ERC Advanced Grant 2011
Technical Description[1]

Broad Spectrum Proof Certificates

ProofCert

Principal Investigator: Dale Miller
Host institution: INRIA Saclay - Île-de-France
Duration in months: 60
Submission date: 9 February 2011

Contents

[1] This technical description is taken directly from the submitted proposal. The material in Section 2 and Section 3 was intended to be read separately. As a result, there is some duplication in the text of these sections.

1 Proposal Summary

There is little hope that the world will know secure software if we cannot make greater strides in the practice of formal methods: hardware and software devices with errors are routinely turned against their users. The ProofCert proposal aims at building a foundation that will allow a broad spectrum of formal methods—ranging from automatic model checkers to interactive theorem provers—to work together to establish formal properties of computer systems. This project starts with a wonderful gift to us from decades of work by logicians and proof theorist: their efforts on logic and proof has given us a *universally accepted* means of communicating proofs between people and computer systems. Logic can be used to state desirable security and correctness properties of software and hardware systems and proofs are uncontroversial evidence that statements are, in fact, true. The current state-of-the-art of formal methods used in academics and industry shows, however, that the notion of logic and proof is severely fractured: there is little or no communication between any two such systems. Thus any efforts on computer system correctness is needlessly repeated many time in the many different systems: sometimes this work is even redone when a given prover is upgraded. In ProofCert, we will build on the bedrock of decades of research into logic and proof theory the notion of *proof certificates*. Such certificates will allow for a complete reshaping of the way that formal methods are employed. Given the infrastructure and tools envisioned in this proposal, the world of formal methods will become as dynamic and responsive as the world of computer viruses and hackers has become.

2 Extended Synopsis

2.1 Software and hardware correctness is critically important

Computer systems are everywhere in our society and their integration with all parts of our lives is constantly increasing: along with this wide scale use of computing systems comes an increasing need to deal with their correctness. There are a host of computer systems—such as those in cars, airplanes, missiles, hospital equipment—where correctness of software is paramount. Big changes in the attitude towards correctness is also taking place in the area of consumer electronics. For example, years ago, establishing the correctness of, say, desktop PCs, music players, and telephones was not urgent since rebooting such systems to recover from errors or living without a feature due to bugs were mostly nuisances and not "life-threatening". But today,

> *these same devices are now tightly integrated into networks that require dealing with the security of information, with the anonymity of users, etc. while keeping safe from attacks from malicious software (almost always exploiting bugs within software).*

Attempting to establish various kinds of correctness-related properties of software systems is no longer an academic curiosity. The old chestnut "You can't build a tall building on a sandy beach," which is so often invoked to argue for solid foundations for engineering projects, needs a modern updating that requires moving off the beach to the sea: "If you are in a canoe, a small leak might be okay; if you are in a submarine, a small leak is lethal." As it is painfully clear today, plugging your computer into the Internet is similar to descending into the depth of the sea: if there is a crack in your security, it will be exploited quickly. One cannot be relaxed anymore about leaks.

Our ability to provide at least some formal guarantees about software systems will be directly related to our ability to deploy new functionality and services. If we cannot distinguish applets from viruses, we cannot expect people to really use the rich set of flexible services organized around mobile code. Our future could resemble elements of the world in William Gibson's *Virtual Light*, where network security was so bad that important data was transferred by bikers carrying hard-disks! If we cannot produce software that has some formal guarantees, then the development of all the new features and services—and the concomitant increases in efficiency and productivity—that we all hope to see soon will be greatly delayed.

2.2 The Tower of Babel and formal methods

Even in the earliest days of the theory of computing, there was a range of ways to specify computation: *e.g.*, Church's λ-calculus, Turing's machines, Kleene's recursive schemes, Post rewriting systems, *etc*. Gradually, it was shown that all of these different formal systems were computing the same partial recursive functions. While the functionality of these many different programming paradigms coincided, there was no coincidence in their syntax and their operational behaviors. It is now broadly accepted that the malediction against the construction of the Tower of Babel affected not only natural languages but also programming languages: there are numerous programming languages in common use and they do not easily inter-operate. Our discipline lives with many programming languages and has developed tools and techniques for making the best of this diverse and fragmented approach to programming.

Given that programs and proofs share a common and deep connection, it is natural to ask if this same malediction also applies to proofs as well. A quick review of the state-of-the-art strongly suggests that a similar fragmentation already exists. In the theoretical study of proofs and automated reasoning, there are a plethora of proof systems: resolution, tableaux, sequent calculus, natural deduction, DPLL, etc. In the practical development of theorem proving systems, proofs are often stored in formats that are seldom meaningful to another theorem prover and seldom work as proofs for even a later version of the same prover. For example, both Isabelle and Coq can represent proofs as "proof scripts" (procedural descriptions of how to steer a theorem prover to a proof) but these scripts mean nothing to any other prover. It would seem that the malediction of the Tower of Babel also applies to proof system with the result that it is difficult to communicate proofs among the many practitioners in formal methods.

Given that logic can be used to formalize much across the vast landscapes of computation and mathematics, diversity in prover technology is certainly desirable. Clearly, the techniques and tools needed for producing formal proofs in, for example, algebraic topology and in cache coherence protocols are different. The principle goal of ProofCert is to establish that it is possible for this rich diversity of provers to present their proofs in a single, flexible structure.

2.3 The challenges

We shall use the term "prover" to denote *any* computational logic system (automatic or interactive) that is used to establish a proof about some logical expression. There are a wide range of provers in common use today in both academics and industry:[2] besides the well-known theorem

[2]For a perspective on industrial uses of provers, see D. MacKenzie. *Mechanizing Proof*, MIT Press, 2001.

proving systems (Coq, Isabelle, HOL, NuPRL, PVS, *etc*) there are a number of other systems that prove theorems: for example, model checkers, type inferencers, static analyzers, SAT solvers, and rewriting systems can all be seen as providing proofs of logical formulas.

Selecting the appropriate "universal logic" does not seem too controversial. A higher-order logic (following Church's "Simple Theory of Types") seems natural: it can easily capture propositional formulas and (multi-sorted) first-order logics as well as admit an array of modal operators. Also, many typing systems (*e.g.*, dependent typed λ-calculus) can easily be mapped into formulas of higher-order logic. For these reasons, higher-order logic is a popular choice in many of today's ambitious, interactive theorem provers. The main issue depends on a choice between classical and intuitionistic logic: almost all major theorem provers force this choice in their foundations. Our choice here is to make use of recent work in the "unity of logic"[3] in which both classical and intuitionistic connectives can co-exist.

The challenge here is to design *proof certificates* that satisfy the following four desiderata.[4]

1. Since proof checkers must be trusted, they need to be, in principle, simple program: this will make it possible to verify and trust them.

2. It should be easy for a wide range of computational logic systems to output: in a given prover, there should be a pretty-printer that can output a proof certificate for any proof-like-evidence within that prover. The goal is not to force the prover to conform to some arbitrary notion of proof structure. The prover will be required to maintain some notion of proof but there should be some way for the prover to describe **its** notion of proof to the checker.

3. The process of validating a proof certificate requires discovering that it denotes a proof in a well understood and "declarative" setting. This requirement rules out, for example, procedural scripts that drives a particular prover to a proof. The certificates should denote (via a possibly complex series of computations) a proof object for which a rich set of operations are possible. As a result, certificates will allow rich mechanisms for browsing them and for applying them in new situations.

4. The design of proof certificates must acknowledge that proofs can be huge objects which cannot be communicated with all of their details. Thus proofs must contain mechanism that permit lemmas to be introduced (thus allowing tree structured proofs to be organized into directed acyclic graphs) and permit some proof information (presumably information that can be calculated) to be completely elided. The proof checker should be able to redo the computation.

2.4 The ground-breaking developments

We list a few ground-breaking developments that such a proof certificate format would allow.

Universal communications of the results of proving. 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

[3]See Girard's paper in the Annals of Pure and Applied Logic 1993 and the Liang & Miller paper in LICS 2009.

[4]These desiderata and technical approaches to addressing them appear in the draft paper "Communicating and trusting proofs: Towards a broad spectrum proof certificate" available from Miller's web site.

checker.[5] The first two desiderata above 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's output simply by using its own trusted checker.

Marketplaces for proofs. Proof certificate makes it possible to develop a marketplace for proofs. For example, the Acme company may need a formal proof of its next generation safety critical system (such as can be found in avionics and medical equipment) because the proof is mandated by a government contract.[6] Acme can place in the marketplace a proof certificate that contains the proposed theorem along with a hole instead of a proof. Acme would then offer to pay anyone who can fill that hole in such a way that Acme's trusted proof checker can validate it. 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.

Libraries of proofs. Once a proof certificate is checked, it could be admitted to a library: others might be willing to trust the library and to use its theorems without rechecking the certificate. Besides trust, libraries can also provide other services, including searching their theorems and proofs as well as structuring to large collections of theorems.

Check the certificates, not the prover. The availability of proof certificates means that the "trusted base" of software that must be trusted can be radically reduced. One needs to make certain that the proof checker is correct: much of the work of ProofCert is desired to ensure that such checkers will be relatively simple and static programs (hence, their formal correctness should be straightforward). On the other hand, most provers that are designed to search for proofs are usually sophisticated and constantly evolving: their formal verification would be a rather dubious project. Of course, if a particular prover can be formally verified and the proof certificate related to that verification is checked, the proof certificates it may produce do not need to be checked.

One proof can involve many provers. Since a model checker and an inductive theorem prover will be able to output their proof evidence as such proof certificates, their different strengths can be mixed and formally validated. For example, a model checker may only explore a small part of a model since the remaining part can be guaranteed by symmetries of the model. An inductive theorem prover might be able to prove that that model satisfies such a symmetry. While the internal mechanisms for these two style of provers can be wildly different, the desiderata above imply that both provers will be able to output proof certificates and that these can be combined into one certificate for justifying the full model checking effort.

Distributed development of proofs and counterexamples. Since proofs can be large efforts and since the availability of proof certificates should allow many different provers to contribute to a full proof, the notions of partial proof and of counterexamples will play a critical role in the dynamics of assembling proofs. Partial proofs will provide the framework for not only distributing parts of the proving effort but also dealing with *counterexamples*. If Acme is willing to pay for a proof, it should clearly also place some economic value on a counterexample. Clearly, counterexample should be part of a comprehensive proof certificate format description. A strong theory that allows proofs and counterexamples to interact will allow the scope of counterexamples to be clearly understood: in particular, one should be able to determine if the counterexample

[5] See "The Challenge of Computer Mathematics," by Barendregt and Wiedijk in *Trans. A of the Royal Society* (2005).

[6] For an overview of such government requirements, see J. P. Bowen's paper "Formal methods in safety-critical standards" in *Proc. Software Engineering Standards Symposium* (1993).

affects just one or more of the open premises of a partial proof or is it a counterexample to the initial proposed theorem.

2.5 Methodology

In order to explain how such an ambitious treatment of such a ubiquitous concept as proof can be achieve, we outline the major methodological aspects of ProofCert.

Building molecules of inference from the atoms of inference. Gentzen's sequent calculus and Girard's linear logic have provided us with both "atoms of inference" as well as a framework for assembling those atoms into "molecules of inference." Proof theory provides us with certain "chemistry rules" that describe how atoms may either stick together to make the larger connectives or separate to form the boundaries between connectives. The rules of chemistry can be found in recent work on *focused proof systems*.[7] When provers record evidence internally for a proof, that evidence is not necessarily organized using these atoms. Since the set of "molecules of inference" is surprisingly rich,[8] those molecule can often be designed to match the structure of the proof evidence. In this way, the programmer of the prover will be able to output its evidence using the larger inference rules. That same programmer will need to describe the structure of those molecules using the vocabulary of the underlying chemistry (polarizations, additive/multiplicative, phases, etc). Given this organization of proofs, the proof certificate checker only needs to understand a (small set of) atomic inferences and how to apply the rules of chemistry.

The proof checkers will also do proof search (and computation). In order to make the sizes of proof certificates manageable, proof certificates will allow for a trade-off between their size and the complexity of checking them. In the setting of ProofCert, this trade-off arises from allowing parts of a proof to be completely elided. When the proof checker encounters such a hole in the proof, the checker will need to *search* for a proof. Of course, such a search should be known (by the designer of the certificate writer) to be a simple and bounded process. The use of logic variables and unification in theorem proving is an example of eliding such aspects of proofs: instantiations are named by a *logic variable* in the certificate and unification will eventually determine its specific value. Such holes also make it possible to incorporate a flexible boundary between computations and deduction: in particular, computations are essentially proofs that both the proof producer and the proof checker are expected to do exactly the same. Thus proof certificates need to involve computations but do not need to include their trace.

The integration of model checking with conventional theorem proving. Recent work on the proof theory of fixed points and term equality[9] shows us that proof theory can capture proofs from model checkers just as well as it can capture proofs from conventional theorem provers. In particular, objects such as simulations and winning strategies can easily be seen as proof objects in the proof theory with fixed points. Thus, we can now see model checking as a deductive systems in a style similar to how we view proofs from other deductive systems. Such an integration is extremely valuable but also not without its complexity. In particular, while we know how to describe *complete* proofs for both model checking and theorem proving together, we do not yet know how to *search* for ways to complete *partial* proofs involving both style of proof. The problem

[7]See: Andreoli, *J. of Logic and Computation* (1992) and Liang & Miller, *Theoretical Computer Science* (2009).

[8]Examples are in Miller's paper "Communicating and trusting proofs: Towards a broad spectrum proof certificate."

[9]See David Baelde's 2008 PhD thesis and his recently accepted submission to the Trans. on Computational Logic.

here has to do with the nature of quantified variables and unification in the combined proof system. Addressing this issue of unification will be one of the several technical challenges of ProofCert.

2.6 The risks: high-risks but high-gains

The *unification of the formal method community* around a standard for logic and proof is a major benefit of this proposal. European industry will benefit greatly from the many improvements and tools that will come from this effort. At the same time, there are a number of risks involved in this project: as mentioned below, these risks offer opportunities as well.

What if proof certificates are really too big to communicate? There are at least two approaches to addressing such a problem in this setting. First, checkers could be given more resources (more memory, more processors) and the computations that are part of proof certificates can be optimized using logic programming compilation techniques. Second, one can use *reflection*: if one proves that a particular prover component is correct, its certificates do not need to be checked.

How richly can we integrate model checking and theorem proving? As mentioned above, no one has developed a complete approach for a proof checker to fill in the missing information (unification in that setting is not fully understood). While we shall attempt to find a complete solution to this problem, we may also need to consider studying real world examples to see to what extent these two kinds of partial proofs overlap and cause problems.

3 Scientific Proposal

3.1 The state-of-the-art

In the literature of mathematics and computer science, there is a great deal of skepticism, doubt, and confusion about the value and nature of formal proof. For example, Lakatos saw the desire to have formal proofs as a misguided quest for certainty:

> ... '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. - I. Lakatos, *Proofs and Refutations*

While this criticism of formal proof may be appropriate when aimed at a group of mathematicians which is engaged in discovering mathematical concepts (as is depicted by Lakatos), it is not a valid criticism (nor was it intended to be) of those building, for example, safety critical systems where certainty is a requirement and formal proof are a means to establish certainty [19].

Within mathematics, different domains have different needs of proofs. For example, formal proof probably has at best a minor role to play in the discovery of mathematical concepts and the appreciation of mathematics as an art form. On the other hand, the mathematician V. Voevodsky has recently described a plan [34] for doing mathematics in such a way that one can be certain that a theorem holds independently of possible inconsistencies that may someday appear in mathematics. His plan involves building and normalizing formal proofs.

The attitude towards formal proof in computer science has had a varied past. For example, in 1979 the CACM published an article by De Millo, Lipton, and Perlis [22] that provided many criticisms and problems in the use of formal proof in verification. Thirty one years later, the CACM published another article, this one by Z. Shao [30], that provides a more modern and nuanced program for the use of formal proofs in verification. On a more concrete level, formal proofs of software and hardware are now developing economic value. For example, some professional and contractual standards (for example, DefStan 00-55 of the UK *Defence Standards* [23]) mandate formal proofs for software that is highly critical to system safety (see [6] 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 of one of their processors proved to be extremely costly to Intel: as a result, formal verification is now used within Intel to help improve the correctness of the floating point arithmetic on their processors [15].

Improving the quality and safety of software systems can and must be addressed along many axises: for example, software engineering (via, *e.g.*, requirement capturing and code reviews) and human management certainly play important roles. On the other hand, formal proofs can play a role in increasing the quality and correctness of computer system. Much about the treatment of formal proofs can be automated if we have the proper, broad notion of proof and the right suite of tools surrounding them. Developing a universally accepted format for proof certificates and the tools for checking and producing such certificates is the subject of ProofCert.

3.1.1 There are many proof systems in common use

We shall use the term "prover" to denote *any* computational logic system (automatic or interactive) that is used to establish a proof about some expression in a logic. There are a wide range of provers in use today in academics and industry [19]: besides the well-known theorem proving systems (Coq, Isabelle, HOL, NuPRL, PVS, Vampire, *etc*) there are a host of other computational systems that can be seen as proving theorems. For example, model checkers, type inferencers, static analyzers, SAT solvers, and rewriting systems can all be seen providing proofs of logical formulas.

It has long been recognized that to make provers effective, one must often narrow the domain of application of the prover to specialized domains. For example, type inference and static analysis generally establish rather weak properties of program code (falling far short of "full functional correctness"): at the same time, however, such properties could be very useful to a programmer or a compiler if they could be established or refuted quickly. Similarly, when the domain of interest is finite, complete enumeration of that domain might prove more effective at proving a theorem than, say, a prover with stronger methods (such as induction) for the treatment of possibly infinite domains.

The need for proliferation in provers is more related to developing specialized automated and interactive methods for *discovering proofs*: there is **no** corresponding proliferation of *basic proof principles* (this is in contrast to a possible proliferation of axioms or theories to describe mathematical structures). Unfortunately, most existing provers do not rely on such basic proof principles but instead they produce "proof scripts": these are sequences of instructions that instruct the prover to find a proof. That is, these proof scripts are tied to the specialized methods for finding proofs instead of those more general and basic proof principles. Indeed, very few proof principles (inference rules) are needed to describe proofs in all of these systems: modus ponens, substitution, de Morgan duality, replacing expressions with equivalent expressions, *etc*. The ProofCert project will make it possible for these many provers to communicate and check each others's work by focusing on what is this common notion of proof.

3.1.2 Proofs are document to communicate

One of the goals of logic and proof has always been to be a universally accepted method for establishing the truth of propositions and formulas. One of the critical functions for proofs (both informal and formal) is to communicate across time and space. I might do an informal proof for myself with the goal of a rather short-distance communication: I want to convince just myself now of a truth so I can move to my next conjecture. If I want to write a text book and communicate across a greater distance (to readers in some other country at some future time) then my proofs will take on a more formal structure. If I wish to establish a proof that can convenience all individuals (computer checkers included), my proofs will become far more formal and, as a result, able to communicate across time and with all individuals (at least in principle).

The current state of the art neither addresses nor exploits the potential for proofs to communicate universally. Many provers do not produce proofs: their conclusions convince, at best, their implementers. Many provers produce proofs that their own kernels can check and validate: their proofs convince those who believe in the correctness of their (often tiny and transparent) kernels. Still others actually produce proof objects that have a well described format and semantics and which can be checked by independently implemented checkers [28]: someone can include such a prover into their "trusted-base" by simply including a proof checker of their own construction. Proof checkers are much easier to write and verify then the provers that are designed to discover proofs.

What is missing in the world of formal methods and formal proofs is exactly what ProofCert proposes to build: a truly universal and flexible proof document that can be communicated broadly and for which there is a clear semantics that would allow anyone to write a checker.

Designing a universal proof document is, however, far more complex than one might think at first. Consider the following rather immediate attempt at this. The logic of Church's Simple Theory of Types (a popular higher-order logic dating back to 1940) is certainly syntactically rich enough to capture propositional logic, first-order logic, modal logics, and higher-order logic: hence, the syntax of this logic is extremely flexible. Similarly, we can adopt as a proof system, say, Gentzen's 1935 sequent calculus but extended to allow for higher-order quantification. (We address how to deal with the choice between classical and intuitionistic logic in Section 3.4.2). So, why not just define proof certificates by providing a formal syntax and semantics for sequent calculus proofs over simply typed formulas? After all, implementing proof checkers for such a proposal is certainly a simple matter.

We list here just two of the many reasons for why this is a bad proposal for a universal document format for proofs.

- All prover will need to output their internal proof structures into this one format. It is far from clear that the many important proof structures in use today (resolution, tableaux, DPLL, natural deduction, winning strategies, *etc*) can all be related to sequent calculus proofs. Even if it were possible, the resulting encodings would likely introduce a significant blow-up in proof size and obfuscate their structure.

- Proofs can be huge and various "optimizations" and "compressions" must be made of them in order for them to be effectively communicated: for some early work on addressing proof size in the *proof carrying code* literature, see [24]. While the sequent calculus contains the so-called cut rule (the "lemma" rule), it is unlikely that the insertion of cut rules can really address the full problem of proof compression since proofs without cuts are generally so large that they do not exist in nature (nor in computer memories). Other means for making proofs smaller must be taken seriously.

We now turn our attention to the goal of deploying structured documents for expressing proofs: we shall refer to these documents as *proof certificates*.

3.2 Objective: Communicate, check, and trust proofs

We propose to develop a *broad spectrum proof certificate* so that nearly all deductive systems can present their proof evidence as such a certificate and so that simple certificate checkers can be build and trusted. These proof certificates can then be used to communicate proofs between widely different deductive systems. They can also be stored in libraries and information about their structure can be probed and extracted.

The foundations behind building such an approach to proof certificate can be found in the work on proof theory given by Gentzen's introduction of *analytic* methods for representing proofs [12] and Girard's refinement of that structure given by the introduction of linear logic [13]. These two cornerstones of *proof theory* provide us with what we can call the "atoms of inference." More recent work has now shown us that these atoms of inference can be organized into the "molecules of inference" (synthetic connectives) and that there can be a great deal of computation that can be placed into such synthetic connectives [10, 20]. Proof certificates will be based on these molecules of inference and since the range of molecules one can describe is essentially endless, the structure of such certificates will similarly be endless. On the other hand, checking certificates can be, in principle, simple: the checkers need only be designed to handle the small (closed) set of atomic inferences and the "rules of chemistry" that describe how molecules are assembled from atoms (how some atoms stick together and how others cleave apart). For example, such chemistry makes it possible to place computation within inference rules, thereby allowing the engineer of proof certificates to move the boundary between deduction and computation in any number of ways.

ProofCert will take these basic insights into how inference can be organized and will develop them into technologies that can be concretely tested and universally deployed.

3.3 Objective elaborated: four desiderata for proof certificates

We now list four desiderata (**D1** - **D4**) that are necessary for proof certificates to achieve success and broad acceptance.

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: a proof checker removes the need to have trust in theorem provers [29]. The separation of proof generation from proof checking is a well understood principle: for example, Pollack [26] argues for value of independent checking of proofs and the Coq proof system has a trusted kernel that checks proposed proof objects before accepting them [32].

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

In other words, a given prover should be able to take the internal representation of the witness of a proof object 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 [5]. 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. We examine here a couple implications of desiderata **D1** and **D2**.

Marketplaces for proofs A proof certificate that satisfies desiderata **D1** and **D2** makes 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. If someone working in the marketplace finds a counterexample to a proposed theorem, then one should also get paid for that discovery. In this sense, a comprehensive approach to proof certificates should also formally allow counterexamples: we address this issue of counterexamples later in Section 3.4.4.

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 could 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 they use 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. Besides trust, libraries can also provide other services: they could allow people to search their theorems and they could provide some structuring to collections of theorems.

We now present two additional desiderata for proof certificates that will lead ProofCert in novel and distinctive directions.

D3: A proof certificate denotes a proof in the sense of structural proof theory.

By "structural proof theory" we mean the topic 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 [12, 25, 27, 33]. Checking a certificate will be based on attempting to discover (at least in principle) a formal proof in the sense of structural proof theory that the certificate describes in some but not necessarily all details. 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, *etc*) and rich forms of abstraction. Given this desideratum, it should be possible to meaningfully browse and explore proof certificate: in particular, formal proofs in structural proof theories allow for rich reorganizations of their structure (via permutations of inference rules and via substitutions into proofs). While readability is not a necessary desideratum of proof certificates, being able to explore their meaning should be possible and should make understanding their structure much more rewarding that some more static requirement of "readability".

Our final desideratum (**D4** below) addresses the problem of proof size. Proofs will often be large and, as such, 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 (such a transformation can, for example, change a tree-structured proof into a more efficient directed-acyclic graph). There are clearly situations where cut-introduction can make a big difference in proof size (such as *tabled deduction* used in logic programming and model checking [21]). 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. Desideratum **D4** 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 in which some details have been left out. This desideratum forces the design of proof certificates in rather particular directions.

3.4 Methodology

We now outline the major methodological aspects of the ProofCert project.

3.4.1 Building molecules of inference from the atoms of inference

Gentzen's sequent calculus [12] and Girard's linear logic [13] provide us with both "atoms of inference" as well as a framework for assembling those atoms into "molecules of inference". Proof theory provides us with certain "chemistry rules" that describe how atoms may either stick together to make the larger connectives or cleave so that clear boundaries describe the size of connectives: these rules of chemistry can be found in recent work on *focused proof systems* [1, 16] and *super-deduction* [7]. These rules of chemistry allow those working with proofs to move from one level of abstraction (the micro-connectives, the atoms) to another level of abstraction (the macro-connectives, the molecules). A remarkable aspect of this chemistry is that this next level of abstraction provides another proper logic, in the sense that the important analytic techniques of proof theory (namely, cut-elimination, the existence of various kinds of normal forms, *etc*) all hold for this new level of abstraction. Also remarkable about the chemistry is that it is surprisingly flexible and that there are several "dials" that can be adjusted to achieve widely varying molecules that are built from the same atoms.

By analogy, there are two levels (at least) of abstraction also when studying DNA: at one level, DNA can be described as as sequences of carbon, oxygen, nitrogen, and hydrogen molecules but on another level, it can be described as being composed of the molecules of adenine ($C_5H_5N_5$), cytosine ($C_4H_5N_3O$), guanine ($C_5H_5N_5O$), and thymine ($C_5H_6N_2O_2$). Much about DNA can be studied by starting with the abstraction that sees it as a sequences of A, C, G, and T.

The rules of chemistry, in this setting, provide important distinctions that go to the heart of the nature of logical connectives and their roles in building proof. These distinctions sometimes have little relevance if we are only thinking to the truth-functional natural of logical formulas. For example, the distinction that we have learned from linear logic about *additive* and *multiplicative* versions of logical connectives has no barring on the truth conditions for such connectives: however, that distinction can make a large difference in the size (at the molecular level) of proofs.

For example, using the additive conjunction in the usual definition of the Fibonacci sequence (as a recursive Horn clause specification) leads to proofs that are exponential in size. If we switch instead to the multiplicative conjunction, proofs can be of linear size. Such differences in proof size and structure is critical for any study of proof certificates.

3.4.2 Which logic to choose?

Proofs are, of course, based on logical formulas and judgments about them. Hence, we must select a logic on which to build the entire edifice for our proposal for proof certificates.

Logic instead of type theory There is a continuum between what is a predicate in logic and what is a type in a type theory. Generally, types are predicates for which well established inference procedures can operate (*e.g.*, type inference) while no such procedures are assumed for predicates. Since types seems to be a topic of continual evolution and design, it seems natural to instead pick the logic of predicates as the foundation for proof certificates. There is also a good deal of evidence that many type theories can be effectively encoded in logic without a loss of expressiveness [8, 11, 31]. Since we need a common denominator for a wide range of deductive systems, we shall chose logic over type theory.

Higher-order logic The choice of having higher-order principles in our logic seems natural: inside higher-order logic one can easily identify both propositional and (multi-sorted) first-order logics. For example, it is trivial to see the propositional formulas in SAT solving as also being higher-order formulas. Also, higher-order logics provide rich forms of abstractions (usually via λ-abstraction) that can prove invaluable for specifying encoding. On top of these reasons, it is also the case that many popular theorem provers and type system today directly encode higher-order principles.

Classical *and* intuitionistic logic One of the most unfortunate fractures of logic occurs along the *classical* versus *intuitionistic* divide. A great many provers assume a classical logic foundations while an equally large number of provers assume an intuitionistic ("constructive") logic foundation. In many ways, this divide is necessary and has been around since the very first formal studies of modern proof theory (starting with, for example, Hilbert and Gentzen). In order to make for a truly universal framework for proof certificates, one must be able to represent and to relate proofs for both classical and intuitionistic logic within a *single* framework. Gentzen [12] made progress on such a goal with his presentation of sequent calculus which described the differences between classical proof (LK) and intuitionistic proof (LJ) as essentially one about structural rules. Later Girard's linear logic [13] provided a richer treatment of structural rules and their relationship to the logical connectives. Based on that work, Girard presented his "Logic of Unity" (LU) calculus [14] that combined classical, intuitionistic, and linear logic. In many ways, the LU logic is too ambitious and too awkward: although it has been close to two decades since that logic was invented, no one has yet figured a way to actually use that logic in practice. More recently, Liang and Miller have presented a novel merging of classical and intuitionistic logic into a single logic called "Polarized Intuitionistic Logic" (PIL) [17, 18]. In this logic, proof rules for both of these logics live together and can interact (via, for example, cut-elimination).

Within ProofCert, our logic shall be a higher-order version (in the sense of Church's Simple Theory of Types [9]) of the PIL logic. As a result, we will not need to have two different proof formats: one for classical proofs and one for intuitionistic proofs. One framework will work for both classical and intuitionistic inference.

3.4.3 One logic in two stages

There is, however, one remaining issue that must be addressed when we attempt to integrate a wide range of deduction technologies. In particular, there has been a long standing rift between theorem proving and model checking: in a truly universal approach to proof certificates, evidence of proofs generated by model checkers (often provided by "tables") must be admissible within the same space of proofs that are returned from theorem provers. Notice that we do not need to bring unity to the algorithms and data structures that are used in these different technologies: in fact, a proper framework for proof certificates should permit these different deductive systems to have widely varying implementation. Instead, we only need to capture the *evidence* returned from model checkers within the proof certificate framework.

Recent work into the proof theory of *fixed points* and *inductive* and *co-inductive definitions* allowed us to view model checking as deduction and to allow one to view the results of model checking as proofs within proof systems that directly treat least and greatest fixed point as logical connectives [2, 3]. In such a setting, the result of a model checking process, say the production of a simulation or of a winning strategy, can be viewed directly as a rather direct and natural (sequent calculus) proof object [4, 21].

While we have strong evidence to feel that we have a proof theory that allows encoding proofs from both model checking and theorem proving into one setting, we are aware that the technology for checking such rich proof certificates is still an open question. In particular, checking proofs in both settings separately makes use of the *unification* of term structures. When checking theorem proving style proofs, this unification is for one set of variables (the so called "logic variables" or "existential variables"). On the other hand, when checking model checking style proofs, this unification must instead instantiate "eigenvariables" or "universal variables." When these two unification processes must exist together, we do not yet have the tools and technology to describe fully their interaction. Since this proposal is mainly about developing the technology and tools related to proof certificates, this development must proceed in two stages.

Stage 1: Let the higher-order version of a classical-intuitionistic logic PIL be referred to simply as L. The implementation of proof search and unification for this logic is known, so tool development can proceed immediately.

Stage 2: Let μL be the logic that extends L with the least and greatest fixed point operators (along with the equality connective for terms) along the lines of the development of the $\mu MALL$ and μLJ logics described in [3]. While the proof theory for this logic is mostly understood, a key elements for the implementation for a proof checker (namely, the implementation of unification for *both* eigenvariables and logic variables) is still insufficiently understood. Tool development for this stronger logic must be delayed until these key implementation tasks are better understood.

3.4.4 A single framework for proofs, partial proofs, and counterexamples

The development of a proof for a proposed theorem is likely to be a distributed effort by many people using an assortment of different tools (theorem provers, model checkers, static analyzers, *etc*). Since proof development might take a long time, one must be able to encode *partial* proofs. Thus the proof certificate format must be able to support such partial objects and allow them to be checked and shared between participants. Different groups must then be able to build on different parts of the proposed proof in order to complete it. In fact, a partial proof might be used to distribute work to different groups based on their tool set: for example, some subproofs might be quite shallow and involve a lot of checking and computation. In that case, a model checker might be the tool of choice and the team that is expert with that tool would be delegated that subproof. Of course, other parts of a proof might require more in-depth knowledge of certain mathematical abstractions and inductive invariants: an interactive theorem prover with richer proof principles might be necessary for this.

But the modularity of construction permitted by partial proofs is only part of their possible value. In fact, a partial proof may have significant economic value: that is, although they are partial, they might be useful to someone who requested a (complete) proof. In particular, browsing a partial proof (using appropriate browsing tools) might convince the requesters that their proposed theorem is not, in fact, the theorem that they need: as a result, further developing of a proof could terminate. Certain details of the proposed theorem and the domain might come to light while viewing the partial proof development.

Of course, the following scenario is also likely: Acme places into the marketplace a proposed theorem (a certificate with a missing proof). A team picks up on that proof certificate and discovers a counterexample. Of course, that counterexample could be highly valued by Acme as well, at least if it can be properly communicated and checked by Acme. In this sense, counterexamples must be part of the possibilities of "proof certificates". Notice also the rich possibilities that exist between counterexamples and partial proofs: as teams build a partial proof of a proposed theorem, someone might discover that a particular open subproof actually has a counterexample. Since proofs are based on a declarative proof format (via the structural proof theory requirement of desideratum **D3**), it will be possible to use that counterexample to discover which parts of that partial proof must be retracted or revised. Allowing for partial proofs and counterexamples as part of the concept of proof certificates will allow for truly flexible and dynamic construction of proofs by distributed groups of people and tools.

3.5 Specific tasks

We list below the various major tasks that are required to realize ProofCert. Resources are requested for the following personnel.

- Three PhD students are requested: their appointments are for three years and their start and finish dates will be staggered. They will be given topics that require some foundational development. They will also work on designing prototype implementations and developing examples, both of which will serve to valid and redirect their theoretical investigations.

- Eight years of postdoc support is also requested. We hope to attract postdocs who can stay with the effort for 2 or 3 years. During years 2, 3, and 4, we plan to have two postdocs working simultaneously. We plan to attract postdocs with some maturity in theoretical material but who are looking for means for adding some system design and building effort to their background. Their involvement on ProofCert is one of building and testing the proof certificate formats and implementing proof checker and back-end proof certificate printers.

- A staff programmer is requested for years 2 - 5. S/he will help extend, polish, organize, and

T1: Design the proof certificates specifically for *L*. The goal is to define a certificate format that is flexible and allows for trade-offs between proof checking and (bounded/restricted) proof search.

T2: Build a reference proof checker for proof certificates in *L*. Start with the Teyjus virtual machine for λProlog and develop simplifications and optimizations.

T3: Use the proof certificate and checker to encode a number of proof systems (natural deduction, tableaux, resolution refutations, DPLL, *etc*). Build back-ends to popular provers.

T4: Develop the foundations for moving from *L* to *μL*: combining unification, disunification, and constrained unification. Develop an implementation of these ideas: exploit and improve on the Bedwyr model checker [4].

T5: Use the extended proof certificate and checker to encode a number of additional deductive systems including model checking (tables, simulations, winning strategies, *etc*).

T6: Incorporate partial proofs and counterexamples into proof certificates. Develop the proof theory for these additions.

T7: Build back-end proof certificate printers into a number of popular inductive theorem provers and model checkers.

T8: Build reference proof checker for *μL*. Provide at least two implementation models for it, one based on logic programming and one on functional programming

T9 Performance of checker. Proof certificates will be large logic programs. Determine ways to optimize and compile them in order to improve checking time. Develop approach that will allow us to trust such optimizations.

Figure 1: Nine majors tasks numbered **T1**-**T9** along with a brief description

Personnel	Year 1	Year 2	Year 3	Year 4	Year 5
Staff	-	**T2**	**T3**	**T7**	**T8**, **T9**
PhD1	**T1**	**T1**	**T1**	-	-
PhD2	-	**T4**	**T4**	**T4**	-
PhD3	-	-	**T6**	**T6**	**T6**
PostDocs	**T2**	**T2**,**T3**	**T3**	**T5**,**T7**	**T8**,**T9**

Figure 2: The assignment of tasks to team member and project year.

maintain the systems that will be produced by this effort. Many parts of ProofCert should succeed in attracting attention and users long before all aspects of the project are completed. Having someone making sure that research prototypes can move into a public forum will greatly help attract interest and additional support in the ProofCert effort.

The specific tasks and a time-line that associates tasks to personnel and year is given in Figures 1 and 2. We also plan to invited various colleagues and experts in the area of "computerized proof systems". In addition, we also plan to make use of several internships (via our access to resources from Ecole Polytechnique and via INRIA) in the first years of this project in order to carry out several experiments in encoding various existing proof systems.

3.6 Risk evaluation: high-risk but high-gain

The *unification of the formal method community* around a standard for logic and proof is a major benefit of this proposal. European industry will benefit greatly from the many improvements and tools that will come from this effort. Europe is also well positioned to capitalize on this line of research since there are many European academic and industrial sites that currently understand and use formal methods.

At the same time, there are a number of risks involved in this project: as we detail below, these risks offer opportunities as well.

Risk 1: *What if proof certificates are really too big to communicate?* There are at least two approaches to addressing such a problem in this setting. First, checkers could be given more resources (more memory, more processors) and the computations that are part of proof certificates can be optimized using logic programming compilation techniques. Second, one can use *reflection*: if one proves that a particular prover component is correct, its certificates do not need to be checked.

Risk 2: *How richly can we integrate model checking and theorem proving?* As was mentioned before in describing the difference between the sublogic L of μL, no one has yet developed a complete approach for a proof checker to fill in the missing information (unification in that setting is not fully understood). While we shall attempt to find a complete solution to this problem, we many also need to consider studying real world examples to see to what extent these two kinds of partial proofs overlap and cause problems.

Risk 3: *Aren't theorems and their proofs developed within theories? What about theories?* Most developments of theorems take place within *theories* (sets of assumptions). It is not enough then to relate two theorems and their proofs without accounting for differences between their theories, if any. Standard mathematical techniques for "coercing" between different theories should allow, say, results about complex numbers developed on Cartesian coordinates to be applied in a theory where they are developed with polar coordinates. While (higher-order) logic and proof theory should allow suitable abstraction mechanisms to relate theories, this important topic has not been tested in a proof theoretical setting. Addressing this risk will provide the exciting prospects of developing structured ways to handle mathematical theories both declaratively and effectively.

References

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

[2] D. Baelde. On the proof theory of regular fixed points. In M. Giese and A. Waller, editors, *TABLEAUX 09: Automated Reasoning with Analytic Tableaux and Related Methods*, LNAI 5607, pages 93–107, 2009.

[3] D. Baelde. Least and greatest fixed points in linear logic. Accepted to the *ACM Transactions on Computational Logic*, Sept. 2010.

[4] D. Baelde, A. Gacek, D. Miller, G. Nadathur, and A. Tiu. The Bedwyr system for model checking over syntactic expressions. In F. Pfenning, editor, *21th Conf. on Automated Deduction (CADE)*, LNAI 4603, pages 391–397. Springer, 2007.

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

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

[7] P. Brauner, C. Houtmann, and C. Kirchner. Principles of superdeduction. In *22th Symp. on Logic in Computer Science*, pages 41–50, 2007.

[8] G. Burel. A first-order representation of pure type systems using superdeduction. In *23th Symp. on Logic in Computer Science*, pages 253–263. IEEE Computer Society, 2008.

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

[10] G. Dowek and B. Werner. Proof normalization modulo. *Journal of Symbolic Logic*, 68(4):1289–1316, 2003.

[11] A. Felty. Encoding the calculus of constructions in a higher-order logic. In M. Vardi, editor, *8th Symp. on Logic in Computer Science*, pages 233–244. IEEE, June 1993.

[12] G. 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.

[13] J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50:1–102, 1987.

[14] J.-Y. Girard. On the unity of logic. *Annals of Pure and Applied Logic*, 59:201–217, 1993.

[15] J. 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*, LNCS 1690, pages 113–130. Springer, 1999.

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

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

[18] C. Liang and D. Miller. Kripke semantics and proof systems for combining intuitionistic logic and classical logic. Submitted, Sept. 2011.

[19] D. MacKenzie. *Mechanizing Proof*. MIT Press, 2001.

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

[21] D. Miller and V. Nigam. Incorporating tables into proofs. In J. Duparc and T. A. Henzinger, editors, *CSL 2007: Computer Science Logic*, LNCS 4646, pages 466–480. Springer, 2007.

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

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

[24] G. C. Necula and P. Lee. Efficient representation and validation of proofs. In *13th Symp. on Logic in Computer Science*, pages 93–104, 1998.

[25] S. Negri and J. von Plato. *Structural Proof Theory*. Cambridge University Press, 2001.

[26] R. 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.

[27] D. Prawitz. *Natural Deduction*. Almqvist & Wiksell, Uppsala, 1965.

[28] A. Riazanov and A. Voronkov. The design and implementation of VAMPIRE. *AI Communications*, 15(2-3):91–110, 2002.

[29] N. Shankar. Trust and automation in verification tools. In S. D. Cha, J.-Y. Choi, M. Kim, I. Lee, and M. Viswanathan, editors, *ATVA: Automated Technology for Verification and Analysis*, LNCS 5311, pages 4–17. Springer, 2008.

[30] Z. Shao. Certified software. *Communications of the ACM*, 53(12):56–66, Dec. 2010.

[31] Z. Snow, D. Baelde, and G. Nadathur. A meta-programming approach to realizing dependently typed logic programming. In *ACM SIGPLAN Conference on Principles and Practice of Declarative Programming (PPDP)*, pages 187–198, 2010.

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

[33] A. S. Troelstra and H. Schwichtenberg. *Basic Proof Theory*. Cambridge University Press, 1996.

[34] V. Voevodsky. What if current foundations of mathematics are inconsistent? Talk given at the 80th Anniversary of the Institute for Advanced Study, Sept. 2010.