Proofgold: Blockchain for Formal Methods

Chad E. Brown
Czech Technical University in Prague

Cezary Kaliszyk
University of Innsbruck

Thibault Gauthier
Czech Technical University in Prague

Josef Urban
Czech Technical University in Prague

Abstract

Proofgold is a peer to peer cryptocurrency making use of formal logic. Users can publish theories and then develop a theory by publishing documents with definitions, conjectures and proofs. The blockchain records the theories and their state of development (e.g., which theorems have been proven and when). Two of the main theories are a form of classical set theory (for formalizing mathematics) and an intuitionistic theory of higher-order abstract syntax (for reasoning about syntax with binders). We give examples of definitions and theorems published into the Proofgold blockchain to demonstrate how the Proofgold network can be used to support formalization efforts. We have also significantly modified the open source Proofgold Core client software to create a faster, more stable and more efficient client, Proofgold Lava. Two important changes are the cryptography code and the database code, and we discuss these improvements.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases Formal logic, Blockchain, ProofGold

Digital Object Identifier 10.4230/OASIcs...

1 Introduction

Proofgold is a cryptocurrency network with support for formal logic and mathematics. At the core of Proofgold is a proof checker for intuitionistic higher-order logic with functional extensionality. On top of this framework users can publish *theories*. A theory consists of a finite number of primitive constants along with their types and a finite number of sentences as axioms. A theory is uniquely identified by its 256-bit identifier given by the Merkle root of the theory (seen as a tree). After a theory has been published, *documents* can be published in the theory. Documents can define new objects (using primitives or previously defined objects), prove new theorems and make new conjectures.

When a theory is published, the axioms are associated with public keys which are marked as the *owners* of the propositions. Likewise, when a document proves a theorem within a theory, a public key (associated with the publisher of the document) is associated with the proven proposition. These are the only ways propositions can have declared owners. As a consequence, it is possible to determine if a proposition is known (either as an axiom or as a previously proven theorem) by checking if it has an owner. This method of keeping up with proven propositions by associating them with public keys was described in the Qeditas white paper [20] and was part of the Qeditas codebase.[1] Ownership of propositions also gives a way of redeeming bounties by proving conjectures. A bounty can be placed on an unproven

[1] A large part of Proofgold's code was inherited from the open source Qeditas project. More information about Qeditas is at https://iohk.io/en/projects/qeditas/.

$$\frac{}{\Gamma \vdash s}\, s \in \mathcal{A} \qquad \frac{}{\Gamma \vdash s}\, s \in \Gamma \qquad \frac{\Gamma \vdash s}{\Gamma \vdash t}\, s{\approx}t \qquad \frac{\Gamma, s \vdash t}{\Gamma \vdash s \to t} \qquad \frac{\Gamma \vdash s \to t \qquad \Gamma \vdash s}{\Gamma \vdash t}$$

$$\frac{\Gamma \vdash s}{\Gamma \vdash \forall x.s}\, x \in \mathcal{V}_\alpha \setminus \mathcal{F}\Gamma \qquad \frac{\Gamma \vdash \forall x.s}{\Gamma \vdash s^x_t}\, x \in \mathcal{V}_\alpha, t \in \Lambda_\alpha$$

$$\frac{\Gamma \vdash sx = tx}{\Gamma \vdash s = t}\, x \in \mathcal{V}_\alpha \setminus (\mathcal{F}\Gamma \cup \mathcal{F}s \cup \mathcal{F}t) \text{ AND } s,t \in \Lambda_{\alpha\beta}$$

Figure 1 Proof Calculus for Intuitionistic HOL

proposition where this bounty can only be spent by the owner of a proposition (or the owner of the negated proposition). By publishing a document resolving the conjecture, the bounty proposition (or its negation) will become owned by public keys associated with the publisher of the document. After this the bounty can be claimed.

2 Intuitionistic Higher Order Logic

We briefly describe a formulation of intuitionistic higher order logic (IHOL). We begin with a set $\mathcal{T}$ of *simple types*. One base type o is the type of propositions. In general Proofgold allows finitely many other base types, but we will only consider cases with one other base type ι. All other types are $\alpha\beta$, meaning the type of functions from α to β.

We next define a family of simply typed terms. For each type $\alpha \in \mathcal{T}$, let $\mathcal{V}_\alpha$ be a countably infinite set of variables of type α. Let $\mathcal{C}$ be a finite set of typed constants. We define a set Λ_α of *terms of type* α as follows: For each variable x of type α, $x \in \Lambda_\alpha$. For each constant c of type α, $c \in \Lambda_\alpha$. If $s \in \Lambda_{\alpha\beta}$ and $t \in \Lambda_\alpha$, then $(st) \in \Lambda_\beta$. If x is a variable of type α and $s \in \Lambda_\beta$, then $(\lambda x.s) \in \Lambda_{\alpha\beta}$. If $s,t \in \Lambda_o$, then $(s \to t) \in \Lambda_o$. If x is a variable of type α and $s \in \Lambda_o$, then $(\forall x.s) \in \Lambda_o$. Note that Λ_α also depends on the set $\mathcal{C}$, but this set will be fixed in each theory.

We use common conventions to omit parentheses. We sometimes include annotations on λ and $\forall$ bound variables (e.g., $\lambda x : \alpha.s$ and $\forall y : \beta.s$) to indicate the type of the variable. We define $\mathcal{F}s$ to be the free variables of s and for sets A of terms we define $\mathcal{F}A$ to be $\bigcup_{s \in A} \mathcal{F}s$. We assume a capture avoiding substitution s^x_t is defined. Terms of type o are called *propositions*. A *sentence* is a proposition with no free variables.

The only built-in logical connective is implication ($\to$) and the only built-in quantifier is the universal quantifier ($\forall$). In the context of higher-order logic it is well-known how to define the remaining logical constructs in a way that respects their intuitionistic meaning. In each case we use an impredicative definition that traces its roots to Russell [16] and Prawitz [14]. We define $\bot$ to be the proposition $\forall p : o.p$ where x is a variable of type o. We write $\neg s$ for $s \to \bot$. We define $\wedge$ to be $\lambda qr : o.\forall p : o.(q \to r \to p) \to p$ and write $s \wedge t$ for $(\wedge s)t$. We define $\vee$ to be $\lambda qr : o.\forall p : o.(q \to p) \to (r \to p) \to p$ and write $s \vee t$ for $(\vee s)t$. For each type α we use $\exists x : \alpha.s$ as notation for $\forall p : o.(\forall x : \alpha.s \to p) \to p$ where p is not x and is not free in s. For equality we write $s = t$ (where s and t are type α) as notation for $\forall p : \alpha\alpha o.pst \to pts$ where p is neither free in s nor t. This is a modification of Leibniz equality which we will call *symmetric Leibniz equality*. We write $s \neq t$ to mean $(s = t) \to \bot$. The $\beta\eta$-conversion relation $s{\approx}t$ is defined in the usual way.

Let $\mathcal{A}$ be a set of sentences intended to be axioms of a theory. A natural deduction system for intuitionistic higher-order logic with functional extensionality and axioms $\mathcal{A}$ is given by Figure 1. In particular the rules define when $\Gamma \vdash s$ holds where Γ is a finite set of propositions and s is a proposition. Aside from the treatment of functional extensionality, this is the same as the natural deduction calculus described in [3].

Adding Curry-Howard style checkable proof terms to such a calculus is well-understood and we do not dwell on this here [18]. Proofs published in Proofgold documents are given by such proof terms. There are two practical restrictions Proofgold places on proofs. One restriction is that proofs cannot be too big. A proof is part of a document, a document is published in a transaction and a transaction is published in a block. Proofgold has a block size limit of 500KB, so that proofs larger than 500KB (measured in Proofgold's binary format) cannot be published. Another restriction is that checking a proof is not allowed to be too hard. In an extreme case, checking a proof could require β-normalizing a term of size m to obtain a term of size $2^{2^{2^m}}$ (or even much larger). The Proofgold Core checker avoids such "poison proofs" by maintaining a counter that increments while a document is being checked. Each step of the computation increments the counter. For example, substituting t for a de Bruijn index x in a term $r\ x\ x$ increments the counter at least 5 times since there are two applications, two occurrences of x and one occurrence of r. Depending on the structure of r the counter may be incremented more. Also, in practice the substitution may be beneath a binder so that de Bruijn indices in t may need to be shifted. Such shifting increments the counter in a similar way. If the counter reaches a certain bound (150 million), then an exception is raised and the document is considered to be incorrect.

3 Proofgold Theories

3.1 HF Theory

Proofgold has one built-in theory: a theory of hereditarily finite sets (HF). There are many primitive constants, but only six do not have a defining equation: $\varepsilon : (\iota o)\iota$ (a "choice" operator), $\in : \iota\iota o$ (set membership), $\emptyset : \iota$ (the empty set, also the ordinal 0), $\bigcup : \iota\iota$ (the union operator), $\wp : \iota\iota$ (the power set operator) and $\mathsf{r} : \iota(\iota\iota)\iota$ (the replacement operator). For each of the above constants, there is at least one axiom giving a property the constant must satisfy. Additional axioms are a classical principle ($\forall p.\neg\neg p \to p$), set extensionality, an $\in$-induction principle and an induction principle implying all sets are hereditarily finite.

The theory additionally includes 97 constants with axioms giving a definitional equation for each constant. Examples include a constant indicating a set has exactly 5 elements, a constant indicating that an algebraic structure is a loop and a constant indicating that two untyped combinators (represented as sets) are equivalent under conversion. The HF theory was used to generate pseudorandom bounties for the first 5000 Proofgold blocks. These extra constants make it possible to easily generate sentences targeting certain classes.[2] The last of the pseudorandom bounties was automatically placed in December 2020. As of May 2022, 38% of the conjectures have been resolved and the bounties collected. The fact that 62% are still outstanding after 17 months is an indication of the difficulty of the problems.

[2] More information can be found in http://grid01.ciirc.cvut.cz/~chad/pfghf.pdf.

3.2 Two HOTG Theories

There are two theories axiomatizing higher-order Tarski Grothendieck set theory (HOTG). The two theories follow the two formulations described in [3]. One is based on the Mizar formulation[3] and the other is based on the Egal formulation. Both theories are classical via the Diaconescu proof of excluded middle from choice (at ι) and set extensionality [15]. Most of the documents published into the Proofgold blockchain have been published in the HOTG-Egal theory. These documents target formalization of mathematics. A highlight is the construction of the real numbers via a representation of Conway's surreal numbers [4].

3.3 A Theory for HOAS

A different kind of theory published into the Proofgold blockchain is a theory for reasoning about syntax. Unlike the theories above, this theory does not imply classical principles.

For our theory of syntax, we include one base type ι, two primitive constants $\mathsf{P} : \iota\iota\iota$ and $\mathsf{B} : (\iota\iota)\iota$ and four axioms:

- Pairing is injective: $\forall xyzw : \iota.\mathsf{P}xy = \mathsf{P}zw \to x = z \wedge y = w$.
- Binding is injective: $\forall fg : \iota\iota.\mathsf{B}f = \mathsf{B}g \to f = g$.
- Binding and pairing give distinct values: $\forall xy : \iota.\forall f : \iota\iota.\mathsf{P}xy \neq \mathsf{B}f$.
- Propositional extensionality: $\forall pq : o.(p \to q) \to (q \to p) \to p = q$.

The constant P is a generic pairing operation on syntax and B is a generic binding operation, allowing representation by higher-order abstract syntax (HOAS) [13].

We can embed many syntactic constructs into the theory by building on top of the basic pairing and binding operators. For example, we could embed untyped λ-calculus by taking P to represent application and B to represent λ-abstraction. Instead of adopting this simple approach, we will use tagged pairs when representing application and λ-abstraction, so that there will still be infinitely many pieces of syntax that do not represent untyped λ-terms. To do this we will need one tag, so let us define nil to be $\mathsf{B}(\lambda x.x)$. Now we can define $\mathsf{A} : \iota\iota\iota$ to be $\lambda xy : \iota.\mathsf{P}\ \mathsf{nil}\ (\mathsf{P}\ x\ y)$ and define $\mathsf{L} : (\iota\iota)\iota$ to be $\lambda f : \iota\iota.\mathsf{P}\ \mathsf{nil}\ (\mathsf{B}\ f)$. It is easy to prove A and L are both injective and give distinct values.

We can now impredicatively define the set of untyped λ-terms relative to a set $\mathcal{G}$ (intended to be the set of possible free variables) as follows. Let us write $(\mathcal{G}, x)$ for the term $\lambda y : \iota.\mathcal{G}\ y \vee y = x$. Here $\mathcal{G}$ has type ιo while x and y have type ι (and are different). We will define $\mathsf{Ter} : (\iota o)\iota o$ so that Ter is the least relation satisfying three conditions:

- $\forall \mathcal{G} : \iota o.\forall y : \iota.\mathcal{G}y \to \mathsf{Ter}\ \mathcal{G}\ y$,
- $\forall \mathcal{G} : \iota o.\forall f : \iota\iota.(\forall x : \iota.\mathsf{Ter}\ (\mathcal{G}, x)\ (fx)) \to \mathsf{Ter}\ \mathcal{G}\ (\mathsf{L}\ f)$ and
- $\forall \mathcal{G} : \iota o.\forall yz : \iota.\mathsf{Ter}\ \mathcal{G}\ y \to \mathsf{Ter}\ \mathcal{G}\ z \to \mathsf{Ter}\ \mathcal{G}\ (\mathsf{A}\ y\ z)$.

Technically, the impredicative definition of Ter is given as

$$\begin{aligned}\mathsf{Ter} \quad := \quad & \lambda \mathcal{G} : \iota o.\lambda x : \iota.\forall p : (\iota o)\iota o.(\forall \mathcal{G} : \iota o.\forall y : \iota.\mathcal{G}y \to p\ \mathcal{G}\ y) \\ & \to (\forall \mathcal{G} : \iota o.\forall f : \iota\iota.(\forall x : \iota.p\ (\mathcal{G}, x)\ (fx)) \to p\ \mathcal{G}\ (\mathsf{L}\ f)) \\ & \to (\forall \mathcal{G} : \iota o.\forall yz : \iota.p\ \mathcal{G}\ y \to p\ \mathcal{G}\ z \to p\ \mathcal{G}\ (\mathsf{A}\ y\ z)) \to p\ \mathcal{G}\ x.\end{aligned}$$

[3] More information about the Mizar formulation of HOTG can be found in http://grid01.ciirc.cvut.cz/~chad/pfgmizar.pdf.

We can similarly define one-step β-reduction (relative to a set of variables) as follows:

$$\begin{aligned}\mathsf{Beta}_1 \quad := \quad & \lambda\mathcal{G} : \iota o.\lambda xy : \iota.\forall r : (\iota o)\iota\iota o. \\ & (\forall \mathcal{G} : \iota o.\forall f : \iota\iota.\forall z.(\forall x.\mathsf{Ter}\ (\mathcal{G}, x)\ (fx)) \to \mathsf{Ter}\ \mathcal{G}z \to r\ \mathcal{G}\ (\mathsf{A}\ (\mathsf{L}\ f)\ z)\ (fz)) \\ & \to (\forall \mathcal{G} : \iota o.\forall fg : \iota\iota.(\forall z.r\ (\mathcal{G}, z)\ (fz)(gz)) \to r\ \mathcal{G}\ (\mathsf{L}\ f)\ (\mathsf{L}\ g)) \\ & \to (\forall \mathcal{G} : \iota o.\forall xyz.r\ \mathcal{G}\ x\ z \to \mathsf{Ter}\ \mathcal{G}y \to r\ \mathcal{G}\ (\mathsf{A}\ x\ y)\ (\mathsf{A}\ z\ y)) \\ & \to (\forall \mathcal{G} : \iota o.\forall xyz.r\ \mathcal{G}\ y\ z \to \mathsf{Ter}\ \mathcal{G}x \to r\ \mathcal{G}\ (\mathsf{A}\ x\ y)\ (\mathsf{A}\ x\ z)) \to r\ \mathcal{G}\ x\ y.\end{aligned}$$

We can then define $\mathsf{BetaE}\ \mathcal{G}$ to be the least equivalence relation (relative to the domain $\mathsf{Ter}\ \mathcal{G}$) containing $\mathsf{Beta}_1\ \mathcal{G}$. We omit the details here.

These definitions give us sufficient material to make conjectures that ask for certain kinds of untyped λ-terms. Let $\emptyset$ be notation for the term $\lambda x : \iota.\bot$ (representing the empty set of variables). Consider the following sentences:

$$\exists F : \iota.\mathsf{Ter}\ \emptyset\ F \quad \wedge \quad \forall x : \iota.\mathsf{BetaE}\ (\emptyset, x)\ (\mathsf{A}\ F\ x)\ x \tag{1}$$

$$\exists Y : \iota.\mathsf{Ter}\ \emptyset\ Y \quad \wedge \quad \forall f : \iota.\mathsf{BetaE}\ (\emptyset, f)\ (\mathsf{A}\ Y\ f)\ (\mathsf{A}\ f\ (\mathsf{A}\ Y\ f)) \tag{2}$$

Sentence (1) asserts the existence of an identity combinator while sentence (2) asserts the existence of a fixed point combinator. In order to prove each sentence a combinator with the right property must be given as a witness and then be proven to have the property.[4]

As a demonstration, these sentences were published as conjectures (with bounties) in documents published into the Proofgold blockchain. The solutions were then published as two theorems (with proofs). The solutions contain the witnesses: $\mathsf{L}(\lambda x.x)$ for (1) and the famous Y-combinator $\mathsf{L}(\lambda f.\mathsf{A}(\mathsf{L}(\lambda x.\mathsf{A}\ f\ (\mathsf{A}\ x\ x)))(\mathsf{L}(\lambda x.\mathsf{A}\ f\ (\mathsf{A}\ x\ x))))$ for (2).

These simple examples suggest how Proofgold could be used to publish conjectures for verification conditions of programs or even conjectures asking for a program satisfying a specification. This could especially be useful for working with functional smart contract languages such as Plutus Core[5].

4 Proofgold Lava Client

The existing client, ProofGold Core, has already included all the functionality needed to run the blockchain. However, certain parts of the implementation did not scale well. In particular as the number of proofs already in the blockchain grew operations such as synchronizing new clients or rechecking the blockchain became too costly. For these reasons we reimplemented parts of the client software and provide it as the Proofgold Lava Client and discuss the changes in this section.

4.1 Database Layer

The Proofgold client software uses 19 databases. In the Core software they have been stored in 19 directories, each with an index file and and a data file. Lookups in this database, including locking, became a significant overhead for all Merkle tree operations. For this reason in the Lava implementation we switched to the standard Unix DBM interface, in particular using the GDBM library by default, which in addition to the already used operations provides atomic operations.

[4] Note that in a classical calculus, it would be sufficient to prove such an existential statement by proving it is impossible for a witness not to exist.

[5] https://hydra.iohk.io/build/14133599/download/1/plutus-core-specification.pdf

4.2 Cryptography Layer

Harrison has provided an efficient library[6] of field operations in the various cryptographic fields verified in the HOL Light theorem prover [8]. The library includes the Elliptic curve used by Bitcoin and Proofgold along with a number of other elliptic curves and operations provided for them [5]. In the Lava implementation we switched from the OCaml implementation of the cryptographic primitives to instead allow a low level efficient implementation. We provide the flexibility of switching between two implementations. First, we allow the use of the Bitcoin crypto implementation. It has been tested in Bitcoin and other cryptocurrencies, so it is likely to be correct. However, we also allow the use of the formally verified version (where the verified operations are the addition, multiplication, or inverse modulo in the field, but the verification of the actual additions and multiplication of points on the curve is still future work).

In addition to the much more efficient encryption and signing, we also switched to a low-level implementation of SHA256 used for hashing, including the recursive hashing of all sub-structures used in the Merkle tree. That last operation is used quite often, as all subterms used in proof terms are serialized hashed this way.

4.3 Networking and Proofchecking Layers

The Lava client also includes a number of smaller improvements to the networking layer and to proof checking. We have decided not to change the actual communication protocol between the nodes or the limits used in the proof checking, but rather to improve the implementation. In particular, we have reduced the complexity of preparing block deltas, improved the efficiency of serialization, and replaced the implementation of the checker by a more efficient. More efficient checking for the variant of simple type theory used in Proofgold, including perfect term sharing and preserving a number of invariants ($\beta\eta$-normal forms, negation normalization, etc) is discussed elsewhere [2].

5 A HOL4 Interface for Mining Bounties from the HF theory

HOL4 [17] is an interactive theorem prover (ITP) for higher-order logic (HOL) that helps users to produce formal proofs and thus verify theorems. We are developing a HOL4 interface to Proofgold for two reasons. The first one is to enable people familiar with the HOL4 system to check and share their proofs in Proofgold. This way, HOL4 users would benefit from the additional features provided by Proofgold such as authorship recognition and the bounty system. The second one, which is the focus of this section, is to provide a way to manually or automatically prove bounties in HF. For this task, we chose HOL4 because it is equipped with powerful automation. The source code of this interface can be downloaded at http://grid01.ciirc.cvut.cz/~thibault/h4pfg.tar.gz.

5.1 Importing the HF theory into HOL4

We import the 6 axioms and 97 definitions of the HF theory into HOL4. A translation between the two systems is straightforward since the logics of HOL4 and HF are similar and in particular the formula structures are almost identical. When reading a HF statement, the logical constants of the HF theory in Proofgold (e.g $\wedge, \vee, \forall, \rightarrow, \ldots$) are mapped to their

[6] https://github.com/awslabs/s2n-bignum

$$\dfrac{\dfrac{\dfrac{\dfrac{\text{Definition for } \subseteq}{\vdash (a \subseteq b) \leftrightarrow (\forall y. y \in a \to y \in b)}}{\vdash (t_0 \subseteq t_0) \leftrightarrow (\forall y. y \in t_0 \to y \in t_0)} \quad \dfrac{y \in t_0 \vdash y \in t_0}{\forall y. \vdash y \in t_0 \to y \in t_0}}{\vdash t_0 \subseteq t_0} \quad \dfrac{\dfrac{\dfrac{}{\vdash x_1 = x_1}}{\vdash q t_0 x_1 \to x_1 = x_1}}{\vdash \forall x_1. q t_0 x_1 \to x_1 = x_1}}{\dfrac{\vdash (t_0 \subseteq t_0) \wedge (\forall x_1. q t_0 x_1 \to x_1 = x_1)}{\vdash \exists x_0. (x_0 \subseteq t_0) \wedge (\forall x_1. q x_0 x_1 \to x_1 = x_1)}}$$

Figure 2 A HOL4 Proof of a HF Bounty

HOL4 native versions. For other HF constants (e.g. $\in, \subset$, *exactly5*, ...), new HOL4 constants are created. The same process is used to import HF bounties into HOL4.

5.2 Exporting HOL4 proofs to HF

To verify theorems proved in HOL4 with Proofgold, we first need to derive the HOL4 kernel rules from the IHOL rules and HF axioms. For instance, the HOL4 reflexivity rule can be derived from the IHOL rules in the following way:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{ptt \vdash ptt}}{\vdash ptt \to ptt}}{\vdash \forall p. ptt \to ptt}}{\vdash t = t}$$

Every HOL4 theorem is proved by composing applications of the HOL4 kernel inference rules. Therefore, to produce a HF proof, we trace these applications during the proof process and substitute them by their corresponding derivations in HF.

5.3 Proving Bounties

To reward the first users, a finite set of automatically generated bounties was included at the beginning of the Proofgold blockchain by the developers. The newer bounties proposed by developers and users are now usually based on textbook mathematical knowledge (often from interactive theorem provers) and are considerably harder than the automatically generated ones (see Section 6). We now show how to prove, using the HOL4 interface, some of the first "easy" bounties manually and automatically.

5.3.1 Manual Proof

The following auto-generated bounty has a relatively easy proof and therefore is one of the first we could manually prove:

$$\exists x_0. x_0 \subseteq t_0 \wedge \forall x_1. (\forall x_2. x_2 \subseteq x_1 \to \forall x_3 x_4. (\neg c_0 x_3 x_4 \wedge\ c_1 x_0 \wedge \neg c_2 x_2) \to c_3(c_4(c_5 x_0)) x_4) \to x_1 = x_1$$
where $t_0 = \wp(\wp(\wp(\wp\emptyset))))$ and $[c0, c1, c2, c3, c4, c5] = [\mathit{tuple}, \mathit{exactly5}, \mathit{atleast2}, \mathit{SNo}, \mathit{Sing}, \mathit{SNoLev}]$

The main difficulty, when manually proving such an automatically generated bounty, is to identify the relevant part of the formula. After a careful analysis, we found that the truth of this formula can be derived from this abbreviated version $\exists x_0.(x_0 \subseteq t_0) \wedge (\forall x_1. q x_0 x_1 \to x_1 = x_1)$ where the predicate q is used to hide the irrelevant part. Our proof, shown in Figure 2, relies on the imported definition of $\subseteq$.

5.3.2 Automated Proof

In general, proof automation tools help speed up formalization of theorems in interactive theorem provers. As a demonstration of the possible benefits, we have developed a way to automatically prove HF bounties by relying on the automation available in HOL4.

To prove a bounty, we first call HOL(y)Hammer [6] which is one of the strongest general automation techniques available in HOL4. It tries to prove the conjectured bounty from the 6 HF axioms and the 97 HF definitions by translating the problem to external automated theorem provers (ATPs). When an external ATP finds a proof, it also returns the axioms that are necessary to find that proof. With this information, a weaker internal prover such as Metis [10] is usually able to reconstruct a HOL4 proof. The Metis proofs however typically exceed the Proofgold block size limit of 500kb and include dependencies to HOL4 axioms that are not present (and sometimes not provable) in HF. Thus, we have developed a custom internal first-order ATP for HOL4 that produces small proofs and only relies on the HF axioms. A reduction in proof size is achieved by making definitions for large terms (e.g. irrelevant parts of the conjecture and Skolem functions, similar to the example given in Section 5.3.1) and proving auxiliary lemmas for repeated sequences of proof steps (e.g., when permuting literals in clauses). With these optimizations, the automated proof for the bounty from Section 5.3.1 is only four times as large as the manual one (16kb instead of 4kb). The manual proof for this bounty has been submitted and included in the blockchain and the bounty associated with it has been collected. In addition to that, we have so far automatically found and submitted six proofs of the HF bounties. All these proofs were accepted by the Proofgold proof checker and the rewards for these bounties were collected.

This automated system is currently limited to essentially first-order formulas. In the future, we plan to support automated proofs for higher-order formulas based on existing automated translations to first-order [12].

6 The Bounty System and its Applications

One of the main extra features of Proofgold beyond proof verification is the possibility of for users and developers to attach bounties to propositions. Bounties can be used to reward users for finding proofs in mathematical domains of general interest or subproofs of a larger formalization.

6.1 Current Bounties

As mentioned in Section 3.1 for the first 5000 blocks the Proofgold consensus algorithm automatically placed a bounty of 25 Proofgold bars (half of the block reward) on a pseudorandom proposition. We say more about these pseudorandom propositions below. For the next 10000 blocks 25 Proofgold bars (half of the block reward) were placed into a “bounty fund” which was used to place larger bounties on meaningful propositions decided upon through a community forum. The propositions chosen vary from first-order problems derived from Mizar proofs, finite Ramsey properties (e.g., $R(5,7)$ is larger than the cardinality of $\wp 5$), properties of specific categories (e.g., the category of hereditarily finite sets), and numerous others. Since Block 15000 the full block reward is 25 bars and none of this goes towards the creation of bounties, and so bounties are placed by intention rather than automation.

The pseudorandom propositions from the first 5000 blocks can be classified into 8 classes.

Random

Conjectures in this class are generally not meaningful, but the choices made during the generation are also not uniformly random. The conjecture must start with at least two (possibly bounded) quantifiers. When a term of type ι must be generated and a bound variable is not being chosen, then half the time the binary representation of a number between 5 and 20 is used, a quarter of the time the unary representation of a number between 5 and 20 is used. In the remaining quarter of the cases, half the time a unary function is chosen (leaving the argument to be generated), a quarter of the term a binary function is chosen (leaving two arguments to be generated) and the remaining quarter some other set former is used (e.g., Sep). In case the generation seems to be running out of bits of information, then it restricts the choices available.

There are three subclasses of random conjectures. The first kind is simply a sentence constructed as roughly described above. The second kind is of the form $\forall p : \iota o. \forall f : \iota\iota.s$ where s is generated as above but is allowed to use the (uninterpreted) unary predicate p and unary function f. The third kind is of the form $\forall xyz.\forall f : \iota\iota.\forall pq : \iota o.\forall g : \iota\iota\iota.\forall r : \iota\iota o.s$ where s is a generated as above though it is allowed to use x, y, z, f, g to construct sets, to use p, q, r to construct atomic propositions and is (mostly) disallowed from using the constants from the HF set theory.

The automated miner from Section 5 was tested on problems from this family.

Quantified boolean formulas (QBF)

Conjectures in the QBF class are of the form $Q_1 p_1 : o. \cdots .Q_n p_n : o.s \leftrightarrow t$ where $50 \leq n \leq 55$, each Q_i is $\forall$ or $\exists$ and s and t are propositions such that $\mathcal{F}(s) = \mathcal{F}(t) = \{p_1, \ldots, p_n\}$. The propositions s and t are generated using a similar process.

Set Constraints

One of the most challenging aspects of higher-order theorem proving is instantiating set variables, i.e., variables of a type like ιo [1]. The only known complete procedure requires enumeration of $\beta\eta$-normal terms of this type.

The set constraint conjectures are of the form

$$\forall P_1 : \alpha_1.\forall P_2 : \alpha_2.\forall P_3 : \alpha_3.\forall P_4 : \alpha_4.\varphi^1_1 \to \varphi^1_2 \to \varphi^2_3 \to \varphi^2_4 \to \varphi^3_5 \to \varphi^3_6 \to \varphi^4_7 \to \varphi^4_8 \to \bot$$

where each α_i is a small type of the form $\beta_1 \cdots \beta_{m_i} o$ and each proposition φ^i_j is a lower bound constraint for P_i over $\{P_1, P_2, P_3, P_4\}$ if j is odd and an upper bound constraint for P_i over $\{P_1, P_2, P_3, P_4\}$ if j is even. A lower bound constraint for a variable P is a formula that implies P must at least be true for certain elements. An upper bound constraint for a variable P is a formula that implies P cannot be true for more than some number of elements. Such constraints may also be recursive, e.g., saying if $P\ z$ holds then $P\ (f\ z)$ must hold. Recursive constraints can in principle be both lower bound and upper bound constraints.

The positive version of the conjecture states that there is no solution to this collection of set constraints. The negative version can be proven by giving a solution.

Higher-Order Unification

Unlike first-order unification, higher-order is undecidable. In spite of this Huet's preunification algorithm [9] provides a reasonable method to search for solutions. A great deal of research has been done on higher-order unification and is ongoing today [19].

The generated conjectures in this class are essentially higher-order unification problems with eight flex-rigid pairs and four variables to instantiate. The problems are given in a universal form, so that the positive form states that there is no solution. The negative form could be proven by giving a solution. In general the conjectures have the form

$$\forall X_1:\alpha_1.\forall X_2:\alpha_2.\forall X_3:\alpha_3.\forall X_4:\alpha_4.\varphi^1_1\to\varphi^1_2\to\varphi^2_3\to\varphi^2_4\to\varphi^3_5\to\varphi^3_6\to\varphi^4_7\to\varphi^4_8\to\bot$$

where α_i is a small type not involving o and φ^i_j is a proposition corresponding to a disagreement pair of a unification problem.

Untyped Combinator Unification

Since we are in a simply typed setting the untyped combinators are encoded as sets. The generated conjectures are in the form of eight flex-rigid pairs making using four variables to be instantiated. Each conjecture is stated in a universal form that means there is no solution. Proving the negation of the conjecture will usually mean giving a solution, though given the classical setting it is also possible to provide multiple instantiations and prove one must be a solution. (This was also the case for the previous two classes of conjectures.) The conjectures have the form

$$\forall X.\mathsf{combinator}\ X\to\forall Y.\mathsf{combinator}\ Y\to\forall Z.\mathsf{combinator}\ Z\to\forall W.\mathsf{combinator}\ W\to$$
$$\varphi^X_1\to\varphi^X_2\to\varphi^Y_3\to\varphi^Y_4\to\varphi^Z_5\to\varphi^Z_6\to\varphi^W_7\to\varphi^W_8\to\bot$$

where φ^V_i is a proposition giving a flex-rigid pair with local variables and with V as the head of the left. To be more specific each φ^V_i has the form

$$\forall x.\mathsf{combinator}\ x\to\forall y.\mathsf{combinator}\ y\to\forall z.\mathsf{combinator}\ z\to\forall w.\mathsf{combinator}\ w\to$$
$$\mathsf{combinator\_equiv}\ (V\ v_1\ v_2\ v_3\ v_4\ s_1\ \ldots\ s_n)\ t$$

where each $v_i\in\{x,y,z,w\}$, t is a random rigid combinator and each of $s_1,\ldots,s_n$ is a random combinator. In this context a random rigid combinator is either $K\ t_1$ or $S\ t_1$ where t_1 is a random combinator, or $S\ t_1\ t_2$ where t_1 and t_2 are random combinators, or $v\ t_1\ \cdots\ t_n$ where $v\in\{x,y,z,w\}$ and $t_1,\ldots,t_n$ are random combinators. A random combinator is $h\ t_1\ \cdots\ t_n$ where $h\in\{S,K,X,Y,Z,W,x,y,z,w\}$ and $t_1,\ldots,t_n$ are random combinators.

Each of these problems can be viewed as a first-order problem. In the first-order variant we could assume everything is a combinator (so combinator can be omitted) and use equality to play the role of combinator_equiv. It should generally be possible to mimic the equational reasoning of a first-order proof in the set theory representation by using appropriate lemmas about combinator and combinator_equiv.

Furthermore it should be possible to define a notion of reduction and prove that if two terms are equivalent via combinator_equiv, then they must have a common reduct. This would allow one to prove the positive version of the conjecture (meaning there is no solution).

Abstract HF problems

The conjectures in the Abstract HF class are about hereditarily finite sets, but without assuming the full properties about the relevant relations, sets and functions. We fix 24 distinct variables: r_0, r_1 and r_2 of type $\iota\iota o$, x_0, x_1, x_2, x_3 and x_4 of type ι, f_0 and f_1 of type $\iota\iota$, g_0, g_1 and g_2 of type $\iota\iota\iota$ and p_0, p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9 and p_{10} of type ιo. Each of these variable has an intended meaning which can be given by a substitution θ. For example, $\theta(r_0) = \in$, meaning r_0 is intended to correspond to set membership. Each generated conjecture is of the form

$$\forall r_0 r_1 r_2 : \iota\iota o.\forall x_0 x_1 x_2 x_3 x_4.\forall f_0 f_1 : \iota\iota.\forall g_0 g_1 g_2 : \iota\iota\iota.\forall p_0 \cdots p_{10} : \iota o.$$
$$\varphi_1 \to \cdots \to \varphi_n \to \psi.$$

The propositions $\varphi_1, \ldots, \varphi_n, \psi$ are chosen from a set of 1229 specific propositions which hold for HF sets, but may not hold in the abstract case. The conjecture essential states that the selections of φ_i are sufficient to infer the selected ψ.

AIM Conjecture Problems

There are two kinds of AIM Conjecture [11] related problems: one using Loop_with_defs_cex1 and one using Loop_with_defs_cex2. In both cases the conjecture states that no loop exists with counterexamples of the first or second kind satisfying a number of extra equations. The two kinds of counterexamples assert that the loop has elements violating one of two identities. An AIM loop violating either of the identities would be a counterexample to the AIM Conjecture. The pseudorandom propositions do not assume the loop is AIM, but only assume some AIM-like identities hold. That is, instead of assuming all inner mappings commute, the assumption is that some inner mappings commute. Furthermore, in some cases some specific inner mappings are assumed to have a small order (which would not be true in all AIM loops).

Unfortunately there was a bug in the HF defining equation for loops (omitting that the identity element must be in the carrier). This made the negation of all of the pseudorandom propositions in this class easily provable. A Proofgold developer used this bug to collect the bounties and redistribute the bounties to the corrected versions.

Diophantine Modulo

A Diophantine Modulo problem generates two polynomials p and q in variables x, y and z and a number m (of up to 64 bits). The conjecture states there is no choice of (hereditarily finite) sets x, y and z such that the cardinality of p plus 16 is the same as the cardinality of q modulo m. The negation of the conjecture could be proven by giving appropriate x, y and z and proving they have the property.

Diophantine

The final class is given by Diophantine problems (either equations or inequalities). Two polynomials p and q in variables x, y, z are generated (as described above). Each polynomial uses 256 bits of information. The generated conjecture either states there are no (hereditarily finite) sets x, y and z such that the cardinality of p plus 16 is the same as the cardinality of q, or that the cardinality of p plus 16 is no larger than the cardinality of q.

6.2 Large Formalization Projects

Hales's Flyspeck [7] project formalizing the proof of the Kepler Conjecture has been one of the largest challenges in interactive theorem proving so far, involving several ITP communities and to some extent a centralized bounty system. It took more than 10 years to complete and combined the expertise of proof assistant users of the HOL Light, Isabelle/HOL and Coq systems. With our bounty system, the effort could have been shared with an even wider community of researchers interested in formal verification. Indeed, Hales could have put This would involve making a plan of the steps required to prove the final theorem, splitting the formalization into multiple independent parts, and putting them as conjectures into Proofgold with bounties on them. A knowledgeable independent user of an interactive theorem prover interface capable of producing Proofgold terms, could then decide to provide a proof for a particular part. The final proof is completed when all the bounties have been collected. The reward for a particular proof may be increased if it is harder than initially thought and/or to motivate Proofgold users to solve it sooner. In the long run, an attempt at formally proving Fermat's last theorem in Proofgold could be made using this approach. An even better target to test the effectiveness of the bounty system would be the classification of finite simple groups. Its proof required the combined effort of about 100 authors for 50 years and consists of tens of thousands of pages distributed over several hundred journal articles.

References

1 Chad E. Brown. Solving for set variables in higher-order theorem proving. In Andrei Voronkov, editor, *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings*, volume 2392 of *Lecture Notes in Computer Science*, pages 408–422. Springer, 2002.

2 Chad E. Brown, Mikoláš Janota, and Cezary Kaliszyk. Proofs for higher-order SMT and beyond. http://cl-informatik.uibk.ac.at/cek/submitted/smt2022pfs.pdf.

3 Chad E. Brown and Karol Pąk. A tale of two set theories. In Cezary Kaliszyk, Edwin C. Brady, Andrea Kohlhase, and Claudio Sacerdoti Coen, editors, *Intelligent Computer Mathematics - 12th International Conference, CICM 2019, Prague, Czech Republic, July 8-12, 2019, Proceedings*, volume 11617 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2019.

4 John H. Conway. *On numbers and games, Second Edition*. A K Peters, 2001.

5 Warren E. Ferguson, Jesse Bingham, Levent Erkök, John R. Harrison, and Joe Leslie-Hurd. Digit serial methods with applications to division and square root. *IEEE Trans. Computers*, 67(3):449–456, 2018. URL: https://doi.org/10.1109/TC.2017.2759764, doi:10.1109/TC.2017.2759764.

6 Thibault Gauthier and Cezary Kaliszyk. Premise selection and external provers for HOL4. In Xavier Leroy and Alwen Tiu, editors, *Conference on Certified Programs and Proofs (CPP)*, pages 49–57. ACM, 2015. URL: http://doi.org/10.1145/2676724.2693173.

7 Thomas C. Hales, Mark Adams, Gertrud Bauer, Dat Tat Dang, John Harrison, Truong Le Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Thang Tat Nguyen, Truong Quang Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason M. Rute, Alexey Solovyev, An Hoai Thi Ta, Trung Nam Tran, Diep Thi Trieu, Josef Urban, Ky Khac Vu, and Roland Zumkeller. A formal proof of the kepler conjecture. *CoRR*, abs/1501.02155, 2015. URL: http://arxiv.org/abs/1501.02155, arXiv:1501.02155.

8 John Harrison. HOL light: A tutorial introduction. In Mandayam Srivas and Albert Camilleri, editors, *Proceedings of the First International Conference on Formal Methods in Computer-Aided Design (FMCAD'96)*, volume 1166 of *Lecture Notes in Computer Science*, pages 265–269. Springer-Verlag, 1996.

9 Gérard P. Huet. A unification algorithm for typed lambda-calculus. *Theor. Comput. Sci.*, 1(1):27–57, 1975.

10 Joe Hurd. First-order proof tactics in higher-order logic theorem provers. *Design and Application of Strategies/Tactics in Higher Order Logics, number NASA/CP-2003-212448 in NASA Technical Reports*, pages 56–68, 2003.

11 Michael K. Kinyon, Robert Veroff, and Petr Vojtěchovský. Loops with abelian inner mapping groups: An application of automated deduction. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *Lecture Notes in Computer Science*, pages 151–164. Springer, 2013.

12 Jia Meng and Lawrence C. Paulson. Translating higher-order clauses to first-order clauses. *Journal of Automated Reasoning*, 40(1):35–60, 2008. URL: http://dx.doi.org/10.1007/s10817-007-9085-y.

13 F. Pfenning and C. Elliot. Higher-order abstract syntax. *SIGPLAN Notices*, 23(7):199–208, June 1988.

14 Dag Prawitz. *Natural deduction: a proof-theoretical study*. Dover, 2006.

15 R. Diaconescu. Axiom of choice and complementation. *Proceedings of the American Mathematical Society*, 51:176–178, 1975.

16 Bertrand Russell. *The Principles of Mathematics*. Cambridge University Press, 1903.

17 Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Conference on Theorem Proving in Higher Order Logics (TPHOLs)*, volume 5170 of *LNCS*, pages 28–32. Springer, 2008. URL: http://dx.doi.org/10.1007/978-3-540-71067-7_6.

18 M.H.B. Sørensen and P. Urzyczyn. *Lectures on the Curry-Howard Isomorphism*. Rapport (Københavns universitet. Datalogisk institut). Datalogisk Institut, Københavns Universitet, 1998.

19 Petar Vukmirovic, Alexander Bentkamp, and Visa Nummelin. Efficient full higher-order unification. In Zena M. Ariola, editor, *5th International Conference on Formal Structures for Computation and Deduction, FSCD 2020, June 29-July 6, 2020, Paris, France (Virtual Conference)*, volume 167 of *LIPIcs*, pages 5:1–5:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

20 Bill White. Qeditas: A formal library as a bitcoin spin-off, 2016. URL: http://qeditas.org/docs/qeditas.pdf.