Mechanizing BFT consensus protocols in Agda

Orestis Melkonian, Mauro Jaskelioff, James Chapman, and Jon Rossie

Input Output, Global (IOG)

Introduction. Consensus protocols have been around for a long time [10, 9], but there has been a surge of interest in the last decade motivated by cryptocurrency and blockchain applications, where all participants need to agree on a common order of blocks on the chain.

Consensus protocols can be permissioned or permissionless. Nakamoto-style consensus protocols are permissionless (all participants can be part of the decision procedure) while classical protocols like BFT are permissioned (a few designated ones make the decision). With the advent of proof-of-stake blockchains, permissioned protocols can be adapted to work in a blockchain setting: a committee is formed based on the stake of all participants, which makes all decisions until a new committee is designated.

BFT protocols follow a pattern of propose and vote, where a leader proposes a block and other nodes vote for it. When a block gets enough votes it gets notarized; a notarized chain (i.e. a chain of notarized blocks) is considered finalized when certain protocol-dependent conditions hold, which guarantees that all participants will agree on it.

There are several formalisations of different consensus protocols in which parts or all of the protocol and their corresponding safety and liveness properties are proved correct [13, 1, 12]. In this work, we propose an alternative *refinement*-based approach, where the formalization is divided into layers. At the most abstract layer we only model the deterministic result of the consensus protocol (a finalized chain) together with the information we need to prove that the chain is indeed finalized. The motivation for this is that we want to be able to construct and verify zero-knowledge (ZK) proofs that a given chain is notarized/final. This means that the details of *how* to get to a notarized/final chain are not important at this level of abstraction.

Of course in order to get a finalized chain we will need *a* concrete protocol, which we formalize in the next layer. It corresponds to a mechanization of the paper introducing the protocol informally. Still, one might want to explore more details than the "academic" version of the paper, e.g., to analyze and optimize performance or include the actual networking, which should again be studied in an even lower layer of abstraction.

We conduct our work in a mechanized fashion using the Agda proof assistant [11] and simultaneously develop a new compilation backend to Rust, so that our formal artifact can also be utilized for *conformance testing* against an actual blockchain in production.

Framework: an abstract view of BFT protocols. First and foremost, we stay parametric over an abstract type of transactions so that we can later instantiate that with various ledger models, but as a starting point we define everything on top of a minimal UTxO-based ledger, whose meta-theory has already been extensively studied in prior work [3, 2], with the minor extension that transactions are also allowed to register new committees. Hence, the ledger state does not only consist of the typical UTxO set of currently unspent outputs, but also the committees registered thus far.

BFT consensus relies on a subset of the participants, called the 'committee', that assigns a public key to each participant and requires a certain ratio of votes (in the form of *signed* blocks) in order to reach agreement, a.k.a. forming a *quorum*:

```
record Committee : Type where
  field ratio   : Float
        members : AssocList Pid PublicKey
```

```
quorum? : List Vote → Committee → Bool
quorum? vs com =
  com .ratio * length (com .members) < length vs
```

Block *verification* is formulated as a (labeled) transition system, which builds upon the base transaction-level transition of the ledger inherited from the underlying ledger model [6, 8].

```
data _⊢_–[_]→ᵇ_ : ℕ → Ledger × Hash → Block → Ledger × Hash → Type where
  VerifyBlock :
     • h ≡ b .block .height
     • T (quorum? (b .votes) com)
     • ∀[ v ∈ b .votes ] ∃[ pr ∈ com .members ⁉ v .pid ]
         T (verify-signature pr (v .signature) b♯)
     • _ ⊢ l –[ b .block .transactions ]→* l′
     ──────────────────────────────────────────
     h ⊢ (l , H) –[ b ]→ᵇ (l′ , (H , b♯) ♯)
```

After proving that this relation is in fact *computational*, we can extract the expected decision procedure for verifying blocks.

```
verify-block : Height → Ledger × Hash → Verifiable-Block → Maybe (Ledger × Hash)
```

Case study: the Streamlet protocol, mechanized. We have mechanized the Streamlet protocol [5]; one of the simplest in the BFT literature. As expected, the notions of **notarization** (when every block in a chain has votes from the majority) and **finalization** (when three consecutive epochs are notarized in sequence, the prefix up to the second block is necessarily *final*) are straightforward to make precise given a set of exchanged messages *ms*:

```
NotarizedBlock : Block → Type
NotarizedBlock b = length votes ≥ majority
  where
    votes = filter ((_≟ b) ∘ blockMessage) ms

NotarizedChain : Chain → Type
NotarizedChain = All NotarizedBlock
```

```
data FinalizedChain : Chain → Block → Type where
  Finalize : ∀ {ch b₁ b₂ b₃} →
    • NotarizedChain (b₃ ∷ b₂ ∷ b₁ ∷ ch)
    • b₃ .epoch ≡ suc (b₂ .epoch)
    • b₂ .epoch ≡ suc (b₁ .epoch)
    ─────────────────────────────
    FinalizedChain (b₂ ∷ b₁ ∷ ch) b₃
```

Moreover, the progression of the protocol can naturally be expressed as an inductive step relation; to save space we just show the expected type and one example step that records a finalized chain for a node:

```
data _—→_ : State → State → Type where
  FinalizeBlock : ∀ n ch b →
    FinalizedChain (s •messagesSoFar n) ch b
    ─────────────────────────────────────────
    s —→ finalize n ch s
```

```
s₀   —→⟨ ReceiveMessage? 𝔹 0 ⟩
s₁   —→⟨ Vote? 𝔹 [] [] ⟩
s₂   —→⟨ AdvanceEpoch ⟩
 ⋮    ⋮
sₙ₋₁ —→⟨ FinalizeBlock? 𝔸 [ b₆ ⨾ b₅ ⨾ b₂ ] b₇ ⟩
sₙ   ∎
```

Finally, we prove that all involved logical propositions are *decidable*, thus also producing an *executable specification* which can be used to run example traces as shown on the right (details omitted for brevity, proofs are automatically discharged via *proof-by-computation* [14]).

This is work in progress and we are currently closing in on a mechanized proof of *safety* (otherwise called 'consistency'): honest nodes always agree on their final chains (up to a prefix). We have not yet instantiated the general framework outlined above to the Streamlet case due to some pending decisions w.r.t. finalization, but we are confident that our framework is sufficiently equipped to cover this in principle.

Future work Next steps include formalizing other BFT protocols — in particular Simplex [4] and Jolteon [7] — with the hope that our framework sufficiently generalizes over all examples, as well as extracting executable Rust programs that can then be integrated into more lightweight methods such as simulation-based testing (particularly useful for properties that are difficult to formally prove).

References

[1] Harold Carr, Christa Jenkins, Mark Moir, Victor Cacciari Miraldo, and Lisandra Silva. Towards formal verification of HotStuff-based byzantine fault tolerant consensus in Agda. In Jyotirmoy V. Deshmukh, Klaus Havelund, and Ivan Perez, editors, *NASA Formal Methods - 14th International Symposium, NFM 2022, Pasadena, CA, USA, May 24-27, 2022, Proceedings*, volume 13260 of *Lecture Notes in Computer Science*, pages 616–635. Springer, 2022. doi:10.1007/978-3-031-06773-0_33.

[2] Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Jann Müller, Michael Peyton Jones, Polina Vinogradova, and Philip Wadler. Native custom tokens in the extended UTXO model. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Applications - 9th International Symposium on Leveraging Applications of Formal Methods, ISoLA 2020, Rhodes, Greece, October 20-30, 2020, Proceedings, Part III*, volume 12478 of *Lecture Notes in Computer Science*, pages 89–111. Springer, 2020. doi:10.1007/978-3-030-61467-6_7.

[3] Manuel M. T. Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, and Philip Wadler. The extended UTXO model. In Matthew Bernhard, Andrea Bracciali, L. Jean Camp, Shin'ichiro Matsuo, Alana Maurushat, Peter B. Rønne, and Massimiliano Sala, editors, *Financial Cryptography and Data Security - FC 2020 International Workshops, AsiaUSEC, CoDeFi, VOTING, and WTSC, Kota Kinabalu, Malaysia, February 14, 2020, Revised Selected Papers*, volume 12063 of *Lecture Notes in Computer Science*, pages 525–539. Springer, 2020. doi:10.1007/978-3-030-54455-3_37.

[4] Benjamin Y. Chan and Rafael Pass. Simplex consensus: A simple and fast consensus protocol. In Guy N. Rothblum and Hoeteck Wee, editors, *Theory of Cryptography - 21st International Conference, TCC 2023, Taipei, Taiwan, November 29 - December 2, 2023, Proceedings, Part IV*, volume 14372 of *Lecture Notes in Computer Science*, pages 452–479. Springer, 2023. doi:10.1007/978-3-031-48624-1_17.

[5] Benjamin Y. Chan and Elaine Shi. Streamlet: Textbook streamlined blockchains. In *AFT '20: 2nd ACM Conference on Advances in Financial Technologies, New York, NY, USA, October 21-23, 2020*, pages 1–11. ACM, 2020. doi:10.1145/3419614.3423256.

[6] Jared Corduan, Matthias Güdemann, and Polina Vinogradova. A formal specification of the Cardano ledger. https://github.com/input-output-hk/cardano-ledger/releases/latest/download/shelley-ledger.pdf, 2019.

[7] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In Ittay Eyal and Juan A. Garay, editors, *Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers*, volume 13411 of *Lecture Notes in Computer Science*, pages 296–315. Springer, 2022. doi:10.1007/978-3-031-18283-9_14.

[8] Andre Knispel, Orestis Melkonian, James Chapman, Alasdair Hill, Joosep Jääger, William DeMeo, and Ulf Norell. Formal specification of the Cardano blockchain ledger, mechanized in Agda. https://omelkonian.github.io/data/publications/cardano-ledger.pdf, 2024. under submission.

[9] Leslie Lamport. Paxos made simple. *ACM SIGACT News (Distributed Computing Column) 32, 4 (Whole Number 121, December 2001)*, pages 51–58, 2001.

[10] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

[11] Ulf Norell. Dependently typed programming in Agda. In *International School on Advanced Functional Programming*, pages 230–266. Springer, 2008.

[12] Vincent Rahli, Ivana Vukotic, Marcus Völp, and Paulo Jorge Esteves Veríssimo. Velisarios: Byzantine fault-tolerant protocols powered by Coq. In Amal Ahmed, editor, *Programming Languages and*

Systems - 27th European Symposium on Programming, ESOP 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings, volume 10801 of *Lecture Notes in Computer Science*, pages 619–650. Springer, 2018. doi:10.1007/978-3-319-89884-1_22.

[13] Søren Eller Thomsen and Bas Spitters. Formalizing Nakamoto-style proof of stake. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*, pages 1–15. IEEE, 2021. doi:10.1109/CSF51468.2021.00042.

[14] Paul Van Der Walt and Wouter Swierstra. Engineering proof by reflection in Agda. In *Symposium on Implementation and Application of Functional Languages*, pages 157–173. Springer, 2012.