Summary. Mohsen Lesani is an associate professor at University of California, Riverside. He obtained his PhD from UCLA and spent his postdoc at MIT. His research interests are reliability and security of software systems especially concurrent and distributed systems. He received the SIGPLAN Research Highlight [21] in 2019. He is awarded four NSF grants including the NSF CAREER award in 2020. His research results have been published in 38 papers in diverse venues that include Computer Security (S&P), Verification (CAV), Programming Languages (POPL, ICFP, OOPSLA including a distinguished paper award), Concurrent and Distributed Computing (DISC, PODC, PPoPP and ICBC), Software Engineering (FSE and a best paper at ISSRE) and the Communications of the ACM. In addition to external review committees, he has served in six conference program committees including POPL, OOPSLA and ECOOP, and NSF proposal review panel.

1 Research

1.1 Reliable and Secure Distributed Systems

Resilient Replicated Systems. Inter-organizational systems where subsystems with partial trust need to cooperate are common in healthcare, finance and military. In the face of malicious Byzantine attacks, this project assures end-to-end policies for the three aspects of trustworthiness: confidentiality, integrity and availability. In contrast to the state-of-the-art that often sidesteps the provision and validation of availability, this project guarantees end-to-end policies for availability in addition to confidentiality and integrity. It presents a security-typed object-based language, a partitioning transformation, an operational semantics, and an information flow type inference system for partitioned and replicated classes. The type system provably guarantees that well-typed methods enjoy noninterference for the three properties, and that their types quantify their resilience to Byzantine attacks. Given a class and the specification of its end-to-end policies, the Hamraz tool applies type inference to automatically place and replicate the fields and methods of the class on Byzantine quorum systems, and synthesize trustworthy-by-construction distributed systems. The experiments show the resiliency of the resulting systems; they can gracefully tolerate attacks that are as strong as the specified policies. The results of this project have been accepted for S&P’22 [16].

Automatic Analysis and Synthesis of Distributed Systems. Distributed system replication is widely used as a means of fault-tolerance and scalability. Given a sequential class with the declaration of its integrity and recency requirements, this project automatically synthesizes a correct-by-construction replicated class that simultaneously guarantees integrity, convergence and recency and avoids coordination as much as possible. Traditional strong consistency maintains the same total order of operations across replicas. This total order is the immediate source of multiple desirable consistency properties. However, maintaining the total order has proven to inhibit availability and performance. Weaker notions exhibit responsiveness and scalability; however, they forfeit the total order and hence its favorable properties. This project revives these properties with as little coordination as possible. The approach is based on novel sufficient conditions for integrity, convergence and recency that require certain orders between conflicting and dependent operations, and constrain the set of pending operations. To decide the validity of these conditions, the Hamsaz and Hampa tools apply automatic solvers to analyze both the given class statically and its method calls dynamically. They then reduce coordination avoidance to classical graph minimization problems, and use the results to instantiate parametric coordination protocols to obtain replicated systems. The results of this project have been published in POPL’19 [4] and CAV’20 [15]. Based on these results, we are building replication synthesis tools for Amazon Web Services.

Verification of Distributed Systems. This project aims at building mechanically verified (also known as certified) distributed systems. Distributed systems are critically used in geo-replicated data stores, replicated controllers for planes, cars, power plants, and digital currencies. However, they have proven to be complicated due to node and network failures and combinatorially large state-spaces. Thus, they repeatedly suffer data, currency and service loss. Further, the modern efficient data stores provide weaker notions of consistency such as eventual and causal consistency which make client code prone to bugs.

Our first framework, Chapar, supports modular verification of causal consistency for replicated key-value store implementations and their client programs in the Coq theorem prover. It includes a novel operational
semantics which serves as a specification for implementations and a guarantee for clients, verification of clients based on this specification, and a proof technique that were successfully applied to verify two store implementations. The implementations were extracted from Coq to OCaml to built executable stores. The results of this project have been published in POPL’16 [9].

In order to build certified middleware, our second framework presents a novel approach to compositional verification of distributed system stacks. Network middleware has been traditionally composed as a layered stack of components. This project modularly verifies each component based on only the implementation of the component itself and the specification of lower components. It presents TLC (Temporal Logic of Components), a novel temporal program logic that offers intuitive inference rules for verification of both safety and liveness properties of functional implementations of distributed components. TLC is proven sound with respect to a novel operational semantics for stacks of composed components in partially synchronous networks. TLC is mechanized in a Coq framework. This project has been published in ICFP’20 [3].

Cross-chain Exchange. This project presents secure protocols for cross-chain transactions that exchange assets between multiple parties across multiple blockchains. It can be represented as a directed graph, and may include a sequence of exchanges at each blockchain. Further, it may have off-chain steps and hence may not be strongly connected. If all parties conform to the protocol, all the assets should be transferred. If any party deviates from the protocol, the conforming parties should not experience any loss. Further, if the source parties pay, the sink parties should be paid. This project presents a protocol that guarantees the above properties for general cross-chain transactions with sequenced and off-chain steps when a few certain parties are conforming. It presents a tool called XChain that given a high-level description of a cross-chain transaction, can automatically generate smart contracts in Solidity. Further, we show that it is NP-hard for brokers to minimize transaction fees. The results of these project are published in ICBC’20 [20] and PODC’21 [2].

1.2 Domain-specific Languages

Declarative Graph Analytics. This project presents Grafs, a high-level declarative specification language for graph analytics and an optimizer and synthesizer that automatically generates efficient code for five high-performance graph processing frameworks. Graph analytics elicits insights from large graphs to inform critical decisions for business, safety and security. This project presents novel semantics-preserving fusion transformations that optimize the specifications and reduce them to three primitives: reduction over paths, mapping over vertices and reduction over vertices. Reductions over paths are commonly calculated based on push or pull models that iteratively apply kernel functions at the vertices. This paper presents conditions, parametric in terms of the kernel functions, for the correctness and termination of the iterative models, and uses these conditions as specifications to automatically synthesize the kernel functions. Experimental results show that Grafs significantly simplifies specification and accelerates execution. There results of this project are published in ICFP’21 [5].

Languages and Type Systems for Biochemistry. This project designs principled languages and type systems for specific domains. In particular, we took type-safety to interaction safety for biochemistry programs. This project introduced BioScript, a domain-specific language for programmable biochemistry. We formally defined the core syntax, and type checking and inference systems for BioScript. The type system ensures that certain types of errors, specific to biochemistry, do not occur, including the interaction of chemicals that may be unsafe. The paper was published at OOPSLA’18 [18] and received the distinguished paper award, and later, was recognized as the SIGPLAN Research Highlight in 2019. Every year, a committee representing SIGPLAN’s major conferences and elected officials select 2 to 4 papers that are of high quality and broad appeal as SIGPLAN Research Highlights from all the 18 sponsored conferences. Further, the project was invited to appear in the Communications of the ACM (CACM) [17].

1.3 Reliable Concurrency

Automatic Fence Insertion. In the interest of performance, processors feature relaxed memory models that reorder instructions. However, the correctness of concurrent programs is often dependent on the preservation of the order of certain instructions in each program. Thus, the instruction set architectures offer memory fences that prevent certain classes of reordering. Using fences is a subtle task with a trade-off between performance and correctness. This project presents a high-level programming model and a tool to capture
the orders that the user requires, and algorithms that insert the minimum number of fences to preserve the specified orders. In particular, it presents a greedy and polynomial-time optimum fence insertion algorithm for the class of programs with structured branch and loop statements, in addition to NP-hardness results for other classes. The results of this project appeared in my dissertation [6], OOPSLA'15 [1], PODC'17 [7], DISC'19 [22]. As a direct impact, these algorithms have been integrated into LLVM compiler infrastructure by independent developers [19].

Concurrency Testing and Verification. This project presents testing and verification techniques and tools for concurrent data structures and transactional memory (TM). The results of this project appeared in CAV'14 [11], DISC’13 [13], DISC’14 [14], NFM’19 [8], CONCUR’12 [10], and PPoPP’11 [12].

Mainstream programming languages offer libraries of linearizable concurrent data types. Linearizability supports horizontal composition: methods can be called on multiple objects and each method call appears to take effect atomically. However, they usually don’t support vertical composition: an operation that calls multiple methods on a data structure is not atomic by default. I presented [11] an automatic and modular verification technique for atomicity of vertical composition. I presented a novel sufficient condition called condensability and a tool called Snowflake that generates proof obligations for condensability of composing methods in Java and discharges them using constrain solvers. Snowflake could successfully verify a large fraction of an existing suite of compositions from several open-source applications.

In return for the simpler programming model, TM algorithms are complicated and error-prone. To test concurrent algorithms for common bugs, I developed a testing tool, Samand, that given a concurrent algorithm and the specification of a particular error at the interface of the algorithm, checks if there is a concurrent execution that exhibits that error, and produces an example trace. Thus, it makes regression testing possible for concurrent algorithms. It represents the space of concurrent executions as constraints and applies constraint solvers to search for possible executions. I used Samand to show the incorrectness of DSTM and McRT TM algorithms. They suffered from the write-skew and write-exposure anomalies respectively. These results were surprising as previous work had claimed verification of these algorithms.

Opacity is a correctness condition for TM algorithms. I defined a decomposition of opacity called markability as a conjunction of separate intuitive invariants, and proved that markability is required and sufficient for opacity. Separation has obvious benefits of modularity and scalability for verification. More importantly, the proofs of markability of TM algorithms can mirror the algorithm design intuitions about the location and order of read and commit points. To prove markability, I defined a program logic called synchronization object logic (SOL) that supports reasoning about the execution order and linearization orders of calls on the primitive synchronization objects. It provides inference rules that axiomatize the properties and independence of these orders, and the properties of common synchronization object types. SOL is proved sound based on a denotational semantics. I formalized SOL in the PVS proof assistant and used it to machine-check the markability of the TL2 TM algorithm.

During my internship at Oracle labs, I developed a framework [10] in PVS for verification of TM algorithms. I specified both the specifications and implementations as I/O automata and proved simulation relations between them. I used the framework to prove the correctness of the NOrec TM algorithm. The proofs of new algorithms can leverage the fundamental forward and backward simulation lemmas, and the hierarchy of simulations in the framework.

2 Grants

I received four NSF grants and I am the PI for three of them. The first one, “CRII: SHF: Certified Byzantine Fault-tolerant Systems” (PI, $174,999) is a single PI grant. It investigates verification of reliability and security of distributed systems where faulty or malicious nodes exhibit arbitrary or misleading behavior. These systems have applications ranging from finance to aircraft control. The second grant that Professor Song and I received is titled “SaTC: CORE: Small: Practical Whole Kernel Memory Safety Enforcement” (co-PI, $474,399, equal shares). This research project develops new techniques to eliminate critical memory access vulnerabilities from commodity operating system kernels. It includes both provably sound type systems and efficient implementation techniques. The third grant is “FET: Small: Stochastic Synthesis of Peptides and Small Molecules” (PI, $500,000, equal shares) with my co-PIs, Professors Brisk and Grover. This project accelerates the development of new drugs. It bring the level of automation inherent in computing disciplines to the life sciences. This project applies program synthesis principles to automate the discovery of new
drugs. Given a drug specification, the synthesis process automatically identifies a ligand that complies with the specification, can be safely synthesized and strongly binds to the desired proteins. The fourth grant is “CAREER: Distributed System Synthesis on Certified Middleware” (PI, $523,801). This proposal addresses programmer productivity and reliability of distributed systems that spans both the client applications and the supporting distributed middleware. It includes both novel automatic synthesis techniques for client applications and novel verification techniques for distributed middleware.

3 Service

I was a member of the PC (Program Committee) for OOPSLA’21 (11 papers), PC for POPL’20 (14 papers), PC for OOPSLA’20 (16 papers), PC for DisCoTec’20 (2 papers), PC for ECOOP’18 (8 papers), PC for CPP’17 (8 papers), and ERC (External Review Committee) for POPL’17 (6 papers). I was a panel member of an NSF SHF program in 2018 (8 proposals). Further, I was a committee member in Academic Panel of the Doctoral Symposium at OOPSLA’18. In the day long symposium, we advised students for progress in their dissertations. I also chaired a session at OOPSLA’18, two sessions at POPL’20 and a session at ICBC’20.

References


Notes

1SIGPLAN: ACM Special Interest Group on Programming Languages
2S&P is the top conference in computer security
POPL, PLDI, OOPSLA and ICFP are the top four conferences in programming languages.
CAV is one of the top two conferences in logic and verification.
FSE is one of the two top conferences in software engineering.
PODC and DISC are the top two conferences in distributed computing.
Conference abbreviations:
CACM: Communications of the ACM
CAV: International Conference on Computer-Aided Verification
CPP: ACM SIGPLAN and SIGLOG conference on Certified Programs and Proofs
ECOOP: European Conference on Object-Oriented Programming
ESOP: European Symposium on Programming
FSE: The ACM Symposium on the Foundations of Software Engineering
CONCUR: International Conference on Concurrency Theory
DISC: The International Symposium on Distributed Computing
DisCoTec: International Federated Conference on Distributed Computing Techniques, Program Committee
ICBC: IEEE International Conference on Blockchain and Cryptocurrency
ICFP: ACM SIGPLAN International Conference on Functional Programming
ISSRE: International Symposium on Software Reliability Engineering
OOPSLA: ACM SIGPLAN conference on Object-oriented Programming, Systems, Languages, and
PODC: ACM Symposium on Principles of Distributed Computing
PLDI: ACM-SIGPLAN Symposium on Programming Language Design and Implementation
POPL: ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages Applications
PPoPP: ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming
S&P: IEEE Symposium on Security and Privacy
SHF: Software Hardware Foundation