Ben Greenman October 6, 2015
CS 7480 Summary Gradual Typing for Functional Languages

This paper introduces the catchphrase "gradual typing" and describes a cast-based type system that admits untyped terms [2]. Type safety in this system means that if a term may be assigned a type (with "dynamic" holes), then it will evaluate to either a value or a run-time cast error. To handle diverging terms, the authors parameterize evaluation by a natural number limiting the number of computation steps.

The paper is written "by th' book" [1] and reads nicely. Moreover, all proofs are formalized in Isabelle.

Strengths

- Excellent title.
- Clear presentation.
- Casts are a neat solution to mutable references and untyped functions. (Rather than striving to preserve an invariant, delay the check until necessary.)

Weaknesses

- The benefits of the untyped language are not well motivated (besides terms like "flexible" etc). In particular, I would like to know when gradual typing might be more useful than type inference.
- Eliding blame is a mistake. A CastError is no better than a segfault if it does not accurately tell where the problem occurred.

References

[1] Benjamin C. Pierce. *Types and Programming Languages*. MIT Press, 2002.

[2] Jeremy G. Siek and Walid Taha. Gradual typing for functional languages. In *SFP*, 2006.