Towards Verified Blockchain Architectures: A Case Study on Interactive Architecture Verification*

Diego Marmsoler

Technische Universität München, Germany
diego.marmsoler@tum.de

Abstract. With the emergence of cryptocurrencies, Blockchain architectures have become more and more important. In such architectures, components maintain and exchange a list of records in a way which makes the entries persistent, i.e., resistant to modifications. Thereby, the architecture is dynamic in the sense that components may join or leave the network and connections between them may change over time. The dynamic nature of Blockchain architectures makes their verification a challenge, since it involves reasoning about potentially unbounded number of components. To this end, we developed FACTUM, an approach for the specification and interactive verification of dynamic architectures based on the interactive theorem prover Isabelle. In this paper we report on the outcome of applying the approach to formally specify a version of Blockchain architectures and verify that the list entries of such architectures are indeed persistent.

Keywords: Blockchain, Interactive Theorem Proving, Dynamic Architectures, FACTum, Isabelle

1 Introduction

The concept of Blockchain was first introduced with the invention of the Bitcoin cryptocurrency by a person (or group) known as Satoshi Nakamoto in 2008 [26]. Since then, the technology found several other applications, especially in the domain of cryptocurrencies [4]. However, the technology seems promising also for other domains, such as the medical [3], land management [7], business process management [24], or even identity management [35]. Usually, the term "*blockchain*" refers to a list of records, so-called *blocks*, which contain actual data elements. A *Blockchain architecture*, on the other hand, consists of a network of so-called *nodes*, in which every node maintains a copy of the blockchain and continuously exchanges its copy with other nodes. Thereby, blockchains are required to be *persistent*, i.e., entries should be resistant to modifications. To achieve this, nodes are required to follow a certain protocol consisting of several, so-called, *consensus rules*.

Blockchain architectures are an instance of a more general class of architectures called dynamic architectures [22]. In such architectures, components may join or leave

* This is a post-peer-review, pre-copyedit version of an article to be published in the proceedings of the 39th International Conference on Formal Techniques for Distributed Objects, Components, and Systems, which will appear in Springer's LNCS-IFIP volume series.

the architecture and connections between components can change over time. This dynamics makes the verification of such architectures a challenge, since it involves reasoning about an unbounded number of components.

In an attempt to address this problem, we developed FACTUM [18], an approach for the specification and verification of such architectures. A FACTUM specification consists of three main parts:

- A specification of the involved data types in terms of abstract datatypes.
- A specification of the involved types of components in terms of interfaces and corresponding assertions about the behavior of components of a certain type.
- A set of architectural assertions to specify component activation and reconfiguration of connections between components.

A FACTUM specification can be systematically transferred to a corresponding Isabelle [27] theory where it is subject to interactive verification.

While the general FACTUM approach was already introduced in [18], the focus of [18] was the presentation and discussion of the specification techniques and the algorithm to map a FACTUM specification to a corresponding Isabelle locale. To this end, we demonstrated the algorithm by means of three simple examples: a Singleton architecture, a Publisher-Subscriber architecture, and a Blackboard architecture, amounting up to 500 lines of Isabelle code. With this paper, we build on the work described in [18] and evaluate the approach on a larger case study. To this end, we applied the approach to specify Blockchain architectures based on the description provided in [26] and verify persistency of confirmed blocks. Thus, the contribution of the paper is twofold:

- It describes a case study for FACTUM, which reveals important insights about the use of FACTUM for the verification of dynamic architectures.
- It provides a formal specification of Blockchain architectures, which is guaranteed to resist double spend attacks.

In total, the specification consists of 12 assumptions for Blockchain architectures and verification required roughly 3500 lines of Isabelle/HOL code.

In the next section, we provide some background on Blockchains (Sect. 2) and the FACTUM approach (Sect. 3). We then present a possible specification of Blockchain architectures (Sect. 4) and describe our formalization and verification of the persistence property for blockchain entries (Sect. 5). We continue with a discussion of related work (Sect. 7) in the area of formalizations of blockchain-related concepts and verification of consensus algorithms. We conclude our presentation with a summary of major results and a discussion of its implications as well as directions for future work (Sect. 8).

2 Blockchain Architectures

Blockchain architectures were first introduced with the invention of the Bitcoin cryptocurrency [26]. In cryptocurrencies, a digital coin is usually passed from one owner to the next one by digitally signing an electronic transaction. To ensure that coins are only spent once, a payee has to know whether a received coin is already spent or not at the time he receives it. This problem is known as the *double spend problem* and before the invention of Blockchain, it was solved using a central, trusted identity, which knew every transaction of the system and confirmed that a coin was not already spent. In an

attempt to avoid such central authorities, Bitcoin proposed a system called Blockchain to solve the double spend problem in a distributed, peer-to-peer network. To this end, the network stores a continuously growing list of persistent entries, which contain the actual money transactions. The list is shared among all participants of the network and by inspecting it, a node can independently verify that a coin was not already spent. In this paper, we call such a network a *Blockchain architecture* and in the following we summarize some basic concepts of such architectures. Thereby, we follow the informal description provided in [26].

Blockchain. The term "blockchain" usually refers to the major data structure involved in a Blockchain architecture: a list of records aka. *blocks*. Blocks, on the other hand, contain the actual data elements, for example, money transactions in cryptocurrency applications. Blocks can be added on top of the chain and verified by a process known as *mining*. In Bitcoin, for example, mining involves the guessing of a random number (a so-called *nonce*), adding it to a candidate block and checking whether the corresponding hash exhibits a certain form (starting with a certain number of zeros). This makes mining of a new block computationally expensive, since it usually requires many guesses (and subsequent hashings) to find a number which produces the right hash. On the other hand, ensuring that a given block was indeed successfully mined remains computationally cheap (it only requires a single hashing).

Consensus. In a Blockchain architecture, every node maintains a local copy of the blockchain, which it exchanges with its peers. Due to the distributed nature, it may happen that two different blocks are added concurrently, resulting in two different versions of the blockchain available in the network. In order to reach a *consensus* on which version is the "right" one, a Blockchain architecture usually comes with a strategy of how to select the right version from a set of competing blockchains. This rule is applied by every *honest* node of the network and should guarantee that the nodes eventually reach a consensus.

Consensus rules. There are several different types of strategies used to reach consensus, such as proof-of-work [26] or proof-of-stake [4]. In the proposed specification, we rely on the *proof-of-work* concept also used by Bitcoin and related applications. It is based on the observation that the number of blocks in a blockchain usually represents the amount of computing power involved to build this chain. Thus, the largest chain from a set of competing blockchains must be the one accepted by the majority of the network. Thus, if a honest node is facing two versions of a blockchain, it is required to always choose the longer one.

Confirmation blocks. In a proof-of-work network, every CPU gets one vote and majority decisions can usually only be manipulated if one entity owns more than 50% of the computing power of the network. This might not be true, however, for blocks added to the blockchain only recently. A single node may just be lucky and guess the right nonce fast, without investing a lot of computational power. To cope with such lucky guesses, one usually waits for some blocks to be mined on top of the block containing a certain transaction, to accept this transaction as completed. These blocks are called *confirmation blocks* and in Bitcoin, for example, it is suggested to wait for at least six confirmation blocks to accept a transaction as completed [31].

3 FACTum

FACTUM [18] is an approach for the formal specification and interactive verification of dynamic architectures. It consists of a formal system model for dynamic architectures, techniques to specify architectures over this model, an algorithm to map the specification to a corresponding Isabelle theory, and an Isabelle-based framework to support the interactive verification of architecture specifications. FACTUM is also implemented in terms of an Eclipse/EMF application called FACTUM Studio [21] which supports a user in the development of architecture specifications.

3.1 System Model

In FACTum, an architecture is modeled in terms of *sets* of so-called *architecture traces* [15,22], i.e., streams [6] of architecture snapshots. Thereby, an *architecture snapshot* consists of a set of (active) components with their ports valuated by messages and connections between the ports of the components. Moreover, components of a certain type may be parameterized by a set of messages.

Example 1 (Architecture trace). Assuming that $\mathtt{M}_1$, $\mathtt{M}_2$, ... are sets of messages. Figure 1 depicts an architecture trace t with corresponding architecture snapshots $t(0) = k_0$, $t(1) = k_1$, and $t(2) = k_2$. Architecture snapshot k_0, for example, consists of three active components: c_1, c_2, and c_3. Component c_3 is parameterized with a parameter p with value $\mathtt{M}$. It has two input ports i_0 and i_1, valuated with messages $\mathtt{M}_2$ and $\mathtt{M}_1$, respectively. Moreover, it has two output ports o_0 and o_1, valuated with messages $\mathtt{M}_1$ and $\mathtt{M}_3$, respectively. □

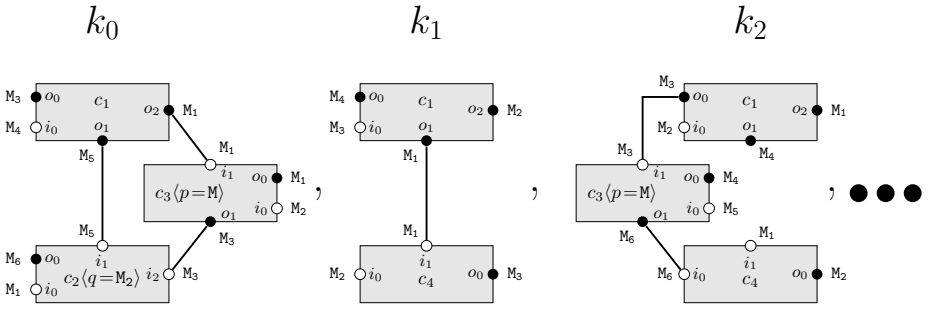

Fig. 1: Architecture trace with its first three architecture snapshots.

Note that the model allows components to be valuated by a set of messages, rather than just a single message, at each point in time. Moreover, components can be activated and deactivated and connections between them may change over time. The model of architecture traces is also implemented by a corresponding Isabelle/HOL theory, which is described in [17].

3.2 Specifying Dynamic Architectures

FACTUM provides several techniques to support the formal specification of dynamic architectures [18]:

- First, the data types involved in an architecture are specified in terms of algebraic specifications [33].
- Then, a set of interfaces is specified graphically using architecture diagrams.
- Component types are then created by adding constraints about component behavior to the corresponding interfaces.
- Finally, a set of architectural assertions is added to specify constraints about component activation and deactivation as well as interconnection.

A FACTUM specification comes with a formal semantics in a denotational style, which is described in [20]. To this end, each specification is interpreted by a corresponding set of architecture traces.

Constraints about component behavior are specified in terms of *behavior trace assertions*, i.e., first order linear temporal logic formulæ using ports of the interfaces as free variables. Architectural constraints are specified in terms of *architecture trace assertions*. These are also a type of first order linear temporal logic formulæ, with variables denoting components and some special terms and predicates:

- With $c.p$, for example, we denote the valuation of port p of a component c.
- With $\{\!|c|\!\}$ we denote that component c is currently active.
- With $c.o \rightsquigarrow c'.i$ we denote that output port o of component c is connected to input port i of component c'.

Architecture diagrams are a graphical formalism to specify interfaces for component types. To this end, component types are represented by rectangles with their ports denoted by empty (input) and filled (output) circles. Architecture diagrams may be annotated to easily express common architectural constraints:

Activation annotations can be added to component types, to specify upper and lower bounds for the number of active components of the corresponding type.

Connection annotations are expressed in terms of annotated lines between the ports of component types, to express upper and lower bounds for connections between the ports of corresponding components.

Note that activation and connection annotations are actually just synonyms for certain architectural assertions and may also be expressed using architecture trace assertions described above.

3.3 Verifying Dynamic Architectures

FACTUM comes with an algorithm to map a given specification to a corresponding Isabelle theory, where it is subject to formal verification. To support the verification, FACTUM provides a framework for the interactive verification of architecture specifications in Isabelle/HOL [17]. Among other things, the framework implements a calculus to support reasoning about component behavior in a dynamic environment [16].

4 Formalizing Blockchain Architectures

In the following, we present our formalization of Blockchain architectures in FACTUM.

4.1 Data Types and Ports

As described in Sect. 2, a key data type for Blockchain architectures is the *blockchain* itself. In the following, we first formalize a blockchain data structure by means of algebraic datatypes. Then, we specify two types of ports to send and receive blockchains, respectively.

Blockchains. A blockchain is modeled as a parametric list, in which the nature of the list entries (the blocks) depends on the concrete application context of the pattern. In cryptocurrency applications, for example, a block is usually a set of transactions. In other applications, however, blocks could be of a different type.

Figure 2a depicts a specification of blockchains by means of an abstract data type specification. First, a parametric sort $\langle \mathsf{B} \rangle \mathtt{BC}$ is introduced as a synonym for a corresponding list. Thereby, the type of blocks is denoted with type parameter B. In addition, we specify a function symbol MAX for blockchains, which takes a set of blockchains, and returns a blockchain with maximal length. Thus, we require two characteristic properties for MAX: Eq. (1) requires that a maximal blockchain of a set of blockchains BC is part of BC itself. In addition, Eq. (2) requires that MAX is indeed maximal, i.e., that the length of every other blockchain of the corresponding set BC is less or equal to the length of MAX. Note that $MAX(BC)$ is guaranteed to exist, whenever $BC \neq \emptyset$ and BC is *finite*.

Port types. Figure 2b specifies two types of ports which can be used to exchange blockchains: *pin* for input ports and *pout* for output ports. They will be used later on for the specification of component type interfaces.

DTSpec Blockchain	**imports** $\langle \mathsf{B} \rangle \mathtt{LIST}$ **as** $\langle \mathsf{B} \rangle \mathtt{BC}$
MAX :	$\wp(\langle \mathsf{B} \rangle \mathtt{BC}) \to \langle \mathsf{B} \rangle \mathtt{BC}$
flex BC :	$\wp(\langle \mathsf{B} \rangle \mathtt{BC})$
bc :	$\mathtt{BC}$
$MAX(BC) \in BC$	(1)
$\forall bc \in BC : \#bc \leq \#MAX(BC)$	(2)

(a) Data type specification.

PSpec BPort	
pin :	$\langle \mathsf{B} \rangle \mathtt{BC}$
$pout$:	$\langle \mathsf{B} \rangle \mathtt{BC}$

(b) Port specification.

Fig. 2: Data types and ports for Blockchain architectures.

4.2 Component Types

As described in Sect. 2, the components involved in a Blockchain architecture are called *nodes*. In the following, we first describe the syntactic interface of such a node component. Then, we introduce some auxiliary definitions for nodes. Finally, we provide a set of characteristic properties for a node's behavior.

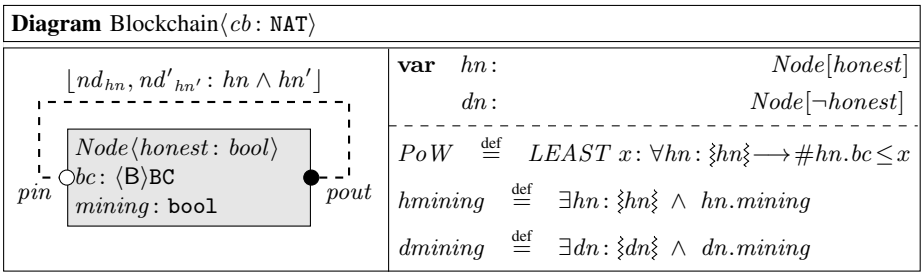

Fig. 3: Architecture diagram for Blockchain architectures.

Interfaces. The architecture diagram depicted in Fig. 3 is parameterized by a number of confirmation blocks cb and specifies the syntactic interface of Blockchain nodes. Actually, the diagram also contains a graphical representation of a connection constraint as well as the definition of three auxiliary definitions for nodes. For now, we may just ignore these additional aspects and focus on the description of the interface. We will, however, come back to the auxiliary definitions in the next section and we will discuss the connection constraint later on in Sect. 4.3.

Recall that a node in a Blockchain may either be honest or dishonest. Thus, a node is parameterized by a boolean value $honest$, which means that every component of type node is associated with a boolean value, which determines its trustworthiness. In addition, a node has two state variables: variable bc keeps a local copy of the blockchain and variable $mining$ signals the mining of a new block. Finally, a node may exchange blockchains via its input port pin and output port $pout$.

Auxiliary definitions. To support subsequent development, the right hand side of Fig. 3 introduces three auxiliary definitions for nodes: honest proof-of-work and honest/dishonest mining.

Honest proof-of-work. Honest proof-of-work (PoW) represents the *maximal* proof-of-work, currently available in the honest community. Since proof-of-work corresponds to the length of a blockchain (Sect. 2), honest proof-of-work is defined as the least upper bound for the length of honest blockchains, i.e. blockchains of active ($\{\!|hn|\!\}$) and honest ($Node[honest]$) nodes. Note the use of the definite description operator $LEAST$ to denote the *least* element x which satisfies a certain condition.

Honest and dishonest mining. Honest mining ($hmining$) is a predicate to denote the successful mining by some honest node. Similarly, dishonest mining ($dmining$) signals the mining by some dishonest node. Both predicates are interpreted over an architecture state and require the existence of a honest/dishonest node, which currently finished mining. Honest and dishonest mining play an important role for the formalization of a fundamental assumption for Blockchain architectures later on.

Behavior. The behavior of nodes is formalized in terms of behavior trace assertions (described in Sect. 3).

Honest nodes. The behavior of honest nodes is specified in Fig. 4 (with $\bigcirc P$ and $\Box P$ we denote taht P is true in the next state or in all future states, respectively). First, we introduce several variables to denote single blocks (b) and blockchains (c and c'). Note

the distinction between "flexible" and "rigid" variables: while "flexible" variables may be newly interpreted at each point in time, "rigid" variables keep their value over time. Then, we require three assertions for a honest node's behavior: Eq. (3) requires that a new node is initialized by the empty blockchain while Eq. (4) requires that every honest node always forwards a copy of its local blockchain to the network through its output port $pout$. Eq. (5) formalizes the consensus rule for honest nodes, which (according to Sect. 2) requires that a honest node always takes the blockchain with maximal proof-of-work as the current one, i.e, if a honest node receives a blockchain on its input with more proof-of-work than its own blockchain, then it will accept that blockchain as the current one. Its formalization consists of two parts: The antecedent characterizes the blockchain taken by a honest node:

$$c = \begin{cases} MAX(pin) & \text{if } \exists c' \in pin\colon \#c' > \#bc, \\ bc & \text{else.} \end{cases}$$

Since the proof-of-work for a blockchain is given by its length, the property fixes a blockchain c, which is either a maximal blockchain from its input port pin (for the case that it is strictly longer than its own blockchain), or its own blockchain bc (for the case that no blockchain from its input is longer than its own blockchain). The consequent formalizes the mining process:

$$\bigcirc\big(\neg mining \wedge bc = c \vee mining \wedge \exists b\colon\ bc = c@b\big).$$

Thereby, a honest node may either mine a new block ($mining$), append it to c and take the resulting chain as its current blockchain bc, or it may not mine any new block ($\neg mining$) and just set c as its current blockchain bc.

BSpec Blockchain		**for** $Node\langle honest\rangle$ **of** Blockchain
flex	b:	B
	c':	$\mathtt{BC}\langle\mathsf{B}\rangle$
rig	c:	$\mathtt{BC}\langle\mathsf{B}\rangle$
$bc = []$		(3)
$\Box\big(pout = bc\big)$		(4)
$\Box\Bigg(c = \begin{cases} MAX(pin) & \text{if } \exists c' \in pin\colon \#c' > \#bc, \\ bc & \text{else.} \end{cases}$ $\longrightarrow \bigcirc\big(\neg mining \wedge bc = c \vee mining \wedge \exists b\colon\ bc = c@b\big)\Bigg)$		(5)

Fig. 4: Specification of behavior for honest nodes.

Dishonest nodes. The attacker model is given by the specification of the behavior for dishonest nodes in Fig. 5.. Similar as for honest nodes, Eq. (6) requires that a new node is initialized by the empty blockchain. Additional behavior is characterized by Eq. (7). Note that, compared to honest nodes, dishonest nodes may not follow the consensus rules. Thus, while honest nodes always take the blockchain with the most proof-of-work

as their current blockchain, dishonest nodes may take every blockchain from its input as their current one. Moreover, in contrast to honest nodes, dishonest nodes may also drop elements from a blockchain, thus trying to modify a blockchain's history. The formalization consists of two parts. The antecedent first characterizes a blockchain c:

$$c \in (pin \cup \{bc\})$$

The consequent is similar to the one for honest nodes:

$$\bigcirc\big(\neg mining \wedge bc \sqsubseteq c \vee mining \wedge \exists b\colon\ bc = c@b\big)$$

Note that, due to computing restrictions, even dishonest nodes may at most mine one single block at a time. Thus, the mining case is indeed the same as for honest nodes. The difference, however, comes with the case in which no new block is mined. While, for such a case, honest nodes are required to take c as their current blockchain, dishonest nodes may take an arbitrary prefix of c as their current blockchain.

BSpec Blockchain		**for** $Node\langle\neg honest\rangle$ **of** Blockchain
flex	b:	B
rig	c:	BC⟨B⟩
$bc = []$		(6)
$\square\left(c \in (pin \cup \{bc\}) \longrightarrow \bigcirc\big(\neg mining \wedge bc \sqsubseteq c \vee mining \wedge \exists b\colon\ bc = c@b\big)\right)$		(7)

Fig. 5: Specification of behavior for dishonest nodes.

4.3 Architectural Constraints

Architectural constraints restrict activation and deactivation of components and connections between component ports [15,22]. They are mainly formulated in terms of architecture trace assertions, i.e., linear temporal logic formulæ, formulated over component ports[1]. Certain constraints, however, can be expressed more easily in a graphical manner, by annotating the pattern's architecture diagram. In the following, we first discuss connection constraints for Blockchain architectures. Then, we present some basic activation constraints for such architectures. Finally, we conclude the section with a description of a fundamental constraint for Blockchain architectures, which is essential to guarantee persistence of blockchain entries.

Connection constraints. Connection constraints restrict connections between component ports and therefore they affect the topology of an architecture. For our pattern of Blockchain architectures, we require a single connection constraint, which is expressed graphically by an annotation of the architecture diagram, depicted in Fig. 3. The dashed

[1] Architecture trace assertions are summarized in Sect. 3

connection between a nodes input and output ports expresses a conditional connection between ports $pout$ and pin of two (possible different) components of type node. The *minimal* condition for the connection to happen is expressed by the annotation

$$\lfloor nd_{hn}, nd'_{hn'} : hn \wedge hn' \rfloor.$$

The condition essentially requires the ports to be connected, whenever two components are $honest$. Roughly speaking, the constraint requires that every honest node is connected to every other honest node of the network. While this constraint is indeed a strong requirement, it is necessary to guarantee persistence of blockchain entries.

Basic activation constraints. Activation constraints affect the activation and deactivation of components of a certain type. We require four basic activation constraints for Blockchain architectures, summarized in Fig. 6 (with $\ominus P$ we denote that P was true in the previous state) and explained in more detail in the following. *Finite number of active*

ASpec Basic		**for** Blockchain
flex	bc:	$\mathtt{BC}\langle \mathsf{B} \rangle$
	nd:	$Node\langle hn \rangle$
	nd':	$Node\langle hn' \rangle$
rig	hn:	$Node[honest]$
$\Box\Big(finite\big(\{\, nd \mid \S nd\S \}\big)\Big)$		(8)
$\Box\Big(\exists hn : \S hn\S \wedge \bigcirc \S hn\S\Big)$		(9)
$\Box\Big(\S hn\S \wedge hn.mining \longrightarrow \ominus \S hn\S\Big)$		(10)
$\Box\Big(\S nd\S \wedge bc \in nd.pin \longrightarrow \exists nd' : \S nd'\S \wedge nd'.bc = bc\Big)$		(11)

Fig. 6: Basic activation constraints for Blockchain architectures.

nodes. Our first activation property for Blockchain architectures is more of technical nature and restricts the number of active components at each point in time. By Eq. (8), we require that at each point in time, only a finite number of node components can be active. The property should be satisfied by every architecture found in practice. However, it is needed to guarantee that at every point in time, a node component receives only a finite number of blockchains which, in turn, is required to guarantee the existence of a maximal blockchain for a component's input port.

Keeping the honest blockchain. The second activation property we require for Blockchain architectures is needed to guarantee that the honest blockchain, i.e., the blockchain accepted by honest nodes as the "correct" one, is not lost. It is formalized by Eq. (9) and requires that at every point in time, there exists an active and honest node, which stays active for at least one time step. Thus, it is guaranteed that the current honest blockchain is stored by the honest network and does not get lost.

Mining on most recent blockchain. Another basic activation property for Blockchain architectures is needed to ensure that the honest network indeed collaborates in the

mining process. The property is formalized by Eq. (10) using the previous operator: it requires that whenever a honest node is mining a new block, this node was active at the time point right before the mining happened. This ensures that the node had indeed access to the most recent version of the honest blockchain and works on extending this version instead of an older one.

Closed architecture. The last basic activation property for Blockchain architectures requires such an architecture to be closed. Eq. (11) formalizes the property and requires that for every blockchain available at the input of any active node component at any point in time, there exists a corresponding active node component which provides the blockchain at its output. In other words, the property guarantees that every blockchain available in the architecture was build up by the network via the mining process and not injected from the outside.

A fundamental assumption for Blockchain architectures. In the following section, we present a fundamental constraint for Blockchain architectures. Since its specification requires to express mining frequencies, we first introduce an operator to express such frequencies in LTL. The operator can be used to express statements of the form: "for every time span in which at least x states can be observed which satisfy a certain property φ, at least y states can be observed to satisfy a certain property φ'".

Definition 1 (Weak until for relative frequencies). *A trace t satisfies $\varphi \ {}_{\lceil x \rceil}\mathcal{W}_{\lfloor y \rfloor} \ \varphi'$, for state predicates φ and φ', at time point n, iff*

$$\begin{aligned} &\exists n' \geq n \colon cc(t, n, n', \varphi') \geq y \ \ \wedge \ \ (\forall n \leq i < n' \colon cc(t, n, i, \varphi) \leq x) \\ &\quad \vee \ \ (\forall n' \geq n \colon cc(t, n, n', \varphi) \leq x), \end{aligned}$$

with $cc(t, n, n', p) \quad \stackrel{def}{=} \quad |\{i \mid i > n \wedge i \leq n' \wedge p(t(i))\}|$.

In Fig. 7 we use the newly introduced operator to formalize a fundamental requirement for Blockchain architectures. Roughly speaking, the property requires that for every time span in which we can observe a number of dishonest minings which is *greater or equal* to the number of confirmation blocks cb, then we can also observe a number of honest minings which is *greater* than the number of confirmation blocks. Note that this is an important requirement needed to guarantee persistence of blockchain entries.

ASpec Blockchain	**for** Blockchain
$\square\Big(umining \ {}_{\lceil cb \rceil} \ \mathcal{W} \ {}_{\lfloor cb+1 \rfloor} \ tmining\Big)$	(12)

Fig. 7: Fundamental assumption for Blockchain architectures.

5 Verifying Blockchain Architectures

We verified an important property for Blockchain architectures which ensures persistence of blockchain entries.

5.1 Persistence of Blockchain Entries

As described in the introduction, Blockchain architectures were invented to solve the double spend problem in a distributed peer-to-peer network. In order to do so, blockchain entries, once accepted by the network, must be resistant to future modifications. This property is summarized by the following theorem:

Theorem 1 (Persistence of blockchain entries). *In a Blockchain architecture, the entries of honest blockchains, which are confirmed by a number of blocks greater or equal to the number of confirmation blocks, are resistant to future modifications.*

The theorem is formally specified by the architectural assertion depicted in Fig. 8 (with $\boxminus P$ we denote that P was true in all previous states). To this end, sbc denotes a blockchain which contains the entries supposed to be persistent and Eq. (13) - Eq. (16) characterize a time point n_s, for which the property actually holds.

ASpec Save		**for** Blockchain
flex	hn:	$Node[honest]$
	dn:	$Node[\neg honest]$
	nd:	$Node$
rig	hn':	$Node[honest]$
	sbc:	$\langle \mathsf{B} \rangle \mathtt{BC}$

$$\boxminus\Big(\Big(\forall hn' \colon \big(\neg \{\!| hn' |\!\}\big)\ \mathcal{W}\ \big(\{\!| hn' |\!\} \wedge sbc \sqsubseteq hn'.bc\big)\Big)\ \wedge \tag{13}$$

$$PoW \geq \#sbc + cb\ \wedge \tag{14}$$

$$\Big(\forall dn \colon \{\!| dn |\!\} \longrightarrow \#dn.bc < \#sbc\Big)\ \wedge \tag{15}$$

$$\ominus\boxminus\Big(\forall nd \colon \{\!| nd |\!\} \longrightarrow \#nd.bc < \#sbc \vee sbc \sqsubseteq nd.bc\Big)\ \wedge \tag{16}$$

$$\longrightarrow\quad \boxminus\Big(\forall hn \colon \{\!| hn |\!\} \longrightarrow sbc \sqsubseteq hn.bc\Big)\Big) \tag{17}$$

Fig. 8: Specification of persistence property for Blockchain architectures.

Eq. (13) requires that sbc is indeed a prefix of the blockchain of every honest node hn' at hn''s first activation after n_s. It basically ensures that the honest network is initialized with blockchains extending sbc.

Eq. (14) requires the proof-of-work at time point n_s to be greater or equal to the length of sbc, increased by the number of confirmation blocks cb. This equation is required to provide the honest network with some lead over a potential attacker, which might want to change sbc. Note, however, that the assumption is indeed feasible, since Thm. 1 ensures persistence only of entries which were confirmed by cb number of blocks.

Eq. (15) requires the length of the blockchain of every active and dishonest node dn to be less than the length of sbc. Together with Eq. (16), this equation ensures that

a potential attacker did not prepare a "false" blockchain before time point n_s, which he could then use later on to cheat the honest network.

Eq. (16) requires for every node's blockchain $nd.bc$, at every time point before n_s, that sbc is either a prefix of $nd.bc$ or that the length of $nd.bc$ is smaller than the length of sbc.

For every time point n_s, for which the above conditions hold, the property depicted in Fig. 8 guarantees that sbc will always be a prefix of every honest node's blockchain (formalized by Eq. (17)).

5.2 Verification Effort

The pattern's specification (as presented in Sect. 4) was formalized in three different Isabelle/HOL theories, which are available via the Archive of Formal Proofs in [19]:

- a theory `Auxiliary`, which contains some auxiliary results, such as custom induction rules;
- a theory `RF_LTL`, which contains a calculus for Blockchain architectures, based on counting LTL;
- a theory `Blockchain`, which is the main theory containing the actual formalization of the pattern.

Theorem 1 was then formalized as theorem *blockchain-save* in theory `Blockchain` and mechanically verified in Isabelle. Its proof consists of roughly 3 500 lines of Isabelle/Isar code and required an effort of roughly three person months (by a person with around two years of experience in using Isabelle).

6 Discussion

We admit that the specification presented in Sect. 4 is somehow idealized and some of the assumptions may not always hold. Thus, to better understand when the results can be applied, we discuss some of these assumptions in more detail.

Cryptographic aspects. Cryptography is an important aspect when it comes to Blockchain. For example, some Blockchain implementations make extensive use of Merkle tree's [25] to ensure integrity of blockchains. With the work presented in this paper, we abstracted from cryptographic aspects. Rather, we assumed integrity of blockchains and focused on the problem of building consensus in a way to resist double spend attacks. Of course, flaws in the implementation of the integrity mechanism might lead to situations in which the results presented in this paper are not valid anymore. Thus, for such applications, one first needs to verify correctness of the employed integrity mechanism. Only then, our results can be applied to support the verification.

Probabilistic aspects. In Blockchain, the process of mining new blocks is usually of probabilistic nature and thus, it is actually difficult to provide any "hard" guarantees. The reason why we could provide such a guarantee here, is the probabilistic nature of the assumption provided by Eq. (12). In a real-world setting, the assumption is usually only valid with a certain probability. Thus, also the corresponding guarantee, provided by Thm. 1, is only valid with a certain probability. Hence, to use the results presented in this paper for a concrete setting, one first needs to verify (or estimate) the probability

of Eq. (12) to be true in this setting. This is then also the probability of Thm. 1 to be true in this setting.

Broadcast. Another limitation of the specification presented in this paper is the connection constraint provided by Fig. 3, which requires honest nodes to be always connected. While this may seem too strict, it indeed reflects a real problem in Blockchain networks, such as Bitcoin, in which "resilience to the double spending attack relies strongly on the assumption that Bitcoin's P2P network is connected, and that honest nodes are able to communicate." [36] Thus, to ensure that Thm. 1 holds, and thus the corresponding Blockchain network indeed resists double spend attacks, the network needs to employ mechanisms to ensure a high degree of connectivity for the honest sub-network.

The attacker model. The attacker model presented in Fig. 5 does not allow the instantaneous modification of blocks within a blockchain. Rather, modifying an entry can only be done by first removing corresponding entries from the top of the blockchain and then to add new blocks over time. This assumption is based on two fundamental design decisions inherent in bitcoin-like Blockchain applications: First, as already discussed above, such Blockchain applications usually employ Merkle tree's to ensure integrity of blockchains. Second, adding new blocks to a blockchain is done through mining, which usually requires some time and cannot happen instantaneous.

7 Related Work

This paper provides a formalization of Blockchain architectures and a mechanized proof of an important safety property regarding integrity of blockchain entries. Thus, related work can be found in formalizations of Blockchain architectures in general, as well as verification of consensus algorithms, specifically.

7.1 Formalizations of Blockchain Concepts

There has been some work in formalizing and investigating different aspects of Blockchain technologies. A lot of research in this area is devoted to the formalization of concrete technological implementations. The Ethereum Virtual Machine and its contract language Solidity, for example, are formalized in Coq [12] and Isabelle/HOL [11], respectively. Another interesting branch of research in this area concerns the study of so-called smart contracts. Such contracts can be used to associate transactions with code, which execution is triggered by certain events. A proposal to formalize such contracts is provided by Bhargavan [5]. Approaches for their verification were made based on behavior models [1], Finite State Machines [23], or interactive theorem proving [2].

Relation to our work: The studies described so far report on the formalization of various types of concepts found in Blockchain technology. Thus, they provide many insights into the formalization and even mechanization of various concepts used in Blockchain. The main difference to our work lies in the scope of these studies: while they focus on the details of these different concepts, we try to integrate them at a more abstract level in a so-called Blockchain architecture. One exception here is Pirlea's recent work [29] which goes in a similar direction to our work. The authors try to come up with an abstract model of Blockchain, which we would consider a Blockchain architecture, in Coq. What

is interesting is that they identify important aspects of Blockchain architectures and provide abstract notions for them. Specifically, they introduce an abstract notion of proof object and a so-called validator acceptance function, which is used to ensure validity of a block w.r.t. a specific proof object. Moreover, they abstract from the concrete consensus agreement, called Fork Choice Rule in an abstract function, which they require to form a total order between blockchains. These abstractions allow their model to be applied to various scenarios. While, with our work, we follow a similar approach, there are some notable differences: (i) First, with our implementation in Isabelle/HOL we provide an alternative framework for Isabelle/HOL users. (ii) A more important difference, however, concerns the scope of the proved property: In their work, the authors verified that a Blockchain architecture, in a consistent state, will eventually reach a consistent state again. In our work, we were rather interested in blockchain integrity, i.e., that additions to the blockchain are guaranteed to be persistent. (iii) Finally, in their work, they do not consider possible attackers. As shown in this work, these nodes may have different behavior and we were interested whether this could influence integrity.

7.2 Verification of Consensus Algorithms

Consensus mechanisms for Blockchain architectures are actually an instance of more traditional, distributed fault tolerance protocols. Such protocols were intensively studied over the last decades and mechanical verifications exist, for example, for Paxos [8,13], Raft [32,34], and the classical Two-Phase Commit [30]. More recently, work in this area focuses on the verification of more Blockchain-specific protocols. Kiayias [14], for example, proposes a verified consensus protocol based on proof-of-stake.

Relation to our work: The work discussed so far provides formalizations of various protocols, useful for the implementation of distributed trust. The pattern proposed and verified in this paper, however, uses a mechanism called "proof-of-work". Thus, approaches using proof-of-work are most closely related to our work and are discussed in more detail. The idea of applying proof-of-work to the problem of establishing distributed trust goes back to Nakamoto in its original bitcoin paper [26]. Here the author provides a mathematical description of the theory behind Blockchain technology and provides probabilistic bounds about certain security concerns. Garay [9,10] and Pass [28] elaborate on these ideas and identify and verify two properties of proof-of-work: *common prefix* and *chain quality*. The former is actually similar to Thm. 1 proved in this paper. While these works provide similar results to ours, there are two notable differences to our work: (i) First, the above approaches exclusively focus on probabilistic boundaries. While such boundaries are important in the area of Blockchain, we try to identify the preconditions which are required in order to establish these properties. (ii) Second, the above works were not mechanized, so far.

8 Conclusion

In this paper, we reported on the outcome of applying FACTUM to specify a variant of Blockchain architectures [26] and verify that blockchains are guaranteed to be persistent for architectures implementing the specification:

– The blockchain itself is modeled as a parametric list over blocks.
– Nodes represent the types of components. They either keep a blockchain and forward copies to other nodes or they may add at most one new block through mining. Thereby, we distinguish between two types of nodes: **Honest nodes** strictly follow the consensus rules and when faced with different copies of a blockchain, they always take the longest one (containing the most amount of work) as the "correct" one. **Dishonest nodes** on the other hand, do not necessarily follow the consensus rules and may also remove blocks from any blockchain they receive, in order to attempt to modify a certain entry.
– A Blockchain architecture is parameterized by a number of confirmation blocks, i.e., a value which determines the number of blocks which need to be mined on top of a block in order to consider this block to be save.

We also propose a formalization of a desired safety property: *persistence of blockchain entries*. Finally, we (mechanically) verified the property from the specification.

Throughout the paper, we describe 11 characteristic properties for Blockchain architectures and one fundamental assumption about relative mining frequencies, which guarantee persistence of blockchain entries. The properties can be used to support the verification of Blockchain architectures. To this end, an architecture specification is verified to satisfy the properties and in return, persistence of blockchain entries is guaranteed by Thm. 1. For the case that nodes are implemented by means of statemachines, this step could even be automated using model checking techniques. In addition, the paper presents a case study about the use of FACTUM for the verification of dynamic architectures. Thereby it reveals interesting insights to direct future research. On the positive side, it shows feasibility of verifying properties for dynamically evolving architectures, even if we need to reason about unbounded number of components. On the negative side, we discovered two main weaknesses: Since the approach is based on interactive theorem proving, the effort required to verify an architecture is still relatively high. For example, the verification of the property presented in this paper required a total effort of roughly three person months. Another weakness concerns the usability of the approach in practice since verification requires expertise in interactive theorem proving, which is not always available.

Based on the outcome of this study, we derive two directions for future work: (i) One direction should focus on extending the preliminary analysis of Blockchain architectures presented in this paper. To this end it should mainly address the limitations identified in Sect. 6: partial broadcasts, cryptographic aspects, explicit consideration of probabilities. (ii) Another direction should address to extend the FACTUM approach based on the lessons learned from this case study. In particular possibilities for proof automation and proof modeling should be investigated.

Acknowledgments. We would like to thank Manfred Broy, Alexander Knapp, Maximilian Junker, and Andreas Lochbihler for their comments and helpful suggestions on earlier versions of this paper. In addition, we are grateful to all the anonymous reviewers of FORTE 2019 for suggesting many improvements to the presentation. Parts of the work on which we report in this paper was funded by the German Federal Ministry of Education and Research (BMBF) under grant no. 01Is16043A.

References

1. Abdellatif, T., Brousmiche, K.: Formal verification of smart contracts based on users and blockchain behaviors models. In: 2018 9th IFIP International Conference on New Technologies, Mobility and Security (NTMS). pp. 1–5 (Feb 2018)
2. Amani, S., Bégel, M., Bortin, M., Staples, M.: Towards verifying ethereum smart contract bytecode in isabelle/hol. In: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. pp. 66–77. ACM (2018)
3. Azaria, A., Ekblaw, A., Vieira, T., Lippman, A.: Medrec: Using blockchain for medical data access and permission management. In: Open and Big Data (OBD), International Conference on. pp. 25–30. IEEE (2016)
4. Bentov, I., Gabizon, A., Mizrahi, A.: Cryptocurrencies without proof of work. In: Clark, J., Meiklejohn, S., Ryan, P.Y.A., Wallach, D.S., Brenner, M., Rohloff, K. (eds.) Financial Cryptography and Data Security. Lecture Notes in Computer Science, vol. 9604, pp. 142–157. Springer (2016)
5. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., et al.: Formal verification of smart contracts: Short paper. In: Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security. pp. 91–96. ACM (2016)
6. Broy, M.: A logical basis for component-oriented software and systems engineering. The Computer Journal 53(10), 1758–1782 (Feb 2010)
7. Chavez-Dreyfuss, G.: Sweden tests blockchain technology for land registry. http://web.archive.org/web/20161024065806/http://www.reuters.com/article/us-sweden-blockchain-idUSKCN0Z22KV
8. Drăgoi, C., Henzinger, T.A., Zufferey, D.: Psync: A partially synchronous language for fault-tolerant distributed algorithms. In: Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages. pp. 400–415. POPL '16, ACM, New York, NY, USA (2016), http://doi.acm.org/10.1145/2837614.2837650
9. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol: Analysis and applications. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 281–310. Springer (2015)
10. Garay, J., Kiayias, A., Leonardos, N.: The bitcoin backbone protocol with chains of variable difficulty. In: Annual International Cryptology Conference. pp. 291–323. Springer (2017)
11. Hirai, Y.: Defining the ethereum virtual machine for interactive theorem provers. In: Int. Conference on Financial Cryptography and Data Security. pp. 520–535. Springer (2017)
12. Hirai, Y.: Ethereum virtual machine for coq (v0. 0.2). Published online on 5 (2017)
13. Jaskelioff, M., Merz, S.: Proving the correctness of disk paxos. The Archive of Formal Proofs. http://afp. sf. net/entries/DiskPaxos. shtml (2005)
14. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: A provably secure proof-of-stake blockchain protocol. In: Annual International Cryptology Conference. pp. 357–388. Springer (2017)
15. Marmsoler, D., Gleirscher, M.: On activation, connection, and behavior in dynamic architectures. Scientific Annals of Computer Science 26(2), 187–248 (2016)
16. Marmsoler, D.: Towards a calculus for dynamic architectures. In: Hung, D.V., Kapur, D. (eds.) Theoretical Aspects of Computing - ICTAC 2017 - 14th International Colloquium, Hanoi, Vietnam, October 23-27, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10580, pp. 79–99. Springer (2017)
17. Marmsoler, D.: A framework for interactive verification of architectural design patterns in isabelle/hol. In: Sun, J., Sun, M. (eds.) 20th Int. Conf. on Formal Engineering Methods, 2018, Gold Coast, QLD, Australia, November 12-16, 2018, Proceedings. Lecture Notes in Computer Science, vol. 11232, pp. 251–269. Springer (2018)

18. Marmsoler, D.: Hierarchical specication and verication of architecture design patterns. In: Fundamental Approaches to Software Engineering - 21th Int. Conf., FASE 2018, Held as Part of the Euro. Joint Conf. on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings (2018)
19. Marmsoler, D.: A theory of architectural design patterns. Archive of Formal Proofs (Mar 2018), http://isa-afp.org/entries/Architectural_Design_Patterns.html, Formal proof development
20. Marmsoler, D.: Axiomatic Specification and Interactive Verification of Architectural Design Patterns in FACTum. Dissertation, Technische Universität München, München (2019)
21. Marmsoler, D., Gidey, H.K.: FACTUM Studio: A tool for the axiomatic specification and verification of architectural design patterns. In: Formal Aspects of Component Software - FACS 2018 - 15th International Conference, Proceedings (2018)
22. Marmsoler, D., Gleirscher, M.: Specifying properties of dynamic architectures using configuration traces. In: International Colloquium on Theoretical Aspects of Computing, pp. 235–254. Springer (2016)
23. Mavridou, A., Laszka, A.: Tool demonstration: Fsolidm for designing secure ethereum smart contracts. In: Bauer, L., Küsters, R. (eds.) Principles of Security and Trust. pp. 270–277. Springer International Publishing, Cham (2018)
24. Mendling, J., Weber, I., Aalst, W.V.D., Brocke, J.V., Cabanillas, C., Daniel, F., Debois, S., Ciccio, C.D., Dumas, M., Dustdar, S., et al.: Blockchains for business process management-challenges and opportunities. ACM Transactions on Management Information Systems (TMIS) 9(1), 4 (2018)
25. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) Advances in Cryptology — CRYPTO '87. pp. 369–378. Springer Berlin Heidelberg, Berlin, Heidelberg (1988)
26. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system (2008)
27. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: a proof assistant for higher-order logic, vol. 2283. Springer Science & Business Media (2002)
28. Pass, R., Seeman, L., Shelat, A.: Analysis of the blockchain protocol in asynchronous networks. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 643–673. Springer (2017)
29. Pîrlea, G., Sergey, I.: Mechanising blockchain consensus. In: Proceedings of the 7th ACM SIGPLAN Int. Conference on Certified Programs and Proofs. pp. 78–90. ACM (2018)
30. Sergey, I., Wilcox, J.R., Tatlock, Z.: Programming and proving with distributed protocols. Proceedings of the ACM on Programming Languages 2(POPL), 28 (2017)
31. The Bitcoin Community: The bitcoin wiki. http://web.archive.org/web/20181106124036/https://en.bitcoin.it/wiki/Confirmation
32. Wilcox, J.R., Woos, D., Panchekha, P., Tatlock, Z., Wang, X., Ernst, M.D., Anderson, T.: Verdi: A framework for formally verifying distributed system implementations. In: Proceedings of the 2015 ACM SIGPLAN Conf. on Prog. Language Design and Implementation (PLDI), Portland, OR (2015)
33. Wirsing, M.: Algebraic specification. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science (Vol. B), pp. 675–788. MIT Press, Cambridge, MA, USA (1990), http://dl.acm.org/citation.cfm?id=114891.114904
34. Woos, D., Wilcox, J.R., Anton, S., Tatlock, Z., Ernst, M.D., Anderson, T.: Planning for change in a formal verification of the raft consensus protocol. In: Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs. pp. 154–165. ACM (2016)
35. Yurcan, B.: How blockchain fits into the future of digital identity. http://web.archive.org/web/20170119054131/https://www.americanbanker.com/news/how-blockchain-fits-into-the-future-of-digital-identity
36. Zohar, A.: Bitcoin: Under the hood. Commun. ACM 58(9), 104–113 (Aug 2015)