Towards Mechanising Probabilistic Properties of a Blockchain

Kiran Gopinathan
University College London, UK
kiran.gopinathan.16@ucl.ac.uk

Ilya Sergey
Yale-NUS College and School of Computing, NUS, Singapore
ilya.sergey@yale-nus.edu.sg

Abstract

We present our progress on the formalisation and mechanisation of a probabilistic model of a blockchain consensus protocol in Coq, taking steps towards the formal verification of its security properties, stated in terms of probabilities, in an adversarial environment.

1 Introduction

Blockchain consensus protocols are a family of *open* distributed byzantine [5] consensus protocols, where an arbitrary number of (potentially malicious) parties can participate at any point of time. Previous formalisation efforts considered a simple model of blockchain-based consensus and established its basic *safety* properties in Coq [8], yet they did not address any *security* properties, which inherently require to incorporate some notion of probability.

Fundamentally, a blockchain consensus protocol allows a set of independent actors communicating over an asynchronous network to maintain a shared public ledger, robust to adversarial attacks; this is achieved by using the calculation of a pre-image of a hash function as a validation tool [6]. Any adversary attempting to disrupt the consensus must first produce a sufficiently long validated chain, an action that becomes increasingly improbable as the length of the public chain increases. The work on Bitcoin Backbone Protocol (BBP) by Garay *et al.* [3] stated the *Chain Growth* and *Common Prefix* properties. For example, the former states that, given certain bounds on the ratio of adversarial parties, a consensus on the *prefix of any chain* held by an honest actor could be guaranteed to a high likelihood in a semi-synchronous setting. In this proposed talk, we will present our ongoing work on the formalisation of the BBP model and the proof of its security properties in Coq.

2 A Model for Bitcoin Backbone Protocol

2.1 State-Space of a Byzantine Distributed System

To capture the semantics of the BBP model, our formalisation must encode the following components of the global system state:

- **Network State.** Our definition simulates a Δ-*bounded-synchronous* network (the network operation is modelled as occurring at the level of discrete rounds, such that all messages will by delivered after Δ rounds) through the use of a queue of fixed length Δ (*i.e.*, a delivery queue). To encode this, all messages sent by hosts in the network during a round are stored in a *message pool*; at the end of the round this collection is then placed into the queue, removing the first entry and delivering all its messages synchronously.
- **Blockchains.** We represent a blockchain in the system as a sequence of `Block` records [6, 8], where each `Block`, contains a sequence of transactions, the hash value ($\mathbb{N}_{[0..\kappa]} < T$) of the prior block and an integer proof-of-work [2]. As in the BBP model, T represents the globally fixed hashing difficulty and κ represents the output size of the hash function.
- **Actors and Adversaries.** We represent the state of *honest* actors as a collection of their local blockchains, received messages and transactions. We encode an *adversary* as an opaque parameterised type, thereby preventing introspection into its state and allowing for arbitrary adversarial strategies. We associate a boolean value with the internal state of each actor to record whether they are honest or not. This simple dichotomy allows the model to represent varying numbers of malicious actors during the execution, allowing the adversaries to corrupt honest actors.
- **Oracle state.** Oracles are a standard way of representing non-determinism within Coq [8]; we utilise an oracle to capture the non-deterministic nature of the hashing operation, whereby the hashing of an unseen block returns a random value. To ensure that the oracle produces consistent results (*i.e.*, hashing the same block twice produces the same output) we represent the state of the oracle as a map between `Block` records and $\mathbb{N}_{[0..2^\kappa]}$—an integer representing a κ-length hash result.

We encapsulate all these components of the state-space within the `World` data type, representing *the entire history* of the protocol execution until a certain round. As the state of the oracle is tied to the world state, representing the semantics of the system as a binary *relation* on worlds would prevent a probabilistic analysis of the random values returned by the oracle, we must instead define the semantics as a *probabilistic computation.*

2.2 Modelling Randomised System Executions

We encode the protocol semantics as the following step function:

$$\texttt{world\_step} : \texttt{World} \longrightarrow \texttt{seq RndGen} \longrightarrow \texttt{Comp}\ (\texttt{option World})$$

The first parameter is the initial world to start an execution from. The second parameter acts as a *schedule* [8], representing by the sequence of *events* and internal choices (`RndGen`) leading to the overall execution result. The system events from `seq RndGen` include generation of a transaction, the corruption of an honest actor, or a single call to a hash-function by an honest actor.

The `world_step` function, when provided an initial `World` and a schedule, iteratively consumes each event in the schedule and probabilistically updates the world state. The probabilistic result is represented by a value of type `option` to allow the execution to fail, as not all sequences of events are valid: for instance, it would not be valid for an honest actor to call the hash function if the adversary was active at that time. The crucial component of the `world_step` definition is its monadic return type `Comp`, representing the outcome of a randomised computation [9]. As embedded into Coq, expressions of type `Comp` define a domain-specific language for representing operations for generating and working with random bits [7]—precisely what we need to encode the random results of hashing used to generating transactions and blocks by actors.

Implemented in the style of the FCF library [7], randomised expressions (of type `Comp A` for some value type `A`, *e.g.*, `World`) provide an ergonomic Haskell-style `do`-notation for constructing randomised computations. For example, the following code snippet shows a part of our definition of `world_step` in Coq that draws random hash values from the corresponding primitive `hash`, while randomly generating a new world with a freshly minted block:

```
(* For a current state of a given honest actor... *)
let: tx_pool := get_honest_tx_pool state in
(* find a set of transactions to include in the new block *)
let: txs := get_latest_txs tx_pool best_chain in
(* calculate the hash of the new block *)
do (hash_result <-$ hash (nonce, txs) oracle_state;
   newWorld <-$ (* create a new world using hash_result *);
   return (Some newWorld))
```

2.3 Reasoning about Randomised Executions

As customary when reasoning about randomised algorithms, we state properties of our BBP executions in terms of probabilities.

Since the results of an execution (`Comp (option World)`) represent the "syntax" of probabilisitc distributions ($\texttt{dist}\ A : A \rightarrow \mathbb{R}_{[0,1]}$), we can convert our computations into distributions of possible results via the probability monad defined by Affeldt and Hagiwara [1]:

$$\begin{aligned} \texttt{bind} &: \texttt{dist}\ A \rightarrow (A \rightarrow \texttt{dist}\ B) \rightarrow \texttt{dist}\ B \\ \texttt{ret} &: A \rightarrow \texttt{dist}\ A \\ \texttt{eval\_dist} &: \texttt{Comp}\ A \rightarrow \texttt{dist}\ A \end{aligned}$$

Now suppose we wish to reason about the probability that a given property $\texttt{F} : \texttt{option World} \rightarrow \texttt{bool}$ holds. We can represent that by stating that for all worlds *reachable* from some initial world w_0 via some schedule *sc*, by expressing the statement in terms of probabilities as follows:

$$\forall sc, \forall w, \texttt{eval\_dist}\ (\texttt{world\_step}\ w_0\ sc)\ w > 0 \implies F\ w$$

This can be reformulated via more transitional notation:

$$\begin{aligned} &\forall sc, \text{P[}\ (\texttt{world\_step}\ w_0\ sc) \ \triangleright\ F\ ] = 1, \text{ where} \\ \text{P[}\ a = b\ ] &\triangleq \texttt{eval\_dist}\ a\ b \\ \text{P[}\ a\ ] &\triangleq \text{P[}\ a = true\ ] \\ \texttt{fmap} &: \texttt{Comp}\ A \rightarrow (A \rightarrow B) \rightarrow \texttt{Comp}\ B \\ c \triangleright f &\triangleq \texttt{fmap}\ f\ c \end{aligned}$$

3 Properties of the Bitcoin Backbone Protocol

Using constructions from Section 2, we state the security properties.

The first property, which asserts that there is an overall "progress" in the system with honest actors, relies on the auxiliary *characteristic* function $X'_i : \mathbb{N} \rightarrow \texttt{World} \rightarrow \{0, 1\}$, such that $X'_i\ w$ returns 1 *iff* the round i was *bounded successful* in a world w, *i.e.*, any honest actors were able to successfully mine a block during that round and all rounds from $i - \Delta$ to i had no successful mining attempts for the globally-fixed network delay Δ; otherwise, $X'_i\ w$ returns 0.

Definition 3.1 (Chain Growth Property)**.** The property CGP : `World` $\rightarrow$ `bool` is defined with respect to a finite number of rounds N_{rounds}, fixed number of actors max_{actors}, and holds *iff* for a given world w, any round $r \in [0, \ldots, N_{\text{rounds}}]$, blockchain c, actor address $addr \in [0, \ldots, max_{\text{actors}}]$, such that *addr* is honest in the world w and has c as its chain at round r, it is the case that for a "later" round s, such that $s > r + \Delta$, and any other actor $addr'$ in the system, whose chain in s is c', $len\ c' \geq \left(len\ c\ + \sum_{i \in [r..s-\Delta]} X'_i\ w\right)$.

The second property defines a "preservation"-like notion similar to the classical consensus in a randomised blockchain-based setting.

Definition 3.2 (Common Prefix Property)**.** The property CPP : `World` $\rightarrow$ `bool` is defined with respect to a consecutive sequence of rounds starting at i to j, a number of blocks k, and holds *iff* for a given world w, for any blockchains c_1, c_2, round $r \in [i, \ldots, j]$, such that chain c_1 is adopted by some actor[1] at round r, and c_2 is either adopted or diffused at round r, it is the case that pruning k blocks off the end of c_1 produces a prefix of c_2 and pruning k blocks off the end of c_2 produces a prefix of c_1.

Typical Executions A key innovation in the BBP proof is the choice to restrict the state of the model used to *exclude* exceptional situations, and instead consider the "average" case. Informally, this *typical execution* property can be described as follows:

Definition 3.3 (Typical Execution Property)**.** The property TEP_ε : `seq RndGen` $\rightarrow$ `World` $\rightarrow$ `bool` is defined *wrt.* a parameter $\varepsilon : \mathbb{R}_{(0,1)}$ and holds *iff* for a schedule *sc* and a world w resulting from *sc*: (a) the number of bounded successful rounds and bounded uniquely successful rounds for the world are no more than an ε-ratio below their expected value given the schedule; (b) the number of successful rounds for the world is less than an ε-ratio above its expected value; (c) the number of blocks hashed by the adversary is less than an ε-ratio above its expected value.

In our development, the typical nature of the considered executions is encoded by the following assumption.

Assumption 3.1 (Typical Executions)**.** $\forall sc : \texttt{seq RndGen}, \varepsilon : \mathbb{R}_{(0,1)}$,

$$\text{P[}\ \texttt{world\_step}\ sc \triangleright \text{TEP}_\varepsilon\ sc\ ] = 1 - e^{-\Omega(\kappa)},$$

where $\Omega(\kappa)$ refers to any function that grows linearly in κ.

3.1 Main Theorems

We state the main theorems in terms of the defined above properties. "Dotted" logical conjunction denotes its point-wise lifting to worlds.

Theorem 3.1 (Chain Growth Lemma)**.** $\forall sc : \texttt{seq RndGen}$,

$$\text{P[}\ \texttt{world\_step}\ sc\ w_0 \triangleright \text{CGP}\ ] = 1$$

Theorem 3.2 (Common Prefix Theorem)**.** $\forall sc, k : \mathbb{N}, k > 2\kappa \frac{T}{2^\kappa}$,

$$\begin{aligned} &\text{P[}\ \texttt{world\_step}\ sc\ w_0 \triangleright (\text{CPP}_k\ \dot{\wedge}\ \text{TEP}_\varepsilon\ sc)\ ] \ = \\ &\text{P[}\ \texttt{world\_step}\ sc\ w_0 \triangleright \text{TEP}_\varepsilon\ sc\ ] \end{aligned}$$

Care has been taken in the formulation of Theorem 3.2 to avoid incorporating the complex probabilities of the typical execution property into the theorem statement. Rather than proving the probability of a typical execution and the Common Prefix Property holding, we formulate the property as that given a typical execution, the Common Prefix Property holds with probability 1.

3.2 Elements of Our Mechanisation and Future Work

We built our mechanisation using the libraries by Affeldt and Hagiwara [1], which provides a probability framework implemented on top of the Ssreflect extension for Coq [4]. The mechanised versions of Definitions 3.1 and 3.2 are given as *decidable* predicates (*i.e.*, returning `bool`), and require a careful choice of types of values they quantify over, which all must be *finite*. This, "Ssreflect-style" approach pays off by giving an access to a large library of rewriting lemmas for Σ-notation of sums, enabling very concise proofs (mostly, by rewriting) of classical probability properties.

Our proofs of Theorems 3.1 and 3.2 are by induction on the length of the schedule and are partially complete; most remaining efforts relate to verifying the message delivery mechanism. In the future, we are also planning to promote the statement of Assumption 3.1 to a Lemma, as its paper-and-pencil proof is given in the BBP paper [3].

[1] We adopt the BBP terminology here, and define an actor as adopting a chain c at round r if the actor accepts chain c as its local chain during round r.

References

[1] Reynald Affeldt and Manabu Hagiwara. 2012. Formalization of Shannon's Theorems in SSReflect-Coq. In *ITP (LNCS)*, Vol. 7406. Springer, 233–249.

[2] Cynthia Dwork and Moni Naor. 1992. Pricing via Processing or Combatting Junk Mail. In *CRYPTO (LNCS)*, Vol. 740. Springer, 139–147.

[3] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin Backbone Protocol: Analysis and Applications. In *EUROCRYPT (Part 2) (LNCS)*, Vol. 9057. Springer, 281–310.

[4] Georges Gonthier, Assia Mahboubi, and Enrico Tassi. 2009. *A Small Scale Reflection Extension for the Coq system.* Technical Report 6455. Microsoft Research – Inria Joint Centre.

[5] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (1982), 382–401.

[6] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf.

[7] Adam Petcher and Greg Morrisett. 2015. The Foundational Cryptography Framework. In *POST (LNCS)*, Vol. 9036. Springer, 53–72.

[8] George Pîrlea and Ilya Sergey. 2018. Mechanising blockchain consensus. In *CPP*. ACM, 78–90.

[9] Norman Ramsey and Avi Pfeffer. 2002. Stochastic lambda calculus and monads of probability distributions. In *POPL*. ACM, 154–165.