Communicating and trusting proofs: The case for broad spectrum proof certificates

Dale Miller

INRIA & LIX/École Polytechnique

Abstract. Proofs, both formal and informal, are documents that are intended to circulate within societies of humans and machines distributed across time and space in order to provide trust. Such trust might lead one mathematician to accept a certain statement as true or it might help convince a consumer that a certain software system is secure. Using this general characterization of proofs, we examine a range of perspectives about proofs and their roles within mathematics and computer science that often appear contradictory. We then consider the possibility of defining a *broad spectrum proof certificate* format that is intended as a universal language for communicating formal proofs among computational logic systems. We identify four desiderata for such proof certificates: they must be (*i*) checkable by simple proof checkers, (*ii*) flexible enough that existing provers can conveniently produce such certificates from their internal evidence of proof, (*iii*) directly related to proof formalisms used within the structural proof theory literature, and (*iv*) permit certificates to elide some proof information with the expectation that a proof checker can reconstruct the missing information using bounded and structured proof search. We consider various consequences of these desiderata, including how they can mix computation and deduction and what they mean for the establishment of marketplaces and libraries of proofs. In a companion paper we proposal a specific framework for achieving all four of these desiderata.

1 Introduction

Logic and proof are formal systems that have proved of some utility in mathematics and computer science. We first survey the landscape of uses for proofs within mathematics and computer science: we take the liberty of developing this survey at a high-level so that we can make sense of the many uses of this notion. We then turn our attention to what appears to be a central theme of proof: they are documents that are used to communicate trust.

While much of the value of proofs comes from their communications, the current state of affairs in computational logic systems makes exchanging proofs the exception instead of the rule. Many theorem proving systems use proof scripts to denote proofs and such scripts are generally not meaningful in other theorem provers: they may also fail to denote proofs for different versions of the same prover. There is also a wide variety of "evidence of proofs" that appear

in computational logic systems: these can range from proof scripts to resolution refutations and tableaux proofs to winning strategies in model checkers. It is hard to imagine that a given theorem prover would be able to read and check proofs in all these forms. Still, there is growing evidence that proofs need to be communicated between different computational logic systems: see, for example, [FMM+06] where an SMT prover is combined with the Isabelle prover. Generally, this work proceeds by integrating two particular prover systems: the general problem of integrating provers is seldom addressed.

In Section 2 we describe proof in rather general terms, admitting informal and formal proofs and allowing consumers of proofs to be, for example, mathematicians, computer programmers, and even computers. This general setting allows one to revisit the various criticisms and controversies surrounding the roles of proof in mathematics and computer science.

Starting with Section 3, we turn our attention exclusively to *formal proof* and we shall focus on their potential roles within computational settings where formal proofs have a limited but growing relevance. In particular, we shall consider the problem of designing a notion of *proof certificate* that can be used to denote a wide variety of proof structures. These desiderata attempt to capture ambitious goals for proof certificates, including the fact that they should have a highly flexible structures that should be easy to check. Since proofs can be large objects that could overwhelm communication, storage, and checking resources, proof certificates must also allow for trade offs between proof size and proof checking. We describe in another paper [Mil11] how it should be possible to design such a notion of proof certificate. In this paper, however, we flesh out several consequences of having proof certificates of this style.

2 Characterizing proofs and their roles

To understand the role and, hence, the nature of proof, we need to take a step back and review why proofs exist and how they are used. A key aspect of proof seems to be that they are documents that are communicated within a group of individuals in order to inspire trust.

2.1 Societies of humans and machines

Communications takes place within various "societies" comprised of individuals dedicated to common ends: we shall also assume that such individuals are human as well as computers. Admitting machines into such societies seems sensible in the many situations where computers are making decisions and are reacting to other individuals to further the goals of a society. We list here various kinds of societies of agents and some possible goals for them: while such societies may have several goals, we select here those for which a notion of proof is helpful.

1. A *sole mathematician* writes an argument that convinces himself and he then moves to address new problems. In this small society, a proof is a communication between the mathematician at one moment (the time he developed

the proof) and some future time. A goal of such the sole mathematician is to continue to develop a line of research with confidence.
2. A collection of *mathematician colleagues* searches for beautiful and deep mathematical concepts. The energies of such a group are put into finding the right definitions and the right connections among ideas.
3. An *author of a mathematics text and his readers* is a society that is typically distributed by both geography and by time: the readers are located in a future after the text is written. The goal of this society is to have a successful *one-way communication*: that is, the author must be able to communicate with readers without getting feedback on how successful was the communications.
4. A group consisting of *programmers*, who are writing code for a popular operating system, and *users*, who are attempting to use that operating system on their computers, has a goal of producing quality software that the users find convenient and secure and for which the programmers get rewarded with purchases.
5. A *group of programmers, users, mobile computers, and servers* can form a society that exchanges money for various services (e.g., email, news, backups, and cloud computing).

Notice that in example 4, machines are not meant as individuals of the group: instead, they are tools used by the individuals. On the other hand, it seems appropriate to classify smart phones, electronic banking systems, and software servers all as individuals in example 5 since the choices and decisions that they take affect the goals of the society.

2.2 Proofs as documents communicated within societies

By logical formulas we mean the familiar notion of syntactic objects composed of logical connectives, quantifiers, predicates, and terms: these have, of course, proved useful for encoding mathematical statements and assertions in computational logic. Proofs can be seen as one kind of document that is communicated within a society of agents (human or computer) distributed in *time and space* with the purpose of instilling trust in an assertion (written as a logical expression). We return to the example societies in the previous section and illustrate roles for proofs in them.

1. The only communication possible within a society consisting of a *sole mathematician* involves that mathematician telling a future instance of himself to trust that a certain formula is a theorem. If at some point in the future, that mathematician trusts his proof, he might take certain actions, such as developing consequences of that theorem.
2. Consider a group of *mathematician colleagues* such as the one featured in Lakatos's *Proofs and Refutations* [Lak76]. This society interacts within a lively and narrow spacial dimension with the agents sitting together discussing. The individual also interact across time, of course, as new examples,

counter-examples, definitions, and proofs appear. The goal of such a society of mathematicians might be to "develop deeper insights and understanding of geometry." The group exchanges messages and makes presentations. Proofs in this setting are generally informal since the energies of the group are put into exploring and discovering the right definitions and the right connections among ideas.

3. A society involving the *author of a mathematics text* and his *readers* generally involves a one-way communication: the readers will have the text only after the book is written and the readers may be physically remote from the authors. A good example of such an author and text is, of course, Euclid and his *Elements*, which has been an important text for the communication of deep results about geometry to readers for two millennia.
4. A *group of programmers and users* of an operating system can be a large group. Many documents might need to circulate in such a group: bug reports from users should alert programmers to things that need to be fixed; programmer ship out new versions of software components; programmers exchange programs, scripts, interfaces, etc. Some of these documents, such as interfaces, probably contain typing information, which can often be seen as formulas for which the program is a proof: type checking is then a simple kind of proof checking for simple assertions about the program. Certain parts of an operating system can be so critical to the proper functioning of the operating system that a formal proof of some correctness conditions might be required: for example, certain guarantees about device drivers (low level code used to control devices attached to a computer) might need to be formally verified by, say, a model checker [BCLR04].
5. In a *group of programmers, users, mobile computers, and servers* can come together to provide numerous services, machines themselves should be classified as individuals since decisions and actions that they can might affect achieving the groups goals. For example, a mobile phone might be asked to maintain certain security policies which could mean that certain mobile code might not be downloaded to the phone. As a result, certain services might not be available to the user of that phone and some income for those services are lost. If the infrastructure behind the movement of code allows for proofs to be attached to mobile code, the phone may allow the execution of that code if the phone could check that the proof established certain security assertions. The development of such an infrastructure has been extensively studied under the title "proof carrying code" [Nec97].

To meet their goals, societies circulate a wide variety of documents. Among mathematicians such documents include those describing examples, counter-examples, lists of definitions, axioms, and hierarchical organizations of theorems (lemmas, corollaries, propositions), and proofs. Among societies involving software and machine individuals, the list of documents includes bug reports, patches, interfaces, programs, scripts, manuals, test cases, etc. Some of these may be formal object (e.g., programs, types, and test cases) and some may be informal (bug reports and documentation). Of these many documents, proofs

can be roughly identified as those that inspire trust in one agent of the conclusions drawn by another agent. One's trust in a program might be inspired by the fact that over its lifetime, no one has found errors in it. While such evidence is an important source of trust, it is not a document nor a proof.

2.3 Formality of proofs

Proofs can be divided into those that are informal and those that are formal.

Informal proofs We generally expect that informal proofs are readable by humans and are didactic. We also expect that they do not contain all details and that they may have errors. Informal proofs are circulated within societies of humans where they can be evaluated in a number of ways: Is the proof proving something interesting? Are the assumptions the right ones? Are the proof methods appropriate? Is this situation an example or a counterexample? If an informal proof is evaluated highly enough, more might be done with it: it might be written for a broader audience and it might be formalized. Typically, an informal proof will be made "more formal" when the group of people with which it is intended to communicate becomes larger and more diverse (involving greater separation in time and space and background).

Formal proofs A formal proof is a document with a precise syntax that is machine checkable: in principle, an algorithm should make it possible to "perform" the proof described in the document. We shall not assume that formal proofs are human readable or that they contain "explanations" of why a formula is actually true. One can generally assume that a human agent does not read and understand formal proofs but rather that their existence and checkability by trusted machine agents means that the human trusts that a formula is, in fact, true.

We should expect more from formal proofs, however, than being a pile of formulas and inference rules that some program has formally checked. If the structure of proofs are based on well understood proof theory designs, it may be possible to perform rich formal manipulations of proofs. For example, they might support the extraction of witnesses and, hence, programs: given a (constructive) proof of $\forall x.A(x) \supset \exists y.B(x,y)$ and a proof of $A(c)$, these two proofs together might be expected to yield a witness d such that $B(c,d)$ holds.

2.4 Revisiting criticisms of proof

Given our characterization of proofs as documents communicated within societies of humans and machines, it seems useful to now revisit some of the criticisms often leveled at formal proofs.

In *Proofs and Refutations* [Lak76], Lakatos provides a couple of perspectives on the nature of proof that nicely illustrates our discussion about proofs. In one case, Lakatos criticizes Euclid's *Elements* for "its awkward and mysterious ordering" of definitions and theorems. Euclid's text is notable for the society

that it has served: given the vast number of readers of the *Elements* that have been distributed over both space and time, it seems that part of that text's success comes, in part, from its formal (sometime unintuitive) structure which increased its universality. Lakatos's other perspective was, of course, on the role of proofs within a small society of mathematicians with limited distribution in time and space: in such a setting, communications is much more informal and the society of mathematicians functions more as explorers of truth and good mathematical design. Even though these two societies involve only humans, their different distribution in time and space leads to rather different requirements on proofs-as-documents.

Connections between proof within mathematics and human psychology was famously pointed out by G. H. Hardy: "There is strictly speaking no such thing as a mathematical proof; we can, in the last analysis, do nothing but point; ... proofs are what Littlewood and I call gas, rhetorical flourishes designed to affect psychology, pictures on the board in the lecture ..." [Har28]. In a comment to Bertrand Russell, Hardy pushed even more the psychological benefits of proofs: "If I could prove by logic that you would die in five minutes, I should be sorry you were going to die, but my sorrow would be very much mitigated by pleasure in the proof" [Cla76, p. 176]. Formal proof probably has no role to play in Hardy's conception of proof.

Consider now a society of agents involved with building and using an operating system. Clearly, the quality of the operating system is important: it should perform various duties correctly as well as maintain certain security standards. Such a society is highly dynamic: new features are added and others are removed; bugs are discovered and patches are issued; and the operating system must allow for extension to its function by allowing new device drivers to be added or new executable code to be loaded and run. In such a setting, it seems futile to expect that there is a unique formal specification of the operating system to which members of the society are attempting to find a formal proof. None-the-less, informal proof and formal proof could still have some role to play among some agents of this society. Some programmers may want informal proofs that their programs satisfy certain requirements while other programmers might want to have completely formal proofs involving possibly weak properties of some other programs.

In light of this description of a society working to develop an operating systems, consider some of the criticisms of formal methods raised De Millo, Lipton, and Perlis in [MLP79]. For example, they argued that formal verification in computer science does not play the same role as proofs do in mathematics: this certainly does not seem problematic because of the differences among the many agents in this society: informal proof may play an important role among some agents while formal proof may play an equally important role among other agents. Those components of an operating system that are static parts of many generations of such a system (such as, for example, sorting algorithms, file system functions, and security protocols) may need to be trusted at a level that formal verification could provide. Those components that are dynamic, experimental,

and constantly changing would not be sensible targets for formal verification. De Millo, Lipton, and Perlis state that "Outsiders see mathematics as a cold, formal, logical, mechanical, monolithic process of sheer intellection; we argue that insofar as it is successful, mathematics is a social, informal, intuitive, organic, human process, a community project." Given the richness of societies that are part of building large software systems, it seems clear that both views of proofs are important and both serve important roles.

If we allow for machine-to-machine communications of proofs, then formal proof plays, of course, a central role. The *proof carrying code* project of Lee and Necula [Nec97] illustrates just such a situation. In that setting, a society of agents contains at least two machine agents, one that provides executable code and the other that is charged with permitting the accumulation of new code as long as that code maintains certain security assurances. Ensuring that security assurances are maintained requires some knowledge about the executable code. Examples of such assurances are that the code does not access inappropriate memory cells or that a typing discipline is maintained: e.g., that a "string" object is not transformed into, say, an "electronic wallet" object. The approach described by Lee and Necula requires that the executable code is paired with a formal proof that that code satisfies the necessary assurances. The proof needs to be checked prior to executing the new code.

To underline again the different roles of proof to different societies consider the following from Lakatos [Lak76]: "'Certainty' is far from being a sign of success, it is only a symptom of lack of imagination, of conceptual poverty. It produces smug satisfaction and prevents the growth of knowledge." While this criticism of formal proof sounds appropriate for those charged with the discovery of mathematical concepts, it is not a valid criticism (nor was it intended to be) of those building safety critical software where formal proof can play an important role in establishing certainty [Mac01].

For the rest of this paper, we turn our attention to formal proof and how these can be designed to be universal and amenable to communicating and checking.

3 Formulas and logical interpretation

Before describing proof certificates in more specifics, we fix the language of formulas and its underlying logical interpretation. In order to provide the most ambitious approach to capturing logics, computations, and proofs, we need to focus on frameworks that are general and inclusive. This goal has lead us to consider the following selections.

Simple Theory of Types Church's Simple Theory of Types (STT) [Chu40] provides a syntactic framework for unifying propositional, first-order, and higher-order logics. Such formulas allow quantification at all higher-order types which in turns allows for rich forms of abstractions to be encoded. A remarkable aspect of this choice of logic is that by making simple syntactic restrictions on the types of constants allowed, one can restrict STT to, say, propositional logic or (multisorted) first-order logic. It is also immediate to add syntactic operators such

as modal operators and choice operators. This choice of formula specification is not only one of the oldest formalized logics but also a common choice in several modern theorem proving systems.

Classical and *Intuitionistic Logic* Church developed the core logic (Axioms 1-6) as well as an extension involving the axioms of choice, description, extensionality, and infinity, used to formalize mathematics more generally. Church's version of STT relied on classical principles and that choice is appropriate for a great deal of formalized mathematics and computer science. Intuitionistic logic also has important roles to play in these settings. The logic we wish to describe here would be, ideally, a single logic that allows for mixing both classical and intuitionistic logic into one logic. Such a possibility seems suggested by Gentzen's original characterization of the difference between these logics [Gen69] as involving the presence or absence of structural rules (on the right of the sequent arrow) and Girard's linear logic [Gir87] which gives a flexible and precise notation for relating logical connectives with structural rules. Although there have been at least a couple of efforts to bring these logics together into one proof system [Gir93,LM09b], we shall assume for concreteness in the rest of this paper that we select classical logic as the interpretation of logic formulas. Much of the technical development below also holds for intuitionistic logic.

Proof search framework The proof search approach to the specification of computation seems particularly well developed for our agenda for proof certificates. While we shall see a number of reasons to highlight these advantages, we simply point out there that the proof search paradigm allows directly capturing a number of principles that help to unify different aspects of proof and computation. In particular, proof search specifications are based on relations: functions are, of course, easily seen as particular kinds of relations. Similarly, such specifications are non-deterministic but, of course, determinism is one specific kind of non-determinism. Finally, the proof theory of proof search makes it possible to capture parallel actions in computation: again, sequent computation is easily seen as a special kind of parallel computation. Thus, it seems better to start with a framework based on relations, non-determinism, and parallelism instead of starting with a proof normalization framework where functional, determinate, and sequential computations are forced.

4 Two desiderata for proof certificates

We now introduce the term "proof certificate" to mean a document that should denote a proof but for which some computation and search might be necessary to formally reconstruct that proof. We list now two of four desiderata for proof certificates.

D1: A simple checker can, in principle, check if a proof certificate denotes a proof.

Proof checkers should be, in principle, simple and well structured so that they can be inspected and possibly proved formally correct. The correctness of a checker should be much easier to establish than the correctness of a theorem prover: in a sense, a proof checker removes the need to have trust in theorem provers. The separation of proof generation from proof checking is a well understood principle: for example, Pollack [Pol98] argues for the value of independent checking of proofs and the Coq proof system has a trusted kernel that checks proposed proof objects before accepting them [Tea02]. Proof checking is likely to be at times computationally expensive, so different proof checkers may perform differently depending on the resources (say, memory and processors) to which they have access.

> **D2:** The proof certificate format supports a wide range of proof systems.

In other words, a given computational logic system should be able to take the internal representation of the "proof evidence" that it has built and output essentially that structure as the proof certificate. Thus, this one proof certificate format should be usable to encode natural deduction proofs, tableaux proofs, and resolution refutations, to name a few. Thus, if a system builds a proof using a resolution refutation, it should be possible to output a certificate that contains an object that is roughly isomorphic to the retained refutation.

A theorem prover is said to satisfy the "de Bruijn criterion" if that prover produces a proof object that can be checked by a simple checker [BW05]. Desiderata **D1** and **D2** together imply a "global" version of the de Bruijn criterion: if every theorem prover can output a proper proof certificate, then any prover can trust any other prover simply by using their own trusted checker. Before presenting two additional desiderata, we examine two implications of desiderata **D1** and **D2**.

The tension between "simplicity" of the checker (promised in **D1**) and the "flexibility" of the certificates (promised in **D2**) is clearly a challenge to address. In [Mil11], we propose to address this challenge by dividing inference rules into "micro" rule and "macro" rules. The proof checker will need to understand the micro rules as well as the language for defining macro rules in terms of these micro rules. Proof certificates will then be documents that are built from the macro rules only. Flexibility is achieved by employing a description language for the macro rules that is highly malleable and well structured.

4.1 Marketplaces for proofs

Formal proofs of software and hardware are developing some economic value. For example, some professional and contractual standards (for example, DefStan 00-55 of the UK *Defence Standards* [Min97]) mandate formal proofs for software that is highly critical to system safety (see [Bow93] for an overview of such standards). The cost of going to market with computer system containing an error can, in some cases, prove so expensive that additional assurances arising from formal

verification can be worth the costs. For example, an error in the floating point division algorithm used in an Intel processor proved to be extremely costly for Intel: formal verification was used within Intel to help improve the correctness of its floating point arithmetic [Har99].

Where there is economic value there are opportunities for markets and proof certificates that satisfies desiderata **D1** and **D2** make it possible to develop a marketplace for proofs in the following sense. Assume that the ACME company needs a formal proof of its next generation safety critical system (such as might be found in avionics, electric cars, and medical equipment). ACME can submit to the marketplace a formula that needs to be proved: this can be done by publishing a proof certificate in which the entire proof is elided. The market then works as follows: anyone who can fill the hole in that certificate in such a way that ACME's trusted proof checker can validate it will get paid. This marketplace can be open to anyone: any theorem prover or combination of theorem provers can be used. The provers themselves do not need to be known to be correct. ACME must make certain that the proposed theorem that it places on the market is really the one that it needs. The people submitting completed proof certificate must also try to ensure that the ACME proof checker, with its restrictions on computational power, can perform the checking: otherwise they would not be paid for their proof certificate.

If someone working in the marketplace finds a counterexample to a proposed theorem, then one should also get paid for that discovery as well. Similarly, partial progress on proving a theorem might well have some economic values. A comprehensive approach to proof certificates should formally allow counterexamples and partial proofs: we will not pursue these issues here.

4.2 Libraries of proofs

Once proof certificates are produced they can be archived within libraries. In fact, libraries might be trusted agents that are responsible for checking proof certificates. Since checking proof certificates is likely to be computationally expensive in many cases, libraries can focus significant computational resources (e.g., large machines and optimizing compilers) on proof checking. Once a proof certificate is checked and admitted to a library, others might be willing to trust the library and to use its theorems without rechecking the certificate. To the extent that formal proofs have economic value, libraries will have economic incentives to make certain that the software that it uses to validate certificates is trustable. If someone else (a competing library, for example) finds that a non-theorem is accepted into a library, trust in that library could collapse along with its economic reason for existing. Libraries can also provide other services such as searching among theorems and structuring collections of theorems.

5 Two more desiderata for proof certificates

We shall now present two additional and important desiderata.

D3: A proof certificate is intended to denote a proof in the sense of structural proof theory.

By "structural proof theory" we mean the literature surrounding the analysis of proofs in which restricting to analytic proofs (e.g., cut-free sequent proofs or normal natural deductions) still preserves completeness. For references to the literature on structural proof theory, see [Gen69,Pra65,TS96,NvP01].

Checking a certificate should mean that a computation on the certificate should yield (at least in principle) a formal proof in the sense of structural proof theory. This desideratum guarantees that the structure of certificates is not based on some ad hoc design: the certificate, as an artifact, should be viewable as a notation for a proof in a framework with rich formal properties (cut-elimination, normalization, constructive content, etc).

Our final desiderata (**D4** below) addresses the fact that formal proofs can be large and that certificates must, somehow, allow proofs to be redacted. Large proofs will tax computational resources to store, communicate, and check them. Thus, any approach to proof certificates must provide some mechanism for making them compact even if the proof they denote is huge. One approach to making proofs smaller could be "cut-introduction": that is, one can examine an existing proof for repeated subproofs and then introduce a lemma that accounts for the commonality in those subproofs. In this way, the lemma could be proved once and the various similar subproofs could be replaced by "cutting-in" instances of that lemma. There are clearly situations where cut-introduction can make a big difference in proof size. Proof certificates must, obviously, permit the use of lemmas (clearly permitted by desideratum **D3**). But this one technique alone seems unlikely to be effective in general since proofs without cuts (without lemmas) can be so large that they cannot be discovered in the first place. Our fourth desideratum suggests another way to compress a proof.

D4: A proof certificate can simply leave out details of the intended proof.

Things that can be left out might include entire subproofs, terms for instantiating quantifiers, which disjunct of a disjunction to select, etc. Thus, proof checking may need to incorporate proof-search in order to check a proof certificate that left out some details. As a result, proof checkers will not be simple programs that just check that all requirements of inference rules match correctly. Instead, they will need to be logic programming-like engines that involve unification and (bounded) backtracking search. An early experiment with using logic programming engines to reconstruct missing proof information was reported by Necula and Lee [NL98].

This desideratum forces the design of proof certificates in rather particular directions. While the other desiderata seems general and even obviously desirable, this fourth desideratum is the most characteristic for our proposal.

5.1 Flexible description of proof systems

Taken together, desideratum **D2** and **D3** require that we can provide a rich set of *inference rules* similar to the *analytic* rules (introduction, elimination, and structural) used in proof theory. One way to achieve such richness is to identify a comprehensive set of "atoms" of inference as well as the rules of "chemistry" that allow us to build the "molecules" of inference. We briefly describe how such an approach might work: a more specific proposal for such a framework is given in [Mil11].

The atoms of inference systems The sequent calculus provides an appealing set of primitive inference rules: these include the introduction of one logical connective and the deletion and copying (weakening and contraction) of formulas. As we have already mentioned, Gentzen was able to use this setting to distinguish classical and intuitionistic logic simply as different restrictions on structural rules [Gen69]. Linear logic [Gir87] provides a finer analysis of the roles of introduction rules and the structural rules: this analysis provides additional atoms of inference by, for example, separating connectives into their multiplicative and additive forms. The decomposition of the intuitionistic implication $B \supset C$ into $!\,B \multimap C$ is another example of this finer analysis of logical connectives. In order to capture inductive and co-inductive reasoning (including model-checking-like inference), the atoms of inference should also include fixed points and equality [Bae08,Bae10]. Since the trusted proof checker needs to only implement the atomic inference rules, the checker can be simple in its design, thus satisfying **D1**.

The molecules of inference systems The discipline by which the atoms of inference can be organized into the molecules of inference is based on the technical notion of *focused proof system* [And92,LM09a]. These proof systems attribute "polarity" to atomic inference rules. Atoms of the same polarity can stick together to form molecules: atoms of different polarities form boundaries between molecules. The resulting collection of molecules of inference form a proper proof system since they satisfy such meta-theoretic properties as cut-elimination. Hence, we can satisfy desideratum **D3**. There is also a lot of flexibility in how polarities are attributed so it is possible to "engineer" the set of molecules to cover a wide range of proof evidence, thus satisfying desideratum **D2**. Finally, when details of a proof are elided in a proof certificate (desideratum **D4**), the proof checker will need to conduct a search and that search should be understood as being conducted at the molecular and not atomic level: when filling in details to a proof, one should not be searching for new molecules via new combinations of atoms. Since adequately representing one proof system within another proof system is central to our development here, we expand on this topic next.

5.2 Three levels of adequacy

When comparing two inference systems, we follow [NM10] by identifying three "levels of adequacy." The weakest level of adequacy is *relative completeness*: a

formula has a proof in one system if and only if it has a proof in another system. Here, only *provability* is considered. A stronger level of adequacy is of *full completeness of proofs*: the proofs of a given formula are in one-to-one correspondence with proofs in another system. If one uses the term "derivation" for possibly incomplete proofs (proofs that may have open premises), an even stronger level of adequacy is *full completeness of derivations*: here, the derivations (such as inference rules themselves) in one system are in one-to-one correspondence with those in the other system. When we state equivalences between proof systems, we will often comment on which level the adequacy that equivalence should be placed: in general, we shall strive to always have our encodings be the third and most demanding level of adequacy. These degrees of adequacy appear to correspond roughly to Girard's proposal [Gir06, Chapter 7] for three levels of adequacy based on semantical notions: the levels of *truth*, *functions*, and *actions*.

The third level of adequacy (which, of course, implies the other two levels) is particularly significant here since it provides a sensible means for addressing desideratum **D4**. If a proof certificate elides an entire subproof then the proof checker will need to reconstruct that subproof. The designer of the proof certificate presumably has elided that subproof become he feels that it is an easy proof for the proof checker to discover. This impression is only useful, however, if the search conducted by the proof checker (which strings together the atoms of inference) can be related directly to the search for the elided proof. This match must hold for successful applications of inference rules as well as for failing applications of inference rules. The notion of full completeness of derivations allows making this match.

6 Mixing computation and deduction

Proofs and computations have, of course, a great deal in common. The Curry-Howard Isomorphism views certain (constructive) proofs as programs themselves. Here, we are interested in another connection between proofs and computation: that is, during checking of (or performing) a proof, certain computations must be made. For example, in order to take a step in a certain proof, a given number must be known to be prime, a condition that could be established by a straightforward computation.

Proof checkers can be divided into those that rely solely on determinate (functional) computations and those that permit the more general, non-deterministic (relational) computations. Proof checkers of proofs in typed λ-calculi generally rely on extensive uses of β-reduction. Via the *deduction modulo* approach to specifying proof system, theories can, at times, be turned into functional computations that sit within inference rules [DHK03]. The Dedukti proof checker [Boe11] uses deduction modulo by compiling such computations into the functional programming language Haskell.

On the other hand, there are proof checkers that are built using non-deterministic search principles and that employ logic programming engines. For example, some of the early proof checkers involved in the *proof carrying code*

effort used logic programming systems based on (subsets of) higher-order unification [AF99,App01,NL98]. These systems experimented with backtracking search (sometimes, even within the unification process). In one paper, the nondeterminism inherent in a Prolog-based proof checker was resolved by supplying the checker with an oracle which was responsible for having all the answers to the question "I have several choices to consider, which should I take?" [NR01].

While placing significant amounts of computation (either functional or relational) into inferences seems necessary for capturing a wide range of proof certificates, this integration comes with some costs. First, one must accept that a compiler and a runtime system for a programming language must be accepted into the core of a proof checker. While both functional and logic programming implementations are well studied and small and well understood compilers and interpreters can be identified for both paradigms, their presence in a proof checker will certainly complicate one's willingness to trust them. Second, proof checkers running on different hardware could have rather different resources available to them: thus, the computation required to check a proof might be available to one checker and not to another. This problem could be addressed by having a network of trusted libraries of proofs: such libraries could publish theorems only after their own proof checkers have checked a given proof certificate. Such libraries could, of course, have computational resources available that might not be available on, say, desktop or mobile computers.

The use of proof search (non-deterministic, logic programming) to do proof checking may introduce some special issues of its own. Most logic programming engines (the efficient ones) usually come with a depth-first search strategy for building proofs. This style of search is notoriously poor when dealing with problems related to deduction. Since a proof checker is only rechecking or reconstructing a object that is already known to exist, the proof certificate could well come with useful bounds on how much search needs to be done in order to reconstruct a particular, elided subproof. For example, a depth-bound for a depth-first-search process should be a natural value to estimate from an existing proof object. Another issue with using a non-deterministic proof checker is that there can be a mismatch between finding *the* proof or finding *a* proof: theorems generally have many proofs. Since a proof certificate might elide information, there is no guarantee that the proof the checker reconstructs is the original proof. This discrepancy does not appear to be serious since the proof checker will, at least, find a proof.

7 Conclusion

We have argued that proofs are used within societies of human and machine agents in order to provide confidence in certain assertions. Communicating and trusting proofs is critical to their use within such societies. When we restrict our attention to formal proofs, checking by simple proof checkers is a common approach to providing trust in proofs. Asking a proof certificate format to also be "broad spectrum" in the sense that it can be used by a wide variety of

computational logic systems, seems to imply that the complexity of the proof checker must increase significantly. We have argue here that by employing the flexibility of focused proof systems to describe macro-level inference rules, we can maintain a simple structure for the proof checker while capturing a rich collection of inference systems.

References

[AF99] Andrew W. Appel and Amy P. Felty. Lightweight lemmas in Lambda Prolog. In *16th International Conference on Logic Programming*, pages 411–425. MIT Press, November 1999.

[And92] Jean-Marc Andreoli. Logic programming with focusing proofs in linear logic. *J. of Logic and Computation*, 2(3):297–347, 1992.

[App01] Andrew W. Appel. Foundational proof-carrying code. In *16th Symp. on Logic in Computer Science*, pages 247–258, 2001.

[Bae08] David Baelde. *A linear approach to the proof-theory of least and greatest fixed points*. PhD thesis, Ecole Polytechnique, December 2008.

[Bae10] David Baelde. Least and greatest fixed points in linear logic. Accepted to the *ACM Transactions on Computational Logic*, September 2010.

[BCLR04] Thomas Ball, Byron Cook, Vladimir Levin, and Sriram K. Rajamani. SLAM and static driver verifier: Technology transfer of formal methods inside Microsoft. In E. A. Boiten, J. Derrick, and G. Smith, editors, *Proc. Integrated Formal Methods, 4th International Conference, IFM 2004, Canterbury, UK*, volume 2999 of *LNCS*, pages 1–20. Springer, 2004.

[Boe11] Mathieu Boespflug. *Conception d'un noyau de vérification de preuves pour le λΠ-calcul modulo*. PhD thesis, Ecole Polytechnique, 2011.

[Bow93] J. P. Bowen. Formal methods in safety-critical standards. In *Proc. 1993 Software Engineering Standards Symposium*, pages 168–177. IEEE Computer Society Press, 1993.

[BW05] H. Barendregt and F. Wiedijk. The challenge of computer mathematics. *Transactions A of the Royal Society*, 363(1835):2351–2375, October 2005.

[Chu40] Alonzo Church. A formulation of the simple theory of types. *J. of Symbolic Logic*, 5:56–68, 1940.

[Cla76] R. W. Clark. *The Life of Bertrand Russell*. Knopf, 1976.

[DHK03] Gilles Dowek, Thérèse Hardin, and Claude Kirchner. Theorem proving modulo. *J. of Automated Reasoning*, 31(1):31–72, 2003.

[FMM+06] Pascal Fontaine, Jean-Yves Marion, Stephan Merz, Leonor Prensa Nieto, and Alwen Fernanto Tiu. Expressiveness + automation + soundness: Towards combining SMT solvers and interactive proof assistants. In Holger Hermanns and Jens Palsberg, editors, *TACAS: Tools and Algorithms for the Construction and Analysis of Systems, 12th International Conference*, volume 3920 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006.

[Gen69] Gerhard Gentzen. Investigations into logical deductions. In M. E. Szabo, editor, *The Collected Papers of Gerhard Gentzen*, pages 68–131. North-Holland, Amsterdam, 1969. Translation of articles that appeared in 1934-35.

[Gir87] Jean-Yves Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[Gir93] Jean-Yves Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.

[Gir06] Jean-Yves Girard. *Le Point Aveugle: Cours de logique: Tome 1, Vers la perfection*. Hermann, 2006.

[Har28] G. H. Hardy. Mathematical proof. *Mind*, 38:1–25, 1928.

[Har99] John Harrison. A machine-checked theory of floating point arithmetic. In Y. Bertot, G. Dowek, A. Hirschowitz, C. Paulin, and L. Théry, editors, *12th International Conference on Theorem Proving in Higher Order Logics*, volume 1690 of *LNCS*, pages 113–130. Springer, 1999.

[Lak76] Imre Lakatos. *Proofs and Refutations*. Cambridge University Press, 1976.

[LM09a] Chuck Liang and Dale Miller. Focusing and polarization in linear, intuitionistic, and classical logics. *Theoretical Computer Science*, 410(46):4747–4768, 2009.

[LM09b] Chuck Liang and Dale Miller. A unified sequent calculus for focused proofs. In *24th Symp. on Logic in Computer Science*, pages 355–364, 2009.

[Mac01] Donald MacKenzie. *Mechanizing Proof*. MIT Press, 2001.

[Mil11] Dale Miller. A proposal for broad spectrum proof certificates. In J.-P. Jouannaud and Z. Shao, editors, *CPP: First International Conference on Certified Programs and Proofs*, 2011. To appear.

[Min97] U. K. Ministry of Defence. UK defence standardization, August 1997. DefStan 00-55.

[MLP79] Richard A. De Millo, Richard J. Lipton, and Alan J. Perlis. Social processes and proofs of theorems and programs. *Communications of the Association of Computing Machinery*, 22(5):271–280, May 1979.

[Nec97] George C. Necula. Proof-carrying code. In *Conference Record of the 24th Symposium on Principles of Programming Languages 97*, pages 106–119, Paris, France, 1997. ACM Press.

[NL98] George C. Necula and Peter Lee. Efficient representation and validation of proofs. In *13th Symp. on Logic in Computer Science*, pages 93–104, Los Alamitos, CA, 1998. IEEE Computer Society Press.

[NM10] Vivek Nigam and Dale Miller. A framework for proof systems. *J. of Automated Reasoning*, 45(2):157–188, 2010.

[NR01] George C. Necula and Shree Prakash Rahul. Oracle-based checking of untrusted software. In *POPL*, pages 142–154, 2001.

[NvP01] Sara Negri and Jan von Plato. *Structural Proof Theory*. Cambridge University Press, 2001.

[Pol98] Robert Pollack. How to believe a machine-checked proof. In G. Sambin and J. Smith, editors, *Twenty Five Years of Constructive Type Theory*. Oxford University Press, 1998.

[Pra65] Dag Prawitz. *Natural Deduction*. Almqvist & Wiksell, Uppsala, 1965.

[Tea02] The Coq Development Team. The Coq Proof Assistant Reference Manual Version 7.2. Technical Report 255, INRIA, February 2002. More recent versions may be obtained from the site `http://coq.inria.fr/`.

[TS96] Anne S. Troelstra and Helmut Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 1996.