Research Statement

RYAN KAVANAGH, Carnegie Mellon University, United States of America

EXECUTIVE SUMMARY

My research vision is to develop programming languages and tools that help practitioners correctly specify, implement, and reason about concurrent and communicating systems. These systems can be found everywhere, from the Internet of things to the cloud to the chips in our phones and computers. For a communicating system to function, each of its components must obey agreed-upon communication protocols; failure to do so can lead to expensive bugs and vulnerabilities. To guarantee that components obey their protocols, we can implement them using *session-typed programming languages*. However, existing session-typed programming languages struggle to capture rich real-world protocols. Over the next five years, I will pursue foundational work to add support for real-world protocols to session-typed programming languages, and I will develop compilers and tools to put this theory into practice. My long-term career goal is to develop a research pipeline supporting my vision that spans from rigorous foundational work to tools and techniques with industrial impact.

1 CONCURRENT AND COMMUNICATING SYSTEMS ARE HARD TO GET RIGHT

The Heartbleed vulnerability allowed attackers to access private data on an estimated 24–55% of websites [Dur+14], and early estimates put its cost to industry at $500 million [Ker14]. At its core, the vulnerability was due to an incorrect implementation of the heartbeat TLS protocol extension. Protocols are agreed-upon rules for communication, and they are critical for system interoperability. Despite this, they are often specified using potentially ambiguous prose documents, making it difficult to ensure that programs correctly implement or obey them. Even if communication protocols were formally specified, the multi-million dollar question remains: how do we guarantee that communicating systems obey their protocols?

Session-typed programming languages are a promising approach. *Session types* formally specify communication protocols and stipulate what messages a system can send or receive, analogously to how data types specify what sorts of data programs can produce or consume. A program written in a session-typed programming language is *guaranteed* to obey its communication protocols. Unfortunately, traditional session types are insufficiently rich to specify the heartbeat extension and many other real-world protocols. Motivated by the above, I am eager to investigate the following claim:

> *Session-typed programming languages can be extended to provide a sound and tractable foundation for specifying, implementing, and reasoning about real-world concurrent and communicating systems.*

My past successes investigating session-typed programming languages and concurrent and communicating systems, described in section 2, uniquely position me to study this problem. In section 3, I describe how I will build on these successes to investigate my claim. I expand on the intellectual merit and broader impact of my research plan in section 4, and I consider funding sources in section 5.

2 PAST SUCCESSES

I have spent the past six years making contributions to the foundations of concurrent computation. This work concerns two areas: *session-typed programming languages* and *weak memory models*.

2.1 Session-Typed Programming Languages

In my dissertation, I study techniques for reasoning about programming languages that combine functional programming with a rich form of message-passing concurrency. Though we know how to reason separately about functional programs and about message passing concurrency, the ways in which these features interact are complex, especially in the presence of non-termination and of code transmission. However, if we are to reason about real-world programs and languages, we must be able to reason about these interactions. My dissertation gives a novel and rigorous framework for doing so.

Concretely, I give the first **denotational semantics** [Kav20a] for a language with the above features and I apply it to program verification. Importantly, denotational semantics let us reason modularly about programs, instead of needing to reason about entire programs at once. This semantics describes program behaviour using mathematical techniques from order theory and category theory. These branches of mathematics are well understood, and I use their results to give insights into message passing computation and to relate it to other parts of mathematics. For example, I observe that program composition behaves unlike function composition, and that it is instead captured by the same category-theoretic operation that defines traces of linear maps. This observation unlocks a wealth of mathematical results that I use to reason about programs. Defining my semantics required that I develop new techniques for reasoning about **parametrized fixed points of functors** [Kav20c]. Functors are special kinds of functions in category theory, and their fixed points arise in many parts of programming languages research. In particular, I showed that their fixed points enjoy a certain structure that implies a large class of identities relevant to programming languages research.

I also give the first **observed communication semantics** [Kav20b] for a language with non-termination. Observed communication semantics provide an intuitively reasonable notion of "observation" for languages with message passing concurrency, and they act as the ground truth when we talk about programs behaving "the same way". To ensure that my semantics is well defined, I develop the first notions of **fairness** [Kav20b] for programming languages defined using substructural operational semantics. Substructural operational semantics are an approach for defining programming languages, where each program step is determined by a rewrite rule. Fairness ensures that schedulers do not neglect portions of parallel or concurrent programs.

Finally, I introduce **bisimulations** for this language. Bisimulations are a kind proof technique for tractably reasoning about programs, and I use them to relate my two semantics.

2.2 Weak Memory Models

Prior to my dissertation work, I investigated the semantics of weak memory models. A memory model specifies what values may be read by a thread from a given memory location during execution. Traditional concurrency research assumes *sequential consistency* as its memory model. Sequential consistency treats operations on global memory atomically, such that one can assume that all operations on memory occur sequentially. Though this simplifies reasoning about concurrent programs, it comes at a performance cost. Consequently, modern architectures and programming languages have long provided weaker guarantees, resulting in *weak memory models*. Unfortunately, classical concurrency algorithms can fail under these weaker guarantees. The situation is made worse by the fact that weak memory models are typically specified in informal prose documents. This informality makes reasoning about the behaviour of algorithms under weak memory models even harder.

To address these problems, I developed a denotational semantics for the SPARC TSO weak memory model [KB19]. This semantics rigorously specifies the behaviour of programs under SPARC TSO. Prior work typically specified weak memory models using *axiomatic semantics*, which require users to reason about whole programs at once. In contrast, my denotational semantics lets practitioners reason modularly about their code.

3 MY RESEARCH ROAD MAP

An important class of real-world protocols that traditional session types cannot specify are *dependent protoctols*. Dependent protocols, e.g., the heartbeat extension of section 1, are protocols where the range of permitted communications *depends* on previous communications. For example, the heartbeat protocol lets a system ensure that its peer is reachable by exchanging "heartbeat" messages. Roughly speaking, it does so by sending its peer the message "Here are n bits of data", followed by said data. Its peer replies "Here are those n bits", followed by the data it received. The only data permitted in this response is the data sent in the original heartbeat message, i.e., the communications allowed in the response *depend* on the values transmitted.

Support for dependent protocols is a natural language extension to pursue as I investigate my claim, and this extension will form the core of my research over the next five years. Various *dependently session-typed languages* [TCP11; TV19; TY18] extend session types to support various kinds of dependent protocols. However, it is unknown how to support these different kinds of dependent protocols in a single language. Also unknown is the extent to which these dependent features can co-exist with other desirable language features, such as *recursively defined session types*.

Denotational semantics are a promising approach to investigating these questions. Not only do denotational semantics let us reason modularly about programs, but they also provide a notion of *program equivalence*. Program equivalence "is arguably one of the most interesting and at the same time important problems in formal verification" [Lah+18], and it is the crux of any dependently typed language. My experience applying denotational semantics to session-typed languages ideally positions me to pursue this approach. Concretely, I intend to pursue four objectives over the next five years.

Dependently session-typed languages have evolved along two separate axes, and my first objective is to extend my denotational semantics along each of these. In the first axis are *value-dependent session types* [TCP11; TY18]. These languages allow programmers to specify protocols that depend on values from an underlying functional language. *Label-dependent session types* [TV19] form the second axis, and they are interesting because they support number-indexed protocols. Number-indexed protocols, e.g., the heartbeat extension, are commonly used in practice, where a transmitted value specifies the number or size of subsequent messages.

My second objective is to unify both styles of dependency into a single language.

My third objective is to explore the extent to which recursive types can exist in dependently session-typed languages. Though dependently session-typed languages with recursive types exist, they only allow for restricted forms of recursion that do not capture many real-world protocols. I expect that my past work using parametrized fixed points to model recursive types will be indispensable.

My final objective is to implement the resulting dependently session-typed language with recursion. This experience will help inform future theoretical work on dependently session-typed languages. Dually, my denotational semantics will likely aid the implementation effort. For example, I may need to use the semantics to justify various program transformations used by the compiler.

These objectives feature many subprojects of varying levels of accessibility that would be appropriate for undergraduate or graduate students. For example, an undergraduate could help explore the trade-offs involved with each kind of dependency through experimentation, or help with compiler implementation. One or more of these objectives could also form the core of a student's PhD thesis.

I am eager to collaborate with members of the programming languages group and with others. For example, I am interested in exploring applications of session-typed languages to concurrent data structures. I am also interested in studying the complexity of concurrent algorithms. A novel approach that I would like to explore involves *ergometric* and *temporal* session types [DHP18a; DHP18b], which describe the complexity of programs in the language itself.

In the long-term, I will pursue foundational work on concurrency motivated by practical concerns, and I will use these results to develop tools for reasoning about and implementing software systems.

4 INTELLECTUAL MERIT AND BROADER IMPACT

My research plan provides the first insights into dependently session-typed languages with general recursive types. Considerable effort went into combining dependent types with recursive types for functional languages, and this work led to many insights on the foundations of programming languages and of formal systems. My research program should provide similar foundational insights.

Denotational semantics for dependent type theories are well understood [Jac99]. However, denotational semantics for session-typed languages are rare, and none exist for dependently session-typed languages. My research plan bridges this gap to provide a deeper understanding of both session-typed and dependently typed languages. In doing so, it provides techniques for reasoning modularly about programs written in dependently session-typed languages.

Ultimately, my program provides richer languages that ensure that communicating systems are correct, thereby minimizing costly implementation errors. This is increasingly important, given the massive and ever-increasing adoption of cloud computing and of software as a service.

5 FUNDING SOURCES

I will actively pursue external funding sources to support these research projects. In particular, I will pursue an NSERC Discovery Grant as an "Early Career Researcher" and various Mitacs grants. I have a track record of successful NSERC fellowship and grant applications: as an undergraduate, I received one of two Canada-wide NSERC/CMS "Math in Moscow" scholarships. I also received a three-year NSERC PGS D fellowship to support my PhD research on denotational semantics for weak memory.

REFERENCES

[DHP18a] Ankush Das, Jan Hoffmann, and Frank Pfenning. "Work Analysis With Resource-Aware Session Types". In: *LICS'18*. 2018. DOI: 10.1145/3209108.3209146.

[DHP18b] Ankush Das, Jan Hoffmann, and Frank Pfenning. "Parallel Complexity Analysis With Temporal Session Types". In: *PACMPL* 2.ICFP (2018). DOI: 10.1145/3236786.

[Dur+14] Zakir Durumeric et al. "The Matter of Heartbleed". In: *IMC'14*. 2014. DOI: 10.1145/2663716.2663755.

[Jac99] Bart Jacobs. *Categorical Logic and Type Theory*. Amsterdam: Elsevier, 1999.

[Kav20a] Ryan Kavanagh. *A Domain Semantics for Higher-Order Recursive Processes*. 2020. arXiv: 2002.01960v3 [cs.PL].

[Kav20b] Ryan Kavanagh. "Substructural Observed Communication Semantics". In: *EXPRESS/SOS 2020*. Ed. by Ornela Dardha and Jurriaan Rot. 2020, pp. 69–87. DOI: 10.4204/EPTCS.322.7.

[Kav20c] Ryan Kavanagh. "Parametrized Fixed Points and Their Applications To Session Types". In: *Electronic Notes in Theoretical Computer Science* 352 (2020), pp. 149–172. DOI: 10.1016/j.entcs.2020.09.008.

[KB19] Ryan Kavanagh and Stephen Brookes. "A Denotational Semantics for SPARC TSO". In: *Logical Methods in Computer Science* 15.2, 10 (2019). DOI: 10.23638/LMCS-15(2:10)2019.

[Ker14] Sean Michael Kerner. *Heartbleed SSL Flaw's True Cost Will Take Time to Tally*. Apr. 19, 2014. URL: https://www.eweek.com/security/heartbleed-ssl-flaw-s-true-cost-will-take-time-to-tally.

[Lah+18] Shuvendu K. Lahiri et al. "Program Equivalence". In: *Dagstuhl Reports* 8.4 (2018). DOI: 10.4230/DagRep.8.4.1.

[TCP11] Bernardo Toninho, Luís Caires, and Frank Pfenning. "Dependent Session Types Via Intuitionistic Linear Type Theory". In: *PPDP'11*. 2011. DOI: 10.1145/2003476.2003499.

[TV19] Peter Thiemann and Vasco T. Vasconcelos. "Label-Dependent Session Types". In: *PACMPL* 4.POPL (2019). DOI: 10.1145/3371135.

[TY18] Bernardo Toninho and Nobuko Yoshida. "Depending on Session-Typed Processes". In: *Foundations of Software Science and Computation Structures*. LNCS 10803. 2018. DOI: 10.1007/978-3-319-89366-2_7.