Scott D. Stoller and Yanhong Liu

Stony Brook University

Knowledge and Reasoning for Drastic Program Improvement: Melding Formal and Statistical Approaches (ONR N00014-20-1-2751) December 1, 2019 – November 30 2022

Final Performance Report

ACCOMPLISHMENTS

What were the major goals and objectives of this project?

Increasingly more sophisticated software are developed for Naval systems. Developing such software increasingly relies on programming the application logic on top of existing components built as libraries. This leads to a dilemma, from two conflicting realities:

On the one hand, a myriad of software libraries have come into existence and grown rapidly in the past 10--15 years---from the prominent NumPy, SciPy, Panda, etc. for scientific computation, data analysis, etc. in the drastically growing user community of the Python language, to all kinds of libraries for all kinds of other languages. They are available as open source for so many important applications and used by everyday programmers in all these application domains, especially for Python due to its ease of use.

On the other hand, developing efficient programs correctly depends critically on knowledge and smart use of these libraries, while sufficient such knowledge is possessed only by the true experts who fully understand relevant functions in relevant libraries and their proper use in the context of the application logic. Even experts cannot keep up due to the growth of such libraries, which are also constantly upgraded for different underlying software versions and hardware architectures. Improper uses can easily lead to drastic slowdowns in the application software.

Solving this dilemma requires learning deep knowledge about programming and programs, and using the knowledge to reason about correct and efficient use of libraries. However, the state of the art is extremely limited. While such knowledge could be programmed in a program optimizer, as is done currently, this approach is impractical and does not scale for maintaining the optimizer with the rapidly growing library knowledge to be captured. A knowledge base about programming and programs is necessary to capture this knowledge declaratively.

Due to the deep and intractable nature of powerful, high-level, dynamic programming languages like Python, acquiring new knowledge of programs and programming automatically through either formal reasoning or statistical learning alone is far from sufficient. Simply using one approach for certain tasks, and the other approach for other tasks, is also inadequate. Each

challenging task requires inherently combined use of both.

Use of a knowledge base about programs and programming will also be critically needed for much beyond program optimization. This is because people believe that great applications will be done by AI, the artificial intelligence, but AI itself is all in computer programs, which must be reasoned about, manipulated, and improved.

This project proposed to develop a general knowledge base framework for programming and programs and that supports drastic program improvement. The framework supports representation of knowledge of programs and programming from two sources: formal descriptions about language semantics and implementations, and statistical information about programs and program executions. The framework represents such knowledge as rules capturing equivalence relations and cost measures, with both formal logic and statistical information. The framework then uses the knowledge base to improve the programs based on the rules. This includes integrated reasoning using formal and statistical knowledge to provide guarantees on the improvement. We also proposed to build an implementation of this framework in Python, use it for analyzing and improving Python programs, and evaluate on real-world applications.

What was accomplished under these goals?

1 Unified Logical Foundation for Knowledge Base Framework

Logic rules are powerful for expressing complex reasoning and analysis problems, especially including program analysis needed for program improvements. They are the foundation for our knowledge base framework for programming and programs. We developed a unified semantics for logic rules with unrestricted negation, quantification, and aggregation; this solved a long open challenge, and is essential for integrated formal and statistical reasoning with guarantees.

1.1 A Unified Semantics of Rules with Unrestricted Negation and Quantification

We developed a simple new semantics for logic rules, founded semantics, and its straightforward extension to another simple new semantics, constraint semantics, that unify the core of different prior semantics. Founded semantics is inferred iteratively as a least fixed-point, while constraint semantics is solved combinatorially. The new semantics support unrestricted negation, as well as unrestricted existential and universal quantifications. They are uniquely expressive and intuitive by allowing assumptions about the predicates, rules, and reasoning to be specified explicitly, as simple and precise binary choices. They are completely declarative and relate cleanly to prior semantics. This resolves well-known challenges like Russell's paradox, that were given conflicting semantics previously.

Our earlier work on founded semantics and constraint semantics was published in LFCS 2018 (Logical Foundations of Computer Science) and was invited for the journal issue; its extended version, with comparisons of semantics for well-known examples and with full proofs of all 19 theorems, appeared in *Journal of Logic and Computation* (JLC) in Dec. 2020.

1.2 A Unified Semantics of Recursive Rules with Aggregation

We made important advances with significant practical implications on extending recursive rules with aggregations, such as max and sum. Whereas recursive rules are essential for program analysis and optimization, aggregations are essential for integrating formal and statistical reasoning, to compute counts, frequency distributions, etc.

We developed a unified semantics for recursive rules with aggregation as well as negation and quantification. Incorporating cost models, this includes also efficient inference using a linear-time derivability relation that gives precise answers to all example problems we studied from the literature and all new problems we encountered.

We also performed experiments on the most challenging examples, exhibiting unexpectedly superior performance over well-known systems when they can compute correct answers. In particular, DLV and clingo cannot compute the desired well-founded semantics for the double-win game we developed, and XSB and SWI-Prolog gave many wrong results.

This work appeared in LFCS 2022 (Logical Foundations of Computer Science).

An extended version, invited for the journal issue, describes further advancements. These include results from new experiments, demonstrating superior performance even after manual optimizations to programs in other rule systems. We also developed a new game, the over-win game, and it revealed that existing semantics and implementation in XSB and SWI-Prolog fundamentally cannot compute the correct results for recursive rules with aggregation. The extended version appeared in *Journal of Logic and Computation* in Dec. 2022.

These results also helped guide improvements to inference in XSB and SWI-Prolog, as described in Section 4.2.

2 Language Design for Programming with Logic and Rules: DA Logic and Alda

For building a powerful knowledge base framework for programming and programs, we designed two languages for programming with logic and rules. DA Logic is a rule-based language that provides the logical foundation. Alda extends DistAlgo—our extension of Python for distributed algorithms—to support logic rules. DistAlgo supports high-level queries with patterns (including quantifications, comprehensions, and aggregations with patterns as queries), key to high-level programming of advanced analysis and optimizations.

2.1 Extending Logic Rules with Meta Knowledge and Knowledge Units

We developed a unified logic rules language, DA logic, for Design and Analysis logic, based on the unifying founded semantics and constraint semantics. It supports the power and ease of programming with different intended semantics with simple unified meta knowledge, and furthermore with knowledge units for building applications modularly. We especially empowered knowledge units to go beyond modules with restricted parameters, and to provide clear module dependencies without circularity.

The earlier version of our work on DA logic appeared in LFCS 2020 and was invited for the journal issue; our refined and extended version appeared in *Journal of Logic and Computation* in Jan. 2021. This work also appeared in KR 2020 (Knowledge Representation and Reasoning, Recently Published Research Track), and ICLP 2020 (International Conference on Logic Programming, Sister Conferences and Journal Presentation Track).

2.2 Integrating Logic Rules and Constraints with Everything Else, Seamlessly

Logic rules are powerful for expressing complex reasoning and analysis problems, but inconvenient for many other aspects that are needed in comprehensive applications. We developed an integrated language, Alda (earlier versions called DA-rules), that supports all of rules, sets, aggregates, functions, updates, and objects as seamlessly integrated built-ins, including concurrent and distributed processes. Key features of the design are:

- Sets of rules can be specified directly as other definitions can, where predicates in rules are simply set-valued variables holding the set of tuples for which the predicate is true. Thus, predicates can be used directly as set-valued variables and vice versa without needing any extra interface, and predicates being set-valued variables are completely different from functions or procedures, unlike in prior logic rule languages and extensions.
- Queries using rule sets are calls to an inference function that computes desired values of derived predicates (i.e., predicates in conclusions of rules) given values of base predicates (i.e., predicates not in conclusions). Thus, queries as function calls need no extra interface either, and a rule set can be used with predicates in it holding the values of any appropriate set-valued variables.
- Values of predicates can be updated either directly as for other variables or by the inference function; declarative semantics of rules are ensured by automatically maintaining values of derived predicates when values of base predicates are updated, through appropriate implicit calls to the inference function.
- Predicates and rule sets can be object attributes as well as global and local names, just as variables and functions can.

We also developed a complete formal semantics for Alda, by operational semantics for an object-oriented language supporting sets, functions, and updates with.declarative semantics for rules.

We further designed a practical compilation framework for Alda and developed a prototype implementation, as described in Section 3.2.

These results are described in a paper published on arXiv, a paper in LPOP 2022 (The 3rd Workshop on Logic and Practice of Programming), and a paper preliminarily accepted (pending revisions) to ICLP 2023 (39th International Conference on Logic Programming) as a journal publication in *Theory and Practice of Logic Programming (TPLP)*.

We also explored extending this language with constraints and interfacing our implementation with powerful constraint solvers, including MiniZinc. Constraints can significantly enhance the knowledge base framework.

3 System Design and Implementation

3.1 Program Transformation System

While parsing programs into abstract syntax trees (ASTs) and building analysis and transformation on top of ASTs has been the gold standard in practice, it has posed significant challenges for advanced analysis and transformations, for two fundamental reasons: (1) while analysis and transformations at the source level are easy to understand by essentially all programmers, expressing analysis and transformations at the underlying AST level requires deep compiler knowledge possessed only by an extremely tiny percentage of programmers, (2) modern-day programming languages and implementations are continuously updated, and even when a language has not changed in a new version, the AST representation in the implementation can be changed, even significantly, breaking the existing analysis and transformations and requiring significant effort to maintain or even reimplement them.

Our solution to this is a powerful program transformation system for specifying sophisticated transformations at the source level, by significantly improving over our earlier work on an invariant-driven program transformation system, InvTS. This started with a total upgrade of the system framework from Python 2 to Python 3, and resulted in a thoroughly rewritten, drastically simplified, general pattern matching and replacement mechanism.

We also automated generation of invariant rules for expensive set queries by exploiting our earlier work on automatic incrementalization of set queries. Such optimization knowledge is critical for drastic program improvements. Additionally, we developed new equivalence and transformation rules, for transformations between loops, comprehensions, and library function calls, as well as translations between data representations, for drastically improving code clarity and efficiency. We applied these rules to optimize Python programs for scientific computation , role-based access control, graph algorithms, game state queries, and for loop optimizations in general.

Each of these tasks addresses a significant challenge in building a knowledge base system for programming and program improvements. The results are described in several reports, and we started writing a paper describing these results.

We also started more experimental evaluation and development of transformations for generation of runtime checks for correctness and efficiency analysis.

3.2 Integrated Language and Compiler

We designed a compilation framework for Alda and implemented a prototype compiler. It is built on our compiler for DistAlgo—an object-oriented language that supports all other features in Alda, including concurrent and distributed processes but not logic rules—and uses XSB, an efficient logic rule system for queries using rules. The framework and compiler are presented in a paper published on arXiv and a paper preliminarily accepted (pending revisions) to ICLP 2023 (39th International Conference on Logic Programming) as a journal publication in *Theory and Practice of Logic Programming (TPLP)*.

Alda is an ideal language for high-level programming of program analysis and transformation, due to its support for greatly simplified programming with logic rules together with sets, functions, updates, and objects. We developed a library, called PS, that supports programming with high-level functionalities for parsing programs, generating logical program representations, and, for efficiency, persistent program databases and compressed representations. Persistent databases can be configured declaratively and used automatically with defaults. These automatic optimizations have enabled significantly reduced analysis time, memory usage, and disk space, and even more drastic improvements for different, repeated, and deeper analyses built on top of earlier analyses of a program. This work is described in a paper submitted to SOAP 2023 (International Workshop on the State of the Art in Program Analysis).

To show Alda's power and flexibility, we developed applications in a variety of areas, including benchmarking, role-based access control, and program analysis. These applications are described in Section 4. We also used these applications as performance benchmarks. Our extensive experiments with them demonstrate the generally good performance of our Alda compiler and PS library. The benchmarks and experimental results are described in a paper published on arXiv and a paper to appear in the ICLP 2023 Technical Communications track.

4. Applications and Experimental Evaluation

4.1 Optimizing Python Programs

Experiments with MADNESS

We used the improved InvTS, described in Section 3.1, to improve madpy, a one-dimensional Python implementation of MADNESS, a multi-million line C++ environment for solving integral and differential equations in many dimensions. MADNESS provides high-level functions implemented using adaptive, fast methods with guaranteed precision. madpy is primarily for pedagogical use in understanding how MADNESS works, and is useful as a prototyping laboratory. madpy as implemented in the MADNESS project uses Python 2, and it uses only the Python standard math library for numerical computations.

We made four main types of improvements to madpy: upgrading to Python 3, replacing classes Vector and Matrix in tensor.py with NumPy arrays and functions, turning many loops into list comprehensions, and replacing remaining loops and some comprehensions with vectorized

NumPy functions. We initially performed these transformations manually and then automated them using InvTS.

This led to some remarkable improvements in speed and conciseness of code. The code size decreased from 1897 to 1630 lines. Using the madpy modules test_onedAST and oned.py for initial testing of correctness and performance, we verified that our transformations to madpy preserve the functionality while also being 4 times faster.

Incrementalization

We demonstrated that the incrementalization method developed in our prior work leads to very efficient implementations of recursive rules with aggregation. We compared four well-known rule systems—XSB, SWI-Prolog, clingo and DLV—with a straightforward least fixed-point computation of our semantics in DistAlgo, and with a more efficient version of that computation obtained using our incrementalization method. We compared their performance on the company control problem, a well-studied challenge problem for recursive rules with aggregation. Our incrementalized program is the fastest among these, even after consulting with XSB and SWI-Prolog experts to optimize the programs for their systems. Detailed experimental results were published in *Journal of Logic and Computation* in Dec. 2022.

Transformation of Loops for Clarity and Efficiency

We developed an analysis that detects `for` loops that can be transformed into comprehensions, and developed transformations that successfully transformed a few hundred `for` loops in the widely-used programs used in our program analysis benchmark (see Section 4.5). Comprehensions are more succinct and easier to read than equivalent `for` loops, and in many cases are more efficient. We also developed an analysis that detects `for` loops that can be eliminated by broadcasting NumPy operations over arrays (a.k.a. vectorization), and wrote InvTS transformation rules for this transformation, which improves readability and performance. We analyzed the programs mentioned above and 11 other popular applications that use NumPy. We found dozens of loops that can be optimized by our InvTS transformation rules and conducted performance experiments with some of them to confirm that this optimization achieves significant speedup. A paper describing this work is in preparation.

4.2 Guiding the Best Rule Systems

The results of our experiments with recursive rules with aggregation described in Section 1.2 helped guide improvements to the best rule systems.

Based on our experiments showing that XSB and SWI-Prolog give wrong results for the double-win game we designed, the developers of both systems have since found and fixed bugs that caused these incorrect results. Experiments with our over-win game revealed that existing semantics and inference in XSB and SWI-Prolog fundamentally cannot compute the correct results for recursive rules with aggregation. Their inference to compute well-founded semantics deals with recursion through explicit negation, but recursion with aggregation in the over-win game is not explicitly negated. Also, their inference does not correctly handle different comparison operations that use the results of the same aggregation. XSB has long been a

leading, most advanced rule system. Using our results, the XSB team is implementing a new inference for XSB, fundamentally different from the existing incorrect inference. Our DistAlgo programs for experiments have been used to generate test cases, especially on larger data, because no other known implementations can compute correct results.

These results appeared in LFCS 2022 (Logical Foundations of Computer Science), an extended version published in *Journal of Logic and Computation* (JLC) in Dec. 2022, and a LinkedIn post by David Warren, a lead developer of XSB.

4.3 OpenRuleBench: all benchmarks + benchmarking itself

OpenRuleBench contains a wide variety of database, knowledge base, and semantic web application problems, written using rules in 11 well-known rule systems, as well as large data sets and a large number of test scripts for running and measuring the performance. Among those 11 rule systems, we compare mainly with XSB, because it is one of the most advanced and fastest, and the most consistent, based on published experimental results for OpenRuleBench.

We easily translated all of the applications in OpenRuleBench into Alda, except we omitted one that tests interfaces of rule systems with databases—database interfaces are a non-issue for Alda, because the excellent ones for Python can be used. It was straightforward to express the applications in Alda, producing programs very similar to those in XSB or even simpler. Additionally, the code for benchmarking itself—for reading data, running tests, timing, and writing results—is drastically simpler and shorter in Alda than in XSB. Experiments with the largest applications in OpenRuleBench show that Alda achieves good performance. These results are described in a paper published on arXiv and a paper to appear in the ICLP 2023 Technical Communications track.

4.4 RBAC: ANSI standard for Role-Based Access Control

The ANSI standard for Role-Based Access Control involves many sets and query and update functions. To program the transitive role hierarchy, a complex and inefficient `while` loop was used in our prior work on RBAC, because an efficient algorithm would be drastically even more complex. With Alda's support for rules as well as sets, functions, updates, and objects, the entire RBAC standard is easily written in Alda, similar as in Python except with rules for computing the transitive role hierarchy, yielding a simpler, clearer program. Furthermore, the Alda version is drastically faster than versions in Python and DistAlgo that use repeated set comprehensions and set queries, respectively. These results are described in a paper published on arXiv and a paper to appear in the ICLP 2023 Technical Communications track.

4.5 PA: program analysis

We developed several program analyses in Alda, including the analyses for optimization described in Section 4.1, and analysis of the class hierarchy of Python programs. Our experience with all of them confirms the ease and power of Alda for expressing knowledge about programs and reasoning, and yields desired program analysis results.

We created a program analysis benchmark, called PA, from our analysis of class hierarchy. It showcases integrated use of rules with aggregate queries and recursive functions. We implemented the analysis in Alda and in XSB and applied the implementations to 9 large, widely-used Python libraries, including NumPy and SciPy for scientific computation and Scikit-learn and PyTorch for machine learning. The Alda version is simpler and clearer, and achieves acceptable performance even on very large data, e.g., 5 million facts representing SymPy's code. The Alda version is significantly faster than our initial XSB version, e.g., 120x faster for PyTorch. After significant performance debugging and manual optimization, we developed an XSB version that is faster than Alda, but we expect Alda to be even faster when our prototype is modified to use an in-memory interface (currently under development), instead of files, to pass data to XSB. These results are described in a paper published on arXiv and a paper to appear in the ICLP 2023 Technical Communications track.

5. Conference/workshop organization and other activities

The co-PI was a Program co-Chair for PADL 2020 (The 22nd International Symposium on Practical Aspects of Declarative Languages), in conjunction with POPL 2020 (The 47th ACM SIGPLAN Symposium on Principles of Programming Languages) (https://popl20.sigplan.org/home/PADL-2020).

The co-PI co-chaired LPOP 2020, The 2nd Workshop on Logic and Practice of Programming (https://2020.splashcon.org/home/lpop-2020, in conjunction with SPLASH 2020), with outstanding invited speakers and active discussions (https://www.youtube.com/watch?v=Q6TVPVJp6sU)

The co-PI was a Program co-Chair for ICLP 2021 (The 37th International Conference on Logic Programming), and organized outstanding invited talks as well as a special session on Datalog (https://sites.google.com/a/cs.stonybrook.edu/lpop/iclp2021).
The PI co-edited two special issues of the journal TPLP (*Theory and Practice of Logic Programming*) for selected papers from the conference and the proceedings of technical communications from the conference.

The co-PI co-Chaired ApPLIED 2021 (The 3rd Workshop on Advanced Tools, Programming Languages, and Platforms for Implementing and Evaluating Algorithms for Distributed Systems), in conjunction with PODC 2021 (The 40th ACM Symposium on Principles of Distributed Computing), and organized outstanding invited talks (http://www.cse.chalmers.se/~elad/ApPLIED2021).

The co-PI served on the jury for the inaugural 2022 Colmerauer Prize, of the Association for Logic Programming (https://logicprogramming.org/2023/02/the-alp-alain-colmerauer-prize/). The co-PI is also serving on the jury for the 2023 Colmerauer Prize and is an Honorary Member of the Scientific Committee of the Year of Prolog, 2022-2023.

The co-PI gave a keynote at PEPM 2022 (ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation) titled: From Meta Frameworks and Transformations to Distributed Computing and More (https://popl22.sigplan.org/details/pepm-2022-papers/9/From-meta-frameworks-and-transformations-to-distributed-computing-and-more).

The co-PI was a General co-Chair and the Program Chair of LPOP 2022 (The 3rd Workshop on Logic and Practice of Programming, with outstanding invited speakers and active discussions (https://lpop.cs.stonybrook.edu/lpop2022).

The co-PI is a General co-Chair of ApPLIED 2023 (The 3rd Workshop on Advanced Tools, Programming Languages, and Platforms for Implementing and Evaluating Algorithms for Distributed Systems), in conjunction with PODC 2023 (The 42nd ACM Symposium on Principles of Distributed Computing) (https://www.cse.chalmers.se/~elad/ApPLIED2023/).

The PI and co-PI will present a tutorial, High-Level Executable Specification and Reasoning for Improving Distributed Algorithms, at PLDI 2023 (The 44th ACM SIGPLAN Conference on Programming Language Design and Implementation) and PODC 2023 (The 42nd ACM Symposium on Principles of Distributed Computing), with FCRC 2023 (ACM Federated Computing Research Conference) (https://pldi23.sigplan.org/details/pldi-2023-PLDI-Workshops/5/High-Level-Executable-Specification-and-Reasoning-for-Improving-Distributed-Algorithm).

The PI and co-PI have been invited to attend and will participate in the DOE/NSF Workshop on Correctness in Scientific Computing (CSC 2023), part of FCRC 2023 (ACM Federated Computing Research Conference).
https://pldi23.sigplan.org/home/csc-2023

What opportunities for training and professional development has the project provided?

Four PhD students, a postdoc, and three undergraduate students received training and professional development while working on this project.

One PhD student focused on compiler implementation of Alda for rules and constraints by building on DistAlgo, XSB, and Minizinc

Another PhD student defended his dissertation in Feb. 2021 and continued as a postdoc working on analysis and optimizations using Alda, until August 2021. He then joined West Virginia University, Institute of Technology as a tenure-track Assistant Professor, and has recently accepted a tenure-track Assistant Professor at California State University, Monterey Bay, to start in Fall 2023

Another PhD student studied programming tools and finding and fixing inefficiencies in programs. His background before joining this project was in computer graphics, not languages and algorithms, and his progress was slow, but he did get trained with essential language, design, and programming skills using Javascript, Python, and DistAlgo, and started to make more progress.

The fourth PhD student to join the project was a beginning PhD student who graduated from our M.S. program in May 2022. The student was introduced to related state-of-the-art research, and was closely supervised to learn more advanced and precise high-level specifications of program properties and more systematic and automated methods for transforming programs and analyzing program correctness and efficiency.

One of the undergraduate students was a talented BS student in our honors program. He completed several research projects for this award, especially including: significant upgrade of InvTS and significant improvement of InvTS pattern matching, generation of InvTS rules, and use of InvTS for optimizations and general transformations. After graduation, he joined Amazon in Seattle. Another student was an outstanding undergraduate TA for co-PI's discrete math course. She learned DA-rules and Datalog with XSB, and experimented with dynamic program analysis. The third student was an excellent student in the discrete math course, who learned and programmed with constraints using board games as examples.

The PI also supervised a high school science research student to learn Python and study AI search strategies.

How were the results disseminated to communities of interest?

Results from this project were disseminated primarily through publications in major conferences and journals and in the Computing Research Repository on arXiv.org, presentations at conferences, and use of the developed software in teaching and in related research.

Honors and Awards

Nothing to report.

Technology Transfer

Nothing to report.

Students

Number of students receiving STEM degrees: 4
Number of undergraduate and graduate STEM participants : 7

PRODUCTS

All journal and conference publications are peer-reviewed.
All publications acknowledge federal support.

Journal Publications

Yanhong A. Liu and Scott D. Stoller. The Founded Semantics and Constraint Semantics of Logic Rules. Journal of Logic and Computation 30(8):1609–1638, December 2020. Oxford University Press, 2020.
Status: published
https://doi.org/10.1093/logcom/exaa056

Yanhong A. Liu and Scott D. Stoller. Knowledge of Uncertain Worlds: Programming with Logical Constraints. Journal of Logic and Computation 31(1):193–212, January 2021. Oxford University Press, 2021.
Status: published
https://doi.org/10.1093/logcom/exaa077

Luca Bortolussi, Francesca Cairoli, Nicola Paoletti, Scott A. Smolka, Scott D. Stoller
Neural Predictive Monitoring and a Comparison of Frequentist and Bayesian Approaches
International Journal on Software Tools for Technology Transfer, May 2021. Springer, 2021.
Status: published
https://doi.org/10.1007/s10009-021-00623-1

Yanhong A. Liu and Scott D. Stoller Recursive Rules with Aggregation: A Simple Unified Semantics. Journal of Logic and Computation 32(8):1659–1693, November 2022. Oxford University Press, 2022.
Status: published
https://doi.org/10.1093/logcom/exac072

Usama Mehmood, Shouvik Roy, Amol Damare, Radu Grosu, Scott A. Smolka, and Scott D. Stoller. A Distributed Simplex Architecture for Multi-Agent Systems. Journal of Systems Architecture 134, January 2023. Elsevier, 2023.
Status: published
https://doi.org/10.1016/j.sysarc.2022.102784

Conference Publications

Yanhong A. Liu and Scott D. Stoller. Knowledge of Uncertain Worlds: Programming with Logical Constraints. 2020 Symposium on Logical Foundations of Computer Science (LFCS '20), volume 11972 of Lecture Notes in Computer Science. Springer, 2020.
Status: published
https://doi.org/10.1007/978-3-030-36755-8_8

Thang Bui and Scott D. Stoller. A Decision Tree Learning Approach for Mining Relationship-Based Access Control Policies. 25th ACM Symposium on Access Control Models and Technologies (SACMAT 2020). ACM Press, 2020.
Status: published
https://doi.org/10.1145/3381991.3395619

Yanhong A. Liu and Scott D. Stoller. Recursive Rules with Aggregation: A Simple Unified Semantics (Extended Abstract). 36th International Conference on Logic Programming (ICLP 2020) (Technical Communications), volume 325 of Electronic Proceedings in Theoretical Computer Science (EPTCS).
Status: published
http://dx.doi.org/10.4204/EPTCS.325 (for the proceedings)

Thang Bui and Scott D. Stoller. Learning Attribute-Based and Relationship-Based Access Control Policies with Unknown Values. 16th International Conference on Information Systems Security (ICISS), volume 12553 of Lecture Notes in Computer Science, pages 23-44. Springer, 2020.
Status: published
https://doi.org/10.1007/978-3-030-65610-2_2

Yanhong A. Liu and Scott D. Stoller. Assurance of Distributed Algorithms and Systems: Runtime Checking of Safety and Liveness. 20th International Conference on Runtime Verification (RV 2020), volume 12399 of Lecture Notes in Computer Science, pages 47-66. Springer, 2020.
Status: published
https://dx.doi.org/10.1007/978-3-030-60508-7_3

Saksham Chand and Yanhong A. Liu Brief Announcement: What's Live? Understanding Distributed Consensus. ACM Symposium on Principles of Distributed Computing (PODC 2021), pages 565-568. ACM Press, 2021.
Status: published
https://dl.acm.org/doi/10.1145/3465084.3467947
https://www.cs.stonybrook.edu/~liu/papers/Liveness-PODC21.pdf

Shouvik Roy, Usama Mehmood, Radu Grosu, Scott A. Smolka, Scott D. Stoller, and Ashish Tiwari. Distributed Control for Flocking Maneuvers via Acceleration-Weighted Neighborhooding. 2021 American Control Conference (ACC), pages 2745-2750. IEEE Press, 2021.
Status: published
https://doi.org/10.23919/ACC50511.2021.9483155

Yanhong A. Liu and Scott D. Stoller Recursive. Rules With Aggregation: A Simple Unified Semantics. Logical Foundations of Computer Science (LFCS 2022), volume 13137 of Lecture Notes in Computer Science, pages 156–179. Springer-Verlag, 2021.
Status: published.

https://doi.org/10.1007/978-3-030-93100-1_11

Usama Mehmood, Scott D. Stoller, Radu Grosu, Shouvik Roy, Amol Damare, and Scott A. Smolka. A Distributed Simplex Architecture for Multi-Agent Systems. 7th International Symposium on Dependable Software Engineering: Theories, Tools and Applications (SETTA 2021), pages 239-257. Springer-Verlag, 2021.
Status: published.
https://doi.org/10.1007/978-3-030-91265-9_13

Yanhong A. Liu. Alda: Integrating Logic Rules with Everything Else, Seamlessly. 3rd Workshop on Logic and Practice of Programming (LPOP 2022), pages 62-64.
Status: published on conference website, to appear on arXiv.
https://lpop.cs.stonybrook.edu/lpop2022/proceedings-slides-and-videos

Yanhong A. Liu, Scott D. Stoller, Yi Tong and Bo Lin. Programming with Logic Rules and Everything Else, Seamlessly. 39th International Conference on Logic Programming (ICLP 2023).
Status: preliminarily accepted, pending revisions.

Yanhong A. Liu, Scott D. Stoller, Yi Tong and K. Tuncay Tekle. Benchmarking for Integrating Logic Rules with Everything Else. 39th International Conference on Logic Programming (ICLP 2023).
Status: accepted to the Technical Communications track.

Yanhong A. Liu, Yi Tong and Scott D. Stoller. High-Level Programming for Program Analysis. 2023 International Workshop on the State Of the Art in Program Analysis (SOAP 2023).
Status: under review.

Books Or Other Non-periodical, One-time Publications

Nothing to report.

Theses

Thang Bui. Mining Relationship-Based Access Control Policies. Ph.D. thesis, Computer Science Department, Stony Brook University, February 2021.

Websites

https://lpop.cs.stonybrook.edu/iclp2021
https://lpop.cs.stonybrook.edu/lpop2022

Other Products: Other

David S. Warren and Yanhong A. Liu, editors. Proceedings of the 3rd Workshop on Logic and Practice of Programming (LPOP 2022).

Status: published on conference website, to appear on arXiv.
https://lpop.cs.stonybrook.edu/lpop2022/proceedings-slides-and-videos

David S. Warren, Peter Van Roy, and Yanhong A. Liu, editors. Proceedings of the 2nd Workshop on Logic and Practice of Programming (LPOP 2020). arXiv:2211.09923 [cs.PL], November 2022.
Status: published
https://arxiv.org/abs/2211.09923

Yanhong A. Liu, Scott D. Stoller, Yi Tong, and Bo Lin, and K. Tuncay Tekle. Programming with Rules and Everything Else, Seamlessly. arXiv:2205.15204 [cs.PL], May 2022.
https://arxiv.org/abs/2205.15204.
Status: published

Other Products: Audio or Video Products

From meta frameworks and transformations to distributed computing and more.
PEPM 2022 Keynote Talk.
Video: https://www.youtube.com/watch?v=wBnswkwepDs&t=138s
Conference: https://popl22.sigplan.org/home/pepm-2022#program

Recursive rules with aggregation: A simple unified semantics.
LFCS 2022 presentation.
Video: https://drive.google.com/file/d/1_61djZDe64G3MoHSHdpI7KMZ9FP0vgCd/view
Conference: https://lfcs.ws.gc.cuny.edu/
Integrating Logic Rules with Everything Else, Seamlessly
LPOP 2022 presentation.
Video: https://youtu.be/NzRZv0bqwrY
Conference: https://lpop.cs.stonybrook.edu/lpop2022

PARTICIPANTS & OTHER COLLABORATING ORGANIZATIONS

Participants during December 1, 2019 - June 15, 2020

Scott D. Stoller
Project Role: PD/PI
Months Worked: 1
International Collaboration: none
International Travel: none

Yanhong Liu
Project Role: co PD/PI
Months Worked: 1
International Collaboration: none
International Travel: none

Matthew Castellana
Project Role: Graduate Student (research assistant)
Months Worked: 1
International Collaboration: none
International Travel: none

Yi Tong
Project Role: Graduate Student (research assistant)
Months Worked: 1
International Collaboration: none
International Travel: none

Eric Keough
Role: undergraduate student
Months worked: 1
International Collaboration: none
International Travel: none

Participants during June 16, 2020 - June 15, 2021

Scott D. Stoller
Project Role: PD/PI
Months Worked: 3
International Collaboration: none
International Travel: none

Yanhong Liu
Project Role: co PD/PI
Months Worked: 3
International Collaboration: none
International Travel: none

Thang Bui
Project Role: Post-Doctoral Scholar
Months Worked: 6
International Collaboration: none
International Travel: none

Yi Tong
Project Role: Graduate Student (research assistant)
Months Worked: 1
International Collaboration: none
International Travel: none

Eric Keough
Role: undergraduate student
Months worked: 2
International Collaboration: none
International Travel: none

Participants during June 16, 2021 - November 30, 2022

Scott D. Stoller
Project Role: PD/PI
Months Worked: 2
Contribution to Project: Worked on all aspects of the project described under Accomplishments.
State, U.S. territory, and/or country of residence: New York, U.S.A
Foreign Collaboration: none
Foreign Travel: none

Yanhong Liu
Project Role: co PD/PI
Months Worked: 7
Contribution to Project: Worked on all aspects of the project described under Accomplishments.
State, U.S. territory, and/or country of residence: New York, U.S.A
Foreign Collaboration: none
Foreign Travel: none

Thang Bui
Project Role: Post-Doctoral Scholar
Months Worked: 2
Contribution to Project: Worked on program performance analysis using rules, machine learning, and line-level profiling.
State, U.S. territory, and/or country of residence: New York, U.S.A
Foreign Collaboration: none
Foreign Travel: none

Matthew Castellana
Project Role: Graduate Student (research assistant)
Months Worked: 3 [was 1]
Contribution to Project: Studied programming tools and finding and fixing inefficiencies in programs.
State, U.S. territory, and/or country of residence: New York, U.S.A
Foreign Collaboration: none
Foreign Travel: none

Kumar Shivam

Project Role: Graduate Student (research assistant)
Months Worked: 2
Contribution to Project: Studied programming tools and finding and fixing inefficiencies in programs.
State, U.S. territory, and/or country of residence: New York, U.S.A
Foreign Collaboration: none
Foreign Travel: none

Eric Keough
Role: Non-Student Research Assistant
Months worked: 1
Contribution to Project: Worked on program transformation system InvTS and using it for optimizations and general transformations.
State, U.S. territory, and/or country of residence: New York, U.S.A
Foreign Collaboration: none
Foreign Travel: none

What other organizations have been involved as partners?
Nothing to report.

Have other collaborators or contacts been involved?
Nothing to report.

IMPACT: What was the impact of the project? How has it contributed?

What is the impact on the development of the principal discipline(s) of the project?
Significant improvements to programming language design, semantics, and compilation, and to program analysis and optimization.

What is the impact on other disciplines?
Improved software development tools incorporating advanced program analysis and optimization technologies directly benefit the many other disciplines that involve software development.

What is the impact on the development of human resources?
Please see the section on opportunities for training and professional development.

What was the impact on teaching and educational experiences?
We and a colleague used the tools and materials we developed in two graduate classes and one undergraduate class, for programming assignments as well as team projects, for over 150 students. In graduate class on principles of programming languages, the PI supervised a team project on complexity analysis using machine learning for Python and Java and three team projects on Python program optimizations using static analysis and transformation.

What is the impact on physical resources that form infrastructure?
Nothing to report.

What is the impact on institutional resources that form infrastructure?
Nothing to report.

What is the impact on information resources that form infrastructure?
As part of this project, we are developing software systems that are freely available from github.com or available by request from the PIs.

What is the impact on technology transfer?
Nothing to report.

What is the impact on society beyond science and technology?
All sectors of modern society are increasingly reliant on software. Improvements in program analysis and optimization help improve the quality of people's lives by supporting the development of more efficient and reliable software.

What percentage of the award's budget was spent in a foreign country?
Zero.

CHANGES/PROBLEMS

Nothing to report.

REPORT DOCUMENTATION PAGE

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED	
		START DATE	END DATE
20230330	Final Report	20191201	20221130

4. TITLE AND SUBTITLE

Knowledge and Reasoning for Drastic Program Improvement: Melding Formal and Statistical Approaches

5a. CONTRACT NUMBER	5b. GRANT NUMBER	5c. PROGRAM ELEMENT NUMBER
N/A	N000142012751	
5d. PROJECT NUMBER	**5e. TASK NUMBER**	**5f. WORK UNIT NUMBER**

6. AUTHOR(S)

Stoller, Scott D.
Liu, Yanhong

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

The Research Foundation for The State University of New York
West 5510 Frank Melville Memorial Library
Stony Brook, NY 11794-0001

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)	11. SPONSOR/MONITOR'S REPORT NUMBER(S)
US Navy Office of Naval Research	ONR	

12. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for Public Release; Distribution is Unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

Sophisticated software developed for Naval systems increasingly relies on programming the application logic on top of growing and evolving software libraries. Developing correct and efficient software depends critically on knowledge and smart use of such existing programs. This project developed a general knowledge base framework for programming and programs. The framework acquires and melds knowledge of programming and programs from two sources: formal descriptions about language semantics and implementations, and statistical information about programs and program executions. The framework uses the knowledge to inform program transformations that optimize and improve programs. The project's results include: founded and constraint semantics for rules with unrestricted negation, quantification, and aggregation, as a unified foundation for logical and statistical reasoning; a powerful language that seamlessly integrates rules, sets, aggregation, functions, updates, and objects; software infrastructure including a compiler for the integrated language, an invariant-based program transformation system with enhanced powerful pattern matching, and a general framework for benchmarking; and a broad set of applications spanning scientific computing, distributed algorithms, guiding rule systems, benchmarking of rule systems and unified languages, role-based access control, and program analysis of numerous large libraries.

15. SUBJECT TERMS

knowledge representation and reasoning, unified semantics, rule systems, program analysis, program optimization, program transformation

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES
a. REPORT U	b. ABSTRACT U	C. THIS PAGE U	UU	

19a. NAME OF RESPONSIBLE PERSON	19b. PHONE NUMBER *(Include area code)*
Scott D. Stoller	631-632-1627

INSTRUCTIONS FOR COMPLETING SF 298

1. REPORT DATE.
Full publication date, including day, month, if available. Must cite at least the year and be Year 2000 compliant, e.g. 30-06-1998; xx-06-1998; xx-xx-1998.

2. REPORT TYPE.
State the type of report, such as final, technical, interim, memorandum, master's thesis, progress, quarterly, research, special, group study, etc.

3. DATES COVERED.
Indicate the time during which the work was performed and the report was written.

4. TITLE.
Enter title and subtitle with volume number and part number, if applicable. On classified documents, enter the title classification in parentheses.

5a. CONTRACT NUMBER.
Enter all contract numbers as they appear in the report, e.g. F33615-86-C-5169.

5b. GRANT NUMBER.
Enter all grant numbers as they appear in the report, e.g. AFOSR-82-1234.

5c. PROGRAM ELEMENT NUMBER.
Enter all program element numbers as they appear in the report, e.g. 61101A.

5d. PROJECT NUMBER.
Enter all project numbers as they appear in the report, e.g. 1F665702D1257; ILIR.

5e. TASK NUMBER. Enter all task numbers as they appear in the report, e.g. 05; RF0330201; T4112.

5f. WORK UNIT NUMBER.
Enter all work unit numbers as they appear in the report, e.g. 001; AFAPL30480105.

6. AUTHOR(S). Enter name(s) of person(s) responsible for writing the report, performing the research, or credited with the content of the report. The form of entry is the last name, first name, middle initial, and additional qualifiers separated by commas, e.g. Smith, Richard, J, Jr.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES). Self-explanatory.

8. PERFORMING ORGANIZATION REPORT NUMBER.
Enter all unique alphanumeric report numbers assigned by the performing organization, e.g. BRL-1234; AFWL-TR-85-4017-Vol-21-PT-2.

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES). Enter the name and address of the organization(s) financially responsible for and monitoring the work.

10. SPONSOR/MONITOR'S ACRONYM(S). Enter, if available, e.g. BRL, ARDEC, NADC.

11. SPONSOR/MONITOR'S REPORT NUMBER(S). Enter report number as assigned by the sponsoring/monitoring agency, if available, e.g. BRL-TR-829; -215.

12. DISTRIBUTION/AVAILABILITY STATEMENT. Use agency-mandated availability statements to indicate the public availability or distribution limitations of the report. If additional limitations/ restrictions or special markings are indicated, follow agency authorization procedures, e.g. RD/ FRD, PROPIN, ITAR, etc. Include copyright information.

13. SUPPLEMENTARY NOTES. Enter information not included elsewhere such as: prepared in cooperation with; translation of; report supersedes; old edition number, etc.

14. ABSTRACT. A brief (approximately 200 words) factual summary of the most significant information.

15. SUBJECT TERMS. Key words or phrases identifying major concepts in the report.

16. SECURITY CLASSIFICATION. Enter security classification in accordance with security classification regulations, e.g. U, C, S, etc. If this form contains classified information, stamp classification level on the top and bottom of this page.

17. LIMITATION OF ABSTRACT. This block must be completed to assign a distribution limitation to the abstract. Enter UU (Unclassified Unlimited) or SAR (Same as Report). An entry in this block is necessary if the abstract is to be limited.