Xiaohong Chen
xc3@illinois.edu

My research interests are in **logic** and **formal methods** (FM), with a focus on improving formal correctness guarantees of semantics-based language implementations and analysis tools. I work on both foundational theories and practical applications. My Ph.D. research proposed **matching logic** (http://matching-logic.org) as a unifying logic foundation for programming languages [3,6], program specification [1,4,5,6], and program reasoning [1,6,7], which enables defining the formal semantics of many real-world languages and reasoning about various program properties. My current research focuses on **proof generation**, where machine-checkable matching logic proof objects are automatically generated as correctness certificates. I am cooperating with the startup Runtime Verification Inc. to transfer proof generation into practical products, to improve formal correctness guarantees of the semantics-based tools in the K framework (https://kframework.org).

My research has had both theoretical and practical impacts. On the theoretical side, my research on matching logic attracted attention from the global PL/FM community for its extreme simplicity, elegance, expressive power, and reasoning power. Research teams, groups, and labs from King's College London (UK), University of Kent (UK), Peking University (China), University of Bucharest (Romania), Alexandru Ioan Cuza University of Iaşi (Romania), Eötvös Loránd University (Hungary), and Masaryk University (Czechia) have started collaborations on the study of matching logic and the development of related tools. I have actively participated in all collaborations and am the leader of the core projects.

On the practical side, my research convinced Runtime Verification Inc. to adopt matching logic as the logic foundation of the K framework to improve its maintenance, reusability, correctness, trustworthiness, and human readability. My recent research on proof generation [2] raised a $30,000-funding from Ethereum Foundation for its potential to significantly increase the safety, reliability, and transparency of cryptocurrencies and smart contracts.

My **research vision** is twofold. On the theoretical side, I propose to keep studying the fundamental problems of matching logic regarding its expressive power, completeness, and decision procedures, with a focus on building connections with the other commonly used logics and formal systems. I will continue developing tool support for matching logic, focusing on an interactive matching logic theorem prover that directly supports reasoning with the many existing formal programming language semantics in the K framework. On the practical side, I propose to study proof-certifying smart contracts and blockchain applications based on my prior successes [2,7]. Complete, rigorous, and machine-checkable correctness certificates are automatically generated for both smart contracts execution and verification, bringing their trustworthiness to a high level, where all certificates are fully transparent and accessible to the stakeholders and can be independently checked by a 240-line proof checker [4].

Prior Work

Background and Significance. The *formal semantics* of a programming language is a rigorous mathematical description of behaviors of all programs of that language. A semantics-based (a.k.a. semantics-parametric or language-agnostic) language tool takes a formal semantics as input and generates an instance of the tool specific to the given language. The K framework (https://kframework.org) is an open-source toolkit with many high-performance semantics-based tools, such as a parser, an interpreter, a deductive program verifier, a program equivalence checker, and so on. Over the past 20+ years, K has been used to formalize real-world languages such as C,

Java, JavaScript, Python, Rust, x86-64, Ethereum VM, and LLVM to model and verify safety- and mission-critical systems by Boeing, DARPA, NASA, and Toyota, and more recently, to model and verify smart contracts, consensus protocols, blockchain-oriented programming languages, and virtual machines by ConsenSys, DappHub, Ethereum Foundation, IOHK, and Runtime Verification Inc.

The wide applications of K have raised concerns among its users and stakeholders, especially from the blockchain community, about the correctness of its semantics-based implementations and tools. Indeed, the development of K in the past was mainly driven by practical needs and user requests rather than a solid theoretical foundation. The existing 500,000-line unverified cross-language code base includes non-trivial algorithms, optimizations, and heuristics for pattern matching, term rewriting, unification, garbage collection, and domain-specific reasoning to ensure high performance. All make it highly challenging to provide formal correctness guarantees.

My Ph.D. research drew a systematic, practical, and logic-based approach to establishing the correctness and trustworthiness of semantics-based language tools. My work so far is twofold. The first is the proposal of *matching logic* (http://matching-logic.org) as a unifying logic foundation for programming languages, program specification, and program reasoning. The second is *proof generation*. Complete, rigorous, accessible, and machine-checkable proof objects are generated as correctness certificates for semantics-based language tools, bringing the best-known assurance levels to software and computers.

Proposal of Matching Logic. I proposed matching logic in [1] as a simple yet expressive logic foundation for programming languages. I showed that many commonly used logics [1,2], calculi [3], type systems [3,4], various formal semantics approaches [5], and especially, formal programming languages semantics [6], are definable in matching logic. I cooperated with Runtime Verification Inc. to merge matching logic and K. I co-designed a low-level intermediate language for K called Kore [9], based on matching logic. I wrote the whitepaper that specified the formal syntax of Kore and implemented the first Kore parser in Scala. Kore is now one of K's core components, and matching logic serves as the logic foundation.

Proof System and Proof Checker. I designed a 15-rule matching logic proof system [1] and implemented a 240-line matching logic proof checker [8], making it one of the smallest of its kind. Matching logic proofs are encoded into proof objects and checked by the proof checker, serving as a minimal trusted code base. More than 100 useful matching logic lemmas were manually proved [1] and later encoded into proof objects, forming a public database of matching logic theorems [2,3]. This database has been used by a research team from Eötvös Loránd University (Hungary) to encode matching logic proofs carried out using the Coq theorem prover into proof objects.

I also proved important completeness results about matching logic. For example, I proved that the first-order (FO) fragment of matching logic is complete [2], extending hybrid completeness [10]. I showed that matching logic is (relatively) complete in proving the functional correctness of programs using directly the formal semantics [1], which showed that matching logic is a (relatively) complete semantics-based verification framework. I proved that initial algebra semantics are definable in matching logic, and various induction principles are derivable using the matching logic proof system [5], which showed that matching logic is a unifying logic foundation for induction and recursion.

An Automated Prover. Like in other logics, the proof rules in the matching logic proof system are too low-level to use in practice. To make matching logic reasoning more efficient, I designed a set of high-level proof strategies and implemented them in a prototype automated theorem prover for matching logic. The prover is equipped with separate modules for reasoning about structure frames, contexts, fixpoints, and recursive properties. The prover was evaluated on benchmark sets across many different logics and was shown to be effective. For example, the prover proved all 280 benchmark heap properties in the SL-COMP competition for separation logic provers and outperformed some specialized separation logic provers [4].

Proposal of Proof Generation. In 2020, I initiated a long-term research agenda in my Ph.D. thesis proposal on proof generation for K. I aim to address the community's long-standing needs for a more trustworthy K. My methodology is to generate complete, rigorous, human-accessible, and machine-checkable matching logic proofs for all language tools in K. These proofs serve as correctness certificates of the correctness of K and eliminate the need for trusting the unverified code base of K. Each proof object clearly states the underlying theory (i.e., the formal semantics of a programming language), the property, and the entire proof steps, which the checker automatically checks.

As a proof of concept, I prototyped a proof generator for K's concrete execution tool [4]. The proof generator takes two inputs: a formal language semantics and an execution trace of a program conducted by K. It outputs a matching logic proof object that the execution is correct concerning the formal semantics. The idea of proof generation has caught the eyes of the industry, especially the blockchain community, for it can provide the best-known assurance levels for smart contracts. In 2022, the proposal "Trustworthy Formal Verification for Ethereum Smart Contracts via Machine-Checkable Proof Certificates," which I helped in writing, raised a $30,000-funding from Ethereum Foundation.

My **other work** included (1) using K and matching logic to specify and reason about computation that takes place beyond traditional scenarios and (2) developing efficient semantics-based tools. I co-designed MediK as an executable formal semantics of an interactive medical guidance system in K [11], which interacts with a web interface and forms an end-to-end guidance system for medical experts and physicians. I used matching logic to specify hybrid systems and properties [12]. I co-designed a unifying logic framework for neural networks based on matching logic, where many different types of neural networks and their properties can be specified in a uniform way [13]. I also participate in developing KPLC, a high-performance executable formal semantics of programmable logic controllers. I helped Runtime Verification Inc. audit K's code base and identified performance bottlenecks. I co-authored [14], which introduces the internal algorithm that K uses to compile formal semantics into LLVM code for efficient concrete execution.

Future Directions

My previous research has shown that matching logic is an expressive logic foundation for programming languages powered by a simple proof system and a small proof checker. My ongoing research has demonstrated that proof generation is a promising and practical technique to improve formal correctness guarantees of semantics-based language tools.

My main objective in the next five years is to develop the proof generation technique further and support more programming languages and language tools. A promising practical application of

potentially high impact is proof-carrying smart contracts, whose execution and verification are certified by machine-checkable proof objects that stakeholders can independently check. On the theoretical side, I propose to keep studying the fundamental problems of matching logic and developing its tool support, with a focus on building connections with other commonly used logics, formal systems, and proof assistants. In the long term, I am also interested in using matching logic and proof generation to specify, reason about, and certify computation that takes place beyond traditional scenarios such as neural networks, systems with human interactions, and cyber-physical systems.

Proof Generation in Practice. My Ph.D. work [4] is a proof of concept showing proof generation's effectiveness in certifying complex semantics-based language tools. Only concrete (i.e., non-symbolic) execution of programs has been considered. It takes more effort to develop efficient proof generation algorithms that support more complex programming languages and formal analysis tools, such as deductive program verifiers and model checkers. My long-term goal is thus to support proof generation in practice. I will focus on proof generation for symbolic execution and formal verification in the short term. Given a successful verification run, a matching logic proof certificate is generated based on the formal semantics of the given language, thus making formal verification results transparent and accessible to humans. For example, in my latest work [5], I present a prototype proof generator that outputs proof certificates for successful verification runs of deterministic programs. At its core are two proof generation algorithms: one for symbolic execution and unification and the other for co-inductive reasoning. Using a similar idea, I plan to develop a proof generator for verifying non-deterministic programs.

Another major research problem is the use of contexts in the existing formal semantics of K. Contexts are a key concept in K that makes it scalable to large, real-world languages. They allow users to specify computation locally wherever it takes place. However, contexts are currently regarded as syntactic sugar and are eliminated during the parsing phase using an unverified, unformalized translation. As a result, frame reasoning becomes more complex and introduces an untrusted code base. I propose to use matching logic to formalize the existing formal semantics in K and generate proof certificates for context-sensitive reasoning. For example, in my latest work [JFP'23], I prove that formal reasoning about contexts is a particular case of matching logic reasoning and thus can be certified using matching logic proof objects.

In terms of practical application, I will focus on transferring proof generation to smart contracts and producing **trustworthy, proof-carrying smart contracts**. I will build upon the existing effort in using K to model and verify smart contracts and their correctness properties and generate matching logic proof certificates. Each certificate will consist of the entire formal semantics of the programming language involved (such as Ethereum virtual machine) as logical axioms and a machine-checkable proof object that derives the intended property of the smart contract. This way, we improve the trustworthiness of smart contracts by eliminating K's complex, unverified verification algorithms from the trust base.

At the same time, I seek to optimize proof certificate sizes and proof checking time. In [4], it is reported that the proof certificate of a 100-step execution trace of a simple imperative program has 1.6 million lines, making it unsuitable to be passed around as a correctness certificate. In the latest New England Verification Day talk, I explored the idea of combing succinct non-interactive arguments of knowledge (SNARKs) and proof generation to obtain succinct cryptographic proofs as

correctness certificates. To test this idea, I have begun cooperating with the startup RISC-zero Inc. in developing a succinct proof generator for matching logic.

Matching Logic: Tool Support. Various formal systems and proof assistants exist in the area of programming languages and formal methods, such as Coq, Isabelle, Lean, and PVS. These proof assistants are widely used in defining formal language semantics and reasoning about programs, with decent support for both interactive and automated reasoning. On the other hand, many real-world programming languages already have their formal semantics defined in K and translated to matching logic theories. There is an excellent need for interactive reasoning power for these languages, especially when the fully automated K tools fail to deliver. Therefore, I plan to export these formal semantics in matching logic to proof assistants to get the best of both worlds. On the one hand, K will obtain new interactive reasoning power from these proof assistants. On the other hand, users who are familiar with reasoning using these proof assistants will get access to the formal semantics of numerous real-world languages. In [5], I co-worked with a research team from Eötvös Loránd University (Hungary) and mechanized matching logic in Coq. I am also involved in another parallel effort led by the Institute for Logic and Data Science at the University of Bucharest in mechanizing matching logic in Lean.

References

TBA