Research Statement

Adithya Murali
University of Illinois Urbana-Champaign
https://muraliadithya.github.io

The overarching focus of my research is to democratize software verification, enabling programmers who are not verification experts to verify their code. My thesis work advances this vision by replacing human creativity currently required in software verification using data-driven logic learning. I combine ideas from programming languages, formal methods, and (symbolic) machine learning.

The desire for bug-free software is as old as computer science. However, we still do not have tools that are powerful enough to ensure software reliability in practice. With the privacy of personal data [14], financial security [15], and physical safety [16, 17] on the line, software testing is not enough. Formal Verification shows a way forward, by describing the desired behavior of a piece of software mathematically and writing a computer-checkable *proof* that the code meets the given specification.

Automation is critical to the **democratization of verification**, i.e., enabling programmers who are not verification experts to verify their software. However, automated verification is at an impasse, both in terms of technical advances as well as practical adoption. This is because **automation for software verification requires crucial technical intervention from verification experts to work**. The requirement for key technical help prevents democratization— very few engineers know the internals of verification tools, and normal programmers are unable to use them. Indeed, existing successes of automated verification (e.g., device drivers [18], operating system kernels [19], or fault-tolerant distributed systems for the cloud [20, 21]) have required significant verification expertise. Furthermore, these technical interventions are challenging and frustrating to provide even for experts! Early in my Ph.D., I set out to build a **verified blockchain** [1][1] and encountered this frustration firsthand, and it has motivated my research in removing these obstacles.

My thesis argues that existing automated reasoning paradigms practice a two-step approach:
(1) A human, typically an expert, uses their creativity (💡) to formally state their high-level arguments about a system's behavior or correctness, and
(2) An automated reasoning engine mechanically (⚙️) verifies the stated arguments.

However, this approach is complex, tedious, and makes automated reasoning inaccessible.
I call this seemingly inescapable requirement for expert ingenuity the **creativity gap** in automated reasoning.

My research identifies the precise creativity gaps in automated reasoning and bridges these gaps using logic learning.

The two-step (💡) + (⚙️) view of reasoning paradigms is a powerful lens that applies to many reasoning problems. For example, in program verification, the expert breaks down the problem into modular specifications in the form of contracts for individual methods and loop invariants, and then an automated engine effectively verifies the methods. Another example is theorem proving, where the expert breaks down a theorem into intermediate lemmas such that SMT solvers [22, 23] (which are automated logic engines) can verify that each step follows from the previous steps. This view even has **applications beyond Programming Languages**. For example, in mathematical and scientific reasoning, humans come up with axioms or laws to characterize a domain and then reason using those laws. As AI moves towards tackling these fields, automatically discovering such laws is a creative task that we must conquer.

My research offers two key insights:

I. **Identifying Creativity Gaps:** Practical problems require expressive reasoning domains which are typically *incomplete* (i.e., can never be fully automated in theory), forcing automation to rely on heuristics. When the heuristics get stuck or simply fail, experts are forced to turn to creativity. We need to understand the shortcomings of these heuristics to identify *how* to automate human help. My work here contributes a set of **foundational theoretical tools and results** that *precisely* characterize the reasoning power of several

[1]Citations appearing in green refer to my publications.

popular heuristics used in verification [OOPSLA'23: 2, OOPSLA'22: 3, POPL'20: 4]. These are the first results that characterize the exact role of human intervention in using automated reasoning for software verification. My work has been accepted to be presented as a tutorial at POPL 2024 [5].

I have also used these insights to fundamentally re-think the field. My recent work submitted to PLDI 2024 [6] develops **new verification paradigms** with the goal of democratizing verification by providing reliable automation with minimal and simple human help.

II. **Bridging Creativity Gaps using Learning:** The second insight is that the solution to automating the creativity gap lies in learning-based synthesis. My thesis work focuses in particular on bridging the creativity gaps in automated reasoning using **data-driven logic learning**, namely, learning logical formulas from examples [OOPSLA '22: 3, OOPSLA '22: 7, POPL'20: 4, IJCAI'22: 8]. My work tackles complex logics containing quantification and complex example data in the form of first-order models. This is a challenging open problem, and my work here contributes new learning frameworks and algorithms that are applicable to many creative tasks. I have built open-source tools that implement these advances.

Apart from my thesis research, I have worked broadly on program verification and program synthesis. My work covers the spectrum from theory to practice and has consistently appeared at the most competitive venues, including POPL [4], OOPSLA [2, 3, 7], CAV [9], ESOP [10], FMCAD [1], TOPLAS [11], and IJCAI [8].

Thesis Research

The table on the right provides an overview of the various creative tasks that my thesis work automates using logic learning. Asterisks link to ongoing efforts that I describe in the section on future work.

Problem Domain	Creative Task
Program Verification (with recursive definitions)	Inductive Lemmas [A,B]
Program Verification (with intrinsic specifications)	Ghost Map Updates [C,*]
Theorem Proving	Axioms [D]
Verification with large libraries	Library Specifications [*]

* Future Work

(A) Identifying Creativity Gaps in Program Verification Practical verification problems involve complex specifications that are stated in *incomplete* logics. This means that no technique can prove all valid programs, so developers of verification tools often end up building a set of complex and opaque heuristics. Consequently, when the heuristics fail, users do not know *why* they fail. This is a source of great frustration even for expert users.

Motivated by this observation, I study a **popular heuristic for verifying functional programs called USMT**: Unfolding definitions followed by SMT solving. This is a very effective heuristic that has been used in many verification tools [24, 25, 26] over the years. However, despite its efficacy, USMT fails on very simple problems and users did not know why— until now.

In our work [2] we develop a **new logic called FLUID** (First-Order Logic Under Inductive Definitions) which provides a sound abstraction of the verification problems posed by users. We find that **USMT is complete for FLUID**, i.e., USMT works *precisely* when the FLUID abstraction is provable, and fails when the abstraction is too weak. Further, we show that when USMT fails, there always exist spurious counterexamples to the FLUID abstraction called *rogue nonstandard models*. This shows that the role of human creative help (💡) in aiding tools is to provide hints (in the form of inductive lemmas) that eliminate such spurious counterexamples.

This work goes **from practice to theory**. It is one of only two such results [2] [27] in the literature to precisely characterize heuristics used in verification. On the practical side, our result **informs the design of verification tools** that deploy predictable techniques and provide precise, useful feedback to the user when they fail. I am working with developers of verification tools to improve their design using the insights from this work.

(B) Filling Creativity Gaps in Program Verification The second major thrust of my thesis is the automation of human creative effort in verification. In this direction, I address the creativity gap that arises when reasoning using USMT in the context of imperative programs over dynamic heaps [3]. Verification problems in this setting are stated in a very general and powerful logic: First-Order Logic with Recursive Definitions. A previous result [27] on identifying creativity gaps for this problem shows that **the role of human help is *precisely* the creative task of providing inductive lemmas** that bridge the gaps left by the heuristic. In other words, the verification problem splits into two precise pieces: (1) creatively (💡) come up with inductive lemmas such that (2) USMT can mechanically (⚙️) verify the given program using the lemmas. This motivated me to investigate automating **inductive lemma synthesis**.

We develop in the resulting work [3] a **new data-driven logic learning framework called FOSSIL** (a First-Order Solver with Synthesis of Inductive Lemmas) that is based on learning complex quantified lemmas from **novel counterexample first-order models**. This work is a **landmark that contributes many firsts**: we formalize for the first time the idea of counterexample models, create several new kinds of counterexamples beyond the simple positive and negative examples known in the literature, and show how to design frameworks for doing what we call 'Model-Guided Synthesis'— a new and powerful synthesis approach. Unlike synthesis problems in literature, the salient feature of this creative task is that the **synthesis problem has no logical specification!** To handle this challenge our technique uses several different carefully designed algorithmic components in concert with a logic learner to gradually *elicit* valid and useful lemmas. My collaborators and I are currently using this framework to build a practical tool [13] for verifying heap manipulating programs where verification is **completely automated**.

(C) Rethinking Verification Paradigms: Predictable Verification using Intrinsic Specifications My study into the opaqueness of verification tools motivated me to rethink the problem through the lens of combining creativity (💡) and mechanical reasoning (⚙️). In recent work [6] I define the **new problem** of **predictable verification** seeking a verification paradigm such that (1) the user is asked to provide upfront a fixed set of annotations *independent* of the underlying verification mechanism, and (2) the verification, given the annotations, can be completely automated. In other words, we want *no more frustrating proof engineering* (e.g., inductive lemmas, instantiation triggers, etc.)! This would yield a framework where the user simply expresses arguments for the correctness of their programs at a high level, and then an automated engine checks their work effectively.

We create the **Intrinsic Specifications (IS) paradigm** for predictable verification based on two innovations. First, where specifications generally involve recursively defined functions that compute over unbounded regions of the program state, we formulate novel *intrinsic* specifications that talk about program states *locally* without using recursive definitions. Intuitively, an intrinsic specification says what a program state looks like when one is standing 'inside' it, whereas recursive definitions offer a 'global' view of the state. The second innovation is a **new verification methodology** that allows the user to argue the correctness of intrinsically defined specifications by augmenting the program with *ghost code* (code which does not execute but provides a proof of the analyzed program in a computational way). Verifying programs augmented with ghost code against intrinsic specifications is in fact *decidable*! It can therefore be automated very effectively using SMT solvers [22, 23].

IS provides users a simple, programmatic technique and offers a predictable verification experience. We used IS to verify 50 programs over complex data structures such as **overlaid binary search trees** used in a **Linux I/O scheduler**. I am actively working on using IS to verify programs that manipulate concurrent data structures.

(D) Filling Creativity Gaps by Discovering Laws One of the most beautiful aspects of human intelligence is our ability to **formulate abstract rules for navigating new domains**. This is especially prominent in mathematical and scientific inquiry, where the scientist may even understand the phenomenon at hand precisely, but they want to analyze it at a higher level of abstraction and draw interesting conclusions by operating within the abstraction. For example, doctors have a detailed understanding of the human lung and its dynamics, but when analyzing X-rays of lungs they talk about 'masses' or 'honeycombing' in the images and use these concepts in their diagnosis. Automating this creative aspect of reasoning encompasses a great many technical problems such as acquiring relevant concepts, forming compositional abstractions, learning to reason within the abstract space, and continually learning better abstractions with experience.

As a first step towards this larger goal, I study in my work [7] the problem of **axiom synthesis for reasoning domains in programming languages** like Kleene Algebras. Although the problem of finding axiomatizations is centuries old, I define in this work a **new formulation of axiom synthesis** as a computational problem. This novel problem definition opens up the use of computational tools for finding axioms in many complex domains, including those where the objects of study may not even have a logical specification. I develop **Learning-based Axiom Synthesis**, a framework based on data-driven logic learning for synthesizing axioms. Similar to my work on lemma synthesis for verification [3], the **synthesis problem has no logical specification!** The ability to elicit logical formulas desired in creative tasks without a logical specification is a distinguishing feature of my work.

We used the learning-based framework to automatically synthesize axioms for modal logics and Kleene algebras, finding axioms that were hitherto only known to have been formulated by expert logicians! Our work has received a lot of enthusiasm from the community, and I am currently pursuing several multi-institute collaborations to automatically discover axioms for domains that are tedious or hard to axiomatize manually.

Future Outlook

My long-term vision is to build machines that learn and reason in synergy. This is a vast space where my thesis explores the specific challenge of automating the creative help needed in software verification using learning. In the future, I want to work more broadly on applications that require inter-operating between learning and reasoning techniques, by (a) working on PL applications to fill creativity gaps that are filled today by humans using learning and, (b) working on AI applications to incorporate reasoning into Deep Learning-based technology. As a forward-looking programming languages researcher, I will develop techniques at the intersection of programming languages and program synthesis/machine learning to advance this vision.

I outline below some directions that I plan to pursue in the short term. The unifying theme in these directions is the use of logic learning and automated reasoning, which I am uniquely positioned to apply successfully owing to my research experience.

Democratizing Verification Developing verified systems typically requires large teams consisting of many verification experts as well as systems programmers. I want to enable systems programmers to be able to verify their software without help from verification experts. While systems programmers are already able to think about and argue the correctness of their software at a high-level, they often need verification experts to help them convince a verification engine (a.k.a "proof engineering") that their arguments are correct.

In my thesis work, I develop the Intrinsic Specifications (IS) paradigm [6] which allows a programmer to verify their program by simply writing a special kind of code called *ghost code*. Crucially, the paradigm requires no proof engineering! However, a potential obstacle is the complexity of the required ghost code. I will use program synthesis and machine learning techniques to automatically synthesize ghost code, lowering the cognitive burden involved in verification for systems programmers.

A particular domain where lowering the cognitive burden is of great urgency is the verification of smart contracts, which can be written by systems programmers with no verification expertise. With active and dedicated malicious actors, unsecure code always leads to massive financial losses with no real recourse. I believe that my experience with developing a verified blockchain will enable me to understand the technical challenges deeply and develop effective solutions. I intend to collaborate with systems and security researchers in these efforts.

Learning for Scalable Automated Verification One of the primary challenges in automated verification today is scale. Industrial code can be quite large, and automated logic solvers are unable to handle the size of the verification queries. However, there are several application domains where the size of the code written by the programmer (i.e., the "client" code) is small, but the code may call several standard library functions which can have large implementations. The challenge is that the library functions are themselves unverified and have no formal specifications, which seemingly makes this setting just as intractable as large code.

In ongoing work, my collaborators and I are investigating verification *modulo tested libraries*, a new approach that disentangles reasoning about the library from reasoning about the client code. The key idea is to use learning to infer specifications for libraries that are correct with a high degree of assurance, and then use these inferred specifications to verify the relatively small client code. The assurance for library functions is provided using a test generator, which scales to large code extremely well. I will use the algorithms for logic learning developed in my thesis to infer specifications, adapting them to learning using test generators. This is a promising research direction with manifold applications such as mobile applications, web applications, IoT, etc.

Concept Learning in the Large My thesis work argues that learning concepts (represented as logical formulas) from examples can automate a vast space of creative tasks. However, state-of-the-art techniques for logical concept learning (a) do not scale, even to thousands of examples, and (b) cannot handle noise. I plan to work on fundamental techniques for improving logical concept learning to overcome these challenges. I am currently pursuing several different approaches to solve this problem, including neuro-symbolic techniques.

Language and Logic The growing capabilities of Large Language Models (LLMs) have enabled computers to effectively interpret natural language utterances across many applications. Consequently, it appears highly plausible that we may be able to use natural language in place of formal logic for reasoning. This is an emerging area with a rich set of problems.

One application in this area that I plan to tackle is natural language theorem proving using automated logic engines. Humans write prove theorems by writing down a set of high-level arguments in natural language. These arguments are such that other humans can read the arguments and easily see that each step follows from the previous steps. However, theorem provers today require the user to write a proof as a set of *tactics* specific to the prover.

Contemporary work on machine learning for theorem proving builds on this interface, developing machine learning techniques that try to generate the relevant tactics from natural language text. These techniques perform better with more data and better models, but they can also fail very easily. For example, no known technique achieves 100% accuracy on well-established datasets consisting of high school mathematics problems.

In future research, I plan to investigate augmenting LLMs with automated logic solvers to prove theorems. This will enable users to simply prove their theorems in natural language, and perhaps even allow LLMs to automatically generate natural language proofs (which are then formally verified). I plan to collaborate with researchers in machine learning and education in these efforts. More generally, I plan to collaborate with machine learning researchers to use LLMs augmented with logic solvers to tackle more ambitious goals like ensuring the coherence and consistency of conversational AI models.

References

My Work

[1] Faria Kalim, Karl Palmskog, Jayasi Mehar, **Adithya Murali**, Indranil Gupta, and P. Madhusudan. "Kaizen: Building a Performant Blockchain System Verified for Consensus and Integrity". In: *2019 Formal Methods in Computer Aided Design (FMCAD)*. 2019, pp. 96–104. DOI: 10.23919/FMCAD.2019.8894248.

[2] **Adithya Murali**, Lucas Peña, Ranjit Jhala, and P. Madhusudan. "Complete First-Order Reasoning for Properties of Functional Programs". In: *Proc. ACM Program. Lang.* 7.OOPSLA (Oct. 2023). DOI: 10.1145/3622835. URL: https://doi.org/10.1145/3622835.

[3] **Adithya Murali**, Lucas Peña, Eion Blanchard, Christof Löding, and P. Madhusudan. "Model-Guided Synthesis of Inductive Lemmas for FOL with Least Fixpoints". In: *Proc. ACM Program. Lang.* 6.OOPSLA (Oct. 2022). DOI: 10.1145/3563354. URL: https://doi.org/10.1145/3563354.

[4] Umang Mathur, **Adithya Murali**, Paul Krogmeier, P. Madhusudan, and Mahesh Viswanathan. "Deciding Memory Safety for Single-Pass Heap-Manipulating Programs". In: *Proc. ACM Program. Lang.* 4.POPL (2020). DOI: 10.1145/3371103. URL: https://doi.org/10.1145/3371103.

[5] **Adithya Murali** and Madhusudan Parthasarathy. *Automated Datastructure Verification using Unfoldings and SMT Solving: Foundations and FO-Completeness (POPL 2024 - TutorialFest) - POPL 2024*. publisher: POPL 2024. URL: https://popl24.sigplan.org/details/POPL-2024-tutorialfest/10/-Automated-Datastructure-Verification-using-Unfoldings-and-SMT-Solving-Foundations-a (visited on 11/24/2023).

[6] **Adithya Murali**, Cody Rivera, and P. Madhusudan. "Predictable Verification using Intrinsic Definitions of Datastructures". In: *Under Submission* (2023).

[7] Paul Krogmeier, Zhengyao Lin, **Adithya Murali**, and P.Madhusudan. "Synthesizing Axiomatizations Using Logic Learning". In: *Proc. ACM Program. Lang.* 6.OOPSLA (Oct. 2022). DOI: 10.1145/3563348. URL: https://doi.org/10.1145/3563348.

[8] **Adithya Murali**, Atharva Sehgal, Paul Krogmeier, and P. Madhusudan. "Composing Neural Learning and Symbolic Reasoning with an Application to Visual Discrimination". In: *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*. Ed. by Luc De Raedt. Main Track. International Joint Conferences on Artificial Intelligence Organization, July 2022, pp. 3358–3365. DOI: 10.24963/ijcai.2022/466. URL: https://doi.org/10.24963/ijcai.2022/466.

[9] Paul Krogmeier, Umang Mathur, **Adithya Murali**, P. Madhusudan, and Mahesh Viswanathan. "Decidable Synthesis of Programs with Uninterpreted Functions". In: *Computer Aided Verification*. Ed. by Shuvendu K. Lahiri and Chao Wang. Cham: Springer International Publishing, 2020, pp. 634–657. ISBN: 978-3-030-53291-8.

[10] **Adithya Murali**, Lucas Peña, Christof Löding, and P. Madhusudan. "A First-Order Logic with Frames". In: *Programming Languages and Systems*. Ed. by Peter Müller. Cham: Springer International Publishing, 2020, pp. 515–543. ISBN: 978-3-030-44914-8.

[11] **Adithya Murali**, Lucas Peña, Christof Löding, and P. Madhusudan. "A First-Order Logic with Frames". In: *ACM Trans. Program. Lang. Syst.* 45.2 (May 2023). ISSN: 0164-0925. DOI: 10.1145/3583057. URL: https://doi.org/10.1145/3583057.

[12] **Adithya Murali** and P. Madhusudan. "Delta Logics: Logics for Change". In: *Under Submission* (2023).

[13] **Adithya Murali**, Hrishikesh Balakrishnan, Aaron Councilman, and P. Madhusudan. "Automating Program Verification for Frame Logic". In: *Under Submission* (2023).

Other References

[14] Vidhi Doshi. "Analysis | A security breach in India has left a billion people at risk of identity theft". en-US. In: *Washington Post* (Dec. 2021). ISSN: 0190-8286. URL: https://www.washingtonpost.com/news/worldviews/wp/2018/01/04/a-security-breach-in-india-has-left-a-billion-people-at-risk-of-identity-theft/ (visited on 11/12/2023).

[15] Scott Matteson. *Report: Software failure caused $1.7 trillion in financial losses in 2017.* en-US. Jan. 2018. URL: https://www.techrepublic.com/article/report-software-failure-caused-1-7-trillion-in-financial-losses-in-2017/ (visited on 11/12/2023).

[16] California Department of Motor Vehicles. *Autonomous Vehicle Collision Reports.* en-US. URL: https://www.dmv.ca.gov/portal/vehicle-industry-services/autonomous-vehicles/autonomous-vehicle-collision-reports/ (visited on 11/12/2023).

[17] Bryan Pietsch. "2 Killed in Driverless Tesla Car Crash, Officials Say". en-US. In: *The New York Times* (Apr. 2021). ISSN: 0362-4331. URL: https://www.nytimes.com/2021/04/18/business/tesla-fatal-crash-texas.html (visited on 11/12/2023).

[18] Thomas Ball, Byron Cook, Vladimir Levin, and Sriram Rajamani. *SLAM and Static Driver Verifier: Technology Transfer of Formal Methods inside Microsoft.* Tech. rep. MSR-TR-2004-08. Microsoft Research, Jan. 2004, p. 22. URL: https://www.microsoft.com/en-us/research/publication/slam-and-static-driver-verifier-technology-transfer-of-formal-methods-inside-microsoft/.

[19] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. "SeL4: Formal Verification of an OS Kernel". In: *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles.* SOSP '09. Big Sky, Montana, USA: Association for Computing Machinery, 2009, pp. 207–220. ISBN: 9781605587523. DOI: 10.1145/1629575.1629596. URL: https://doi.org/10.1145/1629575.1629596.

[20] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. "How Amazon Web Services uses formal methods". In: *Communications of the ACM* (2015). URL: https://www.amazon.science/publications/how-amazon-web-services-uses-formal-methods.

[21] Neha Rungta. "A billion SMT queries a day". In: *CAV 2022.* 2022. URL: https://www.amazon.science/publications/a-billion-smt-queries-a-day.

[22] Leonardo De Moura and Nikolaj Bjørner. "Z3: An Efficient SMT Solver". In: *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems.* TACAS'08. Budapest, Hungary: Springer-Verlag, 2008, pp. 337–340. ISBN: 978-3-540-78799-0.

[23] Clark Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanović, Tim King, Andrew Reynolds, and Cesare Tinelli. "CVC4". In: *Computer Aided Verification.* Ed. by Ganesh Gopalakrishnan and Shaz Qadeer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 171–177. ISBN: 978-3-642-22110-1.

[24] Robert S. Boyer and J. Strother Moore. *A Computational Logic Handbook.* USA: Academic Press Professional, Inc., 1988. ISBN: 0121229521.

[25] Patrick M. Rondon, Ming Kawaguci, and Ranjit Jhala. "Liquid Types". In: *SIGPLAN Not.* 43.6 (June 2008), pp. 159–169. ISSN: 0362-1340. DOI: 10.1145/1379022.1375602. URL: https://doi.org/10.1145/1379022.1375602.

[26] Jad Hamza, Nicolas Voirol, and Viktor Kunčak. "System FR: Formalized Foundations for the Stainless Verifier". In: *Proc. ACM Program. Lang.* 3.OOPSLA (Oct. 2019). DOI: 10.1145/3360592. URL: https://doi.org/10.1145/3360592.

[27] Christof Löding, P. Madhusudan, and Lucas Peña. "Foundations for natural proofs and quantifier instantiation". In: *PACMPL* 2.POPL (2018), 10:1–10:30. DOI: 10.1145/3158098.