Research Statement

Xin Zhang

Program reasoning techniques, including program analysis, type systems, and automated theorem provers have been widely applied to ensure software quality. With the software industry experiencing a major shift to ML, the program reasoning community is facing both opportunities and challenges. One one hand, advances in ML bring new toolkits to build better program reasoning techniques; on the other hand, it is now of paramount importance to ensure the quality of ML programs as they are being increasingly used in making critical decisions. My research focuses on this interplay between program reasoning and machine learning (ML). **On one hand, I use ML techniques to improve the usability of program reasoning techniques in general. On the other hand, I design new program analyses and languages to help programmers build ML systems of higher quality.**

Past Research

I have improved efficiency and accuracy of program reasoning techniques by leveraging ML ideas at all levels:

- **Data-Driven Program Reasoning.** I have developed data-driven program reasoning by using both offline learning and online learning. I have built a framework that enables a program reasoning system learning from its past runs [OOPSLA'16] for better efficiency. It builds summaries on shared library code in the training phase and uses these summaries to short-circuit analyzing the same code when the system is applied to a new program. I have used this framework to speedup important program analyses like call-graph construction by 2.6×. I have also developed a framework that systematically enables a program reasoning system to improve its accuracy by learning from runtime information. Such information includes user feedback on truth of bug reports [FSE'15, Distinguished Paper Award] and user feedback on truth of key intermediate results [OOPSLA'17]. By incorporating only limited feedback from end users, my framework is able to reduce over 70% false alarms for important program analyses like datarace detection and pointer analysis.

- **Representation That Combines Logic and Probabilities.** To enable data-driven program reasoning and build more flexible techniques in general, I am the first to systematically combine logic and probabilities in their representations [PLDI'14, FSE'15], whereas representations of all existing program reasoning techniques are all logical. While the logical part encodes expert knowledge from a designer and correctness conditions (e.g., soundness), the probabilistic part offers a unified way to handle uncertainties. This approach results in a general framework that synthesizes a highly specialized program reasoning system for a given usage scenario such that the system is optimal in efficiency and accuracy [PLDI'14, Distinguished Paper Award]. For example, by generating program analyses that are specialized to individual assertions, my framework is able to resolve 10× more assertions for important analyses like pointer analysis and type-state analysis compared to the state-of-the-art.

- **Effective Inference by Exploiting Domain Knowledge.** To support inference under this new representation, I have leveraged and improved upon existing probabilistic inference algorithms by exploiting domain insights. I developed query-guided inference that speeds up inference by focusing on the part in the representation that is relevant to a specific analysis result [POPL'16]; I developed a guess-and-check algorithm that lazily considers constraints induced by the analysis problem [AAAI'16, SAT'16]; finally, I developed the first incremental algorithm for solving Maximum Satisfiability which the inference algorithm is built upon [CP'16]. With these techniques, my inference solver is able to solve inference problem comprising up-to 10^{30} clauses which cannot be solved by previous techniques in a fast, optimal, and sound (not violating logical/hard constraints) manner. Moreover, because these insights are not unique to program analyses, my solver also outperforms existing solvers on problems from other domains such as information retrieval.

I have developed new program analyses and languages for improving quality of ML programs:

- **Program Analyses for Trustworthy ML.** I have developed novel program analyses to address two key issues in trustworthy ML: interpretability and fairness. I designed the first analysis that generates minimum, robust, and

symbolic corrections as actionable explanations when a judgment made by an ML system is undesirable [NeurIPS'18]. For example, when an ML system rejects a mortgage application, my approach will generate explanations to the applicant like "you will get the loan if you improve your credit score by at least 50". I have applied my analysis to neural networks comprising up-to 4092 neurons that make mortgage decisions, predict theorem solver performance, and recognize drawings. I also designed the first specification language to encode different fairness definitions and the first scalable analysis for verifying these specifications on large real-world models [OOPSLA'19]. My analysis can verify neural networks with up-to 16 millions neurons, which is five orders of magnitude larger than the largest previously-verified neural network [1].

- **Probabilistic Programming with Distributional Inference and Causality.** While analyses are used to improve quality of existing programs, I have also developed languages to help write better new programs. In particular, I have built two extensions to probabilistic programming [2], which has recently attracted significant interest due to its abilities to construct arbitrarily complex distributions and condition on rich predicates. The first extension enables my language to condition on distributional properties, which are properties that summarize the whole output distribution (e.g., expectation, variance, KL divergence, fairness, and robustness) [TR'19a]. I demonstrated its effectiveness by repairing classifiers that are unfair or not robust. The second extension augments probabilistic programming with intervention and counterfactual [TR'19b]. These two language constructs are fundamental to causal inference [3]. Causal inference is drawing increasing attention as most existing ML programs excel at capturing correlations but not causality. I demonstrated the effectiveness of my extension by examples that predicts population dynamics, performs inverse planning, and generates but-for causes [4].

Future Directions

I plan to continue working on the projects under the two previously described directions in the near future. In the longer term, I plan to combine these two directions together and develop ML augmented program reasoning techniques to ensure the quality of ML programs.

Near-Future

I plan to continue to apply ML techniques to program reasoning in the following directions:

- **Learning Effective Designs.** Existing program reasoning systems rely on an expert designer to make various design choices, such as how to handle procedure calls, at what granularity to reason about dynamically allocated objects. These design choices essentially describe how to approximate often unbounded program behaviors and are therefore called "abstractions". A key challenge is how to choose effective abstractions that balance trade-off between efficiency and accuracy. I plan to leverage data-driven approaches to resolve this challenge. In particular, I plan to learn effective abstractions by training them on a large corpse of programs. As a starting point, I am working with a Ph.D. student from Peking University to generate abstractions of library code using offline learning. The idea is that if an abstraction is effective for a library when reasoning about many programs, it is very likely it is also effective when reasoning about a new program. Note this project is complementary to the previous project of reusing results on library code as there are many cases we cannot directly reuse the results but the abstraction.

- **Interactive Reasoning via Active Learning.** I believe that I can further reduce the user effort in the online learning project by applying active learning. The objective in choosing a question to the user would be to maximize the expected false alarm reduction. To estimate how likely an analysis fact is valid, I plan to leverage both statistical methods and dynamic program analysis. Once obtaining the estimations, I plan to encode the question selection problem itself as a probabilistic inference problem.

- **Compositional Inference.** A key reason that modern program reasoning techniques scale to large programs is because they exploit modularity in a program. For example, a wide range of program analyses build reusable summaries on procedures [5]. I plan to exploit this insight in probabilistic inference. As a starting point, I plan to develop compositional solving for Maximum Satisfiability that my solver is built upon. I plan to divide the constraints according to which procedures they are generated from and solve them in the order of calls. To reduce time on analyzing different calls to same procedures, my approach will generate summaries for individual procedure constraints in the form of "assignment $\mapsto$ weight". A key challenge would be how to handle recursions.

I plan to extend my analyses and languages for ML programs in the following directions:

- **Interpretability Analysis for Richer Models and Domains.** I plan to extend my work on interpretability to recurrent neural networks (RNNs) and networks for problems in natural language processing and computer vision. If we treat an RNN naively as a feedforward network by unrolling, its size depends on the size of the input sequence which can be arbitrarily long. To address this scalability challenge, I view an RNN as a loop program and plan to leverage ideas in conventional program analyses that infer loop invariants [6]. These loop invariants will capture the relation between input and output of each RNN node, therefore speed-up the analysis. I believe this idea has broader applications to other analyses for RNNs such as analyses that check robustness and other safety properties. In order to extend my analysis to networks for natural language processing and computer vision, I need to address the challenge that explanations in their input space is not human-understandable. For example, if my current approach is directly applied to a network that recognizes images in pixels, it will generate explanations like "this picture would be a stop sign if you increase the R value of pixel 17-24 by 50". On the other hand, explanations like "this picture is not a stop sign because the letter T misses a cross" are much easier for a user to understand. To bridge this gap, I plan to generate models that map from a description space to the encoding space, and then apply my method to generate explanations in the description space.

- **Synthesizing Actual Causes.** Building atop my causal probabilistic programming language, I plan to develop a framework for synthesizing actual causes [7] in a probabilistic program. I want to answer questions like "we observed a crash in a self-driven car simulator, what caused it?" Potential applications include debugging robotic systems, estimating the effects of medical treatments, and allocating responsibilities in convictions. I plan to solve the problem in three stages: 1) extending the theory of actual causality from graphical models to probabilistic programs, 2) inventing an efficient algorithm to check whether a given predicate is an actual cause, and 3) developing a technique to synthesize a candidate cause that is understandable to a human.

Long-Term

In the next stage of my career, I plan to use ML augmented program reasoning to ensure the quality of ML programs. Such a combination is necessary as unique properties of ML programs pose new challenges that go beyond the scope of conventional program reasoning techniques. I will discuss several key challenges and how I plan to address them with ML techniques below:

- **Learning Incomplete Specifications.** The specification about the application scenario of an ML system (i.e., what inputs are valid) is often incomplete. For example, there often exists uncertainties in the environment of a robotics system (e.g., how a human would react to a robot, what scenarios are possible). Existing program reasoning techniques cannot do much if the specification is not complete. Currently, a user has to manually specify the models about these inputs. This is both time consuming and error-prone. Instead, I want to learn these models automatically. While there exist techniques on learning these models for data generation purpose, these learnt models are not suitable for verification purpose. In particular, there are several properties they do not satisfy. First, the model needs to be compact to be analyzable, whereas most existing models are based on deep neural networks. To address this challenge, I plan to leverage techniques in model compression and interpretability, to extract simpler models from learnt complex models. Second, the models need to be sound. This means that the model should be a super set of the possible legal inputs, which is critical to make a safety argument. While it might be challenging to have fully sound models, I hope to have a model that covers most common cases and address the rest by applying runtime monitors to the system under analysis. Last, the models need to be causal. This is important if the user wants to fix the program or the environment setting when the analysis shows the system to be unsafe.

- **Effective Reasoning of Probabilistic and Distributional Properties.** Many properties of interest in ML programs are either probabilistic or distributional (i.e., properties that summarize the output distribution such as fairness). This is out of the scope of conventional program reasoning techniques which typically deal with simple properties (properties that can be rejected by a single trace) or hyper properties (properties that can be rejected by a finite set of traces). While there exist several instances of program reasoning techniques [1, 8] for such properties including ours [OOPSLA'19], there are unaddressed challenges. First of all, existing approaches, especially white-box approaches that provide strong guarantees, have trouble scaling to large programs (in contrast, my fairness analysis is a black-box approach that provides probabilistic guarantees). Fundamentally, these techniques are solving probabilistic inference problems. To improve the scalability of inference, besides developing novel inference algorithms, I plan to leverage ideas

from convectional program reasoning techniques to create abstractions of programs [9]. An interesting problem would be how to create abstractions that guarantee soundness (i.e., does not miss bugs) for these new properties. Moreover, this direction can potentially improve the performance of probabilistic inference for non-analysis problems. Second, it is unclear how a program reasoning system would provide guidance on bug fixing to a user when it fails to prove a property. Conventional systems reason about simple properties so they provide feedback in the form of counterexample execution traces when detecting a bug. However, in the case of distributional properties such as fairness, it is unclear what useful information an analysis could provide to a user when it fails. This severely affects the usability of these systems.

- **A Language that Unifies Logic and Probability.** There is an increasing need for designing a language that unifies logic and probability. While the logical part encodes the prior from a human user, the probabilistic part encodes uncertainties and enables learning from data. Conventional logical AI such as inductive learning has tractability issues to learn data from complex domains such as computer visions, while it is hard to express human priors succinctly using neural networks. I believe probabilistic programming is the right direction for combining these two extremes. However, we need to address two key issues in inference: 1) there lack efficient algorithms such as gradient descent for neural networks, and 2) most existing algorithms are sampling-based, therefore their results can violate logical constraints, which is undesirable for safety-critical applications. To address the first issue, I plan to leverage ideas in program synthesis to create highly-specialized algorithms for problems from different domains. To address the second issue, I plan to combine symbolic methods such as lifted inference [10] with existing sampling-based methods.

In the next decade, I plan to develop novel program analyses and languages for writing safe, interpretable, and fair ML programs. Using these techniques, I hope to revolutionize the way people write ML programs and build end-to-end safety cases for important systems such as self-driving cars.

References (Past, See CV for Complete List))

[OOPSLA'16] S. Kulkarni, R. Mangal, X. Zhang, and M. Naik. Accelerating program analyses by cross-program training. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2016, part of SPLASH 2016, Amsterdam, The Netherlands, October 30 - November 4, 2016*, pages 359–377, 2016.

[OOPSLA'17] X. Zhang, R. Grigore, X. Si, and M. Naik. Effective interactive resolution of static analysis alarms. *PACMPL*, 1(OOPSLA):57:1–57:30, 2017.

[PLDI'14] X. Zhang, R. Mangal, R. Grigore, M. Naik, and H. Yang. On abstraction refinement for program analyses in datalog. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, pages 239–248, 2014.

[FSE'15] R. Mangal, X. Zhang, A. V. Nori, and M. Naik. A user-guided approach to program analysis. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 462–473, 2015.

[POPL'16] X. Zhang, R. Mangal, A. V. Nori, and M. Naik. Query-guided maximum satisfiability. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2016, St. Petersburg, FL, USA, January 20 - 22, 2016*, pages 109–122, 2016.

[AAAI'16] R. Mangal, X. Zhang, A. Kamath, A. V. Nori, and M. Naik. Scaling relational inference using proofs and refutations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, February 12-17, 2016, Phoenix, Arizona, USA*. Pages 3278–3286, 2016.

[SAT'16] R. Mangal, X. Zhang, A. V. Nori, and M. Naik. Volt: A lazy grounding framework for solving very large maxsat instances. In *Theory and Applications of Satisfiability Testing - SAT 2015 - 18th International Conference, Austin, TX, USA, September 24-27, 2015, Proceedings*, pages 299–306, 2015.

[CP'16] X. Si, X. Zhang, V. M. Manquinho, M. Janota, A. Ignatiev, and M. Naik. On incremental core-guided maxsat solving. In *Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings*, pages 473–482, 2016.

[NeurIPS'18] X. Zhang, A. Solar-Lezama, and R. Singh. Interpreting neural network judgments via minimal, stable, and symbolic corrections. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.* Pages 4879–4890, 2018.

[OOPSLA'19] O. Bastani, X. Zhang, and A. Solar-Lezama. Verifying fairness properties via concentration. *PACMPL*, 3(OOPSLA), 2019.

[TR'19a] Z. Tavares, X. Zhang, E. Minaysan, J. Burroni, R. Ranganath, and A. Solar-Lezama. The random conditional distribution for higher-order probabilistic inference. *CoRR*, abs/1903.10556, 2019. URL: `http://arxiv.org/abs/1903.10556`.

[TR'19b] Z. Tavares, J. Koppel, X. Zhang, and A. Solar-Lezama. A language for counterfactual generative models, 2019.

References (External)

[1] A. Albarghouthi, L. D'Antoni, S. Drews, and A. V. Nori. Fairsquare: probabilistic verification of program fairness. *PACMPL*, 1(OOPSLA):80:1–80:30, 2017.

[2] A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, pages 167–181, 2014.

[3] J. Pearl. *Causality*. Cambridge university press, 2009.

[4] J. Y. Halpern and C. Hitchcock. Actual causation and the art of modeling. *CoRR*, abs/1106.2652, 2011. arXiv: `1106.2652`. URL: `http://arxiv.org/abs/1106.2652`.

[5] T. W. Reps, S. Horwitz, and S. Sagiv. Precise interprocedural dataflow analysis via graph reachability. In *Conference Record of POPL'95: 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Francisco, California, USA, January 23-25, 1995*, pages 49–61, 1995.

[6] C. A. Furia, B. Meyer, and S. Velder. Loop invariants: analysis, classification, and examples. *ACM Comput. Surv.*, 46(3):34:1–34:51, 2014.

[7] J. Y. Halpern and J. Pearl. Causes and explanations: A structural-model approach - part II: explanations. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence, IJCAI 2001, Seattle, Washington, USA, August 4-10, 2001*, pages 27–34, 2001.

[8] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: verification of probabilistic real-time systems. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 585–591, 2011.

[9] P. Cousot and R. Cousot. Abstract interpretation frameworks. *J. Log. Comput.*, 2(4):511–547, 1992.

[10] G. V. den Broeck and J. Davis. Conditioning in first-order knowledge compilation and lifted probabilistic inference. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada*, 2012.