GARRIGUE, Jacques Professor

Office: Rm 405 in Math. Bldg.

Telephone: +81 (0)52–789–4661 (ext. 4661)

E-mail: garrigue@math.nagoya-u.ac.jp

Website: http://www.math.nagoya-u.ac.jp/~garrigue/

Membership of academic societies:
ACM, JSSST (Japanese Society for Software Science and Technology), IPSJ (Information Processing Society of Japan)

Research Interest:

- Theory of Programming Languages
- Typed lambda-calculi and type inference
- Applications of proof assistants

Research Summary:

My reasearch aims at eradicating bugs from programs while preserving expressiveness. This is becoming a fundamental need, as programs are everywhere. There are several approaches to reach this goal, but I have been working mostly on type systems, and proof of programs.

Participating in the development of the functional programming language OCaml, I have worked at enriching expressiveness without losing safety. Functional programming languages are based on the typed lambda-calculus, and have a strong connection to logic. From logic comes the concept of *soundness*. A type system is *sound* if typed programs are guaranteed not to cause type-related errors at runtime.

One important property of advanced type systems is to allow the reuse of code in differently typed contexts. This is called *polymorphism*. I made several contributions in that domain [1, 2, 5].

However, it is often hard to prove that the algorithm checking that a program has a type is correct. For this reason, I have also proved in Coq the soundness and completeness of such an inference algorithm [3].

In another attempt, I have been working with Reynald Affeldt and Akira Tanaka from AIST on applying Coq to proofs of systems and mathematical theories. Together we have proved some linear codes from coding theory [4], and worked on code generation from certified algorithms [6]

Outside of the above work on programming languages and proof assistants, I am also interested in logic, and computability theory.

Major Publications:

[1] J. Garrigue and D. Rémy, Extending ML with semi-explicit first class polymorphism. Information and Computation **155** (1999), 134–171.

[2] J. Garrigue and D. Rémy, Ambivalent Types for Principal Type Inference with GADTs. In *11th Asian Symposium on Programming Languages and Systems*, Springer LNCS (2013).

[3] J. Garrigue, A Certified Implementation of ML with Structural Polymorphism. Mathematical Structures in Computer Science (2014) 11:1–25.

[4] J. Garrigue and R. Affeldt, Formalization of Error-correcting Codes: from Hamming to Modern Coding Theory. In *6th Conference on Interactive Theorem Proving*, Springer LNCS (2015).

[5] J. Garrigue and J. Le Normand. GADTs and exhaustiveness: Looking for the impossible. In *Proc. ML Family / OCaml Workshops 2015*, number 241 in EPTCS (2017), 23–35.

[6] A. Tanaka, R. Affeldt, J. Garrigue. Safe low-level code generation in coq using monomorphization and monadification. Journal of Information Processing (2018) 26:54-72.

Awards and Prizes:

- JSSST Takahashi encouragement prize (2010)

Education and Appointments:

1995 Doctor of Science of the University of Tokyo
1995 Research Associate, Kyoto University
2004 Associate Professor, Nagoya University
2018 Professor, Nagoya University

Message to Prospective Students:

My research area is directly connected to functional programming languages, but the subject of my small class is often more logic oriented. The main subjects are computability, lambda-calculus and type theory, and logic.

Computability (or theory of computation) is the study of which functions can be computed by a concrete algorithm. Among mathematical functions, some can be computed by a program, and some cannot. Morevover, this difference is not related to the programming language used. From a logical point of view, this means that no program can discover all the possible theorems. We have used the following textbook.

1. Neil D. Jones, *Computability and Complexity from a Programming Perspective.* MIT Press, 1995.

Lambda-calculus is a framework connecting logic and computer science. Its type theory can be applied to the construction of both programming languages and theorem provers. The π-calculus gives an insight into concurrent programming. In small classes, we have used the following textbooks.

1. Benjamin C. Pierce, *Types and Programming Languages.* MIT Press, 2002.
2. Yves Bertot, Pierre Castéran, *Interactive Theorem Proving and Program Development.* Springer, 2004.
3. Franz Baader, Tobias Nipkow, *Term Rewriting and All That.* Cambridge Univ. Press, 1998.

Logic provides a foundation to both areas, and also some techniques to prove the valvidity of logical statements (using the resolution procedure for instance). We have used the following textbooks.

1. Jean Gallier, *Logic for computer science.* Online edition, 1986.
2. John Harrison, *Handbook of practical logic and automated reasoning.* Cambridge University Press, 2009.
3. Jean-Yves Girard, *The Blind Spot: Lectures on Logic.* European Mathematical Society, 2011.

No special knowledge is required to start working on thes subjects, but logic is a welcome basis. For the master thesis, I can help in various areas of theoretical computer science.