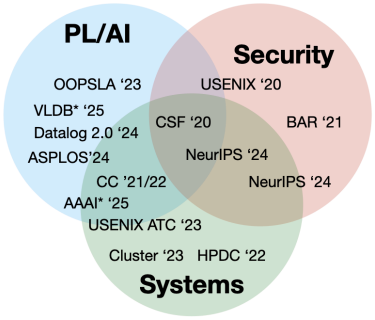
Research Statement

Kristopher Micinski, Syracuse University

In my research, I scale state-of-the-art reasoning systems to unprecedented levels of expressivity and parallelization to enable the next frontier in program analysis, security, and symbolic artificial intelligence. Throughout my career, I have published top papers in a breadth of areas, inspired by an overarching goal: *achieve orders-of-magnitude algorithmic and implementation breakthroughs necessary to understand (the security of) production software systems.* During my PhD, I worked on static and dynamic analysis for relational security properties (information flow), collaborating on foundational work (e.g., HyperLTL [1]) and using techniques such as abstract interpretation and symbolic execution to analyze security properties of production Android applications [6, 9, 10]. At Syracuse, I have built a large and diverse research group which works to build state-of-the-art reasoning engines for expressive declarative analytics tasks with applications to security, program analysis, and symbolic AI. My students and I target the most selective venues (including papers at HPDC, ASPLOS, and USENIX), and I have raised over $2.1M of federal research funding to support my work from NSF, DARPA, and NSA.



The venn diagram on the right categorizes my recent research interests, which apply methods from programming languages and symbolic AI to problems in security and systems. Circles contain papers accepted since starting at SU. At a high level, my main "publishing home" is in programming languages and symbolic AI (SIGPLAN, AAAI, and sometimes SIGMOD). I draw upon a background in security, wherein I developed static/dynamic analyses to reason about (low-level) code in production systems [18, 8, 6]. I was frustrated with the poor implementation story for the rich work in formal security (such as hyperproperties, information flow, and relational reasoning), which motivated my switch to developing the world's fastest Datalog engines [11, 4, 12, 5, 2]; implementing SOTA Datalog engines on modern, leadership-class systems has required serious work in computer systems [15, 14, 3]. In tandem, I have been collaborating with the US Laboratory for Physical Sciences (LPS) on novel methods for binary analysis, culminating in two NeurIPS '24 (datasets and benchmark track) papers and a two-week-long NSA workshop leveraging my work in developing the largest open-source datasets for binary analysis [7, 13]. My long-term vision is to use my state-of-the-art reasoning engines to revolutionize the way we reason about security properties of our code, leveraging rich logical reasoning to interactively understand (security, or otherwise) properties of low-level code. My in-progress NSF CAREER proposal, "Declarative Reverse Engineering via Analysis-View Maintenance," presents a combined vision for research, teaching, and outreach.

The World's Fastest Reasoning Engines For the past several years, *I have led the design of the fastest and most scalable Datalog engines on CPUs, GPUs, and supercomputing clusters [17, 11, 15, 14, 4, 16].* Datalog is a logic programming language which enables solving complex reasoning problems by allowing users to write declarative rules which sketch an acceptable space of solutions; intuitively, Datalog extends SQL with efficient recursive rules. The growing excitement in Datalog—a language which has become the de-facto standard for complex problems in static analysis (the context-sensitive Java analysis DOOP), security (the ddisasm binary disassembler), ontological reasoning (the industrial RDFox)—is due to its unique price point, being expressive enough for solving complex problems (*e.g.*, whole-program static analyses, drug discovery, and graph analytics) in a concise, obviously-correct manner while simultaneously exposing massive potential for parallelization. Unfortunately, current-generation CPU-based Datalog solvers reach their best performance around just 8 cores. My team and I have made several distinct orders-of-magnitude

$$
\begin{aligned}
&\ldots \\
\mathit{MemoryAlias}(x,w) \;&\leftarrow\; \mathit{dereference}(y,x),\\
&\qquad\quad \mathit{ValueAlias}(y,z),\\
&\qquad\quad \mathit{dereference}(z,w).\\
\mathit{ValueAlias}(x,y) \;&\leftarrow\; \mathit{ValueFlow}(z,x),\\
&\qquad\quad \mathit{MemoryAlias}(z,w),\\
&\qquad\quad \mathit{ValueFlow}(w,y).\\
\mathit{ValueFlow}(y,x) \;&\leftarrow\; \mathit{assign}(y,x).\\
\mathit{ValueFlow}(x,x) \;&\leftarrow\; \mathit{assign}(x,y).
\end{aligned}
$$

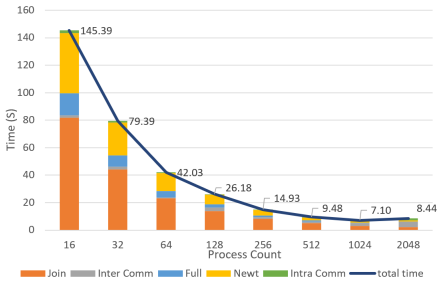


Table 1: GPULog vs. Soufflé

Prog.	Time (s)		Δ
	GPULog	Soufflé	
Httpd	**1.33**	49.48	**37.2x**
Linux	**0.39**	13.44	**34.5x**
Postgres	**1.27**	57.82	**44.9x**

Figure 1: (left) A fragment of a context-sensitive points-to analysis in Datalog, (center) scaling analysis of the Linux kernel to 1k cores of the Theta supercomputer via Slog, (right) the same analysis in GPULog (ASPLOS '25), which beats optimally-configured (16 threads) CPU-based competitor (Soufflé) by 45×.

improvements in state-of-the-art Datalog engines, publishing key advancements in its expressivity [4, 11, 5] and parallelization on modern hardware [3, 17, 15, 16]; our most recent engine (VFLog, AAAI submission), a columnar GPU-based Datalog, beats the best CPU-based competitor by up to 200×. This work was initially funded by a $400k DARPA grant (V-SPELLS), followed by a one-year NSF PPoSS planning grant; last year, we received a $5M PPoSS LARGE grant ($1M to me), funding the project from 2023–2028.

As a motivating application demonstrating the power and scalability of Datalog, Figure 1 gives a representative set of rules from a context-sensitive program analysis; the center and right of the figure show two of my systems, Slog and GPULog applying this analysis to the Linux kernel (center) and scaling to leadership-class supercomputers and cutting-edge GPUs (the NVIDIA H100). Compared to traditional static analysis systems (*e.g.*, WALA, consisting of >100k lines of code), Datalog enables implementing rich, context-sensitive analysis of production languages in orders-of-magnitude less code (*e.g.*, DOOP handles all of Java with just hundreds of rules). The declarative, logic-defined nature of Datalog is a perfect match for applications which involve logical reasoning, including symbolic AI (*e.g.*, provenance and neurosymbolic AI), binary analysis (e.g., the SOTA Datalog-based disassembler ddisasm), business analytics, and even medical reasoning (our NSF PPoSS large aims to apply Datalog to drug discovery and expert systems).

My group has developed the most scalable Datalog engines to date, and has applied our engines to achieve the fastest and most scalable static analyses in history. The center of Figure 1 shows results of Slog, the first data-parallel Datalog engine built on MPI, which we built using innovations in all-to-all communication [4, 3]. My former PhD student Arash implemented an award-winning CPU-based Datalog engine, Ascent, embedded in Rust via procedural macros. Ascent includes the "Bring Your Own Data Structures" technique, enabling Datalog engines to harmoniously cooperate with user-provided data structures; this work won distinguished paper at OOPSLA '23. Ascent beats competitor systems in both speed and scalability, and was used to implement (collaboratively with Galois) the yapall analysis of LLVM; our experiments of yapall scaled a 1-context-sensitive analysis of httpd to 32 threads of a large server. Last, my student Yihao and I have developed the Hash-Indexed Sorted Array (HISA), a GPU-based data structure for implementing Datalog on the GPU. Our system GPULog (right side of Figure 1, to appear at ASPLOS '25), demonstrates the first-ever GPU-based Datalog engine to beat a SOTA CPU-based competitor. Table 1 shows a comparison of GPULog against Soufflé: GPULog beats Soufflé by up to 45×.

Declarative Reverse Engineering via Analysis-View Maintenance Reverse engineering (RE) is a ubiquitous aspect of software understanding. For example, in binary reverse engineering, an expert analyst interactively discovers ("reverses") properties of a low-level software artifact such as a Windows PE or ELF binary. Recent observational studies of RE professionals (my USENIX '20 paper [18]) show that they

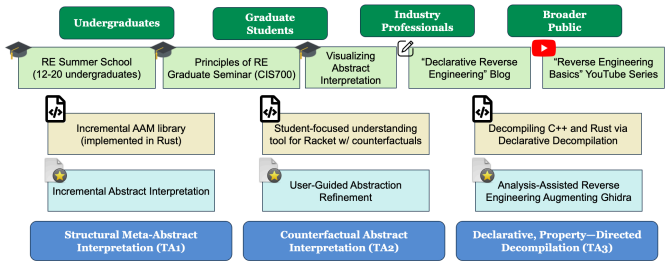


Figure 2: NSF CAREER High-Level Vision

Table 2: Assemblage (NeurIPS '24 D&B) vs. other binary corpora. OSs include Linux (L)/Windows (W). Toolchains include GCC (G), clang (C), and MS Visual C++ (M).

Dataset	Binaries (#)	Available data format	Projects (#)	OS		Toolchain		
				L	W	G	C	M
Ubuntu	87,853	Binaries	22,040	✓	✓	✓		
PSI	188,253	N/A	14,000	✓	U	U		
DIRE	164,632	Binaries	U	✓	U	U		
BinKit	243,128	Binaries	51	✓		✓	✓	
BinaryCorp	48,130	Binaries	9,819	✓	✓	✓		
BinBench	1,127,479	Binaries	U	✓	✓	✓		
Assemblage	**1,536,171**	**Binaries +PDB**	**220,792**	✓	✓	✓	✓	✓

employ iterated rounds of hypothesis formation and falsification, assisted via a variety of automated and ad-hoc methods. Beyond binary RE, programmers employ RE skills throughout their career, in debugging their own code, understanding their collaborators' code, or integrating with APIs and frameworks.

Having spent the bulk of my career developing large-scale analyses of binaries, I have observed the unique challenges and pains of RE firsthand. Based on this experience, my pending NSF CAREER grant proposes a new methodology for reverse engineering, wherein the artifact under analysis is decompiled to an arbitrarily-higher-level representation via declarative rules; decompilations are then incrementally-maintained views of tiers of analyses, interspersed with refinements (assumptions, sources of well-understood unsoundness). Each analysis tier constitutes a decompilation into a higher-level artifact, formally implemented as an aggregation (post-processing of) an abstract interpretation. In my approach, we view reverse engineering as an iterative kind of *reverse programming*, consisting of rounds of saturated reasoning (guided by my advances in logic programming) alongside user- and heuristic-guided refinement. Figure 2 sketches the main thrusts, outcomes, and deliverables for my CAREER proposal. The key research contributions comprise three central efforts: (a) the development of a systematic methodology for *meta*-abstract interpretation, which enables tracking the provenance of analysis facts as they propagate through a static analysis, (b) counterfactual abstract interpretation, which enables the user to use analysis results as a basis for interactive, counterfactual reasoning and (c) declarative, property-directed decompilation, in which I build logic-defined systems for interactive decompilation which scale to production-level RE workloads.

Security-Centric Data Processing Pipelines While RE focuses on an expert reasoning about a single binary (often for days or weeks), a related challenge is to reason about binary *corpora*. For the past several years, I have been collaborating with the NSA's Laboratory for Physical Sciences to build large-scale data processing pipelines which use AI to reason about internet-scale corpora of binary code. This setting poses distinct challenges due to the sheer scale of the problem, which often requires processing millions of binaries in a streaming-based fashion. Unfortunately, research in this space is significantly challenged by a data availability crisis: there are no significant public, open-source datasets of Windows PE binaries to use for training or evaluating AI-based data processing pipelines. In response to this concern, I led the development of ASSEMBLAGE (see Table 2), a distributed system which builds the largest-ever corpus of Windows PE binaries—with rich source-to-binary metadata—for the purposes of training and building the next generation of AI-based binary reasoning systems. These efforts have led to the acceptance of two NeurIPS '24 papers in the datasets and benchmarks track [13, 7]. Additionally, the ASSEMBLAGE data is being used by 12 government-adjacent groups and was the basis for an (unclassified) two-week-long workshop on language modeling for binary analysis, hosted by NSA. Using ASSEMBLAGE, I am collaborating with LPS to build scalable data processing pipelines for massive binary corpora, exploring new directions in language modeling, symbolic AI, and static analysis for tasks such as binary similarity, malware detection, author attribution, and reverse engineering. My long-term goal is to leverage my background in high-performance reasoning to build the most scalable security-centric data processing pipelines to date.

References

[1] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher Micinski, Markus N. Rabe, and César Sánchez. Temporal Logics for Hyperproperties. In Martín Abadi and Steve Kremer, editors, *Principles of Security and Trust (**POST '14**)*, volume 8414 of *Lecture Notes in Computer Science*, pages 265–284. Springer Berlin Heidelberg, 2014.

[2] Bruno Rucy Carneiro Alves de Lima, Merlin Kramer, Kalmer Apinis, and Kristopher Micinski. Incremental Evaluation of Dynamic Datalog Programs as a Higher-order DBSP Program. *the 5th International Workshop on the Resurgence of Datalog in Academia and Industry (**Datalog 2.0 '24**)*, 2024.

[3] Ke Fan, Thomas Gilray, Valerio Pascucci, Xuan Huang, Kristopher Micinski, and Sidharth Kumar. Optimizing the Bruck Algorithm for Non-Uniform All-to-All Communication. In *Proceedings of the 31st International Symposium on High-Performance Parallel and Distributed Computing (**HPDC '22**)*, HPDC '22, page 172–184, New York, NY, USA, 2022. Association for Computing Machinery. (Acceptance rate: 19%).

[4] Thomas Gilray, Sidharth Kumar, and Kristopher Micinski. Compiling Data-Parallel Datalog. In *Proceedings of the 30th ACM SIGPLAN International Conference on Compiler Construction (**CC '21**)*, page 23–35, New York, NY, USA, 2021. Association for Computing Machinery. (Acceptance rate: 37%).

[5] Thomas Gilray, Arash Sahebolamri, Yihao Sun, Sowmith Kunapaneni, Sidharth Kumar, and Kristopher Micinski. Datalog with first-class facts (revision submission). *Submission to **VLDB '25***, 2025. (Revision of a VLDB '25 draft which received a previous "revision" decision).

[6] Jinseong Jeon, Kristopher Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein. Dr. Android and Mr. Hide: Fine-grained Permissions in Android Applications. In *ACM CCS Workshop on Security and Privacy in Smartphones and Mobile Devices (**SPSM '12**)*, pages 3–14, Raleigh, NC, USA, October 2012.

[7] Chang Liu, Rebecca Saul, Yihao Sun, Edward Raff, Maya Fuchs, Townsend Southard Pantano, James Holt, and Kristopher Micinski. Assemblage: Automatic Binary Dataset Construction for Machine Learning. *Thirty-Eighth Annual Conference on Neural Information Processing Systems (**NeurIPS '24** datasets and benchmarks track)*, 2024.

[8] Kristopher Micinski, David Darais, and Thomas Gilray. Abstracting Faceted Execution. In *2020 IEEE 33rd Computer Security Foundations Symposium (**CSF '20**)*, pages 184–198, 2020. (Acceptance rate: 25%).

[9] Kristopher Micinski, Jonathan Fetter-Degges, Jinseong Jeon, Jeffrey S. Foster, and Michael R. Clarkson. Checking Interaction-Based Declassification Policies for Android Using Symbolic Execution. In *European Symposium on Research in Computer Security (**ESORICS '15**)*, volume 9327 of *Lecture Notes in Computer Science*, pages 520–538, Vienna, Austria, September 2015.

[10] Kristopher Micinski, Daniel Votipka, Rock Stevens, Nikolaos Kofinas, Jeffrey S. Foster, and Michelle L. Mazurek. User Interactions and Permission Use on Android. In *Conference on Human Factors in Computing Systems (**CHI '17**)*, 2017.

[11] Arash Sahebolamri, Langston Barrett, Scott Moore, and Kristopher Micinski. Bring Your Own Data Structures to Datalog. *The ACM's Conference on Object-Oriented Programming, Systems, Languages and Applications 2023 (**OOPSLA '23**)*, 7, oct 2023. (Acceptance rate: **36%**, won **distinguished paper award** at OOPSLA '23).

[12] Arash Sahebolamri, Thomas Gilray, and Kristopher Micinski. Seamless Deductive Inference via Macros. In *Proceedings of the 31st ACM SIGPLAN International Conference on Compiler Construction (**CC '22**)*, page 77–88, New York, NY, USA, 2022. Association for Computing Machinery. (Acceptance rate: 32%).

[13] Rebecca Saul, Chang Liu, Noah Fleischmann, Richard Zak, Kristopher Micinski, Edward Raff, and James Holt. Is Function Similarity Over-Engineered? Building a Benchmark. *Thirty-Eighth Annual Conference on Neural Information Processing Systems (**NeurIPS '24** datasets and benchmarks track)*, 2024.

[14] Ahmedur Rahman Shovon, Thomas Gilray, Kristopher Micinski, and Sidharth Kumar. Towards iterative relational algebra on the GPU. In *2023 USENIX Annual Technical Conference (**USENIX ATC '23**)*, pages 1009–1016, Boston, MA, July 2023. USENIX Association. (Acceptance rate: **18%**).

[15] Yihao Sun, Thomas Gilray, Sidharth Kumar, and Kristopher Micinski. Communication-Avoiding Recursive Aggregation. *Proceedings of the IEEE Conference on Cluster Computing (**IEEE Cluster '23**)*, (Cluster), Oct 2023. (Acceptance rate: **24%**).

[16] Yihao Sun, Sidharth Kumar, Thomas Gilray, and Kristopher Micinski. Column-Oriented Datalog on the GPU. *Submission to **AAAI '25***, 2025.

[17] Yihao Sun, Ahmedur Rahman Shovon, Thomas Gilray, Sidharth Kumar, and Kristopher Micinski. Optimizing Datalog for the GPU. *ACM International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS '25**)*, March 2025. Acceptance rate: **12.7%**, (65 out of 510)).

[18] Daniel Votipka, Seth M. Rabin, Kristopher Micinski, Jeffrey S. Foster, and Michelle M. Mazurek. An observational investigation of reverse engineers' processes. In *Proceedings of the 29th USENIX Security Symposium (**USENIX '20**)*, Proceedings of the 29th USENIX Security Symposium, pages 1875–1892. USENIX Association, 2020. (Acceptance rate: 16.1%).