Report on Trustworthy Proofs and Programs

Daniel Murfet

December 29, 2018

Abstract

This report addresses technical aspects of the problem of *trust* in mathematics and software engineering, that is, in the human cultural activities of constructing proofs and programs. We survey the use of classifying topoi to organise mathematical knowledge and software, how that organisation could potentially be reflected in computational tools like proof assistants, and how the widespread use of such tools could be incentivised by conjecture markets.

Contents

> Civilisation advances by extending the number of important operations which we can perform without thinking about them.
>
> Alfred North Whitehead

The economist Kenneth Arrow has argued that trust serves as a catalyst of economic activity, increasing efficiency and saving on transaction costs [1, p.23], [14] and that trust is therefore a primary source of economic wealth:

> Now trust has a very important pragmatic value, if nothing else. Trust is an important lubricant of a social system. It is extremely efficient; it saves a lot of trouble to have a fair degree of reliance on other people's word. Unfortunately

> this is not a commodity which can be bought very easily. If you have to buy it, you already have some doubts about what you've bought. Trust and similar values, loyalty or truth-telling, are examples of what the economist would call "externalities." They are goods, they are commodities; they have real, practical, economic value; they increase the efficiency of the system, enable you to produce more goods or more of whatever values you hold in high esteem. But they are not commodities for which trade on the open market is technically possible or even meaningful.

Trust is indispensable in professional mathematics, where the content of our books and papers are typically informal proofs, with an implicit assertion that the well-educated and sufficiently motivated reader could transform the informal argument into a formal proof. Today this transformation is rarely performed in practice, because it is time consuming, not economically incentivised, and perhaps not very interesting. The cultural practice of mathematics depends, to various degrees, on networks of trust.[1] The edifice of modern mathematics attests to the fact that this cultural practice has been enormously successful; however, these trust networks do sometimes break down, with recent examples in number theory [25] and symplectic geometry [20]. Such breakdowns cause transaction costs to increase and efficiency to decrease: individuals must expend additional effort to determine whether or not to rely on the claims of a given paper, and may even switch fields at high personal cost. Some have argued that the increasing complexity of mathematics, and the specialisation of its practitioners, is outstripping the capacity of our current culture [40].

There same problem afflicts software engineering, which is of greater concern to broader society because of our increasing economic and military dependence on software systems. The purpose of software engineering is to construct programs which achieve a given goal (e.g. transfer money between banks) and do no more than that (e.g. the program does not make payments to corrupt bank managers). Today these goals or *specifications* are rarely written down in formal language, and it is even more rare that a program is proven to fulfill the specification. While software engineering has produced enormous economic value, many have argued that the increasing complexity of software systems, and our dependence as societies on these systems for our wealth and security, requires that we must transition to a new culture with a higher standard of specification and proof [22].

Problem statement. Imagine a future where the communities of professional mathematicians and software engineers have successfully made the transition to a new culture of this kind. What does the path from here to there look like? It seems to involve

A. Developing *technical tools* which enable higher levels of trust,

B. Embedding those technical tools into a *new culture*, and

[1] Most mathematicians have verified parts of mathematics, but the coverage becomes less complete as one moves towards the frontiers of their research. A relevant joke among algebraic geometers is "we don't need formal methods or proof assistants, because we have Ofer Gabber."

C. Transitioning a complex system of cooperating agents to this new culture.

For example, computer proof assistants are technical tools that can assist mathematicians in formalising their proofs, and software engineers in writing specifications and verifying that their programs fulfill the specification (Problem A). While there are passionate communities within mathematics and software engineering that use, and indeed evangelise, the use of such tools (Problem B) these subcultures are far from dominant and it is not clear that enculturation will succeed at current rates (Problem C). This document is the report on a scoping study[2] which aimed to identify promising new approaches to various circumscribed parts of these problems.

Aims. The goal of the study was to lay some of the groundwork for the development of a *formal mathematical modelling environment capable of addressing large scale bodies of mathematical, engineering and systems knowledge.* The component of the study addressed by this report focussed on the analysis of areas of mathematics and logic which could be used as the basis for a next generation of formal methods tool, supporting modularisation, composition and re-use through algebraic and categorical techniques. The main purpose of these features is to encourage wider adoption of formal methods. One of the concrete goals was to scope the application of type theory to the development of a *higher-order monadic computation model* to address the defects in current proof assistants.

Outline. The activities of the group of mathematicians at the University of Melbourne was organised around a series of seminars [31] in which the PI Daniel Murfet and other faculty and masters students, including William Troiani and James Clift, worked through references on categorical logic and topos theory including [24, 27, 23]. Lecture notes are posted online, and videos of the talks are publicly available on YouTube. A brief summary of each seminar, and the conclusions of the scoping study, are collected in Section 1.

While these investigations are reasonably abstract, they are aimed at solving the concrete social problems elaborated above. To buttress this claim we include in Section 3 a speculative discussion of potential solutions to Problems B,C organised around computational networks that we call *proofchains* which support *conjecture markets*. We hope with this proposal to encourage the development of networks which can help to accelerate the spread of formal methods by creating new economic incentives.

1 Organising mathematical knowledge using topoi

A primary means of organising mathematical knowledge is the notion of a *theory*. Each theory provides a language and a set of axioms satisfied by the objects described by that theory: examples include abelian groups, rings, and metric spaces. A concrete instance of this structure is called a *model* of the theory. The organisation of software follows, to some extent, the same pattern: a directory structuring a project in the programming language

[2]The title of the scopying study is "Trustworthy Software Systems - Formal Analysis Tools Development" involving the University of Melbourne, Data61 and DSTG.

C contains *.h* files (specifications, which are similar to theories) and *.c* files (models of the theory). Since we cannot claim any deep understanding of the problems specific to software engineering we focus here on the task of organising mathematical knowledge, but it appears that the organisation of large-scale software is analogous: indeed specification languages like Z [35] make use of first-order theories and view implementations as models.

The project goals of enabling *modularisation, composition and re-use*, which have resonance with ideas in category theory, suggest that *theories and theory morphisms* should be central organising principles in the development of a next generation formal methods tool of the desired kind. By a *theory* we will mean a first-order theory [27, §X.1] unless specified otherwise. The first problem is that it is not clear what a *transformation* or *morphism* of theories should be [31, Lecture 1]. Whatever it is, a morphism of theories

$$\phi : T \longrightarrow T'$$

should consist of an interpretation of the sorts, function symbols, relations and constants of T in the "universe of mathematical discourse" provided by T'. From a practical point of view, such a morphism ϕ would tell a proof assistant *how* to import the symbols of T in an environment already populated by the symbols of T'.

The most straightforward approach would be to define a morphism ϕ to be a mapping of the sorts $X, Y, Z, \ldots$ of T to sorts $\phi(X), \phi(Y), \phi(Z), \ldots$ of T', and a mapping of the remaining symbols of T to symbols of T' of the same kind, compatible with the typings. However this notion is too restricted to be useful, because the sorts of a first-order theory are not necessarily closed under type formation operations. It is more natural to allow $\phi(X)$ to be constructed from the sorts of T' using the allowed operations, and to similarly relax the interpretation of the other kinds of symbols in the theory T.

This suggests a refined guess, namely that ϕ should be the data of a model of T in the category referred to in [27] as the *syntactic category* $\mathcal{S}(T')$ where the objects and morphisms are equivalence classes of formulas in the language of T' (one thinks of a formula ψ with free variables $x_1, \ldots, x_n$ as a set comprehension $\{(x_1, \ldots, x_n) \,|\, \psi\} \subseteq X_1 \times \cdots \times X_n$). However this notion still has at least two problems:

- $\mathcal{S}(T')$ is not a topos, so it is not necessarily clear what we mean by a *model* of T, for example, how do we determine the validity of an axiom of the theory T for our model when we don't know how to talk about quantifiers?
- $\mathcal{S}(T')$ lacks arbitrary colimits, so that we are restricted to interpreting a sort X as a *finite* construction from the sorts and other data of T'.

From the computational point of view the first problem is the more serious one. In any case, provided the theory T' is *geometric* (see Section 1.2 for the definition) there is a natural way to resolve both of these problems by passing from $\mathcal{S}(T')$ to a larger category $\mathcal{B}(T')$ which is both a topos and has arbitrary colimits: this is the *classifying topos* of the geometric theory T'.[3]

[3]Technically the classifying topos is defined as a category of sheaves on a site $(\mathcal{S}(T'), J)$ where J is

So a reasonable notion of a morphism $\phi : T \longrightarrow T'$ of theories is the data of a model

$$M \in \mathrm{Mod}(T, \mathcal{B}(T')) .$$

The correctness of this notion is reinforced by the fact that, provided T, T' are geometric theories, the universal property of the classifying topos gives an equivalence

$$\mathrm{Mod}(T, \mathcal{B}(T')) \cong \underline{Hom}(\mathcal{B}(T'), \mathcal{B}(T))$$

where the right hand side is a category whose objects are the *geometric morphisms* of topoi $\mathcal{B}(T) \longrightarrow \mathcal{B}(T')$. A geometric morphism of topoi $f : \mathcal{E} \longrightarrow \mathcal{F}$ is an adjoint pair (f^*, f_*) where the left adjoint part $f^* : \mathcal{F} \longrightarrow \mathcal{E}$ preserves finite limits.[4] So at least for geometric theories, there appears to be a satisfactory answer to our original question: the correction notion of a morphism of theories $T \longrightarrow T'$ is a geometric morphism of topoi $\mathcal{B}(T') \longrightarrow \mathcal{B}(T)$ or what is the same, a model of T in $\mathcal{B}(T')$.

Thus the theory of classifying topoi and geometric morphisms seems to point towards an attractive categorical way of organising mathematical knowledge, and potentially also large-scale software systems. However, several questions need to be answered before this solution can be employed in practice:

- **Q1.** How serious is the restriction to geometric theories?
- **Q2.** Can you compute what a geometric morphism f^* actually does? More precisely, can it be implemented on a computer, say in a proof assistant?
- **Q3.** Geometric morphisms are adjoint pairs, and adjunctions are closely related to monads. The influence of monads on computer programming in recent decades has been profound [2] following Moggi's fundamental contribution [30]. What is the relation of this proposal to Moggi's monadic languages?

The first question was not addressed directly in the seminars, but we argue in Section 1.2 that the answer is *Maybe It's Not So Bad*. The second two points were addressed by talks in the seminar. In short, the answer to Q2 seems to be *Yes* although further work is required to flesh this out in detail, see [31, Lectures 12,15]. The answer to Q3 is that the information in the relevant adjunctions *cannot* be reduced to monads, so what we are proposing is a genuine extension of the old ideas of Moggi [31, Lecture 15].

The rest of this section is structured as follows: we first give a summary of the seminar and then turn to how it addresses the two points above, before making our conclusions.

a Grothendieck topology. An introduction to sheaves and Grothendieck topologies is given in Lectures 6,7,10,11 of the seminar.

[4] These are connected to geometry by the following basic fact: any continuous function between topological spaces $f : X \longrightarrow Y$ gives rise to a geometric morphism $(f^*, f_*) : \mathrm{Sh}(X) \longrightarrow \mathrm{Sh}(Y)$ of sheaf topoi, and provided Y is Hausdorff every geometric morphism arises in this way [27, §VII.1].

1.1 Summary of seminar talks

- **Lecture 1: Daniel Murfet, "An invitation to topos theory".** Introduces the general ideas of categorical logic and topos theory as a means of organising mathematical knowledge, following Lawvere. Briefly defines topoi, gives some examples, introduces the universal property of a classifying topos.

- **Lecture 2: Daniel Murfet, "The Curry-Howard correspondence".** Introduces some of the general ideas surrounding mathematical logic and the theory of computation, in particular the distinction between functions and algorithms, and makes this concrete by introducing Church's simply-typed λ-calculus and its categorical incarnation following Lambek and Scott's [24]. The statement of the Curry-Howard correspondence is given, connecting formulas and proofs in intuitionistic logic to types and terms in λ-calculus, and objects and morphisms in Cartesian closed categories. *This material is necessary to explain the relevance of topoi to software engineering and computation more broadly.*

- **Lecture 3: William Troiani, "Monads and programs".** It is explained how the simply-typed lambda calculus is not very expressive, and the various methods of increasing expressivity and usability that have been explored in the literature, including Moggi's ideas on how to enable reasoning about programs using monads. A formal introduction to categories, functions and monads is provided, along with the proof of Moggi's theorem that there is a bijection between Kleisli triples and monads. *This material is necessary to provide context for the answer, in Lecture 15, of Question 3 above.*

- **Lectures 4, 5: James Clift, "The definition of a topos".** The definition of limits and colimits are given with some examples. The rest of the seminar contains the definition of exponentials in a category and Cartesian closed categories, subobject classifiers and examples, and finally the definition of a topos as a Cartesian closed category which has all finite limits and a subobject classifier.

- **Lectures 6, 7: Patrick Elliott, "Sheaves of sets".** Contains the definition of presheaves on a topological spaces, the Yoneda lemma, sieves and their intepretation as subfunctors, sheaves from various points of view, Grothendieck topologies and the example of the Zariski site. *This material is necessary to state the definition of a classifying topos.*

- **Lecture 8: William Troiani, "Higher-order logic and topoi (Part 1)".** The alternative definition of an *elementary* topos is given, and the representation of predicates in a topos (a foreshadowing of the Mitchell-Benabou language) is developed in order to give the proof that a category is a topos if and only if it is an elementary topos. *This material is necessary to make the connection between higher-order logics and topoi in Lecture 9.*

- **Lecture 9: Daniel Murfet, "Higher-order logic and topoi (Part 2)".** This talk makes the connection between higher-order logics and topoi by constructing an instance of the latter from an instance of the former, following Lambek and Scott. Gives a detailed definition of an intuitionistic type theory / higher-order logic and the associated topos of types and terms. *In a proof assistant adapted to topoi, one would be primarily concerned with two kinds of examples: classifying topoi constructed from first-order theories, and topoi constructed as in this lecture from higher-order logics.*

- **Lectures 10, 11: Patrick Elliott, "Sheaves form a topos".** Proves in detail that the category of presheaves on a small category is a topos, discusses sheafification, and then defines the exponentials and subobject classifier in a category of sheaves on a site and proves that with these structures sheaves form a topos.

- **Lecture 12: Daniel Murfet, "Classifying topoi (Part 1)".** Gives the definition of the classifying topos of a first-order theory, introduces geometric morphisms and then proceeds to develop various points of view on additive and "nonadditive" tensor products as a way of understanding what the inverse image part of a geometric morphism is doing. The primary example is geometric realisation of a simplicial set.

- **Lecture 13: James Clift, "Higher-order logic and topoi (Part 3)".** Explains quantifiers as adjoints, an important part of understanding how to interpret higher-order logic in a topos, and thus in defining models of a theory. Defines the Mitchell-Benabou language.

- **Lecture 14: William Troiani, "The classifying space of rings".** Examines an important example of a first-order theory (the theory of rings) and the construction of its classifying topos.

- **Lecture 15: Daniel Murfet, "Abstraction and adjunction".** Returns to the original goal of using topoi and geometric morphisms to organise mathematical knowledge, and formalises this in terms of the language that has been developed over the course of the seminar. The running examples are two first-order theories (abelian groups, and torsion abelian groups) and a higher-order logic, and various functors between the three topoi associated to this data. This example is used to illustrate the distinction between a purely *monadic* approach to formalising mathematics, and an approach based on the more general notion of adjunctions.

1.2 Geometric theories

A formula ϕ in a first-order theory language is *geometric* if it can be obtained from atomic formulas by conjunction, disjunction and existential quantification; the name originates from the fact that geometric morphisms $f : \mathcal{E} \longrightarrow \mathcal{F}$ have the property that the left

adjoint f^* is compatible with the interpretation of the "set comprehension" $\{x \mid \phi\}$ as an object of $\mathcal{E}, \mathcal{F}$ [27, Theorem X.5]. A theory T is *geometric* if all the axioms are of the form

$$\forall x_1 \ldots \forall x_n (\phi(x_1, \ldots, x_n) \Longrightarrow \psi(x_1, \ldots, x_n))$$

where ϕ, ψ are geometric. In the category theory literature, geometric formulas are sometimes called *positive existential* formulas [28, p.45] and there is a well-understood connection between geometric theories, sketches and accessible categories.[5] For the reader with a computer science background, we mention that geometric logic extends Horn clause logic, on which the programming language Prolog is based [5].

According to Johnstone "It is remarkable how few of the first-order theories encountered in the practice of mathematics fail to be (at least classically equivalent to) coherent theories" [23, §D.1.1]. All algebraic theories, such as group theory and ring theory, are geometric, as are all essentially algebraic theories, such as category theory, the theory of fields, local rings, lattice theory, projective geometry, separably closed local rings. If we allow infinite disjunctions then the infinitary theory of torsion abelian groups is geometric. However it is certainly not true that every first-order theory of mathematical interest is geometric: for example, an infinitary first-order theory which is *not* geometric is given by the theory of metric spaces [23, Example 1.1.7(l)].

Further, it is not clear that every natural software specification can be written down as a geometric theory. It is therefore an important fact that every first-order theory T has a conservative geometric extension T' [12, Theorem 7.7]. This extension is finite and can be implemented on a computer, and the models of T, T' are essentially the same (although the notion of *morphism* is changed, so the categories are not equivalent).[6] The feasibility of a proof assistant based around geometric theories hinges, therefore, on a close analysis of how cumbersome it is to deal with this geometric extension in practice.

The good news is that effective theorem-proving for geometric theories can be automated [4, 5, 6, 13, 21, 10, 36]. For recent practical applications of geometric logic based automatic theorem provers see [29, 11] which use the ArgoCLP system described in [36]. Moreover proofs in geometric logic are "more readable", according to [37]:

> Our proof representation is developed also with readable proofs in mind. Readable proofs (e.g., textbook-like proofs), are very important in mathematical practice. For mathematicians, the main goal is often, not only a trusted, but also a clear and intuitive proof. We believe that coherent logic is very well suited for automated theorem proving with a simple production of readable proofs.

[5] Note that many authors refer to such theories as *coherent* and use the term geometric for more general theories which allow infinite disjunctions. Since [27] was the main reference for our seminar, we prefer to stick to this terminology.

[6] A related construction, referred to as the *Morleyization* of T in [23, D1.5.13], produces a geometric theory T' whose category of models in **Sets** (or more generally in any Boolean coherent category) is the same as that of T. This involves adding infinitely many new symbols to the language of T, and is hardly reasonable from the point of view of automating reasoning in a proof assistant.

It remains unclear to how appropriate geometric theories are in the context of software specifications, however Vickers has argued in favour of such an approach [38, 39].

2 Conclusions and future directions

Our conclusions are as follows:

1. A next generation formal methods tool with the stated goals should be organised around a notion of **theories** and **theory morphisms**.[7]

2. One appropriate candidate are **geometric theories** and **geometric morphisms**.

3. This approach has the advantage of being based on an extensive development in the literature on categorical logic, an active ongoing effort to implement geometric logic in proof assistants [5] and compatibility with future developments in Univalent foundations.[8] The primary disadvantage of the approach is the necessity of **geometrisation** in order to accomodate arbitrary first-order theories [12].

4. A formal methods tool developed along these lines would be based on languages describing **adjunctions between geometric logics**. We suggest that the original vision of a higher-order monadic underlying language does not go far enough: every monad arises from an adjunction, but not every adjunction is monadic.[9]

In order to justify pursuing this approach, one must either find convincing reasons to be satisfied with the restriction to geometric theories, or develop the practical side of geometrisation to an extent that relying on it no longer seems to be a serious drawback. Modulo this caveat, our recommendations for future directions are:

1. Coordinate with and support the projects working to engineer proof assistants based on geometric logic [5, 36].

2. Augment these projects with research into the formalisation of geometric morphisms in proof assistants along the lines of [3], and support practical work on the implementation of the products of this research.

3. Investigate the practical implementation of geometrisation of first-order theories and the role of geometric logic in the specification of software following [38, 39].

It would also be interesting to understand more clearly how such efforts could interface with the work of various groups on Univalent foundations.

[7] See Lectures 1, 12, 15 of the seminar.

[8] Voevodsky has argued that it is not convenient to formalise areas of modern mathematics involving category theory, such as modern algebra and homotopy theory, in proof assistants based on higher-order logic. The semantics of his alternative approach, Univalent foundations, can be presented using higher topoi and geometric morphisms. It seems therefore that the outlook being proposed here is compatible with future integration with a Univalent approach to foundations.

[9] This point is made in detail at the end of Lecture 15 of the seminar.

3 A sketch of proofchains

> Fundamentally, decentralized communities require incentives for its community members to hold and validate the tamper-evident records, making them collectively immutable. With that as a secure foundation, they can then consider whether a token, representing an actual stake in the community's common assets and purpose, fits the circumstances. If it does make sense, then one must ensure that the differential equation for reward redistributes tokens along the gradients of value creation and long-term stability.
>
> W. Scott Stornetta

Distributed consensus is the cryptographic process that members of a cryptocurrency network use to synchronise their individual copies of the complete history of transactions. This globally coherent history is usually referred to as the *blockchain* [33]. The fact of this global coherence, together with the verification of the lack of double-spending at the time of incorporation of transactions into the blockchain, has an emergent consequence: the existence of a unit of cryptocurrency (Bitcoin, say) with certain logical properties.

In this sense the Bitcoin network is a *logical community*: the energy expended in the proof-of-work protocol can be viewed as the cost of instantiating an emergent logical order with desirable properties (scarcity) on top of more fundamental computational degrees of freedom (the bits of the computers in the network) which do not possess those properties. This note explores the general case of this idea, where the logical order is described by an arbitrary first-order theory, and the analogue of the blockchain maintains a distributed globally coherent *model* of that theory, which may be modified locally by any participant in the network according to a prescribed set of incremental transformations. We call this globally coherent history of transformations the *proofchain*. We hope that proofchains can help to enable new forms of cooperation, by creating logical communities with economic incentives that are more productive than the prevailing ones.

The details of the proposal are given in the next section. To motivate the definitions, let us first imagine a decentralised community maintaining a shared repository of formally verified software components. To explain how novel kinds of markets could help incentivise the activity of such communities, we provide the following cartoon:

Example 3.1. Suppose a user wishes to commission software with specification T consisting of minimal requirements S plus security constraints S', so that $T = S + S'$. This request is posted to the proofchain network together with cryptocurrency bounties funding the work, and the following activity ensues:

- Users survey the existing library $\{(T_i, M_i)\}_{i \in I}$ of software maintained in a tamper-evident and distributed way by the proofchain. The library consists of specifications

T_i together with software M_i fulfilling the specification. If necessary new components are commissioned with their own bounties. Once the work is complete and code M satisfying S has been synthesised, users receive the bounty associated to S.

- To determine whether M satisfies the security constraints S' a prediction market is opened, capitalised by the bounty, where users buy and sell shares in the payoff from a successful proof of the conjectures S' about the code M. We refer to this as a *conjecture market*. The price of these shares gives an objective measure of the knowledge of the community about whether M satisfies S' [16].

  A user who possesses a proof π of S' can buy all open shares in the market, and then broadcast π onto the network, at which point the network protocol automatically closes the market and distributes the payoffs. To produce a proof π means, generally speaking, to first prove properties $\psi_1, \ldots, \psi_k$ of the components $N_1, \ldots, N_k$ used to assemble M. The market value of π determined by the bounty implies a market value for $\psi_1, \ldots, \psi_k$, by arbitrage between conjecture markets, in a recursive process. Users react to the market incentive by proving the fine-grained conjectures, and the results are collected into a proof π of S' for M.

The integrity of this process is guaranteed by the distributed consensus mechanism.

Acknowledgements. The work presented in this section is an extract from an ongoing collaboration with Calin Lazaroiu and Dmitry Doryn begun in December 2018.

3.1 Some details

Throughout this section we fix a topos $\mathcal{E}$ where we take models. We assume that there is an effective computational means of describing models in $\mathcal{E}$ by binary strings. We call such strings *codes* and write $c(M)$ for the code of a model M in $\mathcal{E}$.

Definition 3.2. A *logical order* is the data of

- A first-order geometric theory T,
- A finite set of *conjectures*, which are formulas in the language of T,
- A set of *allowed transactions* on models of T.

The allowed transactions are represented by a polynomial time boolean-valued computable function $A(p, c(M), c(M'), \tau)$ of a public key p, a pair M, M' of models represented by their codes, and an algorithm τ which represents the change, that is, which computes M' from M. The function A evaluates to true if and only if the user with public key p is allowed to transition the model from M to M' via the transaction τ. We restrict to geometric theories so that we have a good theory of changes of logical order by geometric morphisms (for example, updates to the specification).

The proofchain, which we will specify in a moment, solves the problem of distributed and decentralised consensus about a single model M of the theory T, whose dynamics is governed by the set of allowed transactions described by A. That is, the proofchain instantiates the given emergent logical order. The details are the obvious modifications of [33]. The proofchain is, roughly speaking, a crytographically secured sequence

$$c(M_1), (p_1, c(M_1), c(M_2), \tau_1), c(M_2), (p_2, c(M_2), c(M_3), \tau_2), \ldots, c(M_n) \tag{1}$$

where M_i are models for $i \geq 1$, the p_i are public keys, and for all $i \geq 1$

$$A(p_i, c(M_i), c(M_{i+1}), \tau_i) = \text{true}\,.$$

In order that the ith transaction is authenticated as being initiated by the user with public key p_i, and in order that this user cannot deny having sent the transaction, we require as in the Bitcoin blockchain that the user digitally signs the pair

$$c(M_i), (p_i, c(M_i), c(M_{i+1}), \tau_i)$$

with their private key. A *block* consists of a sequence (1) satisfying the given constraints, together with the digital signatures and the hash of another block (the parent block) and a nonce. The proof-of-work consists of incrementing the nonce until a value is found that gives the block's hash some required number of zero bits.

The steps to run the network are as follows [33, §5]:

(1) New transactions are broadcast to all nodes

(2) Each node collects new transactions into a block

(3) Each node works on finding a difficult proof-of-work for its block

(4) When a node finds a proof-of-work, it broadcasts the block to all nodes

(5) Nodes accept the block only if all transactions in it are valid

(6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash

The only difference to the Bitcoin blockchain protocol lies in step (5) where we use a more general notion of validity, based on the function A. Nodes always consider the longest chain to be the correct one and will keep working on extending it, and in this way the energy expended in proof-of-work creates a distributed consensus on a model M. The other details (such as the block verification protocol) follow Bitcoin.

Summary: the proofchain secures the fact that the data M is (in the same approximate sense as in the usual blockchain) guaranteed to both be a model of the given logical

theory, and also to have evolved from its initial state according to the allowed set of possible transactions as described by the logical order.

A logical order consists of a first-order language together some formulas called *axioms* and some formulas called *conjectures*. The proofchain protocol verifies that the data of the model satisfies the axioms at all times. However, the conjectures are "soft" constraints in the sense that they are not required to hold. Instead, the state of belief about each conjecture is encoded in a *prediction market* which is maintained as part of the proofchain protocol. Prediction markets are a simple and elegant distributed method of incentivising the sharing of information [16]. Given a proposition P, the market trades in shares of a fixed payoff in the event that P is proven to be true. The more certain the market is that P is true, the higher the price of each share will be to acquire.[10]

Ideally, a user of the network can query this prediction market to obtain an estimation of the likelihood of the truth of conjecture which accurately represents the state of the knowledge of the participants in the network. The key to the prediction market reflecting the true state of knowledge of the participants is that each user is incentivised economically to contribute whatever knowledge they possess about the conjecture, and indeed, that users are incentivised to invest whatever computational or cognitive resources are required to *determine* the truth of the conjecture.

It is desirable that there is no hard boundary between axioms and conjectures: instead, there should be a mechanism in the protocol for conjectures to be converted into axioms, when a proof (or disproof) of the conjecture is found by the market mechanism. In this way an initially modest set of axioms can evolve into a stronger theory over time.[11]

The primary technical problem with realising conjecture markets is the problem of liquidity. Some of the problems of liquidity in such prediction markets have been addressed in the literature on combinatorial prediction markets [17, 7, 9, 34, 9] and theoretical machine learning literature [15]. To return finally to the context of Example 3.1, individual users who observe an outstanding conjecture π and *guess* that the key to resolving such a conjecture are propositions $\psi_1, \ldots, \psi_k$ about sub-components can *front their own funds* to establish conjecture markets about these components. This kind of mathematical activity is in effect a form of market making, and helps to increase consistency across a family of interdependent conjecture markets.

References

[1] K. J. Arrow, *The Limits of Organization*, New York, 1974.

[10] Parties wishing to incentivise the resolution of a conjecture can donate funds to increase the payoff.

[11] This potentially one way to avoid the problem of a high barrier of entry to the authoring of formally verified software.

[2] N. Benton, *Categorical monads and computer programming*, LMS Impact150 Stories, 1, 9-13, 2015.

[3] N. Benton and P. Wadler, *Linear logic, monads and the lambda calculus* In Logic in Computer Science, 1996. LICS'96. Proceedings., Eleventh Annual IEEE Symposium on (pp. 420-431). IEEE, 1996.

[4] M. Bezem, T. Coquand, *Automating coherent logic*, Proceedings of LPAR 2005, LNCS **3835**, pp. 246–260, Springer, 2005.

[5] M. Bezem, T. Coquand and A. Waaler, *Research proposal: automating coherent logic*, http://www.ii.uib.no/acl/description.pdf, 2006.

[6] M. Bezem, T. Hendricks, *On the mechanization of the proof of Hessenbergs theorem in coherent logic*, Journal of Automated Reasoning **40**, pp 61–85, 2008.

[7] Y. Chen and D. M. Pennock, *A utility framework for bounded-loss market makers*, In Proc. of UAI, pages 349358, 2007.

[8] K. Claessen, R. Hähnle, and J.Mortensson, *Verification of hardware systems with first-order logic*, Proceedings of the CADE-18 Workshop-Problem and Problem Sets for ATP. No. 02/10. 2002.

[9] M. Dudik, S. Lahaie, M. D. Pennock, *A tractable combinatorial market maker using constraint generation*, In Proceedings of the 13th ACM Conference on Electronic Commerce (pp. 459-476). ACM, 2012.

[10] H. de Nivelle, J. Meng, *Geometric Resolution: a proof procedure based on finite model search*, Proceedings of IJCAR 2006, LNAI 4130, pp 303-317, Springer, 2006.

[11] S.Durdević, J.Narboux and P. Janičić, *Automated generation of machine verifiable and readable proofs: A case study of Tarskis geometry*, Annals of Mathematics and Artificial Intelligence, 74(3-4), 249-269, 2015.

[12] R. Dyckhoff and S. Negri, *Geometrisation of first-order logic*, Bulletin of Symbolic Logic 21.2 (2015): 123-163.

[13] J. Fisher and M. Bezem, *Skolem machines*, Fundamenta Informaticae **91**, pp 79-103, 2009.

[14] F. Fukuyama, *Trust: The social virtues and the creation of prosperity*, Free Press Paperbacks, 1995.

[15] S. Garrabrant, T. Benson-Tilsen, A. Critch, N. Soares and J. Taylor, *Logical induction*, arXiv preprint arXiv:1609.03543, 2016.

[16] R. Hanson, *Idea Futures*, http://mason.gmu.edu/~rhanson/ideafutures.html, first version 1996.

[17] R. Hanson, *Logarithmic markets coring rules for modular combinatorial information aggregation*, The Journal of Prediction Markets, 1(1), 3-15, 2012.

[18] R. Hanson, *Bets As Signals of Article Quality*, October 26, 2018, http://www.overcomingbias.com/2018/10/bets-as-signals-of-article-quality.html.

[19] R. Hanson, *How to fund prestige science*, November 10, 2018, http://www.overcomingbias.com/2018/11/how-to-fund-prestige-science.html#more-31927.

[20] K. Hartnett, *A Fight to Fix Geometrys Foundations*, Quanta Magazine, retrieved from https://www.quantamagazine.org/the-fight-to-fix-symplectic-geometry-20170209/, 2017.

[21] B. Holen, D. Hovland and M. Giese, *Efficient rule-matching for hyper-tableaux*, 9th International Workshop on Implementation of Logics Proceedings, Easy-Chair Proceedings in Computing Series 22, Easy-Chair, pp 417, 2013.

[22] C. A. R. Hoarse, J. Misra, G. T. Leavens, N. Shankar, *The verified software initiative: A manifesto*, ACM Comput. Surv., 41(4), 22-1, 2009.

[23] P. T. Johnstone, *Sketches of an elephant: A topos theory compendium* Vol. 1. Oxford University Press, 2002.

[24] J. Lambek and P. J. Scott, *Introduction to higher-order categorical logic*, Vol. 7. Cambridge University Press, 1988.

[25] E. Klarreich, *Titans of Mathematics Clash Over Epic Proof of ABC Conjecture*, Quanta Magazine, retrieved from https://www.quantamagazine.org/titans-of-mathematics-clash-over-epic-proof-of-abc-conjecture-20180920/, 2018.

[26] G. Klein et al. *seL4: Formal verification of an OS kernel*, Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. ACM, 2009.

[27] S. MacLane and I. Moerdijk, *Sheaves in geometry and logic: A first introduction to topos theory*, Springer-Verlag 1992.

[28] M. Makkai and R. Paré, *Accessible categories: the foundations of categorical model theory* (Vol. 104). American Mathematical Soc., 1989.

[29] V. Marinković, *Proof simplification in the framework of coherent logic*, Computing and Informatics, 34(2), 337-366, 2015.

[30] E. Moggi, *Notions of computation and monads. Information and computation*, 93(1), 55-92, 1991.

[31] D. Murfet, *The Rising Sea seminar on topos theory and higher-order logic*, http://therisingsea.org/post/seminar-ch/.

[32] T. Murray and P .C .van Oorschot, *BP: Formal Proofs, the Fine Print and Side Effects*, IEEE SecDev 2018.

[33] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, https://bitcoin.org/bitcoin.pdf, 2008.

[34] D. M. Pennock and L. Xia, *Price updating in combinatorial prediction markets with Bayesian networks*, arXiv preprint arXiv:1202.3756.

[35] B. Potter, D. Till and J. Sinclair, *An introduction to formal specification and Z* , Prentice Hall PTR, 1996.

[36] S. Stojanović, V. Pavlović and P. Janičić, *A coherent logic based geometry theorem prover capable of producing formal and readable proofs*, Proceedings of Automated Deduction in Geometry 2010, LNAI 6877, pp 201220, Springer, 2011.

[37] S. Stojanović, J. Narboux, M. Bezem and P. Janičić, *A vernacular for coherent logic*, In Intelligent Computer Mathematics (pp. 388-403). Springer, Cham., 2014.

[38] S. Vickers, *Geometric logic in computer science*, Theory and Formal Methods 1993. Springer, London, 1993. pp.37–54.

[39] S. Vickers, *Geometric logic as a Specification Language*, Theory and Formal Methods, 1994.

[40] V. Voevodsky, *Will Computers Redefine the Roots of Math?*, Quanta Magazine, retrieved from https://www.quantamagazine.org/univalent-foundations-redefines-mathematics-20150519/, 2015.

Department of Mathematics, University of Melbourne
E-mail address: d.murfet@unimelb.edu.au