Frank (Peng) Fu

Email: frank-fu@dal.ca
Website: `https://fermat.github.io`
Address: Chase building 251, 6316 Coburg Rd,
Dalhousie University, Halifax, Canada.
Telephone: +1 902-494-6760

Education

- Ph.D. Department of Computer Science, University of Iowa, Iowa City, Iowa, USA. September 2009 – August 2014. Advisor: Aaron Stump.
- B.Eng. School of Computer Science, Huazhong University of Science and Technology, Wuhan, Hubei, China. September 2005 – June 2009.

Research interests

- Type systems for quantum circuit programming.
- Type systems for programming languages.
- Interactive theorem proving and language-based formal verification.
- Lambda calculus, type theory and their applications.

Professional experience

- Postdoctoral Researcher, Dalhousie University, Halifax, Nova Scotia, Canada. May 2017 – present.
  - Supervisor: Peter Selinger.
  - Developing a linear dependently typed functional programming language for correct quantum circuit construction.
  - Developed a technique to allow programming and reasoning about nested data types in the dependently typed language Agda.
  - Prototype implementations: S1, S2.
  - Manuscripts: M1, M2.
- Postdoctoral Research Assistant, University of Dundee and Heriot-Watt University, Scotland, UK. October 2014 – August 2016.
  - Applied techniques from logic programming and theorem proving to solve a type class problem.
  - Prototype implementation: S5.
  - Publications: C1, C2, J1.

Refereed journal publications

J1 **Operational Semantics of Resolution and Productivity in Horn Clause Logic**.
Peng Fu, Ekaterina Komendantskaya. Formal Aspect of Computing, 2017. Journal Version of C2.

J2 **Efficiency of Lambda-Encodings in Total Type Theory**.
Aaron Stump, **Peng Fu**. Journal of Functional Programming, 2016.

Refereed conference publications

C1 **Proof Relevant Corecursive Resolution**.

Peng Fu, Ekaterina Komendantskaya, Tom Schrijvers, Andrew Pond. International Symposium on Functional and Logic Programming, FLOPS 2016.

C2 **A Type-Theoretic Approach to Resolution**.

Peng Fu, Ekaterina Komendantskaya. International Symposium on Logic-Based Program Synthesis and Transformation, LOPSTR 2015.

C3 **Self Types for Dependently Typed Lambda Encodings**.

Peng Fu, Aaron Stump. Joint 25th International Conference on Rewriting Techniques and Applications and 12th International Conference on Typed Lambda Calculi and Applications, RTA-TLCA 2014.

Refereed workshop publications

W1 **Equational Reasoning about Programs with General Recursion and Call-by-value Semantics**.

Garrin Kimmell, Aaron Stump, Harley Eades III, **Peng Fu**, Tim Sheard, Stephanie Weirich, Chris Casinghino, Vilhelm Sjöberg, Nathan Collins, Ki Yung Ahn. Programming Languages meets Program Verification, PLPV 2012.

W2 **Irrelevance, Heterogeneous Equality, and Call-by-value Dependent Type Systems**.

Vilhelm Sjöberg,Chris Casinghino, Ki Yung Ahn, Nathan Collins, Harley Eades III, **Peng Fu**, Garrin Kimmell, Tim Sheard, Aaron Stump, Stephanie Weirich. Mathematically Structured Functional Programming, MSFP 2012.

W3 **A Framework for Internalizing Relations into Type Theory**.

Peng Fu, Aaron Stump, Jeff Vaughan. International Workshop on Proof-Search in Axiomatic Theories and Type Theories, PSATTT 2011.

Manuscripts

M1 **An Introduction to Quantum Circuit Programming in Dependently Typed Proto-Quipper**, 2019, available on request.

M2 **Dependently Typed Folds for Nested Data Types**, 2018, available at arXiv: `https://arxiv.org/abs/1806.05230`.

M3 **A Type Checking Algorithm for Higher-rank, Impredicative and Second-order Types**, 2017, available at arXiv: `http://arxiv.org/abs/1711.04718`.

M4 **Representing Nonterminating Rewriting with $\mathbf{F}_2^\mu$**, 2017, available at arXiv: `http://arxiv.org/abs/1706.00746`.

Dissertation works

- Title: **Lambda Encodings in Type Theory**.
- Summary: The dissertation explores two approaches to reason about pure functional programs. One is based on a type system extended with a new type construct called self-type. The other is based on minimal higher-order logic using the comprehension principle. Both approaches rely on encoding functional programs as lambda-terms. The higher-order logic approach is implemented to show how to reason about possibly diverging programs in a consistent proof system.
- Committee: Aaron Stump, Cesare Tinelli, Kasturi Varadarajan, Ted Herman, Douglas Jones.
- Prototype implementation: S6.
- Publications: C3, J2, W3.

Prototype implementations

S1 **Dependently Typed Proto-Quipper** (Source code available on request). A prototype programming language for quantum circuits generation. Main features:

- Linear and dependent types for programming quantum circuits.
- A bidirectional type checker that supports linear types, polymorphic types and dependent types.

S2 **Dependently Typed Folds for Nested Data Types** (Source code: `https://github.com/fermat/dependent-fold`). Currently it is hard to program and reason about nested data types in dependently typed languages. We give a collection of examples in the dependently typed language Agda to show how to program and reason about nested data types using dependently typed folds.

S3 **Higher-Rank** (Source code:`https://github.com/Fermat/higher-rank`). A prototype type checker that supports higher-rank, impredicative and second-order types. The type checker can type check a nontrivial class of functional programs that are currently not supported by the mainstream type checkers.

S4 **Functional Certification of Rewriting (FCR)** (Source code: `https://github.com/fermat/fcr`). A prototype type checker for analyzing and proving the nontermination of term rewriting system. Main features:

- The certification of nonterminating rewriting is reduced to type checking.
- The type checking algorithm is based on resolution with second-order matching.
- The type system is a sub-fragment of $\mathbf{F}_\omega$ with additional typing rule for recursion.

S5 **Corecursive Type Class** (Source code: `https://github.com/fermat/corecursive-type-class`). A prototype interpreter and type checker that implements a type class mechanism based on corecursive resolution. Main features:

- It supports dictionary construction for nonterminating type class resolution.
- It uses a goal directed automated proof construction to construct type class evidence.
- It provides a heuristic for generating intermediate lemma during the proof construction.

S6 **The Gottlob System** (Source code: `https://github.com/fermat/gottlob`). A prototype interpreter for typed functional programming and theorem proving. Main features:

- The functional programming fragment is equipped with Hindley-Milner type inference. The core language is based entirely on Scott encodings, without build-in data types and pattern matching.
- The theorem proving fragment can reason about general functional programs that can possibly diverge.
- It can automatically synthesize an induction principle from a regular algebraic data type declaration, the induction principle is not primitive in Gottlob.

Conference and workshop presentations

- **Dependent types in Proto-Quipper**, September 20, 2018, Dagstuhl Seminar: Quantum Programming Languages, Dagstuhl, Germany.
- **Proof Relevant Corecursive Resolution**. June 22, 2016, The Scottish Programming Languages Seminar, Heriot-Watt University, Edinburgh, UK.
- **A Type-Theoretic Approach to Structural Resolution**. July 13, 2015, LOPSTR, Siena, Italy.
- **Self Types for Dependently Typed Lambda Encodings**. July 15, 2014, RTA-TLCA, Vienna, Austria.
- **Dependent Lambda Encoding with Self Types**. September 2013, ACM SIGPLAN Workshop on Dependently-Typed Programming(DTP), Boston, MA.
- **A Framework for Internalizing Relations into Type Theory**. August 2011, PSATTT workshop, Wroclaw. Poland.

Teaching experience

- Lecturer, "Discrete structures I", 2019 Summer, Dalhousie University.
  - Taught basic set theory, relation, formal logic and proofs, basic counting and basic number theory.
  - Delivered three lectures per week (total 35 lectures, class size: around 30).
  - Held office hour twice a week.
  - Developed class materials, homeworks and exams.
- Teaching Assistant, "Introduction to functional programming in Haskell", which is part of the "Algorithm and AI" module, 2015 Spring. Computer Science, The University of Dundee.
  - Taught basic functional programming in Haskell, the first Haskell class taught in University of Dundee.
  - Delivered one lecture per week (total 13 lectures, class size: around 50).
  - Ran one lab session per week.
  - Developed class materials, homeworks and part of the final exam.
- Graduate Teaching Assistant, "Programming Language Concepts", 2013 Spring, 2014 Spring. Department of Computer Science, The University of Iowa.
  - Graded assignments (Class size: around 70 both times).
  - Ran weekly office hours.

- Graduate Teaching Assistant, "Object-Oriented Software Development ", 2013 Fall. Department of Computer Science, The University of Iowa.
  - Graded assignments (Class size: around 70).
  - Ran one lab session per week.
  - Ran weekly office hours.
- Graduate Teaching Assistant, "Computer Networking", 2009 Fall. Department of Computer Science, The University of Iowa.
  - Graded assignments (Class size: around 30).
  - Ran weekly office hours.

Professional service

- External Reviewer. 24th International Conference on Rewriting Techniques and Applications (RTA 2013). Reviewed: 1 paper.
- External Reviewer. 19th International Conference on Foundations of Software Science and Computation Structures (FoSSaCS 2016). Reviewed: 1 paper.
- External Reviewer. 32nd International Conference on Logic Programming (ICLP 2016). Reviewed: 1 paper.