Swarat Chaudhuri's Research Statement

1. Motivation

Software plays an outsized role in the world that we live in, but with this stature come obligations. Given their impact on human lives, software systems must be *trustworthy* and *efficient*. My 20-year old research career, in Programming Languages (PL), Formal Methods (FM), and their interdisciplinary offshoots, has been grounded in the belief that algorithmic tools can help realize these goals. Algorithms for *program analysis* can help us understand complex systems and induce trust in a system's behavior. Algorithms for *program synthesis* can simplify the discovery of complex, reliable, and efficient computations.

Over the years, I have made many foundational contributions to PL and FM. I have applied these results in a range of domains, from automating scripting tasks (Feser et al. 2015) to concurrent programming (Surendran et al. 2014) to robotics and control (Dantam et al. 2016; Wang et al. 2016). My recent research is primarily focused on exploring the interplay between PL/FM and Machine Learning (ML).

ML has fundamentally transformed many areas of computing since the dawn of this century. I am part of a vanguard of researchers who have shown that ML can also address longstanding challenges in PL/FM. Specifically, program analysis and synthesis have traditionally required search through vast combinatorial spaces of programs and correctness proofs. Techniques for *learning to search and generate* can lead to much faster solutions to these problems. Also, FM research has traditionally required detailed formal models of programs and their environments. ML allows such models to be learned from data (gathered through exploration or from pre-existing corpora), thus vastly expanding the charters of the fields.

On the other hand, ML is also an exciting *application domain* for PL/FM ideas. We are careening towards a world in which all significant software systems will contain embedded ML modules. However, trust and efficiency remain essential challenges in ML. The deep neural networks that form the state of the art of ML are black boxes that cannot be interpreted and debugged like traditional software, and that can give incorrect results on out-of-distribution inputs. Because these models lack well-defined specifications, aligning them with a user's goals can be challenging, and composing them with human-engineered software can raise the risk of undefined behavior. Because the models must learn even the most rudimentary knowledge from data, their training often requires inordinate amounts of data and computation.

A central motivation of my research is to address the above challenges using PL/FM insights. High-level language abstractions and symbolic reasoning have long allowed humans to comprehend, debug, verify, and optimize complex software systems. I believe that these tools can also help us develop trustworthy and efficient ML. However, to realize the full potential of these methods, one must go beyond the direct application of PL/FM ideas to ML and instead aim for a deeper synthesis of logical and statistical techniques. My research seeks to create such a synthesis.

2. Neurosymbolic Programming

I concretely realize my vision to unify PL/FM and ML through research on *neurosymbolic programming* (NSP) (Chaudhuri et al. 2021), a new area at the interface of deep learning and program synthesis in which I am a leader. The area has attracted much interest in the recent past and is the subject of an NSF Expedition (`http://neurosymbolic.org`) of which I am a part. One research thrust here is to design algorithms for program synthesis, and related tasks like proof and specification synthesis, that combine logic-based techniques with deep learning. A second thrust is the representation of learning-enabled systems as *neurosymbolic programs* that blend together neural modules and traditional symbolic code. Under such a formulation, learning becomes a program synthesis problem: "Find a program, expressed in a suitable domain-specific language (DSL), that leads to an optimal value of a data-driven quantitative objective and (in some versions of the problem) also satisfies a set of additional syntactic and semantic requirements."

I am excited about this area both as a PL/FM researcher and as a machine learning enthusiast. On the one hand, NSP may hold keys to addressing many of the trust- and efficiency-related concerns about contemporary machine learning. For example, their symbolic components make neurosymbolic programs potentially more interpretable, and easier to analyze, than neural networks. Also, NSP models can make use of prior knowledge specified within a DSL and constructed compositionally using modular abstractions, making learning more data-efficient. On the other hand, NSP promises new lines of attack on long-open PL/FM challenges and also opens up entirely new application spaces to PL/FM techniques.

Now I sketch some of my results in this area, categorized by the application domains that they address.

Programming and Reasoning Systems. The traditional objective of PL/FM research is to build tools that improve the productivity of professional programmers. This goal continues to motivate my research on NSP. However, while traditional PL/FM approaches depend on detailed formal specifications, my work tends to allow these specifications to be *ambiguous* and *incomplete*. At the same time, unlike the purely statistical approaches that are now in vogue in ML-for-code, my methods treat programs as semantic artifacts and seek to satisfy formal specifications whenever they are available.

For example, in 2018, we presented a framework, called Bayou (Murali et al. 2018), for generating idiomatic Java code given the target code's context and some text describing the code. The specification here is incomplete and ambiguous, and we used a neural network to predict the target code's structure given this specification. At the same time, unlike subsequent, purely neural approaches like OpenAI's Codex, Bayou required the generated code to be compilable and type-safe and used a symbolic method to ensure this requirement. In followup work (Mukherjee et al. 2021), we gave a different approach, based on a combination of symbolic attribute grammars and neural models of code, to this problem. We showed that this approach outperforms state-of-the-art neural models (transformers) at longform code generation, both in terms of generating code free of semantic errors, and in terms of syntactically matching a ground truth. I am currently helping a team at Google X deploy some of these ideas in a real-world product.

Another direction (Murali et al. 2017) is *specification synthesis*. Here, we view a specification as a *norm* that programs follow while executing, and learn a model of such norms using a combination of static analysis and deep learning. In downstream correctness checking tasks, we estimate a statistical distance between a program's behavior and the behavior that the model expects from the program. A high distance suggests that the program is anomalous and likely-buggy.

We have also been developing neurosymbolic approaches to formal reasoning tasks like theorem-proving and expression simplification. While these tasks are traditionally solved using rule-based search, recent work has used deep learning to solve them. Our methods closely couple the two threads of work. The results, while not published yet, are promising.

Finally, we have been working on methods for reasoning over natural language text. In Bostrom et al. (2021), we give a way to use language models and data generated from symbolic templates to perform logical inferences over broad-domain natural language. In more recent work (Bostrom et al. 2022), we use a neural-guided search to compose such inference steps into "proofs" representing complex chains of reasoning.

Autonomous Agents. I have also worked extensively on applications of NSP in learning-enabled autonomous agents, e.g., robots and system infrastructures augmented with learning modules. Safety, interpretability, and sample-efficiency are key considerations in these systems. My work has sought to achieve these criteria using neurosymbolic programming.

Specifically, my collaborators and I introduced the *programmatic reinforcement learning* approach, in which policies for reinforcement learning (RL) agents are learned through program synthesis (Verma et al. 2018, 2019; Cheng et al. 2019). For context, an RL policy maps environment states to agent actions, and mainstream RL uses neural networks as policy representations. By contrast, our method represents policies as programmatic compositions of classic control modules (such as PID controllers) and neural networks and synthesizes these programs using neurosymbolic methods. We showed that such programmatic policies can perform complex tasks while having compact, human-interpretable structure, and that they can use the prior knowledge encapsulated in their symbolic modules to generalize more effectively to new environments.

Another research thrust studies the formal verification of neural and neurosymbolic models, and learning algorithms that invoke verifiers during training. In particular, we have developed novel methods for analyzing the safety and robustness of neural networks (Gehr et al. 2018; Anderson et al. 2019). We gave the first RL algorithm (Anderson et al. 2020) that can leverage modern policy gradient methods for learning, while also ensuring provable safety during exploration. Recently, we gave the first method (Yang and Chaudhuri 2022) to learn provably safe, high-performance parameters for programs that compose neural networks with symbolic, human-written code. Through ongoing collaborations, we are working to deploy these methods in real-world robots and learning-enhanced system software (e.g., neural congestion controllers).

Finally, we have been exploring ways to exploit the *compositionality* of high-level programming languages to transfer knowledge across learning settings. For example, in Valkov et al. (2018), we consider an approach to lifelong learning in which agents use higher-order functional combinators to compose and finetune neural functions from an evolving "library". We show that this modular approach improves the data-efficiency of learning. In ongoing work, we are exploring these ideas to transfer skills from simulated to real-world environments.

Data-Driven Discovery. Another objective is to accelerate *data-driven discovery* in the natural sciences and healthcare. ML has found many recent applications in these domains. However, interpretability and data-efficiency are often key considerations here, making blackbox deep learning an imperfect fit. My work has proposed NSP as an alternative approach that allows interpretable, data-efficient learning modulo strong forms of prior knowledge.

In particular, I have explored program synthesis approaches in *behavior analysis* (e.g., analysis of videos of lab animals). Shah et al. (2020) studies the discovery of programs that label behaviors with actions at various points. Zhan et al. (2021) presents neurosymbolic methods for interpretable clustering, and Tjandrasuwita et al. (2021) uses program synthesis to explain differences between different human annotators of laboratory data. We are currently working to extend these ideas to the synthesis of *causal* programmatic models and program synthesis from perceptual data.

A different line of work, with collaborators in cell biology, uses program synthesis to discover mechanistic models of RNA splicing. Yet another collaboration, with precision medicine researchers, is using program synthesis to learn causal, interpretable models of decision-making that doctors in ICUs can follow.

Algorithmic Themes

The success of the NSP agenda critically depends on effective methods for program synthesis. While PL/FM researchers have worked on program synthesis for a long time, the landscape of the area has rapidly changed over the last few years, as new approaches that synergistically combine symbolic and statistical techniques have emerged. My work has contributed several key algorithmic ideas to this space. Now I summarize a few of these ideas.

Sketch Learning. Sketch learning (Murali et al. 2018) was one of my early algorithmic contributions to NSP. The objective here is to learn, from a given code corpus, a generative model of programs that are constrained to satisfy a set of basic invariants (e.g., type-safety). Such a goal is hard to meet using neural models trained directly on program syntax. By contrast, our approach trained a deep generative model over symbolically-constructed *sketches* of programs, and concretized generated abstractions into programs using an incremental symbolic synthesis method. In doing so, it benefited from the excellence of PL/FM methods at getting low-level details right as well as the ability of neural models to recognize high-level patterns in data. Since our paper, sketch learning has come to be used in many other settings.

Symbolically Supervised Neural Models. We have also been exploring program synthesis techniques in which an end-to-end neural model (for example, a transformer) is used to generate code, but symbolic methods are used to expose rich semantic knowledge to the model. Specifically, in our recent work on *neurosymbolic attribute grammars* (Mukherjee et al. 2021), a neural model of code is allowed to query a program analyzer (an attribute grammar) to discover properties of the code that it has generated. The decision of what to generate next is conditioned on these properties. We found this additional semantic information to significantly enhance the neural model's capabilities. In current work, we are exploring versions of this approach involving richer forms of semantic information (for example, equivalences between terms).

Neural Relaxations. One of my key achievements in NSP is the invention of synthesis methods that use neural networks as *relaxations* of symbolic programs. The observation here is that neural networks and symbolic programs are just two different representations of functions. However, neural networks are more expressive, and their differentiability allows efficient gradient-based learning. Hence, a neural network can be used as an easily-learned approximation of a program.

Our first method along these lines (Verma et al. 2018) followed a simple two-step process: learn a neural network that is a good match for a task specification, then distill this network into a symbolic program with similar behavior. A later paper (Verma et al. 2019) proposed an iterative method that repeatedly: (i) *relaxes* a symbolic program into a neurosymbolic program by adding a neural component; (ii) *updates* this neurosymbolic program using approximate gradients; and (iii) distills the updated neurosymbolic program into a new symbolic program. We showed that this process can be framed as a form of mirror descent and gave learning-theoretic analysis and empirical results justifying its value.

A subsequent paper (Shah et al. 2020) uses neural relaxations to direct a top-down heuristic search over programs. The method relaxes the discrete set of completions for each partial program encountered during the search into a parameterized neural network. We showed that the optimal loss for this neural network can be used as an *admissible heuristic* for the higher-level search and found that such learned heuristics significantly accelerate the search process over programs.

Integrating Learning and Verification. Methods that closely couple formal verification and learning form a key dimension of my research. The objective here is to learn neurosymbolic programs (and as special cases, neural networks and symbolic programs) that are guaranteed to satisfy a set of formal correctness constraints. A possible approach to this task is to learn a program first and then to verify it. However, there is no reason why a learner that has not encountered a verification goal during training should converge to a verifiable program. Consequently, my work follows the different strategy of allowing the learning algorithm to *query* a verifier during training.

Scalability is a key challenge here, as neural network verification is computationally expensive. We have given several new approaches to this challenge. For example, in Anderson et al. (2020), we gave a form of RL in which policies are represented as neurosymbolic programs and are formally verified during each training step. The neural components of these programs are learned using gradients. However, the verification steps *abstract* the neurosymbolic programs into purely symbolic programs, avoiding explicit neural network verification. In Yang and Chaudhuri (2022), verified parameter learning is reduced to gradient-based optimization. This is done through an approximation that only performs safety analysis for a sampled set of program paths at each training step, uses lossy abstract interpretation to derive a quantitative safety signal for each path, and stochastically estimates the gradient of the aggregate safety signal across paths.

Neurosymbolic Probabilistic Programming. Finally, we have been working on ways to connect neurosymbolic and probabilistic programming. Our main contribution here is a framework in which neurosymbolic programs are used to encode probabilistic and causal models. Algorithmic problems here include approximating the posterior of a programmatically represented distribution given a set of observations (expressed in a suitable logic), and discovering the causal structure of a model. We approach these problems using a mix of contemporary causal and statistical techniques and symbolic methods for searching over program structures and propagating symbolic information through programs.

3. Other Work

I worked for many years on classical formal methods before getting interested in the intersection of PL/FM and ML. Below, I summarize some of the highlights of my post-Ph.D. research on these themes. I continue to pursue some of these topics — in particular, symbolic program synthesis — in my current work. Work on some of the other topics, e.g., smooth interpretation or robustness analysis of programs, directly influenced my work on PL/FM-meet-ML.

Symbolic Program Synthesis. Over the years, I have worked on a wide variety of symbolic approaches to program synthesis. For example, in Beyene et al. (2014), we gave a constraint-based approach to the automatic synthesis of correct-by-construction reactive controllers. While the synthesis of finite-state reactive controllers has been well-studied since the 1980s, we were the first to study the automatic synthesis of *infinite-state* controllers. In Feser et al. (2015), we presented λ^2, an approach to the type- and example-directed synthesis of higher-order functional programs that has come to be broadly influential. In recent work (Miltner et al. 2022), we gave the first method for performing bottom-up synthesis (an established strategy in the program synthesis literature) of recursive programs.

I have pursued applications of these methods in a wide range of domains. Of these, the work on program synthesis for robotics deserves a specific mention. The goal here was to synthesize robot motions from symbolic models of the robot and its environment and formal correctness properties like reachability and safety. The main challenge was to simultaneously reason about two different levels of abstraction: a discrete *task level* that is concerned with the high-level goals of the robot, and a continuous *motion level* that is concerned with navigating and manipulating a physical space. We approached this challenge using a series of algorithms (Dantam et al. 2016; Nedunuri et al. 2014; Wang et al. 2016, 2018) that combined formal techniques for task-level planning and geometric algorithms for lower-level motion planning.

Smooth Interpretation and Smoothed Proof Search I was one of the first researchers to use continuous optimization techniques in program synthesis. Years before "differentiable programming" became a mainstream term in computer science, we proposed *smooth interpretation* (Chaudhuri and Solar-Lezama 2010, 2011, 2012), a method that systematically approximates programs by smooth mathematical functions, then finds optimal parameters for these programs using numerical optimization techniques. A subsequent paper (Chaudhuri et al. 2014a) proposed *smoothed proof search*, a method for discovering optimal program parameters that are also *provably safe*. This work introduced a notion of smooth parameterized relaxations of provers of program correctness (specifically, abstract interpreters). Program synthesis was then cast as a problem of optimizing parameters of these relaxations. Since this work, differentiable abstract interpretation has become mainstream as an approach to adversarially robust ML, and our recent work on verified neurosymbolic learning (Yang and Chaudhuri 2022) was inspired by this method.

Robustness Analysis of Programs Many researchers, myself included, have recently studied methods for verifying ML models to be *robust* to adversarially perturbed inputs (Gehr et al. 2018; Anderson et al. 2019). Years before these results, I was studying formal methods for verifying the robustness of programs. In Chaudhuri et al. (2010), we proposed a definition of robustness in terms of mathematical continuity and gave an algorithm for verifying that a program is robust in this sense. A followup paper (Chaudhuri et al. 2011) defined robustness in terms of Lipschitz-continuity and gave a method for computing provable bounds on the Lipschitz constant of a program. Our subsequent work explored several other robustness properties, adapting definitions for continuous dynamical systems and geometric algorithms (Samanta et al. 2013a,b; Chaudhuri et al. 2014b). The algorithmic ideas in these papers drew on tools from a varierty of fields, including automata theory, symbolic program verification, and numerical error analysis.

4. Conclusion

Over the last two decades, innovations in machine learning have hit computer science with the force of a tsunami. The power of these ideas is beginning to be felt in the areas of PL and FM as well. I find these developments to be exciting for two reasons. First, I believe that ML holds keys to solving two of the greatest impediments to the adoption of FM: the need for detailed specifications and the complexity of searching for programs and proofs. Second, I believe that the problems that modern ML has with trust, safety, and efficiency cannot be wished away. Principled abstraction and reasoning techniques, developed by PL/FM researchers through decades of painstaking work, can play a critical role in solving these challenges.

In particular, I believe that neurosymbolic programming is a natural way to bring ML and PL/FM ideas together in a way that enables both to live up to their potential. I am also excited about the technical problems in this nascent field, in particular, the way they straddle the grand traditions of logic and statistics. Over my two and a half years at UT Austin, I have built up a large lab that is almost entirely focused on this area. I am very proud of the work that the students and postdocs in the lab are doing, and I expect that we will produce many high-impact results in the years to come.

References

John K. Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 229–239, 2015.

Rishi Surendran, Raghavan Raman, Swarat Chaudhuri, John M. Mellor-Crummey, and Vivek Sarkar. Test-driven repair of data races in structured parallel programs. In Michael F. P. O'Boyle and Keshav Pingali, editors, *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, pages 15–25. ACM, 2014.

Neil T. Dantam, Zachary K. Kingston, Swarat Chaudhuri, and Lydia E. Kavraki. Incremental task and motion planning: A constraint-based approach. In *Robotics: Science and Systems XII*, 2016. URL `http://www.roboticsproceedings.org/rss12/p02.html`.

Yue Wang, Neil T. Dantam, Swarat Chaudhuri, and Lydia E. Kavraki. Task and motion policy synthesis as liveness games. In *Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, ICAPS 2016, London, UK, June 12-17, 2016.*, page 536, 2016. URL `http://www.aaai.org/ocs/index.php/ICAPS/ICAPS16/paper/view/13146`.

Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, Yisong Yue, et al. *Neurosymbolic Programming*. Now Publishers, 2021.

Vijayaraghavan Murali, Letao Qi, Swarat Chaudhuri, and Chris Jermaine. Neural sketch learning for conditional program generation. In *International Conference for Learning Representations (ICLR)*, 2018. URL `http://arxiv.org/abs/1703.05698`. **(Oral Presentation)**.

Rohan Mukherjee, Yeming Wen, Dipak Chaudhari, Thomas Reps, Swarat Chaudhuri, and Christopher Jermaine. Neural program generation modulo static analysis. *Advances in Neural Information Processing Systems*, 34, 2021. **(Spotlight Presentation)**.

Vijayaraghavan Murali, Swarat Chaudhuri, and Chris Jermaine. Bayesian specification learning for finding API usage errors. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017*, pages 151–162, 2017.

Kaj Bostrom, Xinyu Zhao, Swarat Chaudhuri, and Greg Durrett. Flexible generation of natural language deductions. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2021.

Kaj Bostrom, Zayne Sprague, Swarat Chaudhuri, and Greg Durrett. Natural language deduction through search over statement compositions. *arXiv preprint arXiv:2201.06028*, 2022.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, pages 5052–5061, 2018. URL `http://proceedings.mlr.press/v80/verma18a.html`. **(Long Talk)**.

Abhinav Verma, Hoang M. Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. In *Neural Information Processing Systems (NeurIPS)*, 2019.

Richard Cheng, Abhinav Verma, Gábor Orosz, Swarat Chaudhuri, Yisong Yue, and Joel Burdick. Control regularization for reduced variance reinforcement learning. In *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, pages 1141–1150, 2019.

Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. Ai^2: Safety and robustness certification of neural networks with abstract interpretation. In *Security and Privacy (SP), 2018 IEEE Symposium on*, 2018.

Greg Anderson, Shankara Pailoor, Isil Dillig, and Swarat Chaudhuri. Optimization and abstraction: a synergistic approach for analyzing neural network robustness. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019, Phoenix, AZ, USA, June 22-26, 2019*, pages 731–744, 2019. **(Distinguished Paper)**.

Greg Anderson, Abhinav Verma, Isil Dillig, and Swarat Chaudhuri. Neurosymbolic reinforcement learning with formally verified exploration. *Advances in neural information processing systems*, 33:6172–6183, 2020.

Chenxi Yang and Swarat Chaudhuri. Safe neurosymbolic learning with differentiable symbolic execution. In *ICLR*, 2022.

Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles A. Sutton, and Swarat Chaudhuri. Synthesis of differentiable functional programs for lifelong learning. In *Neural Information Processing Systems (NeurIPS)*, 2018.

Ameesh Shah, Eric Zhan, Jennifer Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. Learning differentiable programs with admissible neural heuristics. *Advances in neural information processing systems*, 33:4940–4952, 2020.

Eric Zhan, Jennifer J Sun, Ann Kennedy, Yisong Yue, and Swarat Chaudhuri. Unsupervised learning of neurosymbolic encoders. *arXiv preprint arXiv:2107.13132*, 2021.

Megan Tjandrasuwita, Jennifer J Sun, Ann Kennedy, Swarat Chaudhuri, and Yisong Yue. Interpreting expert annotation differences in animal behavior. *arXiv preprint arXiv:2106.06114*, 2021.

Tewodros A. Beyene, Swarat Chaudhuri, Corneliu Popeea, and Andrey Rybalchenko. A constraint-based approach to solving games on infinite graphs. In Suresh Jagannathan and Peter Sewell, editors, *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 221–234. ACM, 2014.

Anders Miltner, Adrian Trejo Nuñez, Ana Brendel, Swarat Chaudhuri, and Isil Dillig. Bottom-up synthesis of recursive functional programs using angelic execution. *Proc. ACM Program. Lang.*, 6(POPL):1–29, 2022. **(Distinguished Paper)**.

Srinivas Nedunuri, Sailesh Prabhu, Mark Moll, Swarat Chaudhuri, and Lydia E. Kavraki. Smt-based synthesis of integrated task and motion plans from plan outlines. In *2014 IEEE International Conference on Robotics and Automation, (ICRA)*, pages 655–662, 2014.

Yue Wang, Swarat Chaudhuri, and Lydia E. Kavraki. Bounded policy synthesis for pomdps with safe-reachability objectives. In *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS 2018, Stockholm, Sweden, July 10-15, 2018*, pages 238–246, 2018. URL `http://dl.acm.org/citation.cfm?id=3237424`.

Swarat Chaudhuri and Armando Solar-Lezama. Smooth interpretation. In *Proceedings of the 2010 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2010, Toronto, Ontario, Canada, June 5-10, 2010*, pages 279–291, 2010.

Swarat Chaudhuri and Armando Solar-Lezama. Smoothing a program soundly and robustly. In *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, pages 277–292, 2011.

Swarat Chaudhuri and Armando Solar-Lezama. Euler: A system for numerical optimization of programs. In *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, pages 732–737, 2012.

Swarat Chaudhuri, Martin Clochard, and Armando Solar-Lezama. Bridging boolean and quantitative synthesis using smoothed proof search. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 207–220, 2014a.

Swarat Chaudhuri, Sumit Gulwani, and Roberto Lublinerman. Continuity analysis of programs. In *37th ACM Symposium on Principles of Programming Languages (POPL)*, 2010.

Swarat Chaudhuri, Sumit Gulwani, Roberto Lublinerman, and Sara NavidPour. Proving programs robust. In *SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011*, pages 102–112, 2011. **(Distinguished paper; CACM research highlight)**.

Roopsha Samanta, Jyotirmoy V. Deshmukh, and Swarat Chaudhuri. Robustness analysis of string transducers. In *Automated Technology for Verification and Analysis - 11th International Symposium, ATVA 2013, Hanoi, Vietnam, October 15-18, 2013. Proceedings*, pages 427–441, 2013a.

Roopsha Samanta, Jyotirmoy V. Deshmukh, and Swarat Chaudhuri. Robustness analysis of networked systems. In *Verification, Model Checking, and Abstract Interpretation, 14th International Conference, VMCAI 2013, Rome, Italy, January 20-22, 2013. Proceedings*, pages 229–247, 2013b.

Swarat Chaudhuri, Azadeh Farzan, and Zachary Kincaid. Consistency analysis of decision-making programs. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 555–568, 2014b.