Exploring Formal Math on the Blockchain: An Explorer for Proofgold

Chad E. Brown[1], Cezary Kaliszyk[2][0000−0002−8273−6059], and Josef Urban[1][0000−0002−1384−1613]

[1] Czech Technical University in Prague, Czech Republic
josef.urban@gmail.com
[2] University of Melbourne, Australia and University of Innsbruck, Austria
ckaliszyk@unimelb.edu.au

Abstract. Proofgold is a blockchain that supports formalized mathematics alongside standard cryptocurrency functionality. It incorporates logical constructs into the blockchain, including declarations of formal theories, definitions, propositions and proofs. It also supports placing and collecting bounties on proving these propositions, incentivizing the development of the formal libraries contained in Proofgold. In this paper, we present a web-based blockchain explorer for Proofgold. The system exposes not only the usual transactional data but also the formal mathematical components embedded in the chain and allows some interaction with them. The explorer allows users to inspect blocks, transactions, and addresses, as well as formal objects: theories, definitions, theorems and their proofs. We also support the submission of transactions to the blockchain using our interface. We describe the system architecture and its integration with the Proofgold Lava software, highlighting how the explorer supports navigation of formal content and facilitates mathematical knowledge management in a decentralized setting, as well as a number of formalizations in category theory done in the system.

1 Introduction

Formalized mathematics has seen remarkable progress in recent years, with large-scale developments such as the formal proof of the Feit-Thompson theorem in Coq [16] and the proof of the Kepler conjecture in HOL Light and Isabelle [17], the formalization of perfectoid spaces in Lean [10], and the formalization of superposition calculus in Isabelle's Archive of Formal Proofs [13]. Despite these advances, most formalization efforts are still coordinated through centralized repositories, and the ecosystem lacks well-established mechanisms for incentivizing contributions. Blockchain technology offers a promising avenue to address both of these challenges by providing a decentralized infrastructure for storing and verifying formal content, as well as built-in mechanisms for transparent incentives and attribution. By combining blockchains with formalized mathematics, we can create systems in which formal theories, definitions, propositions and their associated metadata are permanently recorded in a ledger. This enables provenance tracking, ensures tamper resistance and facilitates collaborative contributions across a global network without reliance on a central authority.

Proofgold [8] builds upon this idea of integrating formal mathematics with a decentralized system by providing a blockchain that supports the storage, verification and incentivization of formal math contributions. By including formal theories, definitions, theorems and proofs directly into the blockchain, Proofgold ensures that this mathematical knowledge is publicly accessible and verifiable. By integrating a bounty system, where users can place rewards for the verification (or refutation) of specific conjectures, it also encourages development. Despite these advantages, Proofgold is currently a command-line interface-only system, lacking a user interface, which limits its accessibility to experts who are familiar with the CLI and reduces its appeal to a broader audience.

Contributions In this paper, we present the Web-based blockchain explorer for Proofgold, built by extending the Proofgold Lava client [8]. This extension adds various functionalities for querying the blockchain, and includes the implementation of server-side code to facilitate interactions with the client. In particular:

- We describe the system architecture, outlining how the web-based explorer interfaces with the Proofgold blockchain (Section 3);
- We demonstrate how formal mathematical objects – such as theorems, definitions and proofs – are rendered within the explorer, and explain how users can interact with these objects (Section 4);
- As a case study, we present a number of conjectures in category theory along with their proofs or refutations presented in Proofgold (Section 5).

2 Proofgold and Formal Mathematics on the Blockchain

This section briefly introduces Proofgold; for a more complete description, see [8]. Proofgold is a cryptocurrency designed to support formal logic and mathematics. The core of Proofgold is a proof checker for intuitionistic higher-order logic with functional extensionality. We only give an introduction to the higher-order Tarski Grothendieck set theory (HOTG) defined in Proofgold in Section 5. Users can publish theories, which consist of primitive constants, their types, and axioms. Each theory is uniquely identified by a 256-bit identifier derived from its recursive hash (Merkle root). Documents defining new objects, proving theorems, and stating conjectures can be published within a theory. Ownership of propositions is determined by public keys, enabling the redemption of bounties by proving conjectures. Proofgold combines proof-of-stake and proof-of-burn, with the proof-of-burn element involving burning small amounts of Litecoin. This combination enhances security by reusing Litecoin's proof-of-work. The first 5000 Proofgold blocks automatically placed bounties on pseudorandom propositions [8], allowing new participants to increase their stake by proving theorems.

Proofgold has seen significant activity in terms of theories, documents, and formalizations. The platform includes a built-in theory of hereditarily finite sets (HF), which was used to generate pseudorandom bounties for the first 5000 blocks. Additionally, two theories axiomatizing HOTG have been published [9], one corresponding to Mizar [4] and one corresponding to Megalodon, along with a theory for reasoning about syntax using higher-order abstract syntax (HOAS).

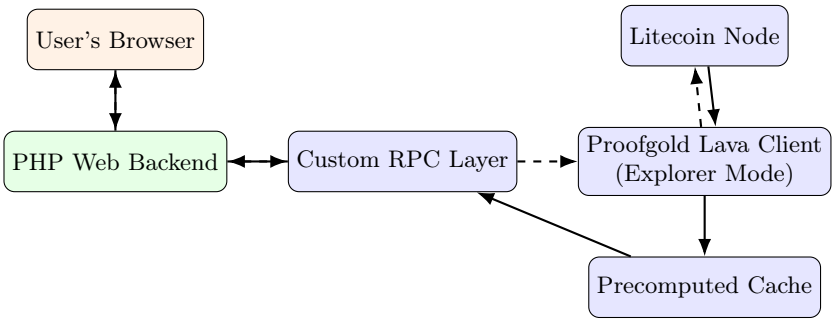


Fig. 1. High-level architecture of the explorer

These theories have facilitated the formalization of various mathematical concepts and the construction of significant mathematical objects, such as the real numbers via Conway's surreal numbers. In particular, the Megalodon proofs of 12 of Wiedijk's 100 theorems [24] are included in Proofgold. Furthermore, the platform has enabled the publication of conjectures and the collection of bounties, with 2836 of the 13142 bounties resolved as of April 2025.

The Proofgold Lava Client, co-developed by the authors, addresses the scalability issues of the original Proofgold Core software. It features an improved database layer using the Unix DBM interface, a more efficient cryptography layer, and enhancements to the networking and proof-checking layers. The client is a command line interface that unfortunately requires somewhat complicated installation (it relies on a litecoin running in RPC mode, and particular versions of the database). Additionally, it supports 210 commands, which may be somewhat overwhelming for users.

3 System Architecture

The design of the Proofgold Explorer focuses on providing accessible and interactive access to both blockchain data and formal mathematical content. In particular, we aim to offer: an intuitive web interface that lowers the entry barrier; structured access to formal and transactional data; and integration with the existing Proofgold infrastructure. To achieve these goals, we propose the architecture consisting of three main components (Fig. 1):

1. A modified Proofgold Lava client, running in a new `explorer` mode that would cache and prepare additional information;
2. A server-side backend written in PHP;
3. An RPC communication layer connecting the backend with both the explorer node and the underlying Litecoin node.

In explorer mode, the Lava client computes and caches additional information about the formal content embedded in the blockchain. This includes the possibility of looking up definitions, propositions, theories, proofs and bounties. In principle, this information is already stored in the blockchain, but it

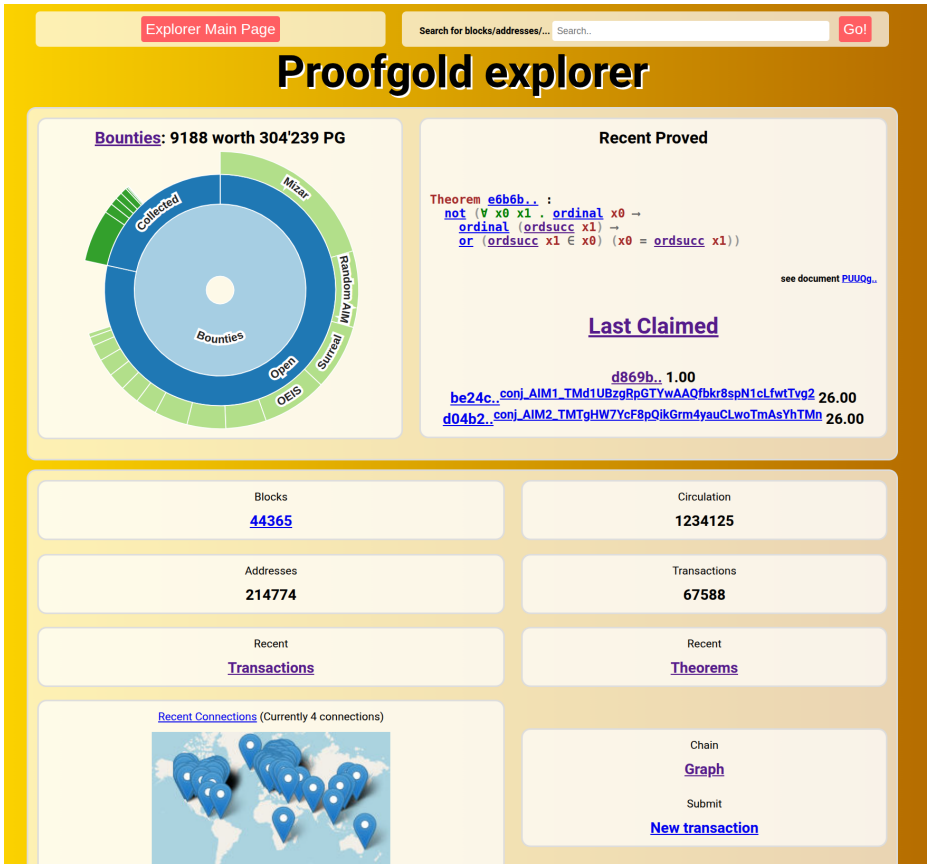


Fig. 2. Explorer Main Page

requires traversing the whole history. The added information is stored in 34 OCaml hashtables that add approximately 10 Gigabytes in memory and that are periodically (every hour) refreshed. These structures allow efficient access to: information about individual formal entities, including their types, statements, and owners; the history and status of bounties and their collection; dependencies between formal objects and their evolution over time; ownership and authorship tracking for mathematical content. It serves this structured information through a custom RPC interface that returns it in a machine-readable format for use by the frontend.

The web backend is implemented in PHP and is responsible for serving user-facing pages that visualize both standard blockchain data (blocks, transactions, addresses, bounties) and formal mathematical constructs (theorems, definitions, proofs, theories, conjectures). The backend queries the explorer node in real time and formats the data into interactive HTML views. It also enables lightweight interaction with the blockchain, such as submitting transactions. These compo-

nents are designed to be kept in sync with the live blockchain state and support both incremental updates and history queries.

The explorer is also connected to the **Megalodon Wiki (mgwiki)**[3] which is a collaborative git-based platform for formal math that enables users to edit and verify Megalodon files directly in the browser. The mgwiki workflow involves cloning or forking the repository, modifying or adding `.mg` files, and committing changes, which triggers automated proof checking and HTML generation via GitHub Actions. Errors, if any, are reported in the GitHub action logs, and successful edits are published as browsable HTML. A unique feature of mgwiki is its integration with Proofgold: when a valid Megalodon file is added, it is automatically converted into a `.pfg` document that can be submitted to Proofgold. This allows users to associate formalized conjectures with bounties and track their progress through the explorer. Mgwiki thus serves also as a gateway for contributing conjectures and proofs to the Proofgold-based decentralized proof bounty system.

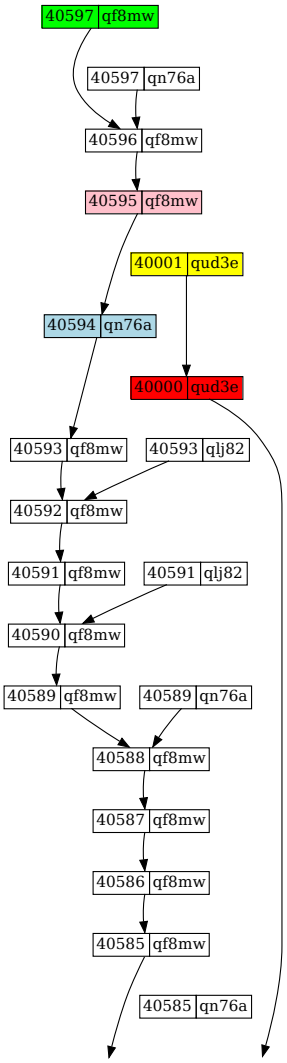


Fig. 3. Chain Graph

4 Explorer Functionality

The Proofgold Explorer provides a structured and interactive interface to both blockchain and formal mathematical data. The main dashboard (Figure 2), available online[4] aggregates statistics about the blockchain such as block height, address count, transaction volume and coin circulation. An important feature is a link to the overview of the current graph of the blockchain. This is normally a single chain with branches in the case of competing chain tips. We show an example from a time when there were multiple competing nodes (Figure 3). Particular nodes in the chain are marked in special colors: nodes that define theories, include proof objects, place bounties on

[3] `https://github.com/mgwiki/mgw_test`, `https://mgwiki.github.io/mgw_test/`
[4] `https://formalweb3.uibk.ac.at/pgbce/`, `http://proofgold.net/explorer/`

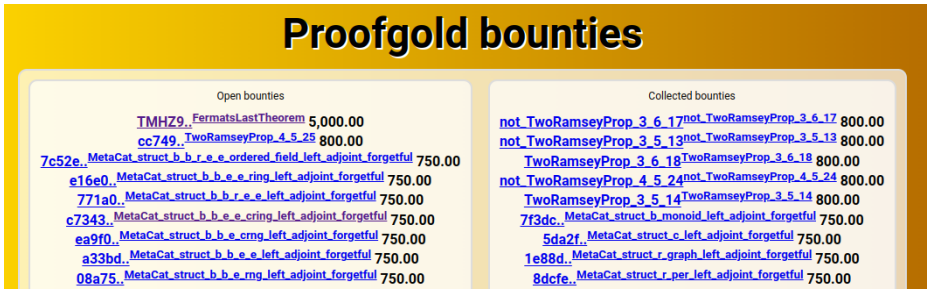


Fig. 4. Highest open and collected bounties

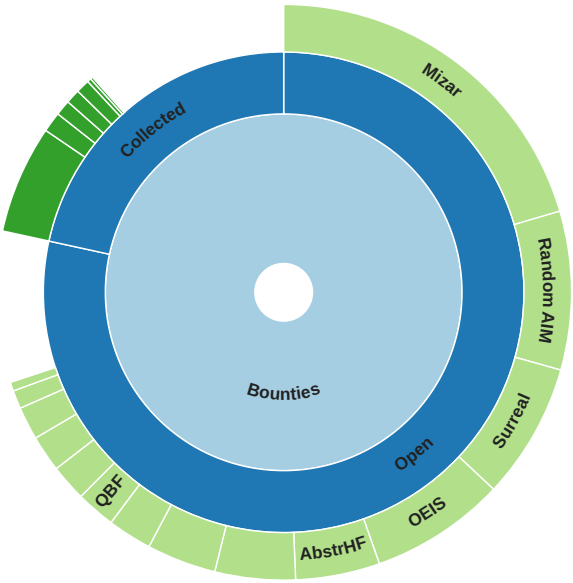
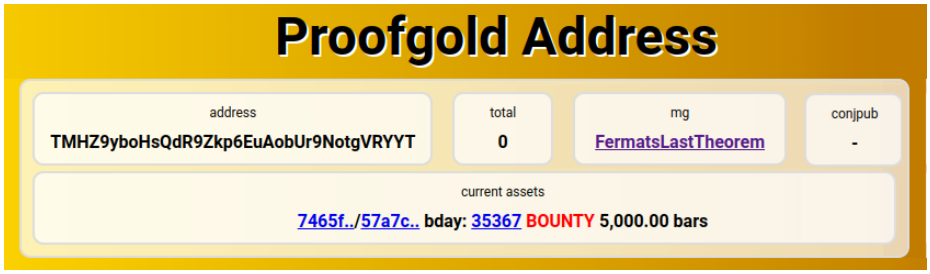


Fig. 5. Open and collected bounty categories

propositions, or just include transactions are colored in green, blue, and pink. Additionally, as the blockchain attempts to be resilient to attacks, the graph also marks missing nodes and invalid nodes (spending non-existing assets, but also invalid proof steps) in yellow and red respectively.

The explorer also visualizes the open and collected bounties, with a more detailed view of the highest open and collected bounties, as well as a categorization of bounties presented in Figures 4 and 5. This allows users to inspect the most valuable open conjectures and focus their proof efforts on them, as well as to see what are the domains of the conejctures users are working on. The individual bounties can be viewed either in the Proofgold format, if their representation is given in a theory document, or in case if they are only given in the opaque formal, they are linked to the Megalodon Wiki. We show this in the example



Theorem. (FermatsLastTheorem)
∀n ∈ int, 2 < n → ∀x y z ∈ int, x ^ n + y ^ n = z ^ n → x = 0 ∨ y = 0 ∨ z = 0
In Proofgold the corresponding term root is a3012f... and proposition id is 6637e9...
Proof:
The rest of the proof is missing.

Fig. 6. Fermat as a bounty in the Proofgold explorer and its corresponding statement in the Megalodon Wiki

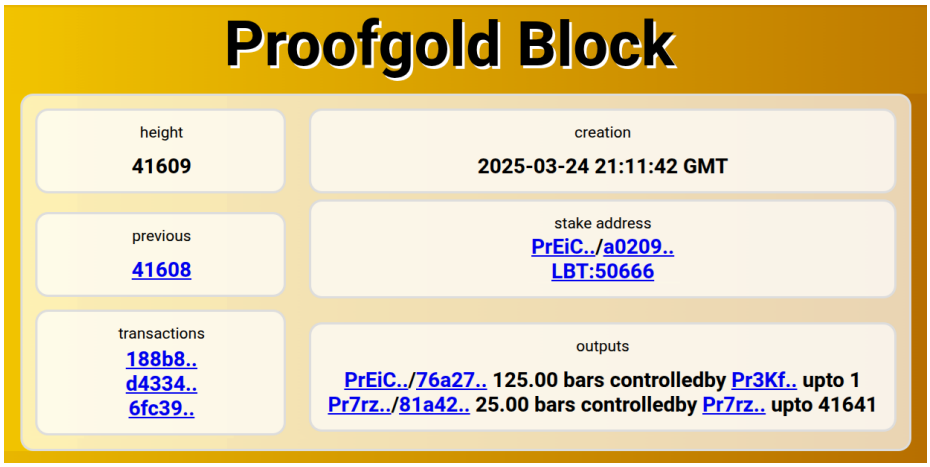


Fig. 7. Explorer view of a block

of Fermat's last theorem, as of April 2025 it is the conjecture with the highest bounty in Proofgold (Figure 6).

The explorer allows viewing blocks (Figure 7) and transactions in them. This is useful, because unlike in other blockchains, Proofgold transactions are not always purely value-based. They can also introduce proof documents. We show an example of such a transaction in Figure 8: The transaction takes a small amount of proofgold along with a marker (used to safely claim ownership of proved objects [8]) and sends them to the address of a proof document. The newly defined objects, shown in the figure, and theorems proved in this document are now owned by the publisher of the document.

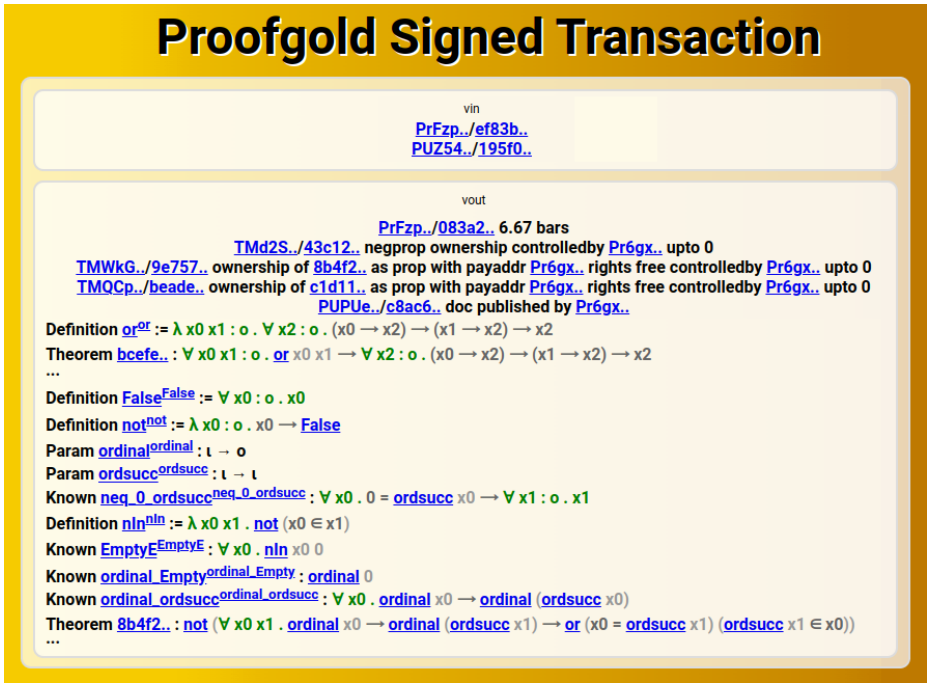


Fig. 8. Explorer view of a transaction. Clicking "..." allows inspecting the proof.

The explorer also allows viewing the list of all theories and viewing individual ones. We show the theory defining document for the Tarski Groethendieck theory in Fig. 9. The axioms directly corresponding to the foundations of Megalodon as known from [9].

4.1 Transaction Submission Interface

Blockchain explorers typically include transaction submission interfaces to allow users to directly broadcast transactions to the network without needing a full node or wallet software. This is useful for users who have already constructed and signed a transaction (usually offline) and simply need to submit it to the blockchain for confirmation. We also include such an interface in the explorer.

5 Use Case and Examples

Some of the examples of conjectures with bounties – many of which have already been proven or disproven – assert the existence of a left adjoint to a forgetful functor. Such left adjoints, when they exist, often correspond to freely generated structures. More generally, the interest in such adjoints can be justified by Slogan IV of [19]: "Many important concepts in mathematics arise as adjoints, right or left, to previously known functors." We begin with an informal description,

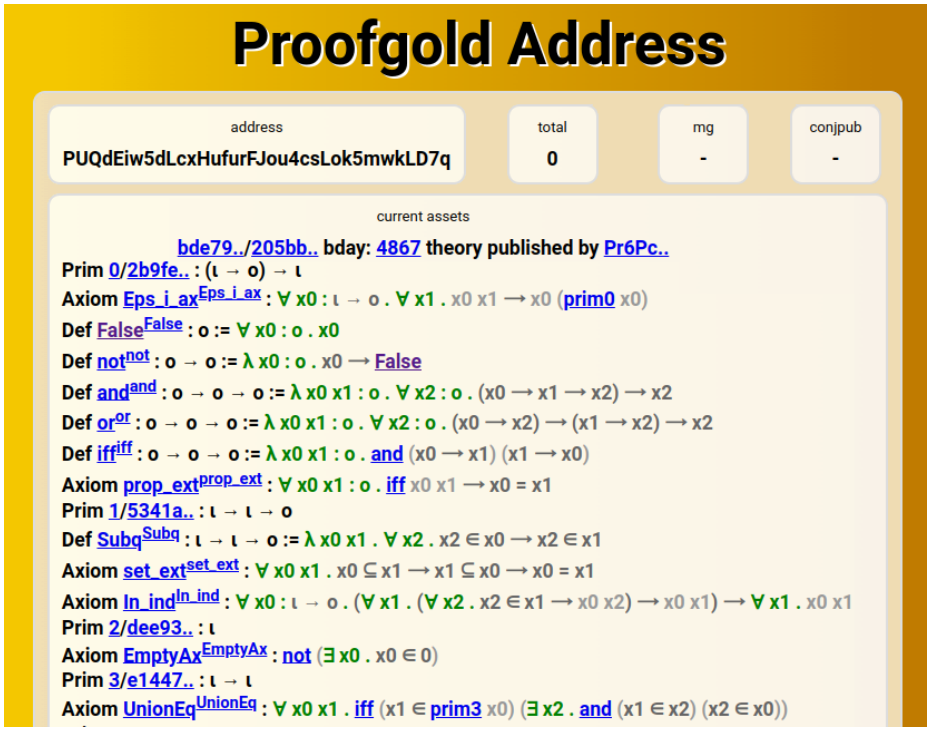


Fig. 9. The higher-order Tarski Groethendieck theory in the explorer

leaving details to [19]. Suppose $\mathcal{C}$ and $\mathcal{D}$ are categories and $\mathcal{F} : \mathcal{C} \to \mathcal{D}$ and $\mathcal{U} : \mathcal{D} \to \mathcal{C}$ are functors. $\mathcal{F}$ is left adjoint to $\mathcal{U}$ if there exist natural transformations $\eta : 1_{\mathcal{C}} \to \mathcal{U}\mathcal{F}$ and $\varepsilon : \mathcal{F}\mathcal{U} \to 1_{\mathcal{D}}$ satisfying certain identities. In each conjecture below, $\mathcal{C}$ will be the category of sets, $\mathcal{D}$ will be some category of structures and $\mathcal{U}$ will be the forgetful functor sending a structure to its carrier set. The conjecture will then state that there exist $\mathcal{F}$, η and ε giving an adjunction. In July 2021, bounties of 750 bars were placed on 33 conjectures of this form. As of May 2025, 14 of the bounties have been collected (11 by proving the conjecture and 3 by disproving the conjecture), and 19 remain open. We discuss a few of these propositions below.

The conjectures and theorems are in the HOTG theory [9], with a base type ι of sets. We use o for the type of propositions and $\alpha\beta$ for the type of functions from α to β. We briefly review elements of set theory required to describe the conjectures. There are the usual logical definitions: $\top : o$, $\bot : o$, $\neg : oo$, $\wedge : ooo$, $\equiv : ooo$, $=$ and $\exists$. Two primitives of the set theory are relevant: $\in : \iota\iota o$ (which we write in infix) and $\emptyset : \iota$ (which we often write as 0 below). After enough infrastructure is defined, we also have the following objects:

– $\mathsf{lam} : \iota(\iota\iota)\iota$: Here $\mathsf{lam}\ X\ f$ is the set encoding the function f restricted to the domain X.

- $\mathsf{ap} : \iota\iota\iota$: $\mathsf{ap}\ f\ x$ corresponds to applying the function (encoded by the set) f to x. Here we often simply write $f\ x$, leaving ap as implicit. (Note that since f and x have type ι, $f\ x$ would be ill-typed if we did not insert ap.)
- $\mathsf{Pi} : \iota(\iota\iota)\iota$: Here $\Pi\ X\ Y$ is the set of functions $f : X \to \bigcup_{x \in X}(Y\ x)$ such that $f\ x \in Y\ x$ for each $x \in X$. We write Y^X for the term $\Pi\ X\ (\lambda x.Y)$. Note that Y^X is simply the set of functions from X to Y.

For the categories of interest, the following definitions are important.

- $\mathsf{lam\_id} : \iota\iota$ where $\mathsf{lam\_id}\ X$ is $\mathsf{lam}\ X\ (\lambda x.x)$. That is, $\mathsf{lam\_id}$ is the set-theoretic encoding of the identity function on X.
- $\mathsf{lam\_comp} : \iota\iota\iota\iota$ where $\mathsf{lam\_comp}\ X\ g\ f$ is $\mathsf{lam}\ X\ (\lambda x.g(fx))$. Assuming f is a function with domain X and g is appropriate, this is the set-theoretic encoding of the composition of f and g.
- $\mathsf{HomSet} : \iota\iota\iota o$ where $\mathsf{HomSet}\ X\ Y\ f$ is $f \in Y^X$.

Some previously proven results can be assumed here, e.g., $\forall X.\mathsf{lam\_id}\ X \in X^X$ and $\forall XYf.f \in Y^X \to \mathsf{lam\_comp}\ X\ f\ (\mathsf{lam\_id}\ X) = f$.

We will only consider structures with a single carrier set and assume the set theoretic representation of each structure is as a function A where $A\ 0$ (i.e., A applied to 0) yields the carrier. To account for this, the identities and composition for categories of structures are defined by slight modifications of $\mathsf{lam\_id}$ and $\mathsf{lam\_comp}$.

- $\mathsf{struct\_id} : \iota\iota$ where $\mathsf{struct\_id}\ A$ is $\mathsf{lam\_id}\ (A\ 0)$.
- $\mathsf{struct\_comp} : \iota\iota\iota\iota\iota$ where $\mathsf{struct\_comp}\ A\ B\ C$ is $\mathsf{lam\_comp}\ (A\ 0)$. Note that this ignores its second and third arguments. Giving the last two arguments explicitly yields $\mathsf{struct\_comp}\ A\ B\ C\ g\ f$ is $\mathsf{lam\_comp}\ (A\ 0)\ g\ f$.

There are many examples of structures defined in Megalodon and Proofgold which differ in two ways: the other components of the structure (a binary operation, a binary relation, etc.) and what properties are assumed of these other components. To give a simple, concrete example, we consider structures with a single binary relation. Avoiding details (which are not relevant here), we note that there is a previously defined object $\mathsf{pack\_r} : \iota(\iota\iota o)\iota$ such that $\mathsf{pack\_r}\ X\ R$ encodes the carrier set X and the binary relation R (restricted to its behavior on X) as a set. We then define $\mathsf{struct\_r} : \iota o$ to be the class of all such sets. We also assume the previously proven $\forall X.\forall R : \iota o.X = \mathsf{pack\_r}\ X\ R\ 0$. That is, the set $\mathsf{pack\_r}\ X\ R$ is a function that yields the carrier set X when applied to 0.

A particular category of such structures will result from adding some restriction of the class $\mathsf{struct\_r}$ as objects. Regardless of the restriction, the arrows should be all functions sending related inputs to related outputs. This is given by $\mathsf{BinRelnHom} : \iota\iota o$ and the previously proven identity

$$\begin{aligned}&\mathsf{BinRelnHom}\ (\mathsf{pack\_r}\ X\ R)\ (\mathsf{pack\_r}\ Y\ Q)\ h\\ &= (h \in Y^X \wedge \forall xy \in X.R\ x\ y \to Q\ (h\ x)\ (h\ y)).\end{aligned}$$

To have a specific category as an example, we consider $\mathsf{IrrPartOrd} : \iota o$ which is the class of all structures with a single binary irreflexive transitive relation (i.e.,

an irreflexive partial order). The details of the definition are not important. It is enough to know we have the following previously proven results:

- If R is irreflexive and transitive on X, then $\mathsf{IrrPartOrd}$ ($\mathsf{pack\_r}$ X R).
- Let A satisfy $\mathsf{IrrPartOrd}$ A and $q : \iota o$ be given. In order to prove q A it is enough to prove q ($\mathsf{pack\_r}$ X R) for all sets X and irreflexive transitive relations R on X.

Let us now turn to the relevant formalization of category theory. In practice, we will be interested in large categories (metalevel categories), so formally $\mathcal{C}$ will not be encoded as a set but will consist of four components (given as four explicit dependencies in the formalization):

- $\mathcal{C}_0 : \iota o$ – the class of all objects of $\mathcal{C}$.
- $\mathcal{C}_1 : \iota\iota o$ – where $\mathcal{C}_1$ X Y is the class of arrows from X to Y in $\mathcal{C}$.
- $\mathsf{id}_{\mathcal{C}} : \iota\iota$ – where $\mathsf{id}_{\mathcal{C}}$ X is the identity arrow for X in $\mathcal{C}$.
- $\mathsf{comp}_{\mathcal{C}} : \iota\iota\iota\iota$ – where $\mathsf{comp}_{\mathcal{C}}$ X Y Z g f is the composition of f (an arrow from X to Y) and g (an arrow from Y to Z).

The second category $\mathcal{D}$ will also be represented by four explicit components: $\mathcal{D}_0$, $\mathcal{D}_1$, $\mathsf{id}_{\mathcal{D}}$ and $\mathsf{comp}_{\mathcal{D}}$. Similarly, the functor $\mathcal{F}$ is represented by two explicit components:

- $\mathcal{F}_0 : \iota\iota$ – where $\mathcal{F}_0$ X is the object of $\mathcal{D}$ to which the object X of $\mathcal{C}$ maps.
- $\mathcal{F}_1 : \iota\iota\iota$ – where $\mathcal{F}_1$ X Y f is the arrow from $\mathcal{F}_0$ X to $\mathcal{F}_1$ Y in $\mathcal{D}$ corresponding to the arrow f from X to Y in $\mathcal{C}$.

Of course, the functor $\mathcal{U}$ is also represented by $\mathcal{U}_0$ and $\mathcal{U}_1$. A natural transformation η is simply of type $\iota\iota$ – mapping an object in one category to an appropriate arrow in the other category.

$\mathsf{MetaCat} : (\iota o)(\iota\iota o)(\iota\iota)(\iota\iota\iota\iota)o$ is defined so that $\mathsf{MetaCat}$ $\mathcal{C}_0$ $\mathcal{C}_1$ $\mathsf{id}_{\mathcal{C}}$ $\mathsf{comp}_{\mathcal{C}}$ holds if the components form a category. Here, we only need to know that two specific (alleged) categories are categories. The category of sets is given by the constant true predicate (every set is an object), HomSet, $\mathsf{Iam\_id}$ and $(\lambda XYZ.\mathsf{Iam\_comp}\ X)$ (where the ignored arguments are needed for the types to match). This has been previously proven to be a category, and the precise proposition is $\mathsf{MetaCat}$ $(\lambda X.\top)$ HomSet $\mathsf{Iam\_id}$ $(\lambda XYZ.\mathsf{Iam\_comp}\ X)$. Likewise, the category of all irreflexive transitive relations has been proven to be a category and the proposition is $\mathsf{MetaCat}$ $\mathsf{IrrPartOrd}$ $\mathsf{BinRelnHom}$ $\mathsf{struct\_id}$ $\mathsf{struct\_comp}$.

We know $\mathcal{F}_0$ and $\mathcal{F}_1$ give a functor between two categories if a number of basic properties hold. Without going into details, there is a previous definition $\mathsf{MetaFunctor}$ such that $\mathsf{MetaFunctor}$ $\mathcal{C}_0$ $\mathcal{C}_1$ $\mathsf{id}_{\mathcal{C}}$ $\mathsf{comp}_{\mathcal{C}}$ $\mathcal{D}_0$ $\mathcal{D}_1$ $\mathsf{id}_{\mathcal{D}}$ $\mathsf{comp}_{\mathcal{D}}$ $\mathcal{F}_0$ $\mathcal{F}_1$ holds precisely if those properties hold. A previously proven result allows us to infer $\mathsf{MetaFunctor}$ $\mathcal{C}_0$ $\mathcal{C}_1$ $\mathsf{id}_{\mathcal{C}}$ $\mathsf{comp}_{\mathcal{C}}$ $\mathcal{D}_0$ $\mathcal{D}_1$ $\mathsf{id}_{\mathcal{D}}$ $\mathsf{comp}_{\mathcal{D}}$ $\mathcal{F}_0$ $\mathcal{F}_1$ by proving those properties. There is also an object $\mathsf{MetaFunctor\_strict}$ which further requires the two (alleged) categories to actually be categories.

We already know the forgetful functor from the category of irreflexive transitive relations to the category of sets is a functor. The forgetful functor (in

general) sends a structure A to its carrier set $A\ 0$ and sends structure morphisms f to f (which is already an appropriate set-theoretic function). Hence, the previously proven theorem is

$$\mathsf{MetaFunctor}\ \mathsf{IrrPartOrd}\ \mathsf{BinRelnHom}\ \mathsf{struct\_id}\ \mathsf{struct\_comp}\ (\lambda X.\top)\ \mathsf{HomSet}\ \mathsf{lam\_id}\ (\lambda XYZ.\mathsf{lam\_comp}\ X)\ (\lambda A.A\ 0)\ (\lambda ABf.f).$$

Similar to the discussion above, we have $\mathsf{MetaNatTrans}$ where

$$\mathsf{MetaNatTrans}\ \mathcal{C}_0\ \mathcal{C}_1\ \mathsf{id}_{\mathcal{C}}\ \mathsf{comp}_{\mathcal{C}}\ \mathcal{D}_0\ \mathcal{D}_1\ \mathsf{id}_{\mathcal{D}}\ \mathsf{comp}_{\mathcal{D}}\ \mathcal{F}_0\ \mathcal{F}_1\ \mathcal{G}_0\ \mathcal{G}_1\ \eta$$

holds if η is a natural transformation from $\mathcal{F}$ to $\mathcal{G}$ (assuming $\mathcal{C}$, $\mathcal{D}$, $\mathcal{F}$ and $\mathcal{G}$ are appropriate inputs). Finally we have $\mathsf{MetaAdjunction}$ where

$$\mathsf{MetaAdjunction}\ \mathcal{C}_0\ \mathcal{C}_1\ \mathsf{id}_{\mathcal{C}}\ \mathsf{comp}_{\mathcal{C}}\ \mathcal{D}_0\ \mathcal{D}_1\ \mathsf{id}_{\mathcal{D}}\ \mathsf{comp}_{\mathcal{D}}\ \mathcal{F}_0\ \mathcal{F}_1\ \mathcal{U}_0\ \mathcal{U}_1\ \eta\ \varepsilon$$

is an adjunction (with witnessing natural transformations η and ε) – again, assuming the inputs are appropriate. We also have $\mathsf{MetaAdjunction\_strict}$ which further ensures the inputs are appropriate by requiring:

- $\mathsf{MetaFunctor\_strict}\ \mathcal{C}_0\ \mathcal{C}_1\ \mathsf{id}_{\mathcal{C}}\ \mathsf{comp}_{\mathcal{C}}\ \mathcal{D}_0\ \mathcal{D}_1\ \mathsf{id}_{\mathcal{D}}\ \mathsf{comp}_{\mathcal{D}}\ \mathcal{F}_0\ \mathcal{F}_1$ – ensuring $\mathcal{C}$ and $\mathcal{D}$ are categories and $\mathcal{F}$ is a functor from $\mathcal{C}$ to $\mathcal{D}$.
- $\mathsf{MetaFunctor}\ \mathcal{D}_0\ \mathcal{D}_1\ \mathsf{id}_{\mathcal{D}}\ \mathsf{comp}_{\mathcal{D}}\ \mathcal{C}_0\ \mathcal{C}_1\ \mathsf{id}_{\mathcal{C}}\ \mathsf{comp}_{\mathcal{C}}\ \mathcal{U}_0\ \mathcal{U}_1$ – ensuring $\mathcal{U}$ is a functor from $\mathcal{D}$ to $\mathcal{C}$.
- $\mathsf{MetaNatTrans}\ \cdots\ \eta$ – ensuring η is a natural transformation from $1_{\mathcal{C}}$ to $\mathcal{U}\mathcal{F}$.
- $\mathsf{MetaNatTrans}\ \cdots\ \varepsilon$ – ensuring ε is a natural transformation from $\mathcal{F}\mathcal{U}$ to $1_{\mathcal{D}}$.

Now, for each category $\mathcal{D}$ of structures, we can form the following conjecture that a left adjoint to the forgetful functor exists:

$$\exists \mathcal{F}_0 : \iota\iota.\exists \mathcal{F}_1 : \iota\iota\iota.\exists \eta\varepsilon : \iota\iota.\ \mathsf{MetaAdjunction\_strict}\ (\lambda X.\top)\ \mathsf{HomSet}\ \mathsf{lam\_id}\ (\lambda XYZ.\mathsf{lam\_comp}\ X)\ \mathcal{D}_0\ \mathcal{D}_1\ \mathsf{struct\_id}\ \mathsf{struct\_comp}\ \mathcal{F}_0\ \mathcal{F}_1\ (\lambda A.A\ 0)\ (\lambda ABf.f)\ \eta\ \varepsilon.$$

As mentioned above, bounties were placed on 33 propositions of this form, of which 19 remain unresolved (neither proven nor disproven). Table 1 lists the 33 propositions and indications whether the proposition has been proven, disproven or is still open. In the particular case of irreflexive transitive relations, $\mathcal{D}_0$ is $\mathsf{IrrPartOrd}$ and $\mathcal{D}_1$ is $\mathsf{BinRelnHom}$. The proof of this special case is given by taking $\mathcal{F}_0$ to be $\lambda X.\mathsf{pack\_r}\ X\ (\lambda xy.\bot)$ – that is, a set X is taken to the structure given by X with the empty relation. Furthermore $\mathcal{F}_1$ is given so that $\mathcal{F}_1\ X\ Y\ f$ is f, η is given so that $\eta\ X$ is $\mathsf{lam\_id}\ X$ and ε is given so that $\varepsilon\ A$ is $\mathsf{lam\_id}\ (A\ 0)$. The remainder of the proof involves checking the relevant properties. This proof was published on the Proofgold blockchain in September 2021.

The most recent of the 33 propositions to be proven has $\mathcal{D}$ as the category of structures with a carrier X and a bijective function $f : X \to X$.[5] A proof of the

[5] The proposition can be viewed on the explorer at the link `https://formalweb3.uibk.ac.at/pgbce/OP.php?b=a69df3cc99230330e94428aa4d4e3bf5ce0405944ff3242f3882144c1c0c5216`.

Name	Status
MetaCat_struct_p_left_adjoint_forgetful	Proven
MetaCat_struct_r_left_adjoint_forgetful	Proven
MetaCat_struct_r_graph_left_adjoint_forgetful	Proven
MetaCat_struct_r_partialord_left_adjoint_forgetful	Proven
MetaCat_struct_r_ord_left_adjoint_forgetful	Disproven
MetaCat_struct_r_wellord_left_adjoint_forgetful	Disproven
MetaCat_struct_r_per_left_adjoint_forgetful	Proven
MetaCat_struct_r_equivreln_left_adjoint_forgetful	Proven
MetaCat_struct_c_left_adjoint_forgetful	Proven
MetaCat_struct_c_topology_left_adjoint_forgetful	Open
MetaCat_struct_c_T1_topology_left_adjoint_forgetful	Open
MetaCat_struct_c_Hausdorff_topology_left_adjoint_forgetful	Open
MetaCat_struct_u_left_adjoint_forgetful	Proven
MetaCat_struct_u_inj_left_adjoint_forgetful	Proven
MetaCat_struct_u_bij_left_adjoint_forgetful	Proven
MetaCat_struct_u_idem_left_adjoint_forgetful	Proven
MetaCat_struct_b_left_adjoint_forgetful	Open
MetaCat_struct_b_quasigroup_left_adjoint_forgetful	Open
MetaCat_struct_b_loop_left_adjoint_forgetful	Open
MetaCat_struct_b_semigroup_left_adjoint_forgetful	Open
MetaCat_struct_b_monoid_left_adjoint_forgetful	Disproven
MetaCat_struct_b_group_left_adjoint_forgetful	Open
MetaCat_struct_b_abelian_group_left_adjoint_forgetful	Open
MetaCat_struct_b_b_e_left_adjoint_forgetful	Open
MetaCat_struct_b_b_e_rng_left_adjoint_forgetful	Open
MetaCat_struct_b_b_e_crng_left_adjoint_forgetful	Open
MetaCat_struct_b_b_e_e_left_adjoint_forgetful	Open
MetaCat_struct_b_b_e_e_semiring_left_adjoint_forgetful	Open
MetaCat_struct_b_b_e_e_ring_left_adjoint_forgetful	Open
MetaCat_struct_b_b_e_e_cring_left_adjoint_forgetful	Open
MetaCat_struct_b_b_e_e_field_left_adjoint_forgetful	Open
MetaCat_struct_b_b_r_e_e_left_adjoint_forgetful	Open
MetaCat_struct_b_b_r_e_e_ordered_field_left_adjoint_forgetful	Open

Table 1. Propositions asserting existence of adjunctions for forgetful functors

```
Definition. We define MetaCat_struct_u_bij_free0 to be λX ⇒ pack_u (int × X) (λu ⇒ (u 0
+ 1,u 1)) of type set → set.
In Proofgold the corresponding term root is fca8b5... and object id is 24e110...

Definition. We define MetaCat_struct_u_bij_free1 to be λX Y h ⇒ λu ∈ int × X ⇒ (u 0,h (u
1)) of type set → set → set → set.
In Proofgold the corresponding term root is b4463b... and object id is f471c2...

Definition. We define MetaCat_struct_u_bij_forgetfree_eps to be λA ⇒ unpack_u_i A (λX h
⇒ λu ∈ int × X ⇒ if_u 0 ≤ 0 then omega_iterate (- (u 0)) (inv X h) (u 1) else
omega_iterate (u 0) h (u 1)) of type set → set.
```

Fig. 10. Definitions for the Adjoint Functor for Bijections in the Megalodon Wiki

```
Theorem. (MetaCat_struct_u_bij_forgetfree)
  MetaAdjunction_strict (λ_ ⇒ True) HomSet (λX ⇒ (lam_id X)) (λX Y Z f g ⇒ (lam_comp X f
  g)) struct_u_bij Hom_struct_u struct_id struct_comp MetaCat_struct_u_bij_free0
  MetaCat_struct_u_bij_free1 (λX ⇒ X 0) (λX Y f ⇒ f) MetaCat_struct_u_forgetfree_eta
  MetaCat_struct_u_bij_forgetfree_eps
```

Fig. 11. Adjunction Theorem for Bijections in the Megalodon Wiki

corresponding proposition was published on the blockchain in June 2024. The witnesses in the proof are as follows:

- $\mathcal{F}_0\ X$ is the structure with carrier $\mathbb{Z} \times X$ (where $\mathbb{Z}$ is the set of integers) and the (bijective) unary function taking $u \in \mathbb{Z} \times X$ to $(u_0 + 1, u_1)$.
- $\mathcal{F}_1\ X\ Y\ f$ is the morphism taking $u \in \mathbb{Z} \times X$ to $(u_0, f(u_1)) \in \mathbb{Z} \times Y$.
- $\eta\ X$ is the function from X to $\mathbb{Z} \times X$ taking x to $(0, x)$.
- $\varepsilon\ (X, f)$ is the function from $\mathbb{Z} \times X$ to X taking u to $f^{u_0}(u_1)$. (Note that if $u_0 < 0$, this means iterating the inverse of the bijection f.)

There are definitions corresponding to these choices of $\mathcal{F}_0$, η and ε in the Megalodon Wiki, and are shown here in Figure 10. The main proof is of a theorem that checks these choices indeed give an adjunction. The proof is long and detailed, so we only show the statement in Figure 11. The actual bounty was on the statement that such an adjunction exists. Since all the details were checked in the previous theorem, we can show the existential theorem with its proof in Figure 12.

The most recent of the 33 propositions to be disproven takes $\mathcal{D}$ to be a category with monoids as objects. Here "disproven" means the negation of the proposition was proven. This should be surprising as it is clear that given a set X one can create a monoid freely generated by X, and this should provide the desired left adjoint. However, the category in question was defined to have semigroup homomorphisms as arrows, and not every semigroup homomorphism (preserving the operation) also preserves the identity element. The definition is arguably a bug in the definition of the category of monoids. The bounty presumably encouraged someone to look closely enough at the definition to find and exploit the bug to prove the negation of the proposition. Using the explorer to examine the proof of the surprising result, others can discover the same bug. The proof follows from the fact that there is no initial object in the category of

```
Proposition. (MetaCat struct u bij left adjoint forgetful)
  ∃F0 _ : set → set, ∃F1 _ : set → set → set → set, ∃eta eps _ : set → set, MetaAdjunction strict (λ_ ⇒ True) HomSet
  (λX ⇒ (lam id X)) (λX Y Z f g ⇒ (lam comp X f g)) struct u bij Hom struct u struct id struct comp F0 F1 (λX ⇒
  X 0) (λX Y f ⇒ f) eta eps
In Proofgold the corresponding term root is 062a21... and proposition id is a69df3...
  Proof:
  Set F0 to be the term λX ⇒ pack u (int × X) (λu ⇒ (u 0 + 1,u 1)) of type set → set.
  Set F1 to be the term λX Y h ⇒ λu ∈ int × X ⇒ (u 0,h (u 1)) of type set → set → set → set.
  Set eta to be the term λX ⇒ λx ∈ X ⇒ (0,x) of type set → set.
  Set epsPhi to be the term λX h ⇒ λu ∈ int × X ⇒ if u 0 ≤ 0 then omega iterate (- (u 0)) (inv X h) (u 1)
  else omega iterate (u 0) h (u 1) of type set → (set → set) → set.
  Set eps to be the term λA ⇒ unpack u i A epsPhi of type set → set.
  We use F0 to witness the existential quantifier.
  We use F1 to witness the existential quantifier.
  We use eta to witness the existential quantifier.
  We use eps to witness the existential quantifier.
  An exact proof term for the current goal is MetaCat struct u bij forgetfree.
  ∎
```

Fig. 12. Existential Adjunction Theorem for Bijections in the Megalodon Wiki

```
Proposition. (MetaCat struct b monoid initial neg)
  ¬ ∃Y _ : set, ∃uniqa _ : set → set, initial p struct b monoid Hom struct b struct id
  struct comp Y uniqa
```

Fig. 13. There is no Initial Monoid in the Megalodon Wiki

monoids and semigroup homomorphisms (and an alleged left adjoint would send the initial object in the category of sets to an initial monoid). The proof that there is no initial monoid (with semigroup homomorphisms) essentially follows from the fact that the two constant functions from a monoid to the multiplicative monoid $\{0,1\}$ are both (always) semigroup homomorphisms, thus guaranteeing an alleged initial monoid would not have a unique morphism to $\{0,1\}$.

In a document in the Megalodon Wiki, the statement of the theorem that there is no initial monoid (with semigroup homomorphisms) is shown in Figure 13. The proof (using the multiplicative monoid on $\{0,1\}$) is also in the same document, but we omit it here as it is long and detailed.[6] The negation of the adjunction conjecture is then proven as a consequence. Its statement and proof in the document in the Megalodon Wiki are shown in Figure 14. The proof begins by assuming there exists an adjunction, i.e., $\exists F_0 : \iota\iota.\cdots$. In general, a proposition $\exists x : \alpha.\varphi$ is considered the same as $\forall q : o.(\forall x : \alpha.\varphi \to q) \to q$. This is why the existential assumption can be "applied" to the current goal (of proving $\bot$), followed by a "let" (essentially giving a fresh name x for the object) and an "assume" (giving the property φ of the object x). This is repeated four times to obtain F_0, F_1, η and ε. There is a previous (unshown) theorem that there is an initial object in the category of sets. (It is witnessed by the empty set, but this is irrelevant here.) We apply that previous theorem to obtain an initial set *Init*

[6] Note that Hom_struct_b : $\iota\iota o$ is defined so that Hom_struct_b A B h holds when h is a function from the carrier of A to the carrier of B preserving the binary operation. A correct definition of the category of monoids would extend the signature to explicitly include the identity element in addition the binary operation. This would presumably have a name like Hom_struct_b_e.

```
Proposition. (MetaCat struct b monoid left adjoint forgetful neg)
  ¬ ∃F0 : set → set, ∃F1 : set → set → set, ∃eta eps : set → set, MetaAdjunction strict (λ_ ⇒ True)
  HomSet (λX ⇒ (lam id X)) (λX Y Z f g ⇒ (lam comp X f g)) struct b monoid Hom struct b struct id struct comp
  F0 F1 (λX ⇒ X 0) (λX Y f ⇒ f) eta eps

In Proofgold the corresponding term root is cbb885... and proposition id is 62ee62...

  Proof:
  Assume H. Apply H to the current goal.
  Let F0 be given. Assume H. Apply H to the current goal.
  Let F1 be given. Assume H. Apply H to the current goal.
  Let eta be given. Assume H. Apply H to the current goal.
  Let eps be given.
  Assume H1: MetaAdjunction strict (λ_ ⇒ True) HomSet (λX ⇒ (lam id X)) (λX Y Z f g ⇒ (lam comp X f g))
  struct b monoid Hom struct b struct id struct comp F0 F1 (λX ⇒ X 0) (λX Y f ⇒ f) eta eps.
  Apply MetaCatSet initial to the current goal.
  Let Init be given.
  Assume H.
  Apply H to the current goal.
  Let uniq be given.
  Assume H2: initial p (λ_ ⇒ True) HomSet (λX ⇒ (λx ∈ X ⇒ x)) (λX Y Z f g ⇒ (λx ∈ X ⇒ f (g x))) Init uniq.
  Apply MetaCat struct b monoid initial neg to the current goal.
  We use F0 Init to witness the existential quantifier.
  An exact proof term for the current goal is LeftAdjointsPreserveInitial (λ_ ⇒ True) HomSet (λX ⇒ (λx ∈ X ⇒
  x)) (λX Y Z f g ⇒ (λx ∈ X ⇒ f (g x))) struct b monoid Hom struct b struct id struct comp F0 F1 (λX ⇒ X 0)
  (λX Y f ⇒ f) eta eps H1 Init uniq H2.
  ∎
```

Fig. 14. There is no Left Adjoint to the Forgetful Functor for Monoids in the Megalodon Wiki

and a function *uniq* that takes each set X to the unique function from $Init$ to X. Our goal is to prove $\bot$. We then apply the theorem from Figure 13 which leaves the subgoal of proving there is an initial monoid. The monoid $F_0\ Init$ is used to witness that there is an initial monoid, and this is justified by a previous (unshown) theorem that ensures left adjoints preserve initial objects.

An example of a proposition with an open bounty is given by taking $\mathcal{D}$ to be the category of groups.[7] In this case, the proposition should be provable by taking $\mathcal{F}_0\ X$ to be the free group generated by X, but no one has yet done this construction and proven the relevant theorem.

6 Related Work

Most proof assistants today are accompanied by tools for exploring their formal libraries via web interfaces. The Mizar Mathematical Library (MML) [4], one of the oldest central repositories of formalized mathematics, in addition to its journal version, has been accessible through HTML renderings and search tools [15,22]. Isabelle's Archive of Formal Proofs (AFP) [7] provides a curated collection of formal developments, rendered online using Isabelle's document preparation system. The Coq proof assistant is supported by an online web interface that allows interactive Coq sessions entirely within the web [12]. Its extension to ProofWeb offers a web interface for multiple proof assistants, aiming to lower the barrier to entry for formal verification [18]. The Lean community

[7] The bounty can currently be seen on the explorer at the link `https://formalweb3.uibk.ac.at/pgbce/q.php?b=TMQvwY1m9iU5rev4qXQjWWGYTZDHwCseEMv`.

has developed a variety of web tools, including the ProofWidgets library [20], with the paper discussing a lot of related work concerning user interfaces for theorem proving.

Beyond these assistant-specific interfaces, more advanced and general-purpose systems have been developed to support exploration, integration, and sharing of formal mathematical knowledge. The `MathHub/MMT` framework provides a logic-independent framework for querying across distributed libraries [5]. The Formal Abstracts aims to explore structured summaries of theorems that link informal and formal mathematics. Its web access to both metadata and formal developments[8] has been impactful, but it appears to be currently inactive.

The Tezos [14] blockchain includes a smart contract programming language that has been designed to make formal verification easier, which means that off-chain formal methods can be used to verify properties of programs/scripts. This supports formal methods, albeit offline [6]. Several blockchain explorers exist for Tezos, but they focus on the usual smart contract information and do not include verification content. Qeditas was an early attempt to combine formalized mathematics with blockchain technology, but the project is no longer maintained; Proofgold builds on its ideas, refining and extending them into a working system.

There have been a number of formalizations of category theory done in type-theoretic proof assistants; examples include UniMath [2] in Coq, the Lean 3 library [1], Agda-Category [23]; several developments in Isabelle/HOL [21] and Isabelle/HOLZF and even Mizar [11]. While type-theoretic systems allow concise and expressive formulations, they often provide less automation; in contrast, Isabelle/HOL offers strong automation but can make some constructions more cumbersome due to the lack of dependent types, and Mizar emphasizes human readability but it has limited expressiveness for higher-category theory.

7 Conclusion

We created a web-based blockchain explorer for Proofgold that allows users to interact with mathematical knowledge contained there. We have demonstrated the utility of the system by presenting several conjectures and their proofs or refutations using the system. Future work includes improving support for faster collaboration, further improvements to the visualization of the formal mathematics, as well as better search [3].

Acknowledgements The results were supported by the Czech Ministry of Education, Youth and Sports within the dedicated program ERC CZ under the project POSTMAN no. LL1902, the ERC PoC grant *FormalWeb3* no. 101156734, Amazon Research Award, and the Czech Science Foundation grant no. 25-17929X.

References

1. The Lean mathematical library. CoRR **abs/1910.09336** (2019), `http://arxiv.org/abs/1910.09336`

[8] `https://formalabstracts.github.io/`

2. Ahrens, B., Matthes, R., Mörtberg, A.: From signatures to monads in UniMath. J. Autom. Reason. **63**(2), 285–318 (2019). https://doi.org/10.1007/S10817-018-9474-4, https://doi.org/10.1007/s10817-018-9474-4
3. Asperti, A., Guidi, F., Coen, C.S., Tassi, E., Zacchiroli, S.: A content based mathematical search engine: Whelp. In: Filliâtre, J., Paulin-Mohring, C., Werner, B. (eds.) Types for Proofs and Programs, International Workshop, TYPES 2004, Jouy-en-Josas, France, December 15-18, 2004, Revised Selected Papers. Lecture Notes in Computer Science, vol. 3839, pp. 17–32. Springer (2004). https://doi.org/10.1007/11617990_2, https://doi.org/10.1007/11617990_2
4. Bancerek, G., Byliński, C., Grabowski, A., Korniłowicz, A., Matuszewski, R., Naumowicz, A., Pąk, K.: The role of the Mizar mathematical library for interactive proof development in Mizar. J. Autom. Reason. **61**(1-4), 9–32 (2018). https://doi.org/10.1007/S10817-017-9440-6, https://doi.org/10.1007/s10817-017-9440-6
5. Bercic, K., Kohlhase, M., Rabe, F.: Towards a unified mathematical data infrastructure: Database and interface generation. In: Kaliszyk, C., Brady, E.C., Kohlhase, A., Coen, C.S. (eds.) Intelligent Computer Mathematics - 12th International Conference, CICM 2019, Prague, Czech Republic, July 8-12, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11617, pp. 28–43. Springer (2019). https://doi.org/10.1007/978-3-030-23250-4_3, https://doi.org/10.1007/978-3-030-23250-4_3
6. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., Zanella-Béguelin, S.: Formal verification of smart contracts: Short paper. In: Murray, T.C., Stefan, D. (eds.) Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security, PLAS@CCS 2016, Vienna, Austria, October 24, 2016. pp. 91–96. ACM (2016). https://doi.org/10.1145/2993600.2993611, https://doi.org/10.1145/2993600.2993611
7. Blanchette, J.C., Haslbeck, M.W., Matichuk, D., Nipkow, T.: Mining the archive of formal proofs. In: Kerber, M., Carette, J., Kaliszyk, C., Rabe, F., Sorge, V. (eds.) Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings. Lecture Notes in Computer Science, vol. 9150, pp. 3–17. Springer (2015). https://doi.org/10.1007/978-3-319-20615-8_1, https://doi.org/10.1007/978-3-319-20615-8_1
8. Brown, C.E., Kaliszyk, C., Gauthier, T., Urban, J.: Proofgold: Blockchain for formal methods. In: Dargaye, Z., Schneidewind, C. (eds.) 4th International Workshop on Formal Methods for Blockchains, FMBC@CAV 2022, August 11, 2022, Haifa, Israel. OASIcs, vol. 105, pp. 4:1–4:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022). https://doi.org/10.4230/OASICS.FMBC.2022.4, https://doi.org/10.4230/OASIcs.FMBC.2022.4
9. Brown, C.E., Pąk, K.: A tale of two set theories. In: Kaliszyk, C., Brady, E.C., Kohlhase, A., Coen, C.S. (eds.) Intelligent Computer Mathematics - 12th International Conference, CICM 2019, Prague, Czech Republic, July 8-12, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11617, pp. 44–60. Springer (2019). https://doi.org/10.1007/978-3-030-23250-4_4, https://doi.org/10.1007/978-3-030-23250-4_4
10. Buzzard, K., Commelin, J., Massot, P.: Formalising perfectoid spaces. In: Blanchette, J., Hritcu, C. (eds.) Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs, CPP 2020, New Orleans,

LA, USA, January 20-21, 2020. pp. 299–312. ACM (2020). https://doi.org/10.1145/3372885.3373830, https://doi.org/10.1145/3372885.3373830
11. Byliński, C.: Introduction to categories and functors. Formalized Mathematics **1**(**2**), 409–420 (1990), http://fm.mizar.org/1990-1/pdf1-2/cat_1.pdf
12. Corbineau, P., Kaliszyk, C.: Cooperative repositories for formal proofs. In: Kauers, M., Kerber, M., Miner, R., Windsteiger, W. (eds.) Towards Mechanized Mathematical Assistants, 14th Symposium, Calculemus 2007, 6th International Conference, MKM 2007, Hagenberg, Austria, June 27-30, 2007, Proceedings. Lecture Notes in Computer Science, vol. 4573, pp. 221–234. Springer (2007). https://doi.org/10.1007/978-3-540-73086-6_19, https://doi.org/10.1007/978-3-540-73086-6_19
13. Desharnais, M., Tóth, B., Waldmann, U., Blanchette, J., Tourret, S.: A modular formalization of superposition in Isabelle/HOL. In: Bertot, Y., Kutsia, T., Norrish, M. (eds.) 15th International Conference on Interactive Theorem Proving, ITP 2024, September 9-14, 2024, Tbilisi, Georgia. LIPIcs, vol. 309, pp. 12:1–12:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2024). https://doi.org/10.4230/LIPICS.ITP.2024.12, https://doi.org/10.4230/LIPIcs.ITP.2024.12
14. Doan, T.T.H., Thiemann, P.: A formal verification framework for tezos smart contracts based on symbolic execution. In: Kiselyov, O. (ed.) Programming Languages and Systems - 22nd Asian Symposium, APLAS 2024, Kyoto, Japan, October 22-24, 2024, Proceedings. Lecture Notes in Computer Science, vol. 15194, pp. 305–324. Springer (2024). https://doi.org/10.1007/978-981-97-8943-6_15, https://doi.org/10.1007/978-981-97-8943-6_15
15. Furushima, H., Yamamichi, D., Shigenaka, S., Nakasho, K., Wasaki, K.: An integrated web platform for the Mizar mathematical library. In: Buzzard, K., Kutsia, T. (eds.) Intelligent Computer Mathematics - 15th International Conference, CICM 2022, Tbilisi, Georgia, September 19-23, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13467, pp. 141–146. Springer (2022). https://doi.org/10.1007/978-3-031-16681-5_9, https://doi.org/10.1007/978-3-031-16681-5_9
16. Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Roux, S.L., Mahboubi, A., O'Connor, R., Biha, S.O., Pasca, I., Rideau, L., Solovyev, A., Tassi, E., Théry, L.: A machine-checked proof of the odd order theorem. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving - 4th International Conference, ITP 2013, Rennes, France, July 22-26, 2013. Proceedings. Lecture Notes in Computer Science, vol. 7998, pp. 163–179. Springer (2013). https://doi.org/10.1007/978-3-642-39634-2_14, https://doi.org/10.1007/978-3-642-39634-2_14
17. Hales, T., Adams, M., Bauer, G., Dang, T.D., Harrison, J., Hoang, L.T., Kaliszyk, C., Magron, V., Mclaughlin, S., Nguyen, T.T., Nguyen, Q.T., Nipkow, T., Obua, S., Pleso, J., Rute, J., Solovyev, A., Ta, T.H.A., Tran, N.T., Trieu, T.D., Urban, J., Vu, K., Zumkeller, R.: A formal proof of the Kepler conjecture. Forum of Mathematics, Pi **5** (2017). https://doi.org/10.1017/fmp.2017.1
18. Hendriks, M., Kaliszyk, C., van Raamsdonk, F., Wiedijk, F.: Teaching logic using a state-of-the-art proof assistant. Acta Didactica Napocensia **3**(2), 35–48 (June 2010)
19. Lambek, J., Scott, P.: Introduction to higher order categorical logic. Cambridge University Press, Cambridge, UK (1986)
20. Nawrocki, W., Ayers, E.W., Ebner, G.: An extensible user interface for Lean 4. In: Naumowicz, A., Thiemann, R. (eds.) 14th International Conference on Interactive Theorem Proving, ITP 2023, July 31 to August 4, 2023, Białystok,

Poland. LIPIcs, vol. 268, pp. 24:1–24:20. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2023). https://doi.org/10.4230/LIPICS.ITP.2023.24, https://doi.org/10.4230/LIPIcs.ITP.2023.24
21. Stark, E.W.: Category theory with adjunctions and limits. Arch. Formal Proofs **2016** (2016), https://www.isa-afp.org/entries/Category3.shtml
22. Tomaszuk, D., Szeremeta, Ł., Korniłowicz, A.: MMLKG: Knowledge graph for mathematical definitions, statements and proofs. Sci Data **10**(791) (2023), https://doi.org/10.1038/s41597-023-02681-3
23. Vezzosi, A., Mörtberg, A., Abel, A.: Cubical Agda: a dependently typed programming language with univalence and higher inductive types. Proc. ACM Program. Lang. **3**(ICFP), 87:1–87:29 (2019). https://doi.org/10.1145/3341691, https://doi.org/10.1145/3341691
24. Wiedijk, F.: Introduction. In: Wiedijk, F. (ed.) The Seventeen Provers of the World, Foreword by Dana S. Scott, Lecture Notes in Computer Science, vol. 3600, pp. 1–9. Springer (2006). https://doi.org/10.1007/11542384_1, https://doi.org/10.1007/11542384_1