.3.3 *We note some* de Morgan *dualities:*

(1) *Suppose X is a set and $f, f' : X \to \mathbf{3}$. Then we have:*

$$\begin{array}{rr}
\neg(f \land f') \equiv (\neg f \lor \neg f') & \neg(f \lor f') \equiv (\neg f \land \neg f') \\
\neg\lozenge(\neg f) \equiv \square f & \neg\square(\neg f) \equiv \lozenge f \\
\neg\diamonddot(\neg f) \equiv \boxdot f & \neg\boxdot(\neg f) \equiv \diamonddot f
\end{array}$$

(2) *Suppose $tv \in \mathbf{3}$. Then $\neg\mathsf{T}\neg tv \equiv \mathsf{TB}\, tv$ and $\neg\mathsf{TB}\neg tv \equiv \mathsf{T}\, tv$.*

Proof. By routine arguments on lattices and truth-tables. $\square$

Remark 2.3.4 Semitopologies [Gab24,Gab25] were designed to generalise the quorum systems like Example 2.1.1 (expositions are in [MR98,ACTZ24]): *quorum* corresponds to *nonempty open set*; *coquorum*

[10] Cf. Remark 2.3.8.

(the sets complement to a quorum) corresponds to *non-total closed set*; and *contraquorum* or *blocking set* (a set that intersects every quorum) corresponds to *dense set*.

There are some nice benefits to working with a semitopology as per Definition 2.3.1, rather than with a concrete structure as per Example 2.1.1:

(1) '$O \in \mathsf{Open}_{\neq\varnothing}$' is shorter, and more mathematically rich, than 'a set O having $2f{+}1$ elements'.
(2) It notes something that is arguably obvious in retrospect, but which does not appear to have been noted before semitopologies: that things typically done in the distributed systems literature are topological in nature.
(3) As Definition 2.3.2 and Lemma 2.3.3 suggest, and the rest of this paper substantiates, we can exploit the connection between topology and logic to reason axiomatically on distributed algorithms.

Remark 2.3.5 T and $\mathbb{TB}$ commute with all monotone connectives: for $\mathsf{M} \in \{\mathsf{T}, \mathbb{TB}\}$ and $tv, tv' \in \mathbf{3}$ we have $\mathsf{M}(tv \wedge tv') \equiv ((\mathsf{M}tv) \wedge \mathsf{M}tv')$ and similarly for $\vee$, $\Diamond$, $\Box$, $\boxdot$, and $\Diamonddot$.

The reader can also easily check facts like $\mathsf{T}\,\mathbb{TB}\,tv = \mathsf{T}\,tv = \mathbb{TB}\,\mathsf{T}\,tv$, and $\mathsf{T}\,\mathsf{B}\,tv = \mathsf{B}\,tv$, and so on.

We may rewrite using these properties without comment henceforth.

We warm up with an easy lemma that connects $\Box$ with $\boxdot$:

Lemma 2.3.6 *Suppose* $(\mathsf{P}, \mathsf{Open})$ *is a semitopology and* $f, f' : \mathsf{P} \to \mathbf{3}$*. Then*

$$(\Box f \wedge \boxdot f') \leq \boxdot(f \wedge f').$$

(Recall that $\mathbf{f} < \mathbf{b} < \mathbf{t}$ *in* $\mathbf{3}$ *as a lattice. The* $\leq$ *above refers to this lattice ordering.)*

As corollaries:

(1) $\vDash \Box f' \wedge \boxdot f'$ *implies* $\vDash \boxdot(f \wedge f')$.
(2) $\vDash \Box f \wedge \boxdot(\mathbb{TF} \circ f)$ *implies* $\vDash \mathsf{T}\boxdot f$.

Proof. By routine lattice calculations from Definition 2.3.2. Or, using topological language: if every $p \in \mathsf{P}$ satisfies $f(p)$, and there exists $O' \in \mathsf{Open}$ such that every $p \in O'$ satisfies $f'(p)$, then clearly every $p \in O'$ satisfies $f(p) \wedge f'(p)$. The first corollary follows by routine reasoning, and the second follows taking $f' = \mathbb{TF} \circ f$. $\Box$

A characteristic fact connecting $\boxdot$, $\Diamonddot$, and $\Diamond$ is this:

Lemma 2.3.7 *Suppose* $(\mathsf{P}, \mathsf{Open})$ *is a semitopology and* $f, f' : \mathsf{P} \to \mathbf{3}$*. Then*

$$(\boxdot f \wedge \Diamonddot f') \leq \Diamond(f \wedge f').$$

As corollaries:

(1) $\vDash \boxdot f \wedge \Diamonddot f'$ *implies* $\vDash \Diamond(f \wedge f')$.
(2) $\vDash \Diamonddot f'$ *implies* $\vDash \Diamond f'$.
(3) $\vDash \boxdot(\mathbb{TF} \circ f')$ *and* $\vDash \Diamonddot f'$ *implies* $\vDash \mathsf{T}\Diamond f'$ *(equivalently:* $\vDash \Diamond\mathsf{T} f'$*).*

Proof. By routine lattice calculations from Definition 2.3.2. Or, using topological language: if we have an open set of fs and a dense set of f's, then there must be some point with both f and f'. Part 1 follows; part 2 follows from part 1 setting $f = \lambda p.\mathsf{T}$ and recalling from Definition 2.3.1 that $\mathsf{Open}_{\neq\varnothing} \neq \varnothing$ so that the universal quantification in $\Diamonddot f$ (Definition 2.3.2) cannot be vacuously satisfied; and part 3 follows from part 1 taking $f = \mathbb{TF} \circ f'$. $\Box$

Remark 2.3.8 In Definition 2.3.1 we assume that the set of points is nonempty, disallowing the *empty semitopology* $(\varnothing, \{\varnothing\})$. In [Gab24, Definition 1.2.2] and [Gab25, Definition 1.1.2] we do not impose this condition, i.e. we allow the empty semitopology. There are two reasons for this difference:

(1) In [Gab24,Gab25] we are studying semitopologies *in the abstract*, so we want to be as general as possible. Here, we use semitopologies specifically to model quorum systems.

    A distributed system with no participants is just not interesting for our purposes is this paper; there is literally nothing to talk about and in particular this semitopology would have no quorums (nonempty open sets). In context, this possibility would be pathological and possibly confusing, so we outlaw it by assuming that the set of participants is nonempty.

(2) Related to the previous point, but more technically, admitting the empty semitopology would mean that $\Diamond(\lambda p.\bot)$ could be equal to $\top$. Mathematically that would be perfectly respectable, but it would slightly complicate the statement of Lemma 2.3.7(2); we would need to explicitly add a condition 'for nonempty semitopology'. [11] It is simpler and neater to just assume this cannot happen.

2.4 3-twined semitopologies

Definition 2.4.1 For $n \geq 0$, call a semitopology $(\mathsf{P}, \mathsf{Open})$ ***n*-twined** when any n nonempty open sets have a nonempty intersection [GZ25b, Section 5]. That is: if $O_1, \dots, O_n \in \mathsf{Open}_{\neq\varnothing}$ then $\bigcap_{1\leq i\leq n} O_i \in \mathsf{Open}_{\neq\varnothing}$.

In this paper we will care most about 3-twined semitopologies; semitopologies such that any three nonempty open sets have a nonempty intersection.

Remark 2.4.2 Indexing the open sets from 1 in Definition 2.4.1 even though we only insist that $n \geq 0$, is not a typo. By convention, $\bigcap\varnothing = \mathsf{P}$, so a semitopology is 0-twined just when P is nonempty. The reader can also check that every semitopology is 1-twined.

So the n-twined condition starts getting more structured from 2-twined onwards.

The terminology '3-twined' follows terminology from the semitopologies literature: it generalises the notion of *intertwined* space considered in [Gab24,Gab25] (intertwined = 2-twined) and see also the Section on n-twined semitopologies in [GZ25b]. However, the general concept is well-known in the literature, albeit not phrased systematically and in (semi)topological terms. For example, being 3-twined is called the Q^3 property in [ACTZ24, Definition 2].

The semitopology in Example 2.1.1 is the canonical example of a 3-twined semitopology, though other examples are easy to generate, e.g. we can take $4f+1$ participants with quorums sets of size at least $3f+1$.

The fundamental logical property of a 3-twined semitopology is this:

Theorem 2.4.3 *Suppose* $(\mathsf{P}, \mathsf{Open})$ *is 3-twined and* $f, f' : \mathsf{P}\to\mathbf{3}$*. Then*[12]

$$(\boxdot f \wedge \boxdot f') \leq \Diamond(f \wedge f').$$

As a corollary

$$\vDash \boxdot f \wedge \boxdot f' \quad \textit{implies} \quad \vDash \Diamond(f \wedge f').$$

Proof. $\boxdot f \wedge \boxdot f' = \mathbf{t}$ implies that there exist $O, O' \in \mathsf{Open}_{\neq\varnothing}$ on which f and f' respectively take the value $\mathbf{t}$. By the 3-twined property, $O \cap O'$ intersects every other $O'' \in \mathsf{Open}_{\neq\varnothing}$, thus $\Diamond(f \wedge f') = \mathbf{t}$.

$\boxdot f \wedge \boxdot f' = \mathbf{b}$ implies that there exist $O, O' \in \mathsf{Open}_{\neq\varnothing}$ on which f and f' respectively take values in $\{\mathbf{t}, \mathbf{b}\}$. By the 3-twined property $O \cap O'$ intersects every other $O'' \in \mathsf{Open}_{\neq\varnothing}$, thus $\Diamond(f \wedge f') \in \{\mathbf{t}, \mathbf{b}\}$. □

Corollary 2.4.4 is a (rather elegant) dual rephrasing of Theorem 2.4.3 and will be useful later:

Corollary 2.4.4 Suppose $(\mathsf{P}, \mathsf{Open})$ is 3-twined and $f, f' : \mathsf{P} \to \mathbf{3}$. Then

$$\boxdot(f \vee f') \leq (\Diamond f \vee \Diamond f').$$

As a corollary,

$$\vDash \boxdot(f \vee f') \quad \text{implies} \quad \vDash \Diamond f \vee \Diamond f'.$$

Proof. By Theorem 2.4.3 $(\boxdot\neg f \wedge \boxdot\neg f') \leq \Diamond(\neg f \wedge \neg f')$. By Proposition 2.2.2(9) $\neg\Diamond(\neg f \wedge \neg f') \leq \neg(\boxdot\neg f \wedge \boxdot\neg f')$. Simplifying using Lemma 2.3.3(1) we obtain $\boxdot(f \vee f') \leq (\Diamond f \vee \Diamond f')$ as required. □

[11] Thanks to Jan Mas Rovira for noting this subtlety.

[12] The reverse of this result also holds: if for every $f, f' : \mathsf{P} \to \mathbf{3}$ we have $(\boxdot f \wedge \boxdot f') \leq \Diamond(f \wedge f')$, then $(\mathsf{P}, \mathsf{Open})$ is 3-twined.

Remark 2.4.5 It might be helpful to sum up some highlights of the maths so far. Suppose $(\mathsf{P}, \mathsf{Open})$ is a semitopology and $f, f' : \mathsf{P} \to \mathbf{3}$.

Suppose further that $\vDash \boxdot(\mathsf{TF} \circ f)$, meaning that f is correct ($=$ **t** or **f**, not **b**) on a quorum ($=$ nonempty open set) of participants. (It is a typical failure assumption in the distributed systems literature that there exists a quorum of correct participants.)

Then we have seen that:

(1) $\vDash \Box f$ implies $\vDash \mathsf{T} \boxdot f$, by Lemma 2.3.6(2).
(2) If $(\mathsf{P}, \mathsf{Open})$ is 3-twined then $\vDash \boxdot f$ implies $\vDash \Diamonddot(\mathsf{T} \circ f)$ and equivalently $\vDash \mathsf{T} \Diamonddot f$, by Theorem 2.4.3.
(3) $\vDash \Diamonddot f$ implies $\vDash \Diamond (\mathsf{T} \circ f)$ and equivalently $\vDash \mathsf{T} \Diamond f$, by Lemma 2.3.7(2).
(4) $\vDash \boxdot f \wedge \Diamonddot(\mathsf{T} \circ g)$ implies $\vDash \mathsf{T} \Diamond f$, by Lemma 2.3.7(1).
(5) If $(\mathsf{P}, \mathsf{Open})$ is 3-twined then $\vDash \boxdot f \wedge \boxdot f'$ implies $\vDash \Diamonddot(f \wedge f')$ and $\vDash \boxdot(f \vee f')$ implies $\vDash \Diamonddot f \vee \Diamonddot f'$, by Theorem 2.4.3 and Corollary 2.4.4.

2.5 A simple voting protocol

We are now ready to consider our first (very simple) protcol:

Definition 2.5.1 Call a tuple $\mu = ((\mathsf{P}, \mathsf{Open}), \mathit{vote}, \mathit{observe})$, where

- $(\mathsf{P}, \mathsf{Open})$ is a(n arbitrary) semitopology and
- $\mathit{vote}, \mathit{observe} : \mathsf{P} \to \mathbf{3}$ are functions,

a **model**. We call a model μ a **model** (of ThyVote) when the axioms in Figure 2 are valid in μ.

Remark 2.5.2 We take a minute to unpack what the notation in Figure 2 is intended to mean:

(1) $\mathit{vote}(p) = \mathbf{t}$ indicates that p voted for **t**. $\mathit{vote}(p) = \mathbf{f}$ indicates that p voted for **f**. $\mathit{vote}(p) = \mathbf{b}$ indicates that p (is hostile because it) sent vote messages for *both* **t** and **f**.[13]
(2) $\mathit{observe}(p) = \mathbf{t}$ indicates that p observed a quorum of vote-**t** messages – some of which may be from hostile participants; p cannot know. Similarly for $\mathit{observe}(p) = \mathbf{f}$. $\mathit{observe}(p) = \mathbf{b}$ indicates that p observed no quorum of messages either way.

Remark 2.5.3 We discuss how the axioms in Figure 2 correspond to the protocol in Example 2.1.1:

(1) (Observe?) is a **backward rule**: it express that *if* a participant p observes **t**, then it must have received a quorum of vote-**t** messages.

   We use weak implication $\to$ (not strong implication $\rightharpoonup$) in (Observe?) because a quorum may include some dishonest (hostile) participants who sent dishonest messages; i.e. vote-**t** to some participants and vote-**f** to others. We represent the combination of vote-**t** and vote-**f** messages from a hostile participant q by setting $\mathit{vote}(q) = \mathbf{b}$. Thus, if $\mathit{observe}(p) = \mathbf{t}$ it does not follows that $\boxdot \mathit{vote} = \mathbf{t}$; it only follows that $\boxdot \mathit{vote} \in \{\mathbf{t}, \mathbf{b}\}$. This highlights that $\mathit{vote}(p)$ in our declarative model is a single *truth-value*, not a *message* or a set of messages.
(2) (Observe¬?) just repeats (Observe?) for the case that p receives a quorum of vote-**f**.
(3) (Observe!)/(Observe¬!) are **forward rules**: they express that if an honest quorum votes for **t**/**f**, then every p observes **t**/**f**.

   Why did we use strong implication here, not weak implication? We could use either, but we make a design choice: since (in our protocol) participants do not report their *observe* to anyone else, there is no useful sense in which lying would be meaningful. So we use the strong implication.
(4) (Correct) asserts that some quorum of participants are honest: they vote **t** or **f**, and not **b**.
(5) (3twined) expresses the characteristic intersection property of a 3-twined semitopology, as per Theorem 2.4.3. For this protocol we will use it once, setting $f = \mathit{vote}$ and $f' = \neg \mathit{vote}$.

Remark 2.5.4 At the level of *vote*, **b** denotes a hostile participant sending mixed votes to different participants. At the level of *observe*, **b** just indicates that a participant has observed no quorum of participants voting for **t** or voting for **f**.

[13] There is no notion here of 'not voted yet', because there is no notion of time! *vote* just records the votes that a participant casts, and our failure assumptions assume that *some* vote is made, as per Remark 2.1.2. Different assumptions would yield different axioms.

This use of $\mathbf{b}$ to mean two things, depending on the context, is a feature, not a bug. It illustrates a fact about truth-values in $\mathbf{3}$: they are just data, which we – via our choice of axioms and intended interpretation – can interpret as we choose and as is mathematically convenient. [14]

Remark 2.5.5 Note in Figure 2 that ($\mathsf{Observe?}$) has a dual ($\mathsf{Observe\neg?}$), and ($\mathsf{Observe!}$) has a dual ($\mathsf{Observe\neg!}$) – but ($\mathsf{Correct}$) has no dual axiom ($\mathsf{Correct\neg}$). Why is this so? Because ($\mathsf{Correct}$) is self-dual: it is a fact of the truth-tables in Figure 1 that $\mathsf{TF}\circ \mathit{vote} = \mathsf{TF}\circ\neg\circ \mathit{vote}$, so that $\boxdot(\mathsf{TF}\circ \mathit{vote}) \equiv \boxdot(\mathsf{TF}\circ\neg\circ \mathit{vote})$.

Lemma 2.5.6 *$\vDash \Diamond\, \mathit{observe} \to \boxdot \mathit{vote}$, $\vDash \boxdot \mathit{vote} \rightharpoonup \Box\, \mathit{observe}$, $\vDash \Diamond\neg \mathit{observe} \to \boxdot\neg \mathit{vote}$ and $\vDash \boxdot\neg \mathit{vote} \rightharpoonup \Box\,\neg \mathit{observe}$.*

Proof. The reasoning is very easy and we prove just the first two parts.

(1) By weak modus ponens (Prop. 2.2.2(4)) it suffices to show that $\vDash \mathsf{T}\Diamond\, \mathit{observe}$ implies $\vDash \boxdot \mathit{vote}$. So suppose $\vDash \mathsf{T}\Diamond\, \mathit{observe}$. Using the meaning of $\Diamond$ from Definition 2.3.2, there exists $p \in \mathsf{P}$ such that $\mathit{observe}(p) = \mathbf{t}$, and so $\vDash \mathsf{T}\, \mathit{observe}(p)$. Then using weak modus ponens and ($\mathsf{Observe?}$) we have $\vDash \boxdot \mathit{vote}$.

(2) By strong modus ponens (Prop. 2.2.2(5)) and the meaning of $\Box$, it suffices to show that $\vDash \mathsf{T}\boxdot \mathit{vote}$ implies $\vDash \mathsf{T}\, \mathit{observe}(p)$ for every $p \in \mathsf{P}$. But this is immediate from ($\mathsf{Observe!}$). $\Box$

We can now prove it is impossible for one participant to observe $\mathbf{t}$ and another to observe $\mathbf{f}$:

Proposition 2.5.7 (Agreement) *If μ is a model of* ThyVote *then $\nvDash \Diamond\mathsf{T}\, \mathit{observe} \wedge \Diamond\mathsf{T}\neg \mathit{observe}$.*

Proof. Suppose $\vDash \Diamond\mathsf{T}\, \mathit{observe} \wedge \Diamond\mathsf{T}\neg \mathit{observe}$. Then there exist $p, p' \in \mathsf{P}$ such that $\mathit{observe}(p) = \mathbf{t}$ and $\mathit{observe}(p') = \mathbf{f}$. Using ($\mathsf{Observe?}$) and ($\mathsf{Observe\neg?}$) $\vDash \boxdot \mathit{vote} \wedge \boxdot\neg \mathit{vote}$. By ($\mathsf{3twined}$) $\vDash \diamonddot(\mathit{vote} \wedge \neg \mathit{vote})$, so using Proposition 2.2.2(7) $\vDash \diamonddot\mathsf{B}\, \mathit{vote}$. Using ($\mathsf{Correct}$) and Lemma 2.3.7(1) $\vDash \Diamond(\mathsf{B}\, \mathit{vote} \wedge \mathsf{TF}\, \mathit{vote})$. But $(\mathsf{B}\, \mathit{vote} \wedge \mathsf{TF}\, \mathit{vote}) \equiv \mathbf{f}$, and $\vDash \Diamond\mathbf{f}$ is impossible. $\Box$

Remark 2.5.8 A more traditional proof for Proposition 2.5.7 would work concretely with the set of quorums from Example 2.1.1, and prove Agreement by counting and calculations (see for example Lemmas 1 to 4 used in the proof of Theorem 1 from [Bra87, page 135]).

Proposition 2.5.7 is not that proof: the semitopology is abstract, except for satisfying the axioms in Figure 2. The reasoning is logical in flavour and follows the structure of the axioms. This is simple, general, and it captures the logical essence of what the concrete counting proofs are actually doing.

Remark 2.5.9 The failure model is encoded in the axioms – typically this shows up in the strength of the implications we use. For example: our failure model in Remark 2.1.2 assumed a reliable network, which is reflected by the use of *strong implication* in the forward rules ($\mathsf{Observe!}$) and ($\mathsf{Observe\neg!}$) in Figure 2.

To model an unreliable network, in which messages sent might not arrive, we would weaken ($\mathsf{Observe!}$) to $\boxdot \mathit{vote} \to \mathit{observe}$, and similarly for ($\mathsf{Observe\neg!}$). This reflects that a quorum of honest participants vote for $\mathbf{t}$, but the network fails to transmit their messages so a participant p (even if honest) may remain unable to observe and so set $\mathit{observe}(p) = \mathbf{b}$.

3 The modal logic

The above concludes our discussion of the simple voting protocol. We are now ready to start studying Bracha Broadcast. We will need two more ingredients:

(1) Our voting example was binary (vote-$\mathbf{t}$ or vote-$\mathbf{f}$). For Bracha Broadcast we assume an arbitrary nonempty set of values Value, over which we will need three-valued quantifiers like unique existence $\exists_1 : \mathsf{Value} \to \mathbf{3}$. We study this in Subsection 3.1.

(2) Our voting example worked directly with semantics; we made assertions directly about functions like $\mathit{vote} : \mathsf{P} \to \mathbf{3}$. For Bracha Broadcast it is better to build the syntax and semantics of a (simple) logic;

[14] An example from real life. If my wife asks me "You didn't forget to put the rubbish out, did you?" and I return a value "Yes", does this mean "Yes, I did forget!" or "Yes, I did put the rubbish out!"? The "Yes" here is raw data; its meaning exists in the (hopefully shared) understanding of the people having the conversation.
In the same way, $\mathbf{b}$ is just data, and is meaningless until we – *we* – use it in axioms and interpret those axioms.

we will have predicate-symbols like echo, which will be interpreted as functions like $\varsigma(\mathsf{echo}) : \mathsf{P} \to \mathsf{Value} \to \mathbf{3}$. We study this in Subsection 3.2.

3.1 (Unique/affine) existence in three-valued logic

Distributed algorithms often require an implementation to make choices, e.g. 'respond only to the first message you receive'. In axioms, we model this using unique existence and affine existence:

Definition 3.1.1 Suppose Value is a set and $f : \mathsf{Value} \to \mathbf{3}$ is a function. Define **existence** $\boldsymbol{\exists} f$, **affine existence** $\boldsymbol{\exists_{01}} f$ (there exist zero or one) and **unique existence** $\boldsymbol{\exists_1} f$ (there exists precisely one) by: [15]

$$\boldsymbol{\exists} f = \bigvee_{v\in\mathsf{Value}} f(v) \quad \boldsymbol{\exists_{01}} f = \bigwedge_{v,v'\in\mathsf{Value}}(f(v) \wedge f(v')) \to v{=}v' \quad \boldsymbol{\exists_1} f = (\boldsymbol{\exists} f) \wedge (\boldsymbol{\exists_{01}} f).$$

Remark 3.1.2 Note that $\boldsymbol{\exists}$, $\boldsymbol{\exists_1}$, and $\boldsymbol{\exists_{01}}$ return truth-values in $\mathbf{3}$, whereas the usual Boolean quantifiers – write them $\exists$, $\exists_1$, and $\exists_{01}$ – return truth-values in $\{\bot, \top\}$. So even if '$\boldsymbol{\exists_{01}}$' looks like '$\exists_{01}$', these are different creatures.

Therefore, continuing the notation of Definition 3.1.1, we spell out cases to check how the Definition works: Then:

(1) If $\exists v \neq v' \in \mathsf{Value}.f(v) = \mathbf{t} = f(v')$, then $\boldsymbol{\exists_1} f = \mathbf{f} = \boldsymbol{\exists_{01}} f$.
(2) If $\exists_1 v \in \mathsf{Value}.f(v) = \mathbf{t}$, and $\forall v' \in \mathsf{Value}.f(v') \in \{\mathbf{t}, \mathbf{f}\}$, then $\boldsymbol{\exists_1} f = \mathbf{t} = \boldsymbol{\exists_{01}} f$.
(3) If $\exists_1 v \in \mathsf{Value}.f(v) = \mathbf{t}$, and $\exists v' \in \mathsf{Value}.f(v') = \mathbf{b}$, then $\boldsymbol{\exists_1} f = \mathbf{b} = \boldsymbol{\exists_{01}} f$.
(4) If $\neg\exists v \in \mathsf{Value}.f(v) = \mathbf{t}$ and $\exists_1 v' \in \mathsf{Value}.f(v') = \mathbf{b}$, then $\boldsymbol{\exists_1} f = \mathbf{b} = \boldsymbol{\exists_{01}} f$.
If there is only one non-$\mathbf{f}$ *value, then arguably* $\boldsymbol{\exists_{01}} f$ *should be* $\mathbf{t}$ *(not* $\mathbf{b}$*). In practice, this will not matter: we will only use Proposition 3.1.3 and that is unaffected either way.*
(5) If $\neg\exists v \in \mathsf{Value}.f(v) = \mathbf{t}$ and $\exists v, v' \in \mathsf{Value}.v \neq v' \wedge f(v) = \mathbf{b} = f(v')$, then $\boldsymbol{\exists_1} f = \mathbf{b} = \boldsymbol{\exists_{01}} f$.
If $\forall v \in \mathsf{Value}.f(v) = \mathbf{b}$*, then* $\boldsymbol{\exists_1} f = \mathbf{b}$ *– note that this is* not $\mathbf{f}$*. This illustrates that in our three-valued setting, 'unique existence' is not the same as 'unique non-false value'.*
(6) If $\forall v \in \mathsf{Value}.f(v) = \mathbf{f}$, then $\boldsymbol{\exists_1} f = \mathbf{f}$ and $\boldsymbol{\exists_{01}} f = \mathbf{t}$.

Our proofs will use unique/affine existence via Proposition 3.1.3:

Proposition 3.1.3 *Suppose V is a set and $f : V \to \mathbf{3}$ is a function and $v, v' \in V$. Then:*

(1) *The following are equivalent:*
  (a) $\vDash \boldsymbol{\exists_{01}} f$.
  (b) $\boldsymbol{\exists_{01}} f \in \{\mathbf{t}, \mathbf{b}\}$.
  (c) $\exists_{01} v{\in}V.(\vDash \mathsf{T}\, f(v))$.
  (d) $\exists_{01} v{\in}V.f(v) = \mathbf{t}$.
  (e) $\forall v, v'{\in}V.f(v) = \mathbf{t} = f(v') \Longrightarrow v = v'$.
(2) *The following are equivalent:*
  (a) $\vDash \boldsymbol{\exists_1} f$.
  (b) $\exists v{\in}V.(\vDash f(v))$ *and* $\vDash \boldsymbol{\exists_{01}} f$.
(3) *The following are equivalent:*
  (a) $\vDash \mathsf{T}\, \boldsymbol{\exists_{01}} f$.
  (b) $(\exists_{01} v{\in}V.f(v) = \mathbf{t}) \wedge (\forall v{\in}V.f(v) \neq \mathbf{b})$.
(4) *The following are equivalent:*
  (a) $\vDash \mathsf{T}\, \boldsymbol{\exists_1} f$.
  (b) $\exists_1 v{\in}V.f(v) = \mathbf{t}$ *and* $\forall v{\in}V.f(v) \neq \mathbf{b}$.
(5) *If* $\vDash \boldsymbol{\exists_{01}} f$ *or* $\vDash \boldsymbol{\exists_1} f$ *and* $\vDash \mathsf{T}\,(f(v) \wedge f(v'))$ *then* $v = v'$.

Proof. We unpack Definition 3.1.1 and check the possibilities in Remark 3.1.2. □

$$\begin{array}{ll} \text{Terms} & t ::= a \mid v \\ \text{Predicates} & \phi ::= \bot \mid \neg\phi \mid \phi \wedge \phi \mid \boxdot\phi \mid \exists a.\phi \mid \exists_{\mathbf{01}} a.\phi \mid \mathbb{TF}\,\phi \mid \mathsf{P}(t) \quad (\mathsf{P} \in \mathcal{P}) \end{array}$$

$$\begin{array}{llll} \text{Sugar} & \diamonddot\phi = \neg\boxdot\neg\phi & \phi \vee \phi' = \neg(\neg\phi \wedge \neg\phi') & \phi \rightarrow \phi' = \neg\phi \vee \phi' \\ & \forall v.\phi = \neg\exists v.\neg\phi & \exists_{\mathbf{1}} v.\phi = \exists_{\mathbf{01}} v.\phi \wedge \exists v.\phi & \mathsf{B}\,\phi = \neg\mathbb{TF}\,\phi \\ & \mathbb{TF}[\mathsf{P}] = \forall a.\mathbb{TF}\,\mathsf{P}(a) & \mathsf{B}[\mathsf{P}] = \forall a.\mathsf{B}\,\mathsf{P}(a) & (\mathsf{P} \in \mathcal{P}) \end{array}$$

Fig. 3. Syntax of a simple modal logic (Definition 3.2.1)

$$\begin{array}{lll} \llbracket\bot\rrbracket_\mu(p) = \bot & \llbracket\neg\phi\rrbracket_\mu(p) = \neg\llbracket\phi\rrbracket_\mu(p) & \llbracket\phi \wedge \phi'\rrbracket_\mu(p) = \llbracket\phi\rrbracket_\mu(p) \wedge \llbracket\phi'\rrbracket_\mu(p) \\ \llbracket\boxdot\phi\rrbracket_\mu(p) = \boxdot\llbracket\phi\rrbracket_\mu & \llbracket\exists a.\phi\rrbracket_\mu(p) = \exists\lambda v.\llbracket\phi[a{:=}v]\rrbracket_\mu(p) & \llbracket\exists_{\mathbf{01}} a.\phi\rrbracket_\mu(p) = \exists_{\mathbf{01}}\lambda v.\llbracket\phi[a{:=}v]\rrbracket_\mu(p) \\ \llbracket\mathbb{TF}\,\phi\rrbracket_\mu(p) = \mathbb{TF}\,(\llbracket\phi\rrbracket_\mu(p)) & \llbracket\mathsf{P}(v)\rrbracket_\mu(p) = \varsigma(\mathsf{P})(p)(v) \quad (\mathsf{P} \in \mathcal{P}) & \end{array}$$

$$\begin{array}{llllll} p \vDash_\mu \phi & \text{means} & \llbracket\phi\rrbracket_\mu(p) \in \{\mathbf{t}, \mathbf{b}\} & \vDash_\mu \phi & \text{means} & \forall p{\in}\mathsf{P}.(p \vDash_\mu \phi) \\ \vDash_\mu \Phi & \text{means} & \forall\phi{\in}\Phi.(\vDash_\mu \phi) & \Phi \vDash_\mu \phi & \text{means} & \vDash_\mu \Phi \Longrightarrow \vDash_\mu \phi \end{array}$$

Fig. 4. Denotation for a simple modal logic (Definition 3.2.2)

3.2 A simple modal logic

Definition 3.2.1 We fix an infinite set of **variable symbols** $a, b \in \mathsf{VarSymb}$. We assume a nonempty set of **values** $v \in \mathsf{Value}$ and a set of **predicate symbols** $\mathsf{P} \in \mathcal{P}$. Specifically:

- For our example of Bracha Broadcast below, we set Value to be any nonempty set and we take $\mathcal{P} = \{\mathsf{bcst}, \mathsf{echo}, \mathsf{ready}, \mathsf{dlvr}\}$.
- For our example of Crusader Agreement below, we set $\mathsf{Value} = \{0, 0.5, 1\}$ and we take $\mathcal{P} = \{\mathsf{input}, \mathsf{echo}_1, \mathsf{echo}_2, \mathsf{output}\}$.

We then define **terms** t, **predicates** ϕ, and syntactic sugar as in Figure 3.

Write $\phi[a{:=}v]$ for **substitution** of a for v; this is the result of replacing every unbound a in ϕ with v. We may abuse notation and write v for quantified variables, e.g. writing $\exists v.\mathsf{echo}(v)$ instead of $\exists a.\mathsf{echo}(a)$; it will always be clear what is meant.

Definition 3.2.2

1. Given a semitopology $(\mathsf{P}, \mathsf{Open})$, an **interpretation** ς is an assignment for each $\mathsf{P} \in \mathcal{P}$ of a function $\varsigma(\mathsf{P}) : \mathsf{P} \rightarrow \mathsf{Value} \rightarrow \mathbf{3}$.
2. A **model** $\mu = (\mathsf{Value}, (\mathsf{P}, \mathsf{Open}), \varsigma)$ consists of a set of values, a semitopology, and an interpretation.
3. Given a model μ, we define a **denotation** $\llbracket\phi\rrbracket_\mu : \mathsf{P} \rightarrow \mathbf{3}$ and **validity** $p \vDash_\mu \phi$ and $\vDash_\mu \phi$ as per Figure 4.
4. Call an axiom ϕ **valid** in μ when $\vDash_\mu \phi$. Call a set of axioms $\textsc{Thy}$ a **theory**.
5. Write $\vDash_\mu \textsc{Thy}$ when $\forall\phi{\in}\textsc{Thy}.(\vDash_\mu \phi)$ (i.e. the axioms of $\textsc{Thy}$ are all valid in μ), in which case we call μ a **model of** $\textsc{Thy}$.
6. Write $\textsc{Thy} \vDash_\mu \phi$ when $\vDash_\mu \textsc{Thy}$ implies $\vDash_\mu \phi$ (i.e. every model of $\textsc{Thy}$ satisfies ϕ).

Notation 3.2.3 In what follows, we may use the fact that $p \vDash_\mu \Diamond\,\phi$ is equivalent to $\vDash_\mu \Diamond\,\phi$ without comment, and similarly for the other modalities $\Box$, $\boxdot$, and $\diamonddot$, and for compound predicates like $\Diamond\,\phi \rightarrow \Box\,\phi$. This pattern will be quite common in the proofs that follow, because many of our axioms have the form $\phi \rightarrow \boxdot\phi'$, so when we use weak modus ponens (Prop. 2.2.2(4)) we may assume $p \vDash_\mu \mathsf{T}\,\phi$ and deduce $\vDash_\mu \boxdot\phi'$ (not $p \vDash_\mu \boxdot\phi'$); the p is still there in some sense, but it no longer matters.

[15] The standard symbol for unique existence is $\exists!$. I am not aware of a standard symbol for affine existence. I use $\exists_1$ and $\exists_{01}$ for clarity and consistency.

Protocol description.

(1) A designated *sender* participant sends messages to all participants, **broadcasting** a value v. Message-passing is reliable, so messages always arrive.

(2) If a participant receives a broadcast v message from the sender – it is assumed that every participant knows who the sender is, and other participants cannot forge the sender's signature – it sends an **echo** v to all other participants.

Each participant will only echo *once*, so if it receives two broadcast messages with different values then it will only echo one of them.

(3) If a participant receives a quorum of echo messages for a value v, or a contraquorum of ready messages for a value v, then it sends messages to all participants declaring itself **ready** with v.

(4) If a participant receives a quorum of ready messages for a value v, then the participant **delivers** v.

Failure assumptions. Our failure assumption will be that a quorum of participants follow the algorithm, and a coquorum (the complement of a quorum, i.e. a nontotal closed set) of participants may not follow the algorithm (more on this terminology in Remark 2.3.4). For instance, a hostile participant may broadcast or echo multiple values v.

Fig. 5. Informal (English) description of Bracha Broadcast (Remark 4.1.1)

Backward rules		*Forward rules*	
(BrDeliver?)	$\mathsf{dlvr}(a) \to \boxdot\mathsf{ready}(a)$	(BrDeliver!)	$\boxdot\mathsf{ready}(a) \to \mathsf{dlvr}(a)$
(BrReady?)	$\mathsf{ready}(a) \to \boxdot\mathsf{echo}(a)$	(BrReady!)	$\boxdot\mathsf{echo}(a) \to \mathsf{ready}(a)$
(BrEcho?)	$\mathsf{echo}(a) \to \Diamond\,\mathsf{bcst}(a)$	(BrEcho!)	$\Diamond\,\mathsf{bcst}(a) \to \exists a.\mathsf{echo}(a)$
		(BrReady!!)	$\Diamonddot\mathsf{ready}(a) \to \mathsf{ready}(a)$
Other rules			
(BrEcho01)	$\exists_{\mathbf{01}} a.\mathsf{echo}(a)$	(BrCorrect)	$\boxdot\mathbb{TF}[\mathsf{ready}] \wedge \boxdot\mathbb{TF}[\mathsf{echo}]$
(BrBroadcast1)	$\exists_{\mathbf{1}} a.\Diamond\,\mathsf{bcst}(a)$	(BrCorrect′)	$\mathbb{TF}[\mathsf{P}] \vee \mathsf{B}[\mathsf{P}]\ \ (\mathsf{P}{\in}\{\mathsf{ready},\mathsf{echo}\})$
		(BrCorrect″)	$\Box\,\mathbb{TF}[\mathsf{bcst}] \vee \Box\,\mathsf{B}[\mathsf{bcst}]$

Free a above are assumed universally quantified (i.e. there is an invisible $\forall a$ at the start of any axiom with a free a). The axioms above assume predicate symbols $\mathcal{P} = \{\mathsf{bcst}, \mathsf{echo}, \mathsf{ready}, \mathsf{dlvr}\}$, as per Definition 3.2.1.

Fig. 6. ThyBB: axioms of Bracha Broadcast (Definition 4.1.2)

(BrValidity?)	$\mathrm{T{\scriptsize HY}BB} \vDash_\mu \Diamond\,\mathsf{bcst}(v) \to \Box\,\mathsf{dlvr}(v)$	(Proposition 4.2.9)
(BrNoDup)	$\mathrm{T{\scriptsize HY}BB} \vDash_\mu \exists_{\mathbf{01}}\mathsf{dlvr}$	(Proposition 4.2.11)
(BrIntegrity)	$\mathrm{T{\scriptsize HY}BB} \vDash_\mu \mathsf{dlvr}(v) \to \Diamond\,\mathsf{bcst}(v)$	(Proposition 4.2.12)
(BrConsistency)	$\mathrm{T{\scriptsize HY}BB} \vDash_\mu \exists_{\mathbf{01}}\Diamond\,\mathsf{dlvr}(v)$	(Proposition 4.2.11)
(BrTotality)	$\mathrm{T{\scriptsize HY}BB} \vDash_\mu \Diamond\,\mathsf{dlvr}(v) \to \Box\,\mathsf{dlvr}(v)$	(Proposition 4.2.13)

Fig. 7. Correctness properties for ThyBB (Remark 4.2.2)

4 Declarative Bracha Broadcast

4.1 Definition of the theory

Remark 4.1.1 Bracha Broadcast is a classic broadcast algorithm [Bra87]. An English description of the algorithm is given in Figure 5. It is designed to allow a group of participants to agree on at most one value, in the presence of some hostile participants.

Definition 4.1.2 Define **Declarative Bracha Broadcast** to be the theory (set of axioms) ThyBB in

Figure 6. Axioms are taken to be universally quantified over any free variables (there is only one: a).

Remark 4.1.3 A fundamental difference between a logical theory and an algorithm is:

- With a theory, we are presented with a model and we ask "Are the axioms valid in this model?".
- With an algorithm, we start from nothing and we construct a model by following the algorithm.

So in this paper, we assume a model is just presented to us, and our business is to check whether it validates axioms. All of the complexity inherent in studying an implementation – i.e. a machine to compute such a model – is elided by design.

Remark 4.1.4 As for our voting example from Figure 2, the axioms in Figure 6 are of three kinds:

(1) *Backward rules* have names ending with ? and represent reasoning of the form "if *this* happens, then *that* must have happened".

(2) *Forward rules* have names ending with ! and represent forward reasoning of the form "if *this* happens, then *that* must happen".

(3) *Other rules* reflect particular conditions of the algorithms, e.g. having to do with failure assumptions.

Remark 4.1.5 [Discussion of the axioms] We consider the axioms in turn. Each is implicitly universally quantified over a if required; so the a in (BrDeliver?) represents 'any value $v \in \mathsf{Value}$'. The truth-values $\mathbf{t}$ and $\mathbf{f}$ represent honest behaviour, and $\mathbf{b}$ represents dishonest behaviour. Each axiom is evaluated at an arbitrary $p \in \mathsf{P}$:

(1) (BrDeliver?) says: if $\mathsf{dlvr}(v)$ is $\mathbf{t}$ at p then for a quorum of participants $\mathsf{ready}(v)$ is $\mathbf{t}$ or $\mathbf{b}$.
This reflects that in the algorithm as outlined in Remark 4.1.1, if p honestly delivers v then a quorum of participants must have sent ready messages for v, though some of those messages may have been sent by dishonest participants.
(2) (BrReady?) says: if $\mathsf{ready}(v)$ is $\mathbf{t}$ at p then for a quorum of participants $\mathsf{echo}(v)$ is $\mathbf{t}$ or $\mathbf{b}$.
(3) (BrEcho?) says: if $\mathsf{echo}(v)$ is $\mathbf{t}$, then somebody $\mathsf{bcst}(v)$ with $\mathbf{t}$ or $\mathbf{b}$.
(4) (BrDeliver!) says: if $\mathsf{ready}(v)$ is $\mathbf{t}$ for a quorum of participants, then $\mathsf{dlvr}(v)$ is $\mathbf{t}$ or $\mathbf{b}$.
This reflects that in the algorithm, if a quorum of honest participants send ready-v messages and if p is honest, then p will send a deliver-v message.
(5) (BrReady!) says: if $\mathsf{echo}(v)$ is $\mathbf{t}$ for a quorum of participants, then $\mathsf{ready}(v)$ is $\mathbf{t}$ or $\mathbf{b}$.
(6) (BrEcho!) says: if bcst is $\mathbf{t}$ for some participant and some value, then echo is $\mathbf{t}$ or $\mathbf{b}$ for some value, though not necessarily the same one.
This reflects that in the algorithm, if somebody honestly broadcasts some value and if p is honest, then p will echo some value. The fact that this will be the *same* value (if both parties are honest) is a Lemma that follows using (BrEcho?); see Lemma 4.2.6(1).
(7) (BrReady!!) says: if $\mathsf{echo}(v)$ is $\mathbf{t}$ for a *contraquorum* of participants, then $\mathsf{ready}(v)$ is $\mathbf{t}$ or $\mathbf{b}$. This reflects the disjunct 'or a contraquorum of ready messages' in Remark 4.1.1(3).
(8) (BrEcho01) says that $\mathsf{echo}(v)$ is $\mathbf{t}$ for at most one value v. It asserts nothing about byzantine behaviour: $\mathsf{echo}(v)$ can be $\mathbf{b}$ for as many values as we like, reflecting the fact that dishonest participants may deviate from the protocol.
(9) (BrBroadcast1) says that $\mathsf{bcst}(v)$ is $\mathbf{t}$ for precisely one value v *or* $\mathsf{bcst}(v)$ is $\mathbf{b}$ for *at least* one value v. This is what $\exists_1$ means in a three-value setting, as discussed in Remark 3.1.2 and Proposition 3.1.3.
(10) (BrCorrect) asserts that there are quorums of honest participants for ready and echo. The literature tends to assume something slightly stronger, that there is one quorum of honest participants for all predicates. We will only need this weaker assumption for our proofs.
(11) (BrCorrect$'$) asserts that a participant p is either entirely honest for ready and echo, in that it returns $\mathbf{t}$ or $\mathbf{f}$ for every value, or it is entirely dishonest, in that it returns $\mathbf{b}$ for every value. This accurately reflects an informal assumption that (for a given predicate) a participant is either honest or dishonest – dishonestly is not attached to a particular message, but to the *participant* who sent it.
(BrCorrect$''$) asserts that either for all participants the bcst predicate returns $\mathbf{t}$ or $\mathbf{f}$ on every v, or for all participants it returns $\mathbf{b}$. This reflects that we do not designate a unique sender:

Remark 4.1.6 Implementations of Bracha Broadcast assume a unique *sender* participant whose identity is known to all participants and whose identity cannot be faked. The sender's task is to pick a unique value v and broadcast v (and, if the sender is honest, only v) to all participants.

In our logic, we model this with a straightforward axiom $\exists_{\mathbf{1}} a.\Diamond\,\mathsf{bcst}(a)$. Our logic is three-valued, and as per Proposition 3.1.3(5) this means that there is at most one value such that somebody **t**-sends it, but there may be many values such that somebody **b**-sends them. We return to this in Remark 4.2.5, after we have a few more proofs.

4.2 Correctness properties for Bracha Broadcast

4.2.1 Statement of the correctness properties

Remark 4.2.1 For the rest of this section, we fix a model (Definition 3.2.2)

$$\mu = (\mathsf{Value}, (\mathsf{P}, \mathsf{Open}), \varsigma)$$

such that the underlying semitopology $(\mathsf{P}, \mathsf{Open})$ is 3-twined (Definition 2.4.1), so that we have Theorem 2.4.3.

Remark 4.2.2 As promised (see end of Remark 4.1.1), we will now use ThyBB in Figure 6 to derive predicates corresponding to the following standard correctness properties for Bracha Broadcast, per Figure 7. We consider them in turn:

1. **Validity:** *If a correct p broadcasts a value v, then every correct p' delivers v.*
   This becomes ThyBB $\vDash_\mu \Diamond\,\mathsf{bcst}(v) \to \Box\,\mathsf{dlvr}(v)$. See Proposition 4.2.9.
2. **No duplication:** *Every correct p delivers at most one value.*
   This becomes ThyBB $\vDash_\mu \exists_{\mathbf{01}}\mathsf{dlvr}$. See Proposition 4.2.11.
3. **Integrity:** *If some p delivers a message m with correct sender p', then m was broadcast by p'.*
   This becomes ThyBB $\vDash_\mu \mathsf{dlvr}(v) \to \Diamond\,\mathsf{bcst}(v)$. See Proposition 4.2.12.
4. **Consistency:** *If correct p delivers v and another correct p' delivers v' then $v{=}v'$.*
   This becomes ThyBB $\vDash_\mu \exists_{\mathbf{01}}\Diamond\,\mathsf{dlvr}(v)$. See Proposition 4.2.11.
5. **Totality:** *If correct p delivers v, then every correct p' delivers v.*
   This becomes ThyBB $\vDash_\mu \Diamond\,\mathsf{dlvr}(v) \to \Box\,\mathsf{dlvr}(v)$. See Proposition 4.2.13.

The English statements above are (in order) adapted from properties BCB1, BCB2, BCB3, and BCB4 from Module 3.11, and BRB5 from Module 3.12 of [CGR11].

Remark 4.2.3 We can make a few observations about the correctness properties above:

1. The logical statements are concise, precise, and (arguably) clear.
2. The rendering of Consistency $\exists_{\mathbf{01}}\Diamond\,\mathsf{dlvr}(v)$ subsumes that for No Duplication $\exists_{\mathbf{01}}\mathsf{dlvr}(v)$. This is less visible in the English because Consistency talks about a 'correct p' and 'another correct p'', which might be taken to indicate that $p \neq p'$. In our logic, it is easier to unify Consistency and No Duplication into a single predicate and proof.
3. The English statement of Integrity was hard to parse, at least for me. Contrast with the predicate $\mathsf{dlvr}(v) \to \Diamond\,\mathsf{bcst}(v)$, which is clear and precise.

4.2.2 Validity: "if a correct p broadcasts v, then every correct p' delivers v"

Lemma 4.2.4 *Suppose $\vDash_\mu$ ThyBB. Then precisely one of the following holds:*

$$\forall p \in \mathsf{P}.p \vDash_\mu \mathsf{TF}[\mathsf{bcst}] \wedge \exists_1 v \in \mathsf{Value}.\forall p \in \mathsf{P}.(p \vDash_\mu \mathsf{T}\Diamond\,\mathsf{bcst}(v)) \quad \textit{or} \quad \forall v \in \mathsf{Value}, p \in \mathsf{P}.(p \vDash_\mu \mathsf{B}\,\mathsf{bcst}(v)).$$

Proof. $(\mathsf{BrCorrect}'')$ asserts that either for all participants the bcst predicate returns **t** or **f** on every v, or for all participants it returns **b**. In the latter case we have the right-hand disjunct above, and in the former case using $(\mathsf{BrBroadcast1})$ we obtain the left-hand disjunct. □

Remark 4.2.5 Lemma 4.2.4 requires some discussion. Our description of Bracha Broadcast in Remark 4.1.1 assumes a designated sender with an unforgeable signature, who might be byzantine. This is an accurate description of the implementation, but logically we do not care that there is one sender. There could be *two* senders, so long as (if honest) they broadcast identical values.

Implementationally we guarantee this by designating a unique sender participant. In the axiomatisation, it is convenient (and more general) to be more abstract: we axiomatise a broadcast predicate that is either somewhere $\mathbf{t}$ for *precisely* one value (corresponding to a value broadcast by an honest sender), or it is $\mathbf{b}$ for all values and all participants (corresponding to arbitrary hostile behaviour of a dishonest sender).[16]

Lemma 4.2.6 *Suppose* $\models_\mu$ ThyBB *and* $v \in \mathsf{Value}$. *Then:*

(1) $\models_\mu \Diamond\mathsf{bcst}(v) \to \mathsf{echo}(v)$.
(2) $\models_\mu \Diamond\mathsf{bcst}(v) \to \Box\,\mathsf{echo}(v)$.
(3) $\models_\mu \boxdot\mathsf{echo}(v) \to \Box\,\mathsf{ready}(v)$.
(4) $\models_\mu \boxdot\mathsf{ready}(v) \to \Box\,\mathsf{dlvr}(v)$.

Proof. We consider each part in turn, freely using weak modus ponens (Prop. 2.2.2(4)) and the semantics in Figure 4:

(1) Consider $p \in \mathsf{P}$ and suppose $p \models_\mu \mathsf{T}\Diamond\mathsf{bcst}(v)$. By weak modus ponens (Prop. 2.2.2(4)) and (BrEcho!) $p \models_\mu \exists a.\mathsf{echo}(a)$. Thus, there exists $v' \in \mathsf{Value}$ such that $p \models_\mu \mathsf{echo}(v')$. There are now two subcases:
   - *Suppose* $p \models_\mu \mathsf{B}\,\mathsf{echo}(v')$. By (BrCorrect′) $p \models_\mu \mathsf{B}\,\mathsf{echo}(v)$, and so $p \models_\mu \mathsf{echo}(v)$.
   - *Suppose* $p \models_\mu \mathsf{TF}\,\mathsf{echo}(v')$. By Proposition 2.2.2(8) (since $p \models_\mu \mathsf{echo}(v')$) $p \models_\mu \mathsf{T}\,\mathsf{echo}(v')$, and so by (BrEcho?) $p \models_\mu \Diamond\mathsf{bcst}(v')$. By Lemma 4.2.4 $v = v'$.

(2) From part 1 of this result, noting that the $p \in \mathsf{P}$ we chose in the proof was arbitrary.
(3) Suppose $p \models_\mu \mathsf{T}\boxdot\mathsf{echo}(v)$. By (BrReady!) $p \models_\mu \mathsf{ready}(v)$. This reasoning did not depend on p, so $\models_\mu \Box\,\mathsf{ready}(v)$ as required.
(4) As the reasoning for part 3, but using (BrDeliver!). $\Box$

Remark 4.2.7 For clarity we spell out what Lemma 4.2.6 asserts. Each part has the form $\models_\mu \phi \to \psi$, meaning that for every $p \in \mathsf{P}$ if $p \models_\mu \mathsf{T}\phi$ (i.e. $[\![\phi]\!]_\mu(p) = \mathbf{t}$) then $p \models_\mu \psi$ (i.e. $[\![\psi]\!]_\mu(p) \in \{\mathbf{t}, \mathbf{b}\}$). So part 1 means '$p \models_\mu \mathsf{T}\Diamond\mathsf{bcst}(v)$ implies $p \models_\mu \mathsf{echo}(v)$'.

Note it does *not* mean '$p \models_\mu \Diamond\mathsf{bcst}(v)$ implies $p \models_\mu \mathsf{echo}(v)$'. That would be a different assertion.

Lemma 4.2.8 *Suppose* $\models_\mu$ ThyBB *and* $v \in \mathsf{Value}$. *Then:*

(1) $\models_\mu \Box\,\mathsf{echo}(v)$ *implies* $\models_\mu \mathsf{T}\boxdot\mathsf{echo}(v)$.
(2) $\models_\mu \Box\,\mathsf{ready}(v)$ *implies* $\models_\mu \mathsf{T}\boxdot\mathsf{ready}(v)$.

Proof. We consider each part in turn:

(1) Suppose $\models_\mu \Box\,\mathsf{echo}(v)$. By (BrCorrect) $\models_\mu \boxdot\mathsf{TF}[\mathsf{echo}]$ – by Figure 3 this means $\models_\mu \boxdot\forall v.\mathsf{TF}\,\mathsf{echo}(v)$ – thus $\models_\mu \boxdot\mathsf{TF}\,\mathsf{echo}(v)$ and by Proposition 2.2.2(8) $\models_\mu \mathsf{T}\boxdot\mathsf{echo}(v)$.
(2) As for part 1, since by (BrCorrect) $\models_\mu \boxdot\mathsf{TF}[\mathsf{ready}]$. $\Box$

Proposition 4.2.9 (Validity) *If* $\models_\mu$ ThyBB *and* $v \in \mathsf{Value}$ *then*

$$\models_\mu \Diamond\mathsf{bcst}(v) \to \Box\,\mathsf{dlvr}(v).$$

Proof. We reason using weak modus ponens (Prop. 2.2.2(4)). Suppose $\models_\mu \mathsf{T}\Diamond\mathsf{bcst}(v)$. By Lemma 4.2.6(2) $\models_\mu \Box\,\mathsf{echo}(v)$, and by Lemma 4.2.8(1) $\models_\mu \mathsf{T}\boxdot\mathsf{echo}(v)$. By Lemma 4.2.6(3) $\models_\mu \Box\,\mathsf{ready}(v)$, and by Lemma 4.2.8(2) $\models_\mu \mathsf{T}\boxdot\mathsf{ready}(v)$. By Lemma 4.2.6(4) $\models_\mu \Box\,\mathsf{dlvr}(v)$ as required. $\Box$

4.2.3 Consistency & No duplication: "if correct p delivers v and correct (possibly equal) p' delivers v' then $v = v'$"

Lemma 4.2.10 *Suppose* $\models_\mu$ (BrCorrect) *(in particular, it suffices that* $\models_\mu$ ThyBB*). Then for* $v, v' \in \mathsf{Value}$*:*

(1) *If* $\models_\mu \boxdot\mathsf{ready}(v)$ *then* $\models_\mu \mathsf{T}\Diamond\mathsf{ready}(v)$.

[16] A longer stronger axiomatisation that is closer to the implementation could be written but would be more complex and not buy us much, since we can prove our correctness results from the weaker and simpler formulation.

(2) *If* $\vDash_\mu \boxdot\mathsf{echo}(v) \land \boxdot\mathsf{echo}(v')$ *then* $\vDash_\mu \mathsf{T}\Diamond(\mathsf{echo}(v) \land \mathsf{echo}(v'))$.
(3) *If* $\vDash_\mu \boxdot\mathsf{ready}(v) \land \boxdot\mathsf{ready}(v')$ *then* $\vDash_\mu \mathsf{T}\Diamond(\mathsf{ready}(v) \land \mathsf{ready}(v'))$.

Proof. We consider each part in turn:

(1) Suppose $\vDash_\mu \boxdot\mathsf{ready}(v)$. From $(\mathsf{BrCorrect})$ also $\vDash_\mu \boxdot\mathbb{TF}\,\mathsf{ready}(v)$. Using Theorem 2.4.3 $\vDash_\mu \Diamonddot\mathsf{T}\,\mathsf{ready}(v)$ and so $\vDash_\mu \mathsf{T}\Diamonddot\mathsf{ready}(v)$ as required.
(2) Suppose $\vDash_\mu \boxdot\mathsf{echo}(v) \land \boxdot\mathsf{echo}(v')$. From $(\mathsf{BrCorrect})$ we have $\vDash_\mu \boxdot\mathbb{TF}[\mathsf{echo}]$. Using Theorem 2.4.3 and Lemma 2.3.7(3) (or just direct from the 3-twined property in Definition 2.4.1) $\vDash_\mu \mathsf{T}\Diamond(\mathsf{echo}(v) \land \mathsf{echo}(v'))$ as required.
(3) As for the proof of part 2, noting that $\vDash_\mu \boxdot\mathbb{TF}[\mathsf{ready}]$ from $(\mathsf{BrCorrect})$. □

Proposition 4.2.11 (Consistency & No Duplication) *Suppose* $\vDash_\mu$ ThyBB *and* $v, v' \in \mathsf{Value}$. *Then*

$$\vDash_\mu \exists_{\mathbf{01}}\Diamond\,\mathsf{dlvr}(v).$$

Proof. By Proposition 3.1.3(1) it suffices to show for every $v, v' \in \mathsf{Value}$ that $\vDash_\mu \mathsf{T}\Diamond\mathsf{dlvr}(v) \land \mathsf{T}\Diamond\mathsf{dlvr}(v')$ implies $v = v'$. So suppose $\vDash_\mu \mathsf{T}\Diamond\mathsf{dlvr}(v)$ and $\vDash_\mu \mathsf{T}\Diamond\mathsf{dlvr}(v')$. By $(\mathsf{BrDeliver?})$ $\vDash_\mu \boxdot\mathsf{ready}(v)$ and $\vDash_\mu \boxdot\mathsf{ready}(v')$, so by Lemma 4.2.10(3) $\vDash_\mu \mathsf{T}\Diamond(\mathsf{ready}(v) \land \mathsf{ready}(v'))$. By $(\mathsf{BrReady?})$ $\vDash_\mu \boxdot\mathsf{echo}(v)$ and $\vDash_\mu \boxdot\mathsf{echo}(v')$, so by Lemma 4.2.10(2) $\vDash_\mu \mathsf{T}\Diamond(\mathsf{echo}(v) \land \mathsf{echo}(v'))$. It follows using $(\mathsf{BrEcho01})$ and Proposition 3.1.3(5) that $v = v'$ as required. □

4.2.4 Integrity: "if some p delivers a message m with correct sender p′, then m was broadcast by p′"

Proposition 4.2.12 (Integrity) *Suppose* $\vDash_\mu$ ThyBB. *Then*

$$\vDash_\mu \mathsf{dlvr}(v) \to \Diamond\,\mathsf{bcst}(v).$$

Proof. We reason using weak modus ponens (Prop. 2.2.2(4)). Suppose $p \in \mathsf{P}$ and $p \vDash_\mu \mathsf{T}\,\mathsf{dlvr}(v)$. By $(\mathsf{BrDeliver?})$ $\vDash_\mu \boxdot\mathsf{ready}(v)$. By Lemma 4.2.10(1) $\vDash_\mu \mathsf{T}\Diamonddot\mathsf{ready}(v)$, and by Lemma 2.3.7(2) $\vDash_\mu \mathsf{T}\Diamond\,\mathsf{ready}(v)$. By $(\mathsf{BrReady?})$ $\vDash_\mu \boxdot\mathsf{echo}(v)$, and by Lemmas 4.2.10(1) Lemma 2.3.7(2) $\vDash_\mu \mathsf{T}\Diamond\,\mathsf{echo}(v)$. Using $(\mathsf{BrEcho?})$ $\vDash_\mu \Diamond\,\mathsf{bcst}(v)$, as required. □

4.2.5 Totality: if correct p delivers v, then every correct p′ delivers v

Proposition 4.2.13 (Totality) *Suppose* $\vDash_\mu$ ThyBB *and* $v \in \mathsf{Value}$. *Then:*

$$\vDash_\mu \Diamond\,\mathsf{dlvr}(v) \;\to\; \Box\,\mathsf{dlvr}(v).$$

Proof. We reason using weak modus ponens (Prop. 2.2.2(4)). Suppose $\vDash_\mu \mathsf{T}\Diamond\mathsf{dlvr}(v)$. By $(\mathsf{BrDeliver?})$ $\vDash_\mu \boxdot\mathsf{ready}(v)$, so by Lemma 4.2.10(1) $\vDash_\mu \mathsf{T}\Diamonddot\mathsf{ready}(v)$. Using $(\mathsf{BrReady!!})$ (applied at every point) $\vDash_\mu \Box\,\mathsf{ready}(v)$, so by Lemma 4.2.8(2) $\vDash_\mu \mathsf{T}\boxdot\mathsf{ready}(v)$. Using $(\mathsf{BrDeliver!})$ (applied at every point) $\vDash_\mu \Box\,\mathsf{dlvr}(v)$. □

Remark 4.2.14 Let us take a moment to unpack the statement of Totality in Proposition 4.2.13 and remember what the symbols mean. As per Figure 4 this asserts that $[\![\Diamond\,\mathsf{dlvr}(v) \to \Box\,\mathsf{dlvr}(v)]\!]_\mu(p)$ has truth-value $\mathbf{t}$ or $\mathbf{b}$ for every $p \in \mathsf{P}$. The choice of p clearly does not matter here because both parts of the implication are modal. Unpacking the truth-table of $\to$ from Figure 1, it suffices to show that if the left-hand side has truth-value $\mathbf{t}$ then the right-hand side has truth-value $\mathbf{t}$ or $\mathbf{b}$.

So suppose $[\![\Diamond\,\mathsf{dlvr}(v)]\!]_\mu = \mathbf{t}$, indicating that some honest participant delivers v. The proof of Proposition 4.2.13 then shows that: every other honest participant delivers v; for any dishonest participants $\mathsf{dlvr}(v)$ has truth-value $\mathbf{b}$; and therefore, the truth-value of $\Box\,\mathsf{dlvr}(v)$ is $\mathbf{t}$ or (if dishonest participants are involved) $\mathbf{b}$.

The statement and proof of Proposition 4.2.13 do not reason explicitly on correct or byzantine participants. They do not need to: this detail is all packaged neatly into how the modalities and implication work. The proof is routine, following the structure of the axioms and applying lemmas in a natural way.

It is easy to take this magic for granted: how logic turns reasoning into (almost) routine symbolic manipulation. But of course, this is why logical specifications can be so helpful. Declarative logical specifications of the kind we see above are particularly powerful, because of how concise they can be.

4.3 Further discussion of the axioms

We take a moment to return to the axioms in Figure 6, to discuss some design alternatives.

Axioms (BrDeliver!) and (BrReady!) in Figure 6 are slightly stronger than absolutely necessary. In the presence of the other axioms, it suffices to assume $\exists a.\boxdot\mathsf{ready}(a) \to \exists a.\mathsf{dlvr}(a)$ and $\exists a.\boxdot\mathsf{echo}(a) \to \exists a.\mathsf{ready}(a)$. It then is not hard to use the backwards rules to derive the stronger forms of the axioms as presented in Figure 6.

The same is not true of (BrReady!!). A weaker axiom $\exists a.\diamondsuit\mathsf{ready}(a) \to \exists a.\mathsf{ready}(a)$ would *not* suffice and the stronger form of the axiom cannot be derived from it.

Conversely, changing (BrEcho!) to $\diamond\,\mathsf{bcst}(a) \to \mathsf{echo}(a)$ would be wrong. Consider the case of a byzantine participant who (dishonestly; contrary to the protocol) broadcasts v and v'. In our logic this would give $\diamond\,\mathsf{bcst}(v)$ and $\diamond\,\mathsf{bcst}(v')$ truth-value $\mathbf{b}$. According to the truth-table for $\to$ in Figure 1, by our modified (BrEcho!) rule, $\mathsf{echo}(v)$ and $\mathsf{echo}(v')$ would then have to have truth-value $\mathbf{b}$ or $\mathbf{t}$. For an honest participant this would mean that both $\mathsf{echo}(v)$ and $\mathsf{echo}(v')$ would have to have truth-value $\mathbf{t}$. This would contradict (BrEcho01), which implies that no honest participant may echo more than one value.

Clause 3 of Remark 4.1.1 renders into logic as a forward rule $(\boxdot\mathsf{echo}(a) \lor \diamondsuit\mathsf{ready}(a)) \to \mathsf{ready}(a)$. This rule appears more-or-less verbatim in Figure 6 as (BrReady!) and (BrReady!!). Yet in Figure 6 we only have the backward rule (BrReady?) as $\mathsf{ready}(a) \to \boxdot\mathsf{echo}(a)$. Why not a rule (BrReady?′) of the form $\mathsf{ready}(a) \to (\boxdot\mathsf{echo}(a) \lor \diamondsuit\mathsf{ready}(a))$? Algorithmically this is inefficient, because if somebody sends a ready message then by an inductive argument a quorum of echoes must exist. In the logic, (BrReady?′) would actually be *too weak*; it would permit a model in which every participant does $\mathsf{ready}(v)$, and for each p (BrReady?′) would be valid at p because of the contraquorum of all the *other* p' who also do $\mathsf{ready}(v)$. This is related to the observations Remark 4.1.3: in the algorithmic world we build things, so things *can't* happen unless we say they *do*; in the logical world we verify things, so things *can* happen unless we say they *don't*.

5 Crusader Agreement

5.1 Motivation and presenting the protocol

Remark 5.1.1 In Section 4 we gave a declarative axiomatisation of the Bracha Broadcast protocol and we showed how to prove its correctness properties by axiomatic reasoning.

We will now do the same for a Crusader Agreement algorithm as presented in an online exposition [IA22]. This exposition was written for the Decentralized Thoughts website [17] by the authors of a journal paper [ABDY22], to provide an accessible introduction to their paper.

Agreement is arguably a moderate step up in complexity from Broadcast, because:

(1) With Broadcast, we want all correct participants to agree on a value broadcast by a single designated (possibly faulty) broadcasting participant.
(2) With Agreement, we want all correct participants to agree on some value, where all participants start off with their own individual values. While still simple, it is slightly more challenging in the sense that correct participants can propose different values whereas in Broadcast they cannot.

Definition 5.1.2

(1) Figure 8 presents the pseudocode from [IA22].
(2) Figure 9 presents THYCA, a corresponding declarative specification in axiomatic style.

Remark 5.1.3 The authors of [IA22] are not completely explicit, but it becomes clear from context, that:

(1) Clauses in Figure 8 are intended to be executed in parallel (not necessarily sequentially).
(2) Each participant starts with their own input value, which may be different from the input value(s) of other participants.

Intuitively, in this algorithm participants try to agree participants on a value 0 or 1, and if a participant can see no clear choice of value to agree on, then it might output a special 'agreement failed' value. Here

[17] Decentralized Thoughts is a well-regarded source of technical commentary on decentralised protocols.

```
input: v (0 or 1)
send <echo1, v> to all
if didnt send <echo1, 1-v> and see f+1 <echo1, 1-v>
    send <echo1, 1-v> to all
if didnt send <echo2, *> and see n-f <echo1, w>
    send <echo2, w>
if see n-f <echo2, u> and n-f <echo1, u>
    output(u)
if see n-f <echo1, 0> and n-f <echo1, 1>
    output(0.5)
```

Fig. 8. Pseudocode Crusader Agreement algorithm from [IA22] (Definition 5.1.2)

Backward rules

$(\mathsf{CaEcho1?})$	$\mathsf{echo}_1(a) \rightharpoonup \Diamond\,\mathsf{input}(a)$
$(\mathsf{CaEcho2?})$	$\mathsf{echo}_2(a) \to \boxdot\mathsf{echo}_1(a)$
$(\mathsf{CaOutput?})$	$(\mathsf{output}(0) \to \boxdot\mathsf{echo}_2(0)) \land (\mathsf{output}(1) \to \boxdot\mathsf{echo}_2(1))$
$(\mathsf{CaOutput'?})$	$\mathsf{output}(0.5) \to (\boxdot\mathsf{echo}_1(0) \land \boxdot\mathsf{echo}_1(1))$

Other rules

$(\mathsf{CaCorrect})$	$\boxdot\mathsf{TF}[\mathsf{input}, \mathsf{echo}_1, \mathsf{echo}_2, \mathsf{output}]$
$(\mathsf{CaCorrect'})$	$\mathsf{TF}[\mathsf{P}] \lor \mathsf{B}[\mathsf{P}] \quad (\mathsf{P}{\in}\{\mathsf{input}, \mathsf{echo}_1, \mathsf{echo}_2, \mathsf{output}\})$
$(\mathsf{CaInput})$	$\big((\mathsf{input}(0) \oplus \mathsf{input}(1)) \land \neg\mathsf{input}(0.5)\big)$
$(\mathsf{CaEcho2}_{01})$	$\exists_{\mathbf{01}} a.\mathsf{echo}_2(a)$

Forward rules

$(\mathsf{CaEcho1!})$	$(\mathsf{input}(a) \lor \Diamond\mathsf{echo}_1(a)) \to \mathsf{echo}_1(a)$
$(\mathsf{CaEcho2!})$	$\exists a.\boxdot\mathsf{echo}_1(a) \to \exists a.\mathsf{echo}_2(a)$
$(\mathsf{CaOutput!})$	$\boxdot\mathsf{echo}_2(a) \to \mathsf{output}(a)$
$(\mathsf{CaOutput'!})$	$(\boxdot\mathsf{echo}_1(0) \land \boxdot\mathsf{echo}_1(1)) \to \mathsf{output}(0.5)$

As standard, free a above are assumed universally quantified (i.e. there is an invisible $\forall a$ at the start of any axiom with a free a). The axioms above assume predicate symbols $\mathcal{P} = \{\mathsf{input}, \mathsf{echo}_1, \mathsf{echo}_2, \mathsf{output}\}$ (as explained in Definition 3.2.1).

Fig. 9. ThyCA: axioms of Crusader Agreement (Definition 5.1.2)

we write this value as 0.5; in [IA22] they write $\bot$ (a riff on the domain-theoretic 'undefined' element), but that clashes with our use in this paper of $\bot$ for logical falsity in Figure 3.

Definition 5.1.4 Relevant correctness properties for Crusader Agreement as per [IA22] are presented in Figure 10. We consider them in turn:

(1) **Weak Agreement:** *If two non-faulty parties output values v and v', then either $v = v'$ or one of the values is* 0.5.
This becomes ThyCA $\vDash_\mu (\Diamond\,\mathsf{output}(v) \land \Diamond\,\mathsf{output}(v')) \rightharpoonup v{=}v'$. See Proposition 5.3.2.

(2) **Validity:** *If all non-faulty parties have the same input, then this is the only possible output* of any

$(\mathsf{CaAgree})$	$\textsc{ThyCA} \models_\mu (\Diamond\,\mathsf{output}(v) \land \Diamond\,\mathsf{output}(v')) \rightharpoonup v{=}v'$	(Proposition 5.3.2)
$(\mathsf{CaValid1})$	$\textsc{ThyCA} \models_\mu (\mathbb{TB}\,\Box\,\mathsf{input}(v) \land \mathsf{output}(v')) \rightharpoonup v{=}v'$	(Proposition 5.3.5)
$(\mathsf{CaValid2})$	$\textsc{ThyCA} \models_\mu \Diamond\,\mathsf{output}(v) \rightharpoonup \Diamond\,\mathsf{input}(v)$	(Proposition 5.3.5)
(CaLive)	$\textsc{ThyCA} \models_\mu \Box\,\exists v.\mathsf{output}(v)$	(Proposition 5.3.10)

Fig. 10. Correctness properties for ThyCA (Definition 5.1.4)

non-faulty party.[18] *Furthermore, if a non-faulty party outputs $v \neq 0.5$, then v was the input of some non-faulty party.*
These become $\textsc{ThyCA} \models_\mu (\Box\,\mathsf{input}(v) \land \mathsf{output}(v')) \rightharpoonup v{=}v'$ and $\textsc{ThyCA} \models_\mu \Diamond\,\mathsf{output}(v) \rightharpoonup \Diamond\,\mathsf{input}(v)$. See Proposition 5.3.5.

(3) **Liveness:** *If all non-faulty parties start the protocol then all non-faulty parties eventually output a value.*
This becomes $\textsc{ThyCA} \models_\mu \Box\,\exists v.\mathsf{output}(v)$. See Proposition 5.3.10.

5.2 *Discussion of the axioms*

Remark 5.2.1 As for the axioms in Figures 2 and 6, the axioms in Figure 9 are grouped into *backward rules* with names ending in ? which represent reasoning of the form "if *this* happens, then *that* must have happened"; *forward rules* have with ending with ! which represent forward reasoning of the form "if *this* happens, then *that* must happen"; and *other rules* which reflect particular conditions of the algorithms.

Recall that each axiom is implicitly universally quantified over a if required, and it is valid of a model when it is valid (i.e. it returns $\mathbf{t}$ or $\mathbf{b}$, but not $\mathbf{f}$) when evaluated at every $p \in \mathsf{P}$.

Remark 5.2.2 We discuss each axiom in Figure 2 in turn:

(1) $(\mathsf{CaEcho1?})$ says: if $\mathsf{echo}_1(v)$ is $\mathbf{t}$ at p then $\mathsf{input}(v)$ is $\mathbf{t}$ at some p'.
This axiom does not immediately correspond to lines 2 to 4 in Figure 8 — contrast with $(\mathsf{CaEcho!})$, which more clearly corresponds to the algorithm. We explain and discuss this difference below in Remark 5.2.3.

(2) $(\mathsf{CaEcho2?})$ says: if $\mathsf{echo}_2(v)$ is $\mathbf{t}$ at p then for a quorum of participants $\mathsf{echo}_1(v)$ is $\mathbf{t}$ or $\mathbf{b}$. This corresponds to part of lines 5 and 6, since it is clear that if a participant performs $\mathsf{echo}_2(v)$ then it must have seen a quorum of $\mathsf{echo}_1(v)$.
The condition `if didnt send <echo2, *>` on line 5 is captured by $(\mathsf{CaEcho2_{01}})$, which just asserts that no correct participant may `echo2` *twice*.

(3) $(\mathsf{CaOutput?})$ and $(\mathsf{CaOutput!})$ reflect lines 7 to 10, except for one detail: lines 7 to 10 suggest a pair of axioms

$$\mathsf{output}(a) \rightarrow (\boxdot\mathsf{echo}_2(a) \land \boxdot\mathsf{echo}_1(a)) \qquad (\boxdot\mathsf{echo}_2(a) \land \boxdot\mathsf{echo}_1(a)) \rightarrow \mathsf{output}(a)$$

but instead in Figure 9 we have axioms

$$\mathsf{output}(a) \rightarrow \boxdot\mathsf{echo}_2(a) \qquad \boxdot\mathsf{echo}_2(a) \rightarrow \mathsf{output}(a).$$

Why the difference?
Because, in fact, the right-hand conjunct in the algorithm from [IA22] is redundant.[19] We could include it in the axioms and it would do no harm, but it also makes no difference to the proofs and would take up space.

[18] I added the 'for any non-faulty party' (the original source material [IA22] did not say this) because from context it is clear that this is what is intended. Faulty parties are not constrained.
[19] This became clear while doing the axiomatic proofs; the right-hand conjunct was never used. In fact the same is true in the source article, but this is harder to spot because its reasoning is less explicitly axiomatic.

One of the algorithm's authors [20] confirms this: the right-hand conjunct was left in by accident. It gives stronger correctness properties that are treated in the paper [ABDY22], but they are not mentioned in the (simplified) online article [IA22].

(4) ($\mathsf{CaOutput}'$?) asserts that if we output 0.5 then we must have seen quorums of echo_1 for both 0 and 1, as per lines 9 and 10 in Figure 8.

(5) ($\mathsf{CaCorrect}$) asserts that there exists a quorum of correct participants, by which we mean precisely that that they all return $\mathbf{t}$ or $\mathbf{f}$ truth-values for input, echo_1, echo_2, and output, and they never return $\mathbf{b}$.

(6) ($\mathsf{CaCorrect}'$) asserts that *either* a participant is correct and always returns correct truth-values — *or* it always returns the faulty truth-value $\mathbf{b}$.

The reader might ask why a faulty participant must always return $\mathbf{b}$; what if it returns $\mathbf{b}$ just sometimes? The reason is that this logically captures worst-case behaviour. Specifically, the forward and backward rules are all implications, and it is a fact that if the left-hand side of an implication is $\mathbf{b}$, then the implication overall returns $\mathbf{b}$ or $\mathbf{t}$ and so is valid; the reader can check this in Figure 1, just by observing that the row for $\mathbf{b}$ in the truth-tables for $\rightarrow$ and $\rightharpoonup$ is '$\mathbf{t}$ $\mathbf{b}$ $\mathbf{b}$' and this does not mention $\mathbf{f}$.

So, this makes $\mathbf{b}$ behave like a 'worst case truth-value' in the sense that all forward and backward rules are valid if the left-hand side returns $\mathbf{b}$, so that in effect the rule does not constrain behaviour involving faulty participants. But, we do not need to keep saying 'for a non-faulty participant', because the three-valued logic takes care of all this for us, automatically.

(7) ($\mathsf{CaInput}$) expresses line 1: a correct participant inputs either 0 or 1.

(8) ($\mathsf{CaEcho1!}$) expresses the forward version of lines 2 to 4; echo1 the value of your input and echo1 the value of any contraquorum that you see.

(9) ($\mathsf{CaEcho2!}$) expresses the forward version of lines 5 and 6. We need the existential quantifier because in this case, we only echo2 the value of the first quorum of echo1 that we see.

(10) ($\mathsf{CaOutput!}$) was discussed above. We would just add here that $\mathsf{echo}_2(0.5)$ is impossible by Lemma 5.4.2(3): that is, in the presence of the other axioms, we do not need an explicit side-condition that $a \neq 0.5$ because that turns out to be a lemma.

(11) ($\mathsf{CaOutput}'$!) corresponds directly to lines 9 and 10 in Figure 8 (and is just the reverse of ($\mathsf{CaOutput}'$?)).

Remark 5.2.3 (CaEcho?) is rather special. A direct reading lines 2 and 3 of the algorithm from Definition 5.1.2 suggests we write the following property:

$$\mathsf{echo}_1(v) \rightarrow (\mathsf{input}_1(v) \vee \diamondplus \mathsf{echo}_1(v)).$$

Indeed, the forward rule ($\mathsf{CaEcho!}$) looks just like that, just in the other direction. So why did we not write the predicate above as our (CaEcho?)?

Because it is subtly too weak. The axiom above would permit a model in which (for example) all participants are correct, and they all input 0, but they all perform $\mathsf{echo}_1(1)$. Note that $\mathsf{echo}_1(1)\rightarrow\diamondplus\mathsf{echo}_1(1)$ is valid of this model. Clearly we want to exclude it, and no practical run of the algorithm would generate it, but logically speaking it perfectly well satisfies the restriction above.

What is going on here is that our model has no notion of time, so our model has no notion of a 'first' echo_1 that would need to be justified by an input.

We could now be literal and introduce a notion of time, either as an explicit timestamp parameter to each echo_1, or we could add a notion of time to the modal context in the logic itself. There would be nothing wrong with that — it reflects what the algorithm actually does and is technically entirely feasible.

But we will prefer a simpler, more general, and arguably more elegant approach. [21] To explain how this works, we reason as follows: we observe of the algorithm that if a correct participant performs $\mathsf{echo}_1(v)$ then either *it* has input v, or $f{+}1$ participants — and thus in particular one correct participant — also have performed $\mathsf{echo}_1(v)$. By an induction on time, we see that if a correct participant performs $\mathsf{echo}_1(v)$ then some some correct participant has input v. This justifies writing (CaEcho?) as we do in Figure 9.

[20] Ittai Abraham; private correspondence.

[21] But as always: what you axiomatise depends on what you want to represent. Today, we might make one design decision about how much detail we want to include in our model. Tomorrow, or for a different protocol, or for a different intended audience, it would be perfectly legitimate to choose different tradeoffs.

It is not immediately obvious that this suffices to prove our correctness properties; we have to check that in the proofs. And, it turns out that the proofs work well, which gives a formal sense in which the ($\mathsf{CaEcho?}$) axiom in Figure 9 is correct and in fact, it is more general than the more literal rendering.[22]

5.3 *Proofs of correctness properties*

5.3.1 *Weak Agreement*

Remark 5.3.1 The proof of Proposition 5.3.2 is straightforward. Note how the 'non-faulty' precondition is integrated into the use of $\rightharpoonup$, which assumes that the left-hand side returns $\mathbf{t}$ (the non-faulty truth-value).[23]

Proposition 5.3.2 (Weak Agreement) *Suppose* $\vDash_\mu$ ThyCA *and* $v, v' \in \{0,1\}$. *Then*

$$\vDash_\mu (\Diamond \mathsf{output}(v) \wedge \Diamond \mathsf{output}(v')) \rightharpoonup v{=}v'.$$

This formalises the following informal statement:

"If two non-faulty parties output values v and v', then either $v = v'$ or one of the values is 0.5"

Proof. Using strong modus ponens (Prop. 2.2.2(5)) it suffices to assume $\vDash_\mu \mathsf{T}\Diamond \mathsf{output}(v) \wedge \mathsf{T}\Diamond \mathsf{output}(v')$ and prove $v = v'$.

So suppose $\vDash_\mu \mathsf{T}\Diamond \mathsf{output}(v) \wedge \mathsf{T}\Diamond \mathsf{output}(v')$. By ($\mathsf{CaOutput?}$) $\vDash_\mu \boxdot \mathsf{echo}_2(v) \wedge \boxdot \mathsf{echo}_2(v')$. By ($\mathsf{CaCorrect}$) $\vDash_\mu \boxdot \mathsf{TF}[\mathsf{echo}_2]$. By Theorem 2.4.3 and Lemma 2.3.7(3) $\vDash_\mu \mathsf{T}\Diamond(\mathsf{echo}_2(v) \wedge \mathsf{echo}_2(v'))$. By ($\mathsf{CaEcho2}_{01}$) and Proposition 3.1.3(1.a&1.e) $v = v'$ as required. □

Example 5.3.3 output is not functional, by which we mean that $[\![\exists_{\mathbf{01}} v.\mathsf{output}(v)]\!]_\mu$ need not be equal to $\mathbf{t}$. We give an example to show that is possible to have $p \vDash_\mu \mathsf{T}(\mathsf{output}(0.5) \wedge \mathsf{output}(1))$.

Consider a model with an even number of participants; all participants are correct; and half of them input 0 and the other half input 1. The reader can check that every participant necessarily produces $\mathsf{echo}_1(0)$ and $\mathsf{echo}_1(1)$, so that all participants $\mathsf{output}(0.5)$. It is also consistent with the axioms for every participant to produce $\mathsf{echo}_2(1)$ and $\mathsf{output}(1)$.

5.3.2 *Validity*

We work towards Proposition 5.3.5; we will need Lemma 5.3.4:

Lemma 5.3.4 *Suppose* $\vDash_\mu$ ThyCA *and* $v \in \{0, 0.5, 1\}$ *and* $p \in \mathsf{P}$. *Then:*

(1) $p \vDash_\mu \mathsf{input}(v)$ *if and only if* $p \vDash_\mu \neg\mathsf{input}(1\text{-}v)$.
(2) $p \vDash_\mu \mathsf{T}\,\mathsf{input}(v)$ *and* $p \vDash_\mu \mathsf{T}\,\mathsf{input}(v')$ *implies* $v = v'$.
(3) $p \vDash_\mu \Box\,\mathsf{input}(v)$ *and* $p \vDash_\mu \mathsf{T}\Diamond\,\mathsf{input}(v')$ *implies* $v = v'$.

Proof. We consider each part in turn:

(1) We consider three subcases:
- *Suppose* $[\![\mathsf{input}(v)]\!]_\mu(p) = \mathbf{b}$.
  Then by ($\mathsf{CaCorrect}'$) (for input) $[\![\mathsf{input}(v')]\!]_\mu(p) = \mathbf{b}$ for *every* $v' \in \{0, 0.5.1\}$ and in particular $[\![\mathsf{input}(1\text{-}v)]\!]_\mu(p) = \mathbf{b} = \neg\mathbf{b}$ so we are done.
- *Suppose* $[\![\mathsf{input}(v)]\!]_\mu(p) = \mathbf{t}$.
  Then by ($\mathsf{CaCorrect}'$) (for input) $[\![\mathsf{input}(v')]\!]_\mu(p) \in \{\mathbf{t}, \mathbf{f}\}$ for *every* $v' \in \{0, 0.5.1\}$ and in particular $[\![\mathsf{input}(1\text{-}v)]\!]_\mu(p) \in \{\mathbf{t}, \mathbf{f}\}$. By ($\mathsf{CaInput}_1$) and Proposition 3.1.3(1.e), $\mathsf{input}(v)$ is true for precisely *one* value $v \in \{0, 1\}$. The result follows.
- The case that $[\![\mathsf{input}(v)]\!]_\mu(p) = \mathbf{f}$ is precisely symmetric to the case of $\mathbf{t}$.

[22] Why is a weaker axiom better? Surely strong axioms with lots of structure are good? No: generally speaking, the game is to derive strong and well-structured correctness properties from weak and minimally-structured axioms; our game is to derive the strongest possible theorems from the weakest possible assumptions.
[23] On the right-hand side, $[\![v{=}v']\!] \in \{\mathbf{t}, \mathbf{f}\}$, so equality always returns a correct truth-value.

(2) By routine logical reasoning from part 1 of this result.
(3) Suppose $\vDash_\mu \Box\mathsf{input}(v)$ and $p \in \mathsf{P}$ and $p \vDash_\mu \mathsf{T}\,\mathsf{input}(v')$. Then $p \vDash_\mu \mathsf{input}(v)$ and $p \vDash_\mu \mathsf{T}\,\mathsf{input}(v')$. By ($\mathsf{CaCorrect}'$) (for input) and Proposition 2.2.2(8), $p \vDash_\mu \mathsf{T}\,\mathsf{input}(v)$. We use part 2 of this result. □

Proposition 5.3.5 (Validity) *Suppose* $\vDash_\mu$ ThyCA *and* $v \in \{0,1\}$ *(so* $v \neq 0.5$*). Then:*

(1) $\vDash_\mu \Diamond\mathsf{output}(v) \rightharpoonup \Diamond\mathsf{input}(v)$.
"If a non-faulty party outputs $v \neq 0.5$, then v was the input of some non-faulty party."
(2) $\vDash_\mu \Box\mathsf{input}(v)$ *and* $\vDash_\mu \mathsf{T}\,\mathsf{output}(v')$ *implies* $v{=}v'$*, for every* $v' \in \{0, 0.5, 1\}$.
"If all non-faulty parties have the same input, then this is the only possible output of any non-faulty party.*"*

Proof. We consider each part in turn:

(1) By strong modus ponens (Prop. 2.2.2(5)) it suffices to show that $\vDash_\mu \mathsf{T}\Diamond\mathsf{output}(v)$ implies $\vDash_\mu \mathsf{T}\Diamond\mathsf{input}(v)$.
So suppose $\vDash_\mu \mathsf{T}\Diamond\mathsf{output}(v)$. By (CaOutput?) $\vDash_\mu \boxdot\mathsf{echo}_2(v) \wedge \boxdot\mathsf{echo}_2(v)$ and by (CaCorrect) $\vDash_\mu \boxdot\mathbb{TF}[\mathsf{echo}_2]$. Using Theorem 2.4.3 and Lemma 2.3.7(3) (or just direct from the 3-twined property in Definition 2.4.1) $\vDash_\mu \mathsf{T}\Diamond\mathsf{echo}_2(v)$.
By similar reasoning using (CaEcho2?) in place of (CaOutput?), we conclude that $\vDash_\mu \mathsf{T}\Diamond\mathsf{echo}_1(v)$. By (CaEcho1?) $\vDash_\mu \mathsf{T}\Diamond\mathsf{input}(v)$ as required.
(2) Suppose $\vDash_\mu \Box\mathsf{input}(v)$ and suppose $\vDash_\mu \mathsf{T}\Diamond\mathsf{output}(v')$; we will show $\vDash_\mu \mathsf{T}\,(v{=}v')$, i.e. $v = v'$.
By part 1 of this result $\vDash_\mu \mathsf{T}\Diamond\mathsf{input}(v')$. But we assumed $\vDash_\mu \Box\mathsf{input}(v)$, so by Lemma 5.3.4(3) $v = v'$ as required. □

Remark 5.3.6 We take a minute to discuss the precise form of the validity properties in Proposition 5.3.5. Consider this English sentence, where $v \neq 0.5$:

"If a non-faulty party outputs $v \neq 0.5$, then v was the input of some non-faulty party."

We render it as

$$\text{ThyCA} \vDash_\mu \Diamond\mathsf{output}(v) \rightharpoonup \Diamond\mathsf{input}(v).$$

By strong modus ponens (Prop. 2.2.2(5)), it means that $[\![\mathsf{output}(v)]\!]_\mu(p) = \mathbf{t}$ for some $p \in \mathsf{P}$ implies $[\![\mathsf{input}(v)]\!]_\mu(p') = \mathbf{t}$ for some $p' \in \mathsf{P}$. We are using $\mathbf{t}$ and $\mathbf{f}$ as the truth-values of correct (non-faulty) participants, and $\mathbf{b}$ as the truth-value for incorrect (faulty) participants, so this corresponds precisely to the English sentence quoted above.

It might be instructive to consider a subtly incorrect rendering of

"If all non-faulty parties have the same input, then this is the only possible output of any non-faulty party*"*

as

$$\text{ThyCA} \vDash_\mu \boxdot\mathsf{input}(v) \rightharpoonup \mathsf{output}(v).$$

This is wrong, for two reasons:

(1) output is just a predicate-symbol; it is not a function-symbol. As such, $\mathsf{output}(v)$ does not *a priori* have to return $\mathbf{t}$ or $\mathbf{b}$ on just one value v.[24] $\mathsf{output}(v)$ could be $\mathbf{t}$ and also $\mathsf{output}(v')$ might be $\mathbf{t}$ for some other v'.
(2) More subtly, just because $\vDash_\mu \mathsf{T}\boxdot\mathsf{input}(v)$ holds does not *a priori* mean that $\vDash_\mu \mathbf{t}\Diamond\mathsf{input}(1\text{-}v)$ cannot hold. Remember that we are working over an abstract 3-twined semitopology; there might be a quorum of correct participants who $\mathbf{t}$-do $\mathsf{input}(v)$ and *also* some other correct participant who $\mathbf{t}$-does $\mathsf{input}(v')$ for some other v'.

This is why we write $\mathbb{TB}\Box\mathsf{input}(v)$ in (CaValid1) in Figure 10: it means that every participant is either faulty or, if it is not faulty, it inputs v. The use of exclusive-or $\oplus$ in (CaInput) ensures that input is functional, so this does indeed render the idea of the English sentence quoted above.

[24] Indeed, in the case of this particular protocol output can be $\mathbf{t}$ on two values; see Example 5.3.3.

5.3.3 Liveness

We work towards Proposition 5.3.10. We will need Lemma 5.3.7 and two corollaries of it:

Lemma 5.3.7 *Suppose* $\vDash_\mu$ ThyCA *and* $v \in \{0, 0.5, 1\}$. *Then:*

(1) $\vDash_\mu \boxdot\mathsf{echo}_1(v)$ *implies* $\vDash_\mu \mathsf{T}\,\diamonddot\mathsf{echo}_1(v)$.
(2) $\vDash_\mu \mathsf{T}\,\diamonddot\mathsf{echo}_1(v)$ *implies* $\vDash_\mu \Box\,\mathsf{echo}_1(v)$.
(3) $\vDash_\mu \Box\,\mathsf{echo}_1(v)$ *implies* $\vDash_\mu \mathsf{T}\boxdot\mathsf{echo}_1(v)$.

Proof. Suppose $\vDash_\mu \boxdot\mathsf{echo}_1(v)$. By (CaCorrect) (for echo_1) and Theorem 2.4.3 $\vDash_\mu \mathsf{T}\,\diamonddot\mathsf{echo}_1(v)$. By (CaEcho1!) (right-hand disjunct, for $\diamonddot\mathsf{echo}_1$) $\vDash_\mu \Box\,\mathsf{echo}_1(v)$. By (CaCorrect) (for echo_1) and Lemma 2.3.6(2) $\vDash_\mu \mathsf{T}\boxdot\mathsf{echo}_1(v)$. ☐

Corollary 5.3.8 Suppose $\vDash_\mu$ ThyCA and $v \in \{0, 0.5, 1\}$. Then $\vDash_\mu \mathsf{echo}_2(v) \rightharpoonup \boxdot\mathsf{echo}_1(v)$.

Proof. Suppose $p \vDash_\mu \mathsf{T}\,\mathsf{echo}_2(v)$. By (CaEcho2?) $\vDash_\mu \boxdot\mathsf{echo}_1(v)$. By Lemma 5.3.7 $\vDash_\mu \mathsf{T}\boxdot\mathsf{echo}_1(v)$. ☐

Corollary 5.3.9 Suppose $\vDash_\mu$ ThyCA. Then:

(1) $\vDash_\mu \mathsf{T}\,\diamonddot(\mathsf{input}(0) \land \mathsf{TF}[\mathsf{echo}_1]) \lor \mathsf{T}\,\diamonddot(\mathsf{input}(1) \land \mathsf{TF}[\mathsf{echo}_1])$.
(2) $\vDash_\mu \mathsf{T}\,\diamonddot\mathsf{echo}_1(0) \lor \mathsf{T}\,\diamonddot\mathsf{echo}_1(1)$.
(3) $\vDash_\mu \mathsf{T}\boxdot\mathsf{echo}_1(0) \lor \mathsf{T}\boxdot\mathsf{echo}_1(1)$.
(4) $\vDash_\mu \Box\,(\mathsf{echo}_2(0) \lor \mathsf{echo}_2(1))$.
(5) $\vDash_\mu \mathsf{T}\boxdot(\mathsf{echo}_2(0) \lor \mathsf{echo}_2(1))$.

Proof. We consider each part in turn:

(1) Using (CaInput) and (CaCorrect) $\vDash_\mu \Box\,\big(\mathsf{input}(0) \lor \mathsf{input}(1)\big) \land \boxdot\mathsf{TF}[\mathsf{echo}_1, \mathsf{input}]$, so by Lemma 2.3.6(1) $\vDash_\mu \boxdot\big((\mathsf{input}(0) \lor \mathsf{input}(1)) \land \mathsf{TF}[\mathsf{echo}_1, \mathsf{input}]\big)$, and rearranging we obtain

$$\vDash_\mu \boxdot((\mathsf{input}(0) \land \mathsf{TF}[\mathsf{echo}_1, \mathsf{input}]) \lor (\mathsf{input}(1) \land \mathsf{TF}[\mathsf{echo}_1, \mathsf{input}])).$$

By Corollary 2.4.4 (since we assume $(\mathsf{P}, \mathsf{Open})$ is 3-twined)

$$\vDash_\mu \diamonddot(\mathsf{input}(0) \land \mathsf{TF}[\mathsf{echo}_1, \mathsf{input}]) \lor \diamonddot(\mathsf{input}(1) \land \mathsf{TF}[\mathsf{echo}_1, \mathsf{input}]).$$

By routine calculations from Figure 1 we conclude that

$$\vDash_\mu \mathsf{T}\,\diamonddot(\mathsf{input}(0) \land \mathsf{TF}[\mathsf{echo}_1]) \lor \mathsf{T}\,\diamonddot(\mathsf{input}(1) \land \mathsf{TF}[\mathsf{echo}_1])$$

as required.

(2) We combine part 1 of this result with (CaEcho1!) (left-hand disjunct, for input) using weak modus ponens and Proposition 2.2.2(8).

(3) From part 2 of this result using Lemma 5.3.7.

(4) From part 3 of this result using (CaEcho2!).

(5) From part 4 of this result using (CaCorrect) (for echo_2) and Lemma 2.3.6(2). ☐

Proposition 5.3.10 (Liveness) *If* $\vDash_\mu$ ThyCA *then* $\vDash_\mu \Box\,\exists v.\mathsf{output}(v)$.

"If all non-faulty parties start the protocol then all non-faulty parties eventually output a value."

Proof. By Corollary 5.3.9(5) there exists a quorum $O \in \mathsf{Open}_{\neq\varnothing}$ such that

$$\forall p \in O.(p \vDash_\mu \mathsf{T}\,(\mathsf{echo}_2(0) \lor \mathsf{echo}_2(1))).$$

Note in passing that by ($\mathsf{CaEcho2}_{01}$) and Proposition 3.1.3(1.e), these two disjuncts are mutually exclusive. There are now two cases:

(1) Suppose $\forall p \in O.(p \vDash_\mu \mathsf{T}\,\mathsf{echo}_2(v))$ for some $v \in \{0, 1\}$, so that $\vDash_\mu \mathsf{T}\boxdot\mathsf{echo}_2(v)$.
Then using (CaOutput!) $\vDash_\mu \Box\,\mathsf{output}(v)$.

(2) Suppose there exist $p, p' \in O$ such that $p \vDash_\mu \mathsf{T}\,\mathsf{echo}_2(0)$ and $p' \vDash_\mu \mathsf{T}\,\mathsf{echo}_2(1)$, so that $\vDash_\mu \mathsf{T}\Diamond\mathsf{echo}_2(0) \land \mathsf{T}\Diamond\mathsf{echo}_2(1)$.
By Corollary 5.3.8 $\vDash_\mu \mathsf{T}\boxdot\mathsf{echo}_1(0)$ and $\vDash_\mu \mathsf{T}\boxdot\mathsf{echo}_1(1)$ and by $(\mathsf{CaOutput'!})$ $\vDash_\mu \Box\,\mathsf{output}(0.5)$.

In either case, $\vDash_\mu \Box\,\exists v.\mathsf{output}(v)$, as required. □

Remark 5.3.11 We continue the theme of Remark 5.3.6 and comment on some design subtleties of the logical correctness property for Liveness:

(1) This property for Liveness would be subtly incorrect:

$$\textsc{ThyCA} \vDash_\mu \boxdot\exists v.\mathsf{output}(v).$$

It might look right, but we are working over an abstract 3-twined semitopology so it might be that there is a quorum of correct participants who output some value, and *also* some other correct participant not in that quorum, who does not.

(2) The English statement of Liveness includes a proviso 'if all non-faulty parties start the protocol', which is omitted in $\boxdot\exists v.\mathsf{output}(v)$. This is for two reasons: first, if a party does not start the protocol then arguably from our point of view it is faulty, so the proviso trivially holds by definition; but second, our model has no notion of time so talking about when and whether a party starts or does not start doing something is just not in the scope of our analysis.

(3) Does $\Box\,\exists v.\mathsf{output}(v)$ not mean that *everyone* outputs a value, including faulty parties? A faulty participant p will return $\mathbf{b}$ for $\mathsf{output}(v)$ for any v, which makes $[\![\exists v.\mathsf{output}(v)]\!]_\mu(p) = \mathbf{b}$. This is a valid truth-value.
So $\Box\,\exists v.\mathsf{output}(v)$ elegantly and accurately captures the idea that *'every correct/non-faulty participant outputs a value'*.

5.4 Further discussion of the axioms and proofs

Remark 5.4.1 Compare $(\mathsf{CaEcho1?})$ from Figure 9 with $(\mathsf{BrEcho?})$ from Figure 6. Note how similar they are: the only difference, in fact, is that $(\mathsf{CaEcho1?})$ uses the strong implication $\rightharpoonup$ whereas $(\mathsf{BrEcho?})$ uses the weak implication $\rightarrow$. This illustrates a pattern: treating distributed algorithms as axiom-systems abstracts them, and this reveals their structure, sometimes in non-obvious ways.

In Remark 5.2.2(10) we mentioned that $(\mathsf{CaOutput!})$ does not have an explict side-condition that $a \neq 0.5$, because this is a Lemma. As promised, here is the statement and proof:

Lemma 5.4.2 *Suppose* $\vDash_\mu \textsc{ThyCA}$ *and* $v \in \{0, 0.5, 1\}$ *and* $p \in \mathsf{P}$. *Then:*

(1) $\vDash_\mu \mathsf{input}(v) \rightharpoonup (v{=}0 \lor v{=}1)$.
(2) $\vDash_\mu \mathsf{echo}_1(v) \rightharpoonup (v{=}0 \lor v{=}1)$.
(3) $\vDash_\mu \mathsf{echo}_2(v) \rightharpoonup (v{=}0 \lor v{=}1)$.

Proof. We consider each part in turn, freely using strong modus ponens (Prop. 2.2.2(5)):

(1) Suppose $p \vDash_\mu \mathsf{T}\,\mathsf{input}(v)$. Then $v = 0 \lor v = 1$ is direct from $(\mathsf{CaInput})$.
(2) Suppose $p \vDash_\mu \mathsf{T}\,\mathsf{echo}_1(v)$. By $(\mathsf{CaEcho1?})$ $p \vDash_\mu \mathsf{T}\Diamond\mathsf{input}(v)$, and we use part 1 of this result.
(3) Suppose $p \vDash_\mu \mathsf{T}\,\mathsf{echo}_2(v)$. By $(\mathsf{CaEcho2?})$ $\vDash_\mu \boxdot\mathsf{echo}_1(v)$, and by $(\mathsf{CaCorrect})$ (for echo_1) and Theorem 2.4.3 $\vDash_\mu \mathsf{T}\,\diamonddot\mathsf{echo}_1(v)$, so by Lemma 2.3.7(2) $\vDash_\mu \mathsf{T}\Diamond\mathsf{echo}_1(v)$ and we can use part 2 of this result.

□

Remark 5.4.3 There are some subtle differences in the treatment of correctness between ThyBB and ThyCA. In ThyBB we insist that ready and echo are correct on quorums (= nonempty open sets), but we do not insist that these quorums must be equal. In contrast, in ThyCA we insist that there is a single quorum on which input, echo_1, echo_2, and output are all correct.

Why the difference? The correctness properties for ThyCA in Definition 5.1.4 use a notion of 'non-faulty participant' as a participant that is correct throughout a run of the algorithm; ThyBB mentions correct participants too, of course, but the notion of correctness is subtly more lax in that it tends to refer to correctness *at a particular moment* rather than correctness throughout the run. The axioms reflect this.

It might be possible to make slightly weaker correctness assumptions of ThyCA: it would suffice to just check every mention of 'CaCorrect' in the proofs, and see whether a weaker axiom would work that (for example) does not assume the same quorum for all predicates. We would then just have to carefully define what we mean by a 'faulty' participant. Something is most likely possible here, but we leave it for future work.

Such questions reflect the fact that logic is doing its job, by making things every explicit and precise. In English it is easy to write 'faulty participant' or 'non-faulty participant', but what this actually *means* can be, and often is, left implicit (or left to the reader to fill in from background culture and expertise). In the logical world, such concepts can be nailed down using axioms and given precise meaning; then design decisions can be discussed within this logical framework.

6 Conclusions and related and future work

Conclusions

We have shown how a distributed algorithm can be represented declaratively via axioms. We use three-valued logic, such that the third truth-value helps us to represent byzantine behaviour; and we use semitopologies to represent quorums; and then we exploit the close connection between (semi)topology and logic to capture a declarative essence of the distributed algorithm.

The result is simple, yet powerful. It is mathematically novel – three-valued modal logic over a semitopology is a new thing, as is (to the best of my knowledge) declarative axiomatic formal specification of distributed protocols – it illuminates the algorithm in new and informative ways.

It is also practically worthwhile: a high-level formal specification in logical style is useful to inform new implementations, for modularity, for updatability, to promote diversity of implementations, for verifying implementation correctness, for provable security, for analysis and formal checking of security properties, and for more. It keeps only what is essential about the object of study, and behaves as an abstract reference implementation of the logical essence of what it *is*. Any implementation can legitimately claim to be 'an implementation of Bracha Broadcast', and will necessarily enjoy the correctness properties of such, if the models that it generates satisfy the axioms in Figure 6. [25]

The examples in this paper are simple by the standards of modern distributed algorithms. This is because this paper focuses on the underlying technique: the more complex the algorithm we apply these ideas to, the *more favourable* the comparison with its declarative specification is likely to become, because logic helps us to control complexity, elide detail, and focus on essential system properties.

Related work

We can think of the logic of this paper as a declarative complement to TLA+, which (while also a modal logic) is more imperative in flavour. TLA+ is an industry standard which is widely and successfully used to specify and verify implementations of distributed protocols [Lam94,Lam02,YML99]. However, TLA+ is a tool for specifying transition systems. This is therefore in the spirit of imperative programming, and this paper complements that technique with a declarative approach.

Declarative specification itself is well-studied and its benefits are well-understood: this is what makes (for example) Haskell and LISP different from C and BASH. What this paper does is show how to bring such principles to bear on a new class of distributed algorithms, of which Bracha Broadcast is a simple but canonical representative. Perhaps axioms like those in this paper might one day be executed as logic clauses in a suitable declarative programming framework, but in the first instance their natural home is likely to be as a source of truth in a specification document, in a formal verification in a theorem-prover, or in a model-checker.

[25] Recall for example the evolution to taking such specifications as standard practice in the world of blockchain tokens: e.g. the ERC20 token standard eips.ethereum.org/EIPS/eip-20 or the FA2 token standard archetype-lang.org/docs/templates/fa2/. The point here is cultural (not technical): it is now generally recognised that a logical specification of a token, is essential. Standard formal verification tools can then be applied [GJS21].

In practice, including in industrial practice, broadcast and consensus protocols are typically specified in English (as in Example 2.1.1 or Remark 4.1.1) or in pseudocode (as in Figure 8). The webpage [IA22] or the book [CGR11] are good examples of the genre. If formalisation is undertaken, it proceeds in TLA+ (or possibly in LEAN or a similar system), but what gets formalised is still basically a (larger or smaller) description of a (larger or smaller) concrete transition relation.

The essential feature of the logical approach in this paper is to treat the algorithm declaratively at a higher level of an abstraction, as an axiomatic theory, rather than as a transition system. It is not obvious that this should preserve the essence of the algorithm, but it does.

Unrelated work

There are fields adjacent to, or superficially similar to, distributed algorithms, and many applications of logic and topology. There is a danger of being misled by a keyword like 'topology' to imagine that this paper is doing one thing, when it is doing another.

So it might be useful to spell out what this paper is not.

This is not a paper on algebraic topology. We mention this because algebraic topology has been applied to the solvability of distributed-computing tasks in various computational models (e.g. the impossibility of wait-free k-set consensus using read-write registers and the Asynchronous Computability Theorem; a good survey is in [HKR13]). Semitopology is topological in flavour, but it is not the same as topology, and this is not a paper about algebraic topology applied to the solvability of distributed-computing tasks.

This is not a paper about large examples or difficult theorems. The examples in this paper (voting, broadcast, agreement) are toy and/or textbook algorithms. This a feature, not a bug! The point of this approach is to attain simplicity via abstraction in logic. So for example: if the proof of Proposition 4.2.11 looks boringly similar to the proof of Proposition 5.3.2, then this is evidence that our abstraction via axiomatic logic has worked its magic and let us turn *complex informal verbal arguments* into *routine, regular symbolic ones*. This flavour of boredom is something that the field of distributed algorithms (in my opinion) needs more of.

And this simplicity via abstraction in logic is not mathematically trivial; far from it. The use of three truth-values, semitopologies, modal logic, and the precise forms of the axioms, are novel. It was hardly obvious that this combination of tools could be usefully applied to study distributed algorithms. Yet we have seen that this is so, and once we have set the machinery up, axioms become compact and precise, and proofs become — if not completely easy — at least precise, symbolic, compact, and clear.

This is not a paper on distributed programming, it is a paper on distributed *algorithms.* Process algebras (perhaps guided by session types), or imperative programs with (say) concurrent separation logic, are valid approaches to designing distributed programs, but so far as I am aware they are no help to design (say) an efficient blockchain consensus algorithm. Those approaches are also typically operational in nature, aiming to model and reason about the dynamic behaviour of systems; whereas the approach proposed in this paper is more static, focussing on design-level evaluation rather than runtime behaviour. While that is true, the fundamental difference of process calculi, session types, and similar systems is that they are fundamentally concerned with writing correct programs, whereas the focus in this paper (and in distributed algorithms in general) is on designing specific algorithms, like 'agreement' or 'broadcast', that are resilient to faulty behaviour.[26] This is a different part of the design space, with its own distinctive literature and norms.

Reflection on the axiomatisations

Developing an axiomatisation is hard work, even for a simple protocol. But, the work is constructive in the sense that it forces us to understand what the protocol is. It is easy enough to read a protocol and to fool ourselves into thinking that we understand it.[27] Converting it into an axiomatisation forces us to

[26] A supplementary word here: *algorithms* and *programs* live on fundamentally different levels of the development stack. Quicksort is a classic list sorting algorithm; `l.sort()` is a (line in) a program; these are not equivalent. Thinking of algorithms as 'just special cases of programs' is a category error that puts the cart before the horse; similarly *distributed algorithms* are not just special a case of *distributed programs.*

[27] Though speaking for myself, I could not even manage that. When I was first exposed to these distributed protocols — written in some combination of English and pseudocode — I did not understand them *at all.* Abstracting away

be explicit about the nature of the thing itself.

I would actually argue that the pseudocode in presentations like Figure 8 is not even really code, or even pseudocode. It is written in code-like style, but consider that clauses are 'executed' in parallel and they follow a Horn clause like structure: Horn clauses are declarative and logical in flavour (coming from logic programming), not imperative and procedural. So I would argue that the logical content is already there, even in Figure 8, and it is just struggling to escape from a code-flavoured presentation.

How do we know when an axiomatisation is 'good'? One useful test is: does it make the proofs easy? As is often the case, small (or large!) differences in how axioms are presented can make big differences to how cleanly proofs go through. This is a criterion for prefering one axiomatisation over another, and it is also a way to understand the protocol: we can say that we have truly understood a distributed protocol when we have expressed it as an axiomatisation that is simple and gives simple proofs. Finding this axiomatisation is often not trivial, but once found, its simplicity is powerful evidence that we have arrived at something *mathematically essential* about what the protocol is trying to express.

We also see that axiomatic proofs are very precise. A predicate has an unambiguous meaning, and logical proofs are just about manipulating axioms using the rules of logic. This is a very useful support for precision and correctness.

Future work

Natural future work is to study these ideas applied to other distributed algorithms. Ones that are foundational for many modern blockchain systems include: Paxos (for this, a draft is already available [GZ25a]), Practical Byzantine Fault Tolerance (PBFT) [CL99], HotStuff [YMR+19], and Tendermint [BKM18]; another family of protocols organises transactions into a directed acyclic graph (DAG) rather than a strict chain (for parallelism and scalability), including DAG-rider [KKNS21], Narwhal and Tusk [DKSS22], and Bullshark [SGSK22]. Many other protocols and protocol-families exist, and studying what logics and families of logics they might correspond to, is an open problem.

Other natural future work is to integrate the logics thus produced into model-checkers and theorem-provers. There is nothing necessarily mathematically complex about this (which is a feature, not a bug): a three-valued logic over a semitopology is not a mathematically complex entity. For example: the TLC model checker [YML99] for TLA+ provides state exploration to verify safety and liveness properties; the PlusCal translator simplifies specification by offering a high-level pseudocode-like interface; and TLA+ Toolbox [KLR19] provides a development environment. It would be good to integrate the logic from this paper into this toolchain or one like it. If there is a catch here, it is that semitopologies involve powersets, which can lead to an exponential state space explosion when looking for counterexamples, so we might do well to choose a tool that would handle this well. On this topic semitopologies have an algebraic analogue of *semiframes* [Gab24, part III]; these provide additional abstraction and so might be useful in the search for counterexamples, by somewhat reducing state space.

References

[AAZ11] Ofer Arieli, Arnon Avron, and Anna Zamansky, *Ideal paraconsistent logics*, Studia Logica **99** (2011), no. 1-3, 31–60.

[ABDY22] Ittai Abraham, Naama Ben-David, and Sravya Yandamuri, *Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement*, Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (New York, NY, USA), PODC'22, Association for Computing Machinery, 2022, Available online at `https://eprint.iacr.org/2022/711.pdf`, p. 381–391.

[ACM20] Ignacio Amores-Sesar, Christian Cachin, and Jovana Micic, *Security analysis of Ripple consensus*, 24th International Conference on Principles of Distributed Systems, OPODIS 2020, December 14-16, 2020, Strasbourg, France (Virtual Conference) (Quentin Bramas, Rotem

the pseudocode to logical axioms was my way of making sense of the field. Axioms do not lie: they mean what they mean. If you mean something else, then you tweak the syntax or change the logic.

Oshman, and Paolo Romano, eds.), LIPIcs, vol. 184, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020, pp. 10:1–10:16.

[ACT22] Ignacio Amores-Sesar, Christian Cachin, and Enrico Tedeschi, *When is spring coming? A security analysis of Avalanche consensus*, 26th International Conference on Principles of Distributed Systems, OPODIS 2022, December 13-15, 2022, Brussels, Belgium (Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, eds.), LIPIcs, vol. 253, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, pp. 10:1–10:22.

[ACTZ24] Orestis Alpos, Christian Cachin, Björn Tackmann, and Luca Zanolini, *Asymmetric distributed trust*, Distributed Computing **37** (2024), no. 3, 247–277.

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic, *The latest gossip on BFT consensus*, CoRR **abs/1807.04938** (2018).

[Bra87] Gabriel Bracha, *Asynchronous byzantine agreement protocols*, Information and Computation **75** (1987), no. 2, 130–143.

[CGR11] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues, *Introduction to reliable and secure distributed programming (2. ed.)*, Springer, 2011.

[CL99] Miguel Castro and Barbara Liskov, *Practical byzantine fault tolerance*, Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999 (Margo I. Seltzer and Paul J. Leach, eds.), USENIX Association, 1999, pp. 173–186.

[DKSS22] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman, *Narwhal and tusk: a dag-based mempool and efficient BFT consensus*, EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022 (Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, eds.), ACM, 2022, pp. 34–50.

[Gab24] Murdoch J. Gabbay, *Semitopology: decentralised collaborative action via topology, algebra, and logic*, College Publications, August 2024, ISBN 9781848904651.

[Gab25] ———, *Semitopology: a topological approach to decentralised collaborative action*, The Journal of Logic and Computation (2025), `https://doi.org/10.1093/logcom/exae050`.

[GJS21] Murdoch J. Gabbay, Arvid Jakobsson, and Kristina Sojakova, *Money Grows on (Proof-)Trees: The Formal FA1.2 Ledger Standard*, 3rd International Workshop on Formal Methods for Blockchains (FMBC 2021) (Dagstuhl, Germany) (Bruno Bernardo and Diego Marmsoler, eds.), Open Access Series in Informatics (OASIcs), vol. 95, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 2:1–2:14.

[GZ25a] Murdoch J. Gabbay and Luca Zanolini, *A declarative approach to specifying distributed algorithms using three-valued modal logic*, 2 2025.

[GZ25b] ———, *A declarative approach to specifying distributed algorithms using three-valued modal logic*, 2025, Journal draft submitted for publication. Available at `https://arxiv.org/abs/2502.00892`.

[HKR13] Maurice Herlihy, Dmitry Kozlov, and Sergio Rajsbaum, *Distributed computing through combinatorial topology*, Morgan Kaufmann, 2013.

[IA22] Sravya Yandamuri Ittai Abraham, Naama Ben-David, *Asynchronous agreement part 4: Crusader agreement and binding crusader agreement*, https://decentralizedthoughts.github.io/2022-04-05-aa-part-four-CA-and-BCA/ (permalink), April 2022, Online article in Decentralized Thoughts.

[KKNS21] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman, *All you need is DAG*, PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021 (Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, eds.), ACM, 2021, pp. 165–175.

[KLR19] Markus Alexander Kuppe, Leslie Lamport, and Daniel Ricketts, *The TLA+ toolbox*, Proceedings Fifth Workshop on Formal Integrated Development Environment, F-IDE@FM

2019, Porto, Portugal, 7th October 2019 (Rosemary Monahan, Virgile Prevosto, and José Proença, eds.), EPTCS, vol. 310, 2019, pp. 50–62.

[Lam94] Leslie Lamport, *The temporal logic of actions*, ACM Trans. Program. Lang. Syst. **16** (1994), no. 3, 872–923.

[Lam02] ______, *Specifying systems, the TLA+ language and tools for hardware and software engineers*, Addison-Wesley, 2002.

[MR98] Dahlia Malkhi and Michael K. Reiter, *Byzantine quorum systems*, Distributed Computing **11** (1998), no. 4, 203–213.

[NTT21] Joachim Neu, Ertem Nusret Tas, and David Tse, *Ebb-and-flow protocols: A resolution of the availability-finality dilemma*, 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021, IEEE, 2021, pp. 446–465.

[Pri02] Graham Priest, *Paraconsistent logic*, Handbook of Philosophical Logic, 2nd Edition (D.M. Gabbay and F. Guenthner, eds.), vol. 6, Kluwer, 2002, pp. 287–393.

[PS17] Rafael Pass and Elaine Shi, *The sleepy model of consensus*, Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II (Tsuyoshi Takagi and Thomas Peyrin, eds.), Lecture Notes in Computer Science, vol. 10625, Springer, 2017, pp. 380–409.

[SGSK22] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias, *Bullshark: DAG BFT protocols made practical*, Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022 (Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, eds.), ACM, 2022, pp. 2705–2718.

[Sho22] Victor Shoup, *Proof of history: What is it good for?*, May 2022, https://www.shoup.net/papers/poh.pdf.

[SNM$^+$22] Caspar Schwarz-Schilling, Joachim Neu, Barnabé Monnot, Aditya Asgaonkar, Ertem Nusret Tas, and David Tse, *Three attacks on proof-of-stake ethereum*, Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers (Ittay Eyal and Juan A. Garay, eds.), Lecture Notes in Computer Science, vol. 13411, Springer, 2022, pp. 560–576.

[SW89] Nicola Santoro and Peter Widmayer, *Time is not a healer*, STACS 89 (Berlin, Heidelberg) (B. Monien and R. Cori, eds.), Springer Berlin Heidelberg, 1989, pp. 304–313.

[SWvRM20] Isaac Sheff, Xinwen Wang, Robbert van Renesse, and Andrew C. Myers, *Heterogeneous paxos: Technical report*, 2020, https://arxiv.org/abs/2011.08253.

[SWvRM21] Isaac Sheff, Xinwen Wang, Robbert van Renesse, and Andrew C. Myers, *Heterogeneous Paxos*, 24th International Conference on Principles of Distributed Systems (OPODIS 2020) (Dagstuhl, Germany) (Quentin Bramas, Rotem Oshman, and Paolo Romano, eds.), Leibniz International Proceedings in Informatics (LIPIcs), vol. 184, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 5:1–5:17.

[Yak18] A. Yakovenko, *Solana: A new architecture for a high performance blockchain v0.8.13*, Whitepaper, 2018, https://solana.com/solana-whitepaper.pdf.

[YML99] Yuan Yu, Panagiotis Manolios, and Leslie Lamport, *Model checking TLA$^+$ specifications*, Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 27-29, 1999, Proceedings (Laurence Pierre and Thomas Kropf, eds.), Lecture Notes in Computer Science, vol. 1703, Springer, 1999, pp. 54–66.

[YMR$^+$19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham, *Hotstuff: BFT consensus with linearity and responsiveness*, Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019 (Peter Robinson and Faith Ellen, eds.), ACM, 2019, pp. 347–356.