The paper argues that software contracts should be integrated with a language's type system [1]. Types should be preferred where they can remove the need for dynamic checks, and contracts should be utilized to formulate precise correctness specifications. The main technical result of the paper is a cast insertion algorithm and logical relations proof that inserting upcasts preserves and reflects contextual equivalence.

Strengths

- Hybrid checking sounds very practical. In particular, I like that refinement types are written in the term language rather than asking developers to program in types as well as in programs.
- Similarly, the hybrid approach makes it easy to specify a single function precisely. My experience with other dependently-typed languages is that proving precise specifications requires global reasoning—you may need to give helper functions precise specs, refactor callers of your function to ensure they meet the new preconditions, or (worst of all) modify your datatype and propogate that change throughout the codebase.

Weaknesses

- The string"SMT" doesn't appear in this paper. But we should first push the limits of automated solvers before we give up and resort to dynamic checks.
- The size and complexity of dynamic checks is not considered (besides "we should avoid run-time checks whenever possible"). But the size of an assertion matters very much. I believe a static typechecker could help reduce the size of terms by compiling its incomplete proofs to assertions, so then only a few goals need to be asserted at runtime rather than the entire proposition. It sounds like CCured did similar optimizations [2].

While reading, I realized one of my criticisms from Tuesday was wrong: the dependent subtyping rule is correct. My "counterexample", $\Pi\, x : \mathbf{int} \,.\, x <: \Pi\, x : \mathbf{nat} \,.\, x$, does not mean that any context expecting a function on naturals may be given a function on integers. All values of those types are identity functions, and the integer identity function can safely be used in place of a natural number identity function because it will never be called with an integer argument.

[1] Kenneth Knowles and Cormac Flanagan. Hybrid type checking. In *TOPLAS*, 2009.

[2] George C. Necula, Jeremy Condit, Matthew Harren, Scott McPeak, and Westley Weimer. Ccured: Type-safe retrofitting of legacy software. In *TOPLAS*, 2005.