Prof. Sam Tobin-Hochstadt — SICSA Distinguished Visitor
22 January – 05 February 2017

Sam Tobin-Hochstadt, Assistant Professor at Indiana University Bloomington, visited Scotland between 22nd January and 5th February 2017. He was hosted by Prof. Phil Trinder and Dr. Patrick Maier at the University of Glasgow. The main purpose of the visit was to explore the potential for collaboration between Sam Tobin-Hochstadt's research group at Indiana and the programming language research community in Scotland.

During his stay, Prof. Tobin-Hochstadt visited three different SICSA universities for a day each, engaging in discussions about dynamic typing, compilers, and teaching programming with academic and research staff. He also gave three seminar talks. Details about these activities are listed below; abstracts of the talks can be found at the end of this report.

- **Typed Racket and Gradual Typing.** Seminar at the University of Edinburgh, 24th January, hosted by Dr. Sam Lindley.
- **Languages as Libraries.** Seminar at the University of St Andrews, 27th January, hosted by Dr. Edwin Brady and Prof. Kevin Hammond.
- **Pycket: A Tracing JIT for a functional language.** Seminar at the University of Glasgow, 1st February, hosted by Dr. Patrick Maier and Prof. Phil Trinder.

Besides these visits and seminars, Prof. Tobin-Hochstadt spent several days at Glasgow collaborating with Prof. Trinder and Dr. Maier. The collaboration focused on the Pycket compiler developed in Prof. Tobin-Hochstadt's group, and its use in the AJIT-Par project of Prof. Trinder and Dr. Maier. Concrete outcomes of the collaboration include:

- Pycket extensions for TCP networking. These will be merged into the main Pycket repository.
- Pycket extensions for shared-memory access to data. Prototype extensions have been implemented and tested; this development will continue.

Overall, this visit was very productive, and will shape the work carried out in the programming language research community in Scotland. The visit has strengthened the ongoing collaboration between Indiana and Glasgow on the Pycket compiler.

Typed Racket and Gradual Typing (seminar)

The trend toward constructing large-scale applications in scripting languages has inspired recent research in gradual typing, which adds types incrementally to existing languages. This idea has also now been adopted in industry, with Typed Clojure, TypeScript, and Facebook's Hack as recent example. Over the last decade, my collaborators and I have developed Typed Racket, the first practical gradual type system, to enable adding types to existing untyped Racket programs. Building Typed Racket has required work at every level of programming language research, from runtime systems and compilers, to type and contract system design, to IDE tool support, and even to new proof techniques. In this talk, I'll survey this landscape of work, explain how the needs of Typed Racket has driven all of these areas, and discuss future challenges that remain to be tackled.

Languages as Libraries (seminar)

Programming language design benefits from constructs for extending the syntax and semantics of a host language. While C's string-based macros empower programmers to introduce notational short-hands, the parser-level macros of Lisp encourage experimentation with domain-specific languages. The Scheme programming language improves on Lisp with macros that respect lexical scope.

The design of Racket — a descendant of Scheme — goes even further with the introduction of a full-fledged interface to the static semantics of the language. A Racket extension programmer can thus add constructs that are indistinguishable from "native" notation, large and complex embedded domain-specific languages, and even optimizing transformations for the compiler backend. This power to experiment with language design has been used to create a series of sub-languages for programming with first-class classes and modules, numerous languages for implementing the Racket system, and the creation of Typed Racket, a complete and fully integrated typed sister language to Racket's untyped base language.

In this talk, I'll review the power of Lisp macros for metaprogramming, describe how Scheme introduced lexical scope for macros, and then show how Racket builds upon these foundation to support the development of full-fledged languages as libraries.

Pycket: A Tracing JIT for a functional language (seminar)

Functional languages have traditionally had sophisticated ahead-of-time compilers such as GHC for Haskell, MLton for ML, and Gambit for Scheme. But other modern languages often use JIT compilers, such as Java, Smalltalk, Lua, or JavaScript. Can we apply JIT compilers, in particular the technology of so-called tracing JIT compilers, to functional languages?

I will present a new implementation of Racket, called Pycket, which shows that this is both possible and effective. Pycket is very fast on a wide range of benchmarks, supports most of Racket, and even addresses the overhead of gradual typing-generated proxies.