GARRIGUE, Jacques                                                    Associate 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 (Association for Computing Machinery),

JSSST (Japanese Society for Software Science and Technology)

## 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 serveral approaches to reach this goal, but I've 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. Models of lambda-calculus can also be used to study the behaviour of programs, and simplify proofs.

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, 3, 4]. Recentely I have been working on the implementation of GADTs (generalized algebraic datatypes) [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 [6]. This is a small part of the language, but this demonstrated the feasability of a complete, and I hope to enlarge it in the future.

Outside of the above work on programming languages, 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, Code reuse through polymorphic variants. In *7th Workshop on Foundations of Software Engineering* (2000), 93–100.

[3] J. Garrigue, Relaxing the value restriction. In *7th International Symposium on Functional and Logic Programming*, Springer LNCS 2998 (2004), 7–23.

[4] J. Garrigue and K. Nakata, Path resolution for recursive nested modules. Higher-Order and Symbolic Computation **24** (2012) 207–237.

[5] 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).

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

## Awards and Prizes:

- JSSST Takahashi encouragement prize (2010)

## Education and Appointments:

| | |
|---|---|
| 1995 | Doctor of Science of the University of Tokyo |
| 1995 | Reasearch Associate, Kyoto University |
| 2004 | Associate 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 $\pi$-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. Robin Milner, *Communicating and mobile systems: the $\pi$-calculus*, Cambridge University Press, 1999.

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.

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.