Research Summary

Mooly Sagiv

December 2015

1 Research Interests

My research focuses on easing the task of developing reliable and efficient programs via program analysis. The aim is to create powerful tools for manipulating programs and analyzing their properties that will help programmers develop correct, reliable, efficient, and secure software. The tools are based on understanding the meaning of the underlying programming language as well as some desired specified properties.

2 Research Philosophy

The general belief in the formal methods and the programming language community is that Automation and Expressiveness contradict each other. On the one hand, there has been a lot of progress with approaches like the Coq proof assistant (used for challenging tasks like verifying compilers) which is very expressive but hard to use. On the other hand, simple type checkers are fully automatic but usually settle for proving weak properties (see Figure 1).

In my research I am taking a radical approach by trying to prove that one can achieve high degree of automation and yet prove expressive properties by incorporating domain knowledge. In doing so, I have sought a balance between principled formal systems and practical algorithms and tools. For example, the TVLA system [32, 21] can automatically prove interesting properties of some programs creating dynamically evolving graphs. The IVY system [25] is another point in the design space (see Section 5.3). However, in order to prove this thesis, I believe it is crucial to develop domain specific knowledge on the kind of properties which need to be defined and the ways domain experts work. I find it exciting since it involves interdisciplinary research. This is achieved via collaborations between formal methods and other fields. I am already collaborating with top researchers in finite model theory on the one hand (Neil Immerman), and networks and systems on the application side (Katerina Argyraki, Michael Schapira, Scott Shenker, and Nickolai Zeldovich).

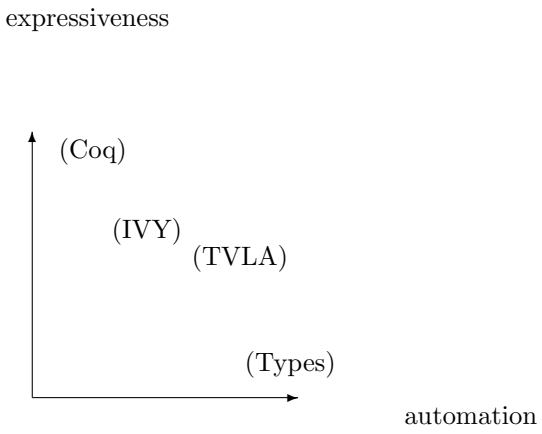

Figure 1: The Tread-offs between Expressive Power and Automation in Formal Verification.

Influence

- Mentored outstanding students in my university and abroad.
- Algorithms integrated into compilers (IBM AS/400) , software understanding tools (Panaya) and software verification (SLAM/SDV).
- Techniques for shape analysis (domains and materialization) implemented in the shape-analysis tools that Facebook is deploying to identify memory errors.
- Closely interacted with research organizations including IBM and MSR.
- Award 1.57 million Euros for 5 years in a prestigious advanced ERC grant. ERC grants allow exceptional established research leaders of any nationality and any age to pursue ground-breaking, high-risk projects that open new directions in their respective research fields or other domains. The ERC Advanced Grant funding targets researchers who have already established themselves as independent research leaders in their own right.
- Most cited in POPL according to a study by Sumit Gulwani.
- The "Kevin Bacon of the PLDI community" according to a study in the Programming Language Enthusiast.

3 Research Highlights

I have contributed to the theory and practice of program analysis. The main contributions are summarized below.

Efficient algorithms for interprocedural dataflow analysis and program slicing [30, 29, 15, 33]. These algorithms improve the asymptotic complexity of analyzing procedures, and enable verification to scale to larger programs. They were used in Microsoft's SLAM system for verifying the safety of Windows device drivers. They are implemented in standard tools for software analysis including Soot [2]. They are also used for analyzing Android applications [3]. The slicing work received an ACM SIGSOFT Retrospective Impact Paper Award (2011). The slicing algorithms were also integrated into the Panaya impact analysis tool for ERP[6]. Panaya was acquired by Infosys.

Shape analysis. Shape analysis is the problem of analyzing programs with dynamically allocated memory. When we started to work on shape analysis, the problem was considered too difficult to solve. We developed a new abstraction framework and key algorithms that solve the problem in most practical cases [31, 32, 21, 22, 20]. These algorithms influenced the shape-analysis tools that Facebook is deploying to identify memory errors. Our paper on Parametric Shape Analysis [32] has over 1,000 Google Scholar citations.

Automatic reasoning about reachability in directed graphs. The correctness of many systems depends on proving the existence (or absence) of paths between sets of nodes in a directed graph evolves as the program runs. Examples include dynamically allocated linked data structures, routing in computer networks, and parameterized distributed protocols. We established lower and upper bounds on the problem [19, 18, 17]. The upper bounds permit reasoning about certain usages of unbounded paths in directed paths using existing SAT solvers. This is surprising since, in general, transitive closure is not even expressible in first-order logic. However, for a rich class of programs, our methodology effectively transforms correctness statements involving reachability to propositional formulas. The formulas are unsatisfiable exactly when the correctness statements are true. Thus SAT solvers provide us with proofs of correctness or counterexample execution sequences.

Composing operations in concurrent data structures. Modern programming languages provide efficient concurrent data structures; however, composing or combining operations on data structures can lead to errors or performance bottlenecks. In [34], we showed that 44% of public-domain usages of Concurrent Map operations are incorrect. This led to modifications to the Java concurrent library by enriching the interface. We also worked on theoretical and practical methods to compose operations on concurrent data structures automatically [11, 13, 14, 12, 7, 8, 35, 39, 9, 38]. Part of the work appears in [10] (2012 ACM Dissertation Award Honorable Mention) and received Best Paper Awards at PLDI'11 and PLDI'12.

4 Opportunities

We are living in an exciting time for programming languages and formal methods. They are finding increasing use in many other areas of computer science including networks, systems and databases. Tools such as proof-assistants (e.g., Coq) and SMT solvers (e.g., Z3) are already deployed in academia and industry. Nevertheless, major theoretical and practical barriers reduce the applicability of these methods. In particular, there are several open problems: (i) the lack of good ways to specify correctness, (ii) the complexity of the verification process, (iii) the steep learning curve required to use formal method tools. There are several ways to improve the situation including the following:

- Identifying domains where correctness can be naturally defined and work with domain experts to understand what needs to be verified.
- Identifying domains in which it is useful to develop automatic methods for core functionality. Examples include distributed protocols, cloud applications, and programs written by students, e.g., for online teaching.
- Involve the programmer in the task. For example, the programmer can define the space of interesting invariants or even assist in defining the ingredients of the invariants.
- Deploy techniques from machine learning in order to infer properties which are understood by the programmer.

Plan In the next 10 years, I am hoping to develop useful techniques in order to change the ways modern software is built addressing the above challenges. These methods will be implemented in real tools and will increase our understanding of the limitations of automatic methods.

5 Current Projects

I am currently involved in three related projects addressing some of the above challenges.

5.1 Analyzing Distributed Systems and Networks

This is a joint project with Katerina Argyraki (EPFL), Michael Schapira (HUJI), Scott Shenker (UCB), and Nickolai Zeldovich(MIT). We are aiming to develop techniques for enforcing safety properties of distributed systems including computer networks. We are also interested in analyzing sophisticated distributed protocols such as Chord.

5.1.1 Preliminary Results

Vericon: Semi-Automatic Verification of SDN control programs In [4], we apply deductive verification for ensuring the correctness of simple Software Defined (SDN) controller programs.

Synthesising provably correct forwarding rules In [23], we show how to automatically install SDN forwarding rules that guarantee a given forwarding policy is maintained.

Reasoning about network states In [26, 27], we show how to verify safety of networks with mutable states using SMT solvers. In [36], we studied the asymptotic complexity of stateful network verification and consider cases in which it is feasible to verify safety.

5.2 Analyzing Cloud Applications

This is a brand new joint project with Aurojit Panda (UCB), Madan Musuvathi(MSR), and Noam Rinetzky (TAU). The theme of this project is to apply static and dynamic program analysis to improve the utility of cloud programming. Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. However, very little tool support, and in particular compile-time tools, are available for helping client programmers produce correct and efficient programs. We are interested in a specific class of cloud applications where distribution is used to scale data processing. In this class, programmers deploy high-level frameworks like Dryad [16], Hadoop [1], and Spark [37] to handle low level system issues such as communication, fault tolerance, and scheduling. The code is written using specialized APIs. This drastically simplifies the programming task and allows novice programmers to build interesting distributed client systems. However, a naive program can suffer from huge performance penalties hidden in the interactions between the client code and system's implementation. Moreover, resource-management is a non-trivial task even for expert users due to irregular input data. Indeed, the use of these frameworks is a double-edged sword as it isolates the developer from how her application actually executes. As cloud-based programming becomes main-stream, the lack of tools that can help client programmers verify that their software works as expected becomes a severe disability. Such tools should consider properties such as functional correctness, service availability, reliability, performance and security guarantees [5]. The kind of applications we are interested in are a perfect fit for formal method techniques as they are often written over frameworks that provide most of the generic services required for distributed processing and expose a domain-specific programming model. This leads to rather small programs, however, efficient use

of these frameworks is predicated on understanding the intricacies of the programming model and the underlying execution engine.

5.3 Diagnosable Semi-Automatic Verification

This is a joint project with Nikolaj Bjørner (MSR), Neil Immerman (UMASS), Ken McMillan (MSR), and Sharon Shoham (TAU). It derives its motivation from the other projects of analyzing distributed systems and cloud applications.

Despite several decades of research, the problem of formal verification of systems with unboundedly many states has resisted effective automation. Our hypothesis is that automated methods are difficult to apply in practice not primarily because they are unreliable, but rather because they are opaque. That is, they fail in ways that are difficult for a human user to understand and to remedy. A practical heuristic method, when it fails, should fail visibly, in the sense that the root cause of the failure is observable to the user. This is in contrast to existing automatic tools which sometimes fail in a non-comprehensible way. We are trying to identify interesting domains in which the verification problem is tractable. We are considering interactive verification.

A standard method for proving safety properties of unbounded systems is by finding *inductive invariants*, i.e., properties which hold in the initial states and are preserved by every step of the execution.

5.3.1 Preliminary Results

Effectively Propositional Reasoning about Unbounded Paths In [18, 17] we showed how to harness existing SAT solvers for reasoning in a sound and complete way about deterministic paths in a dynamically evolving graphs. Here by deterministic we mean that there exist at one outgoing edge used in the path. The main idea is to reduce the verification problem to finding the (un)satisafiability of a formula in Effectively Propositional Logic (EPR)[28]. We believe that these results can be generalized to reason about networks and simple distributed protocols. These results allows to automatically check the correctness of such systems in a sound and complete way. However, they rely on providing inductive invariants by the programmer which can be hard in practice.

Decidability of Inferring Universal Inductive Invariants In [24], we study the decidability of inferring EPR invariants. This article provides a first step towards understanding when invariants can be inferred.

Ivy: Interactive Verification of Parameterized Systems via Effectively Propositional Reasoning The design and implementation of parametric systems can be very tricky even for experienced researchers. In [25], we describe an interactive system — Ivy — for interactively verifying parameterized systems. Ivy is based on the following principles:

- Ivy first attempts to locate counterexamples by bounding the number of protocol actions and symbolically searching for (unbounded) bad inputs.

- Invariants in Ivy are expressed as universal formulas in relational first-order logic. Their inductiveness check reduces to unsatisfiability in Effectively Propositional Logic (EPR). This guarantees that the tool can always decide whether an invariant is inductive or not. Furthermore, our use of universal formulas guarantees that counterexamples to induction can be presented graphically, allowing inspection by humans.

- Users can have better insights than the tool by suggesting candidate local invariants, whose conjunction comprises a global inductive invariant.

6 Usable Formal Methods

My work is part of a larger movement to employ formal methods to reduce concerns about system reliability and security. It can also lead to better performance either by verifying complex systems or by synthesising provably correct optimized code. Finality, it can improve software agility.

I believe that the domain knowledge is a key enabler to the success of formal methods in other areas of computer science. Moreover, domains such as computer systems and networks, distributed protocols and databases are good fit. Programming is tricky in these domains and yet there is a constant demand for innovation. Furthermore, it is possible to develop modular techniques which reason about one piece of a code at the time. I am determined to make the revolution happen with help from outstanding students and colleagues.

References

[1] Open-source implementation of mapreduce. http://hadoop.apache.org.

[2] S. Arzt and E. Bodden. Reviser: efficiently updating ide-/ifds-based data-flow analyses in response to incremental program changes. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 288–298, 2014.

[3] S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. L. Traon, D. Octeau, and P. McDaniel. Flowdroid: precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, page 29, 2014.

[4] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky. Vericon: towards verifying controller programs in software-defined networks. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014*, page 31, 2014.

[5] S. Bouchenak, G. Chockler, H. Chockler, G. Gheorghe, N. Santos, and A. Shraer. Verifying cloud services: present and future. *Operating Systems Review*, 47(2):6–19, 2013.

[6] N. Dor, T. Lev-Ami, S. Litvak, M. Sagiv, and D. Weiss. Customization change impact analysis for erp professionals via program slicing. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2008, Seattle, WA, USA, July 20-24, 2008*, pages 97–108, 2008.

[7] G. Golan-Gueta, G. Ramalingam, M. Sagiv, and E. Yahav. Concurrent libraries with foresight. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, pages 263–274, 2013.

[8] G. Golan-Gueta, G. Ramalingam, M. Sagiv, and E. Yahav. Automatic semantic locking. In *ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP '14, Orlando, FL, USA, February 15-19, 2014*, pages 385–386, 2014.

[9] G. Golan-Gueta, G. Ramalingam, M. Sagiv, and E. Yahav. Automatic scalable atomicity via semantic locking. In *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2015, San Francisco, CA, USA, February 7-11, 2015*, pages 31–41, 2015.

[10] P. Hawkins. *Data Representation Synthesis*. PhD thesis, Stanford University, Apr. 2012. **ACM Doctoral Dissertation Award — Honorable Mention**.

[11] P. Hawkins, A. Aiken, K. Fisher, M. C. Rinard, and M. Sagiv. Data representation synthesis. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 38–49, 2011. **Best Paper Award**.

[12] P. Hawkins, A. Aiken, K. Fisher, M. C. Rinard, and M. Sagiv. Concurrent data representation synthesis. In *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '12, Beijing, China - June 11 - 16, 2012*, pages 417–428, 2012. **Best Paper Award**.

[13] P. Hawkins, A. Aiken, K. Fisher, M. C. Rinard, and M. Sagiv. Reasoning about lock placements. In *Programming Languages and Systems - 21st European Symposium on Programming, ESOP 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia, March 24 - April 1, 2012. Proceedings*, pages 336–356, 2012.

[14] P. Hawkins, M. C. Rinard, A. Aiken, M. Sagiv, and K. Fisher. An introduction to data representation synthesis. *Commun. ACM*, 55(12):91–99, 2012.

[15] S. Horwitz, T. Reps, and M. Sagiv. Demand interprocedural dataflow analysis. In *Proceedings of the Third ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 104–115, New York, NY, Oct. 1995. ACM Press.

[16] M. Isard, M. Budiu, Y. Yu, A. Birrell, and D. Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *Proceedings of the 2007 EuroSys Conference, Lisbon, Portugal, March 21-23, 2007*, pages 59–72, 2007.

[17] S. Itzhaky, A. Banerjee, N. Immerman, O. Lahav, A. Nanevski, and M. Sagiv. Modular reasoning about heap paths via effectively propositional formulas. In *The 41st Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '14, San Diego, CA, USA, January 20-21, 2014*, pages 385–396, 2014.

[18] S. Itzhaky, A. Banerjee, N. Immerman, A. Nanevski, and M. Sagiv. Effectively-propositional reasoning about reachability in linked data structures. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 756–772, 2013.

[19] T. Lev-Ami, N. Immerman, T. W. Reps, M. Sagiv, S. Srivastava, and G. Yorsh. Simulating reachability using first-order logic with applications to verification of linked data structures. *Logical Methods in Computer Science*, 5(2), 2009.

[20] T. Lev-Ami, N. Immerman, and S. Sagiv. Abstraction for shape analysis with fast and precise transformers. In *Computer Aided Verification, 18th International Conference, CAV 2006, Seattle, WA, USA, August 17-20, 2006, Proceedings*, pages 547–561, 2006.

[21] T. Lev-Ami and M. Sagiv. TVLA: A system for implementing static analyses. pages 280–301, 2000.

[22] R. Manevich, E. Yahav, G. Ramalingam, and S. Sagiv. Predicate abstraction and canonical abstraction for singly-linked lists. In *Verification, Model Checking, and Abstract Interpretation, 6th International Conference, VMCAI 2005, Paris, France, January 17-19, 2005, Proceedings*, pages 181–198, 2005.

[23] O. Padon, N. Immerman, A. Karbyshev, O. Lahav, M. Sagiv, and S. Shoham. Decentralizing SDN policies. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2015, Mumbai, India, January 15-17, 2015*, pages 663–676, 2015.

[24] O. Padon, N. Immerman, S. Shoham, A. Karbyshev, and M. Sagiv. Decidability of inferring inductive invariants. In *The 43rdt Annual ACM*

SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL '16, San Petersburg, FL, USA, January 20-22, 2016, 2016. To appear.

[25] O. Padon, K. McMillan, A. Panda, M. Sagiv, and S. Shoham. Ivy: Interactive verification of parameterized systems via effectively propositional reasoning. Submitted for publication, 2015.

[26] A. Panda, K. J. Argyraki, M. Sagiv, M. Schapira, and S. Shenker. New directions for network verification. In *1st Summit on Advances in Programming Languages, SNAPL 2015, May 3-6, 2015, Asilomar, California, USA*, pages 209–220, 2015.

[27] A. Panda, O. Lahav, K. J. Argyraki, M. Sagiv, and S. Shenker. Verifying isolation properties in the presence of middleboxes. *CoRR*, abs/1409.7687, 2014.

[28] R. Piskac, L. M. de Moura, and N. Bjørner. Deciding effectively propositional logic using DPLL and substitution sets. *J. Autom. Reasoning*, 44(4):401–424, 2010.

[29] T. Reps, S. Horwitz, and M. Sagiv. Precise interprocedural dataflow analysis via graph reachability. pages 49–61, New York, NY, 1995. ACM Press.

[30] T. W. Reps, S. Horwitz, S. Sagiv, and G. Rosay. Speeding up slicing. In *SIGSOFT '94, Proceedings of the Second ACM SIGSOFT Symposium on Foundations of Software Engineering, New Orleans, Louisiana, USA, December 6-9, 1994*, pages 11–20, 1994. **Received ACM SIGSOFT Retrospective Impact Paper Award**.

[31] M. Sagiv, T. Reps, and R. Wilhelm. Solving shape-analysis problems in languages with destructive updating. *ACM Trans. Prog. Lang. Syst.*, 20(1):1–50, Jan. 1998.

[32] M. Sagiv, T. Reps, and R. Wilhelm. Parametric shape analysis via 3-valued logic. *ACM Trans. Prog. Lang. Syst.*, 24(3):217–298, 2002.

[33] S. Sagiv, T. W. Reps, and S. Horwitz. Precise interprocedural dataflow analysis with applications to constant propagation. *Theor. Comput. Sci.*, 167(1&2):131–170, 1996.

[34] O. Shacham, N. G. Bronson, A. Aiken, M. Sagiv, M. T. Vechev, and E. Yahav. Testing atomicity of composed concurrent operations. In *Proceedings of the 26th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2011, part of SPLASH 2011, Portland, OR, USA, October 22 - 27, 2011*, pages 51–64, 2011.

[35] O. Shacham, E. Yahav, G. Golan-Gueta, A. Aiken, N. G. Bronson, M. Sagiv, and M. T. Vechev. Verifying atomicity via data independence. In *International Symposium on Software Testing and Analysis, ISSTA '14, San Jose, CA, USA - July 21 - 26, 2014*, pages 26–36, 2014.

[36] Y. Velner, K. Alpernas, A. Panda, A. Rabinovich, M. Sagiv, S. Shenker, and S. Shoham. Some complexity results for stateful network verification. In *22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) 2016*, 2016. To appear.

[37] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauly, M. J. Franklin, S. Shenker, and I. Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, pages 15–28, 2012.

[38] O. Ziv, A. Aiken, G. Golan-Gueta, G. Ramalingam, and M. Sagiv. Composing concurrency control. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 240–249, 2015.

[39] O. Zomer, G. Golan-Gueta, G. Ramalingam, and M. Sagiv. Checking linearizability of encapsulated extended operations. In *Programming Languages and Systems - 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014, Proceedings*, pages 311–330, 2014.