Formal Methods and Software Engineering

Much of my research is focused on the use of rigorous mathematical techniques for high-assurance engineering—for ensuring that digital systems do what they are supposed to do. This goal should hardly need any defense: As our lives become increasingly dependent on digital artifacts, the reliability of these artifacts becomes of paramount importance. We must be able to trust the machines that purport to serve us. This is true both for hardware systems and (especially) for software systems, which are notoriously prone to failure. We need tools to describe and analyze such systems, and to verify that they behave properly.

Perhaps the single most powerful tool for establishing trust is the notion of formal proof. Proofs have been used to convince people that something or other is the case for thousands of years. By formally deriving a proposition $P$ from a set of premises $\Phi$, we reduce the credibility of $P$ to that of $\Phi$: If the elements of $\Phi$ are known to be true and our reasoning is sound, then we can rest assured that $P$ is true as well. In our case the set $\Phi$ might be a description—a model—of a digital system, and $P$ might be a desired property of the system. If our assumptions about the model are correct and we manage to infer $P$ from them, we can be assured that the system satisfies $P$. Note that proofs can not only establish trust, but they also have explanatory value. I.e., they not only tell us that a system behaves in a certain way, they also explain why it does so.

There are alternative, more “light-weight” techniques for analyzing digital systems, some of which (e.g., model checking) have been used with considerable success, especially in hardware design. However, such techniques are usually applicable only to finite-state systems and are better suited for detecting bugs rather than for demonstrating their absence. Formal deduction is the only known technique that can handle infinite-state systems and make strong correctness guarantees.

Unfortunately, deductive techniques have not yet lived up to their full potential. I believe there are two main culprits responsible for that. First, arcane formalisms: Mechanical proof assistants often use notations and methodologies that are highly esoteric—accessible to very few people with highly specialized training. We need more usable proof systems: languages that are more readable and writable, and implementations with simpler and more flexible interfaces. The second major problem is insufficient automation. Formal verification continues to demand an inordinate amount of work from the user, much of it in the form of tedious and routine logical manipulations. It is imperative that we lift more of this burden from the user and relegate it to the computer.

As a graduate student at MIT, I designed and built the proof system Athena [1] with a view to ameliorating these two issues. Athena places a high premium on proof readability and writability. It aims to allow for structured, high-level proofs expressed in the same style and at the same level of abstraction as the informal proofs that computer scientists and mathematicians write in practice. To that end, Athena uses a Fitch style of natural deduction instead of the more customary sequents or...
proof trees. Fitch-style proof systems are widely considered to be the most “natural” proof frameworks, meaning that formal proofs in such systems bear the closest possible resemblance to informal proofs. For that reason, such systems are the most popular pedagogical choice for teaching symbolic logic, used by numerous influential logic textbooks.

Athena formalizes Fitch-style natural deduction by means of novel block-structured syntactic constructs and so-called assumption-base semantics. An assumption base is just a set of premises: a set of propositions that we take for granted for the purposes of a given stretch of logical discourse. This is not a novel notion by itself; it is traditionally known in logic as a “context.” What is novel is the way in which Athena uses assumption bases to give formal semantics to proofs; namely, the idea that the formal meaning of a proof is a function over assumption bases. That is, the meaning of a proof is specified relative to a given assumption base, in the same way that in denotational semantics the formal meaning of an imperative program is a function over stores. This turns out to be a particularly apt viewpoint for giving a rigorous semantics to Fitch-style natural deduction. It also allows for an elegant treatment of proof equivalence and optimization. In a recent paper [5] I have developed a theory of observational proof equivalence based on assumption-base semantics, and used that theory to design and implement an array of simplifying algorithms capable of drastically reducing the size and complexity of certain first-order proofs. This work could have practical relevance for applications such as proof-carrying code (PCC).

Athena is not only a language for writing proofs; it integrates computation and deduction. It includes a higher-order functional programming language in the style of Scheme and ML. More importantly, it offers a novel notion of proof programmability. Athena proofs can be easily abstracted and turned into natural deduction algorithms—called methods—that can be dynamically applied to different arguments. To my knowledge, it is the only system that allows the user to formulate arbitrarily complicated (possibly non-terminating) proof algorithms in a Fitch style, and in such a way that guarantees soundness. This is a significant innovation because proof algorithms are much easier to express in Fitch style, for the same reasons that proofs are easier to express in that style. Indeed, user-written methods are very widely used in Athena and go a long way towards making proofs more automatic and modular. The soundness guarantee is made possible owing to Athena’s assumption-base semantics and is crucial in minimizing the overall trusted base.

The ease with which one can write proof methods in Athena has given rise to the notion of certified computation [10], where instead of writing a conventional algorithm that takes an input $x$ and produces a result $r$, one instead writes a theorem prover that takes $x$ and not only produces $r$ but also proves that $r$ is a correct result for $x$. This type of partial dynamic deductive verification is an intriguing possibility for certain types of software components because it can greatly enhance our confidence in their results; it is much easier than complete static deductive verification; and has several advantages over other techniques for software reliability such as testing. Because methods can be fluidly expressed in it and are guaranteed to produce sound results, Athena is ideally suited for writing such certifying algorithms. It has been used to implement numerous well-known algorithms as certificate-producing theorem provers, ranging from sorting algorithms to compiler optimizations, the Hindley-Milner $W$ algorithm, Prolog engines, and more. I plan to continue to explore this avenue as a means for building more robust software.

Apart from user-defined methods, another way in which Athena provides for proof automation is via the seamless integration of cutting-edge automated theorem provers (ATPs) such as Vampire and Spass, which are included as primitive methods for performing general-purpose reasoning. The combination of user-written methods with powerful external ATPs has shown remarkable potential for facilitating large verification efforts. In a recent project, we used Athena to prove the correctness

---

1 Some systems allow for proofs in Fitch style, but not for proof algorithms with guaranteed soundness.
of the key operations of Unix-style file systems [11]. By using external ATPs and by writing methods that captured recurrent patterns of reasoning, the project was successfully completed in a few weeks.

When automatic proof search fails, users are often left wondering whether their conjectures hold. 2 Proof systems should be able to provide some feedback in such cases. A particularly helpful way of doing so is by way of a countermodel—an interpretation that satisfies the negation of the conjecture. Producing such a countermodel will show that the conjecture is false, and could save the user from much wasteful chimera chasing. Athena provides facilities for automatic model generation [7] through seamless integration with Paradox, a new state-of-the-art model finder. Athena has also been integrated with Alloy [9], a model finder for relational logic. I plan to continue to integrate Athena with existing state-of-the-art systems in order to leverage useful expertise as needed. Such integration will always be seamless, meaning that the Athena user will not be aware that other systems are being invoked under the hood. In the near future I plan to connect Athena with a system such as CVC Lite or haRVey in order to increase automation for certain common types of reasoning. I also plan to augment Athena with a OBDD package; to experiment with implementing Shostak-like algorithms for combining decision procedures as Athena methods; and to increase the efficiency of model execution.

I will continue to tackle challenging verification case studies, which serve as excellent vehicles for further development and fine-tuning of Athena. One such project in the near future will be the verification of the algorithms of INS (Intentional Naming System), a scheme for resource discovery and service location in dynamic networks that was developed by Schwartz and others at MIT. Sarfraz Khurshid of UT Austin (formerly of MIT) used Alloy to analyze the main data structures and algorithms of INS, discovering and fixing several bugs in the process. Sarfraz, Darko Marinov (of UIUC) and I have been exploring the use of Athena to model and verify the corrected INS design. I am also interested in cyber-security, particularly in cryptographic protocol security. I have experimented with encoding the BAN logic of belief and authentication in Athena with a view to automating proof checking and generation in it (for the purpose of verifying protocols relative to their requirements); using countermodel generation for exposing potential attacks to the protocols; and partially mechanizing the process of idealizing protocols as formal objects. In addition, I plan to connect Athena with an implementation of Nancy Lynch’s IO automata language.

Athena is only one member of a whole class of languages known as denotational proof languages (DPLs) [2, 3, 6], which share certain key characteristics such as assumption-base semantics. My dissertation [2] introduced DPLs for various logics and studied them in the general setting of the $\lambda\phi$-calculus, an abstract formal system that is to DPLs what the $\lambda$-calculus is to functional languages.

Going further back, for my Master’s thesis at MIT [4, 15] (with David McAllester as my advisor) I designed a conservative termination analysis for verifying that an algorithm halts on all inputs—an important problem both in verification and in program analysis. I then implemented a simple programming language and tested my method extensively, verifying the termination of dozens of well-known algorithms in less than one minute. Most of my research since then has followed the same pattern of mixing science (analysis) and engineering (synthesis). I enjoy both, and view them as equally important, as I have found that the scientific process of distilling an idea into fundamental principles is at its best when informed by engineering considerations of practicality. Building an efficient implementation, in particular, has always been an excellent way for me to validate and refine my ideas. I intend to continue this practice in the future.

2Theorem proving in interesting logics is undecidable, so failure to find a proof does not mean one does not exist.
Programming Languages

I have had a long-standing interest in programming language theory and implementation. My background in programming language semantics was crucial in the successful design of Athena and other DPLs. Likewise, I had to learn a lot about implementation techniques for functional languages in order to build a robust implementation of Athena. As a PostDoc at MIT, I was the advisor of Teodoro Arvizo and oversaw his Master’s thesis on the design and implementation of a virtual machine for Athena-like DPLs [12]. I also advised Melissa Hao, whose thesis work explored the use of Athena for proving the correctness of dataflow analyses [13]. (In fact, provably correct dataflow analyses and other static program analyses in the context of an extensible optimizing compiler were the initial impetus for the development of Athena.) In a similar vein, I collaborated with Darko Marinov for his Master’s thesis at MIT [14], which used Athena to implement the “credible compilation” scheme of Martin Rinard.

More recently, I collaborated with Alex Salcianu of MIT on a project that uses Athena to prove the correctness of intra-procedural dataflow analyses in an abstract-interpretation framework [18]. This is an ongoing project; we hope to extend and improve our system in the near future, and to increase proof automation. I also plan to collaborate with Olin Shivers and Andrew Hilton of Georgia Tech, who have also been using Athena to reason about dataflow analyses. In additional programming-language-related work, David Musser of RPI has been using Athena to reason about generic software [16], and we intend to collaborate on using Athena to verify the correctness of parts of the C++ STL library, which he designed and implemented [17]. I also plan to informally advise Aytekin Vargun, who is doing his PhD at RPI under David Musser on “Code-Carrying Proofs” and is using Athena for his implementation.

Artificial Intelligence

I am actively interested in several areas of Artificial Intelligence. In a recent paper I explored epistemic issues in multi-agent systems [8], using McCarthy’s famous wise-men puzzle as a springboard. In the summer of 2004 I was supported by a grant from the US Air Force Research Laboratory to investigate the interplay of deontic notions, rational decision making, and game theory in the context of wargaming; I am currently working on a paper with members of that Lab on this topic. In the fall of 2004 I was a co-PI and am currently a co-recipient of a $1.2 million DARPA grant for a machine-learning project that will develop a notion that we call “poised-for learning.” In the context of that project, we are building a system capable of learning astronomy by reading elementary astronomy textbooks, including textual and diagrammatic information, and answering simple queries. I am also currently working with two other AI researchers on an article (forthcoming in AI Magazine) entitled “Logic-based AI for the new millennium.”

Finally, I have lately become very interested in all aspects of intelligent planning. I am working with three other AI researchers on building an intelligent assistant for effects-based operations extended by deep adversarial modeling—a system designed to help a human planner or strategist formulate and decide between plans, which, when implemented in the actual battlespace, defeat the opposing force. I am also becoming interested in the interplay between planning and theorem proving. I believe that with current advances in ATP and first-order model-generation technology, it may be possible to rehabilitate the original conception of planning in the situation calculus. I would also like to explore the possibility of using user-defined methods to do planning in applications that demand domain-specific knowledge, where general-purpose SAT-based planners tend to perform poorly.
References


