Ben Greenman October 5, 2015
CS 7480 Summary Software Contracts for Higher-Order Functions

This paper introduces higher-order contracts for software development [1]. Its contributions include:

- A typed calculus modeling higher-order contracts & their interactions with a type system
- A protocol for assigning blame (when a contract is broken)
- Proofs of contract soundness and compiler correctness
- Notes on a full implementation of the calculus

Ordinary contracts are predicates used to assert the pre- and post-conditions of a function. For example, a function representing the sequence of prime numbers can assert that it takes natural-number arguments and returns a prime number. These predicates, however, are limited to first-order assertions. Higher-order contracts support assertions about higher-order functions. Now a curried division function can assert that when given a real-valued argument, it returns a function on the non-zero real numbers.

Moreover, the paper addresses the issue of *blame* for higher-order contracts. For first-order contracts, it is simple to tell which party is responsible when a contract assertion fails. Either the function caller or the function itself is to blame, and we can stop execution immediately. Higher-order contracts are not checked immediately, therefore we must have a protocol for re-constructing the chain of function calls that led to a contract violation. The paper gives one such a protocol, and reports on the authors' experience using this protocol to build an IDE.

In closing, the authors recommend contracts at the coarsest level possible: module boundaries. This should minimize the number of calls to contract-protected functions, encourage modular development, and enforce API documentation.

Strengths:

- Clear motivation via examples & practical experience.
- The model demonstrates how this work would apply to languages besides Scheme.

Weaknesses:

- No quantitative evaluation of the Dr.Scheme implementation (how many contracts were dependent, flat, etc ...)
- No discussion of future work! I would have liked to compare the authors' predictions with the +10-year research effort that really followed.

Notes

It's funny to me that at least 4 times in this paper, and at least 1 time in his ICFP'14 keynote, Robby emphasizes that contracts are meant to help guide type systems research. I still think the common opinion is that types & contracts are opponents.

At least in Racket, the contracts mostly seem to enforce simple types[1] and until Brian LaChance arrived this summer, you could not even write contracts in Typed Racket.

References

[1] Robert Bruce Findler and Matthias Felleisen. Contracts for higher-order functions. In *ICFP*, 2002.

[2] Michael Greenberg. *Manifest Contracts*. PhD thesis, University of Pennsylvania, 2013.

[1]According to Michael Greenberg's thesis [2], 82% of Racket standard library contracts expressed simple types in 2011.