

Research Plans for Tim Sheard

There is a huge semantic gap between what the programmer knows about his program and the way he has to express this knowledge to a system for reasoning about that program. While many reasoning tools are built on the Curry-Howard isomorphism, it is often hard for the programmers to conceptualize how they can put this abstraction to work. We propose the design of a language that makes this important isomorphism concrete – proofs are real objects that programmers can build and manipulate without leaving their own programming language. Such proofs can express important semantic properties of their programs. We believe that this increases by orders of magnitude the probability that programmers will actually construct programs that they reason about, and this will make measurable differences in the quality of the code produced. It is not that programmers cannot reason about their programs; rather, it is that they find the barriers to entry so high that they would rather not.

In order to make this vision real, we have chosen to explore a new point in the design space of formal reasoning systems. We propose the use of a programming language with a type system in which the user expresses equality constraints between types, which the type checker then enforces. This simple extension to the type system allows the programmer to describe properties of his program in the types of *witness* objects which can be thought of as concrete evidence that the program has the property desired. The addition of two other type extensions, rank-N polymorphism and extensible kinds, will create a powerful new programming idiom for writing programs whose types enforce semantic properties.

These extensions support the construction of meta-programming language that allows the programmer to construct well-typed object language programs as data in the meta-language. This is possible even when the meta-language and object-language are different languages with different type systems. New object language types are defined in using an extensible *kind* mechanism, and the equality types make it possible to define and enforce object-program typing rules. Such a language makes an ideal meta-language as it can be used to combine and reason about multiple layers of system design. Such a meta-language can play an important role in scripting and connecting more powerful tools (such as logical frameworks, theorem provers, generic analysis frameworks, and model checkers) when needed to further enhance system trust.

A language with these features is *both* a practical programming language *and* a logic. This marriage between two previously separate entities further increases the probability that users will apply formal methods to their programming designs. This kind of synthesis creates the foundations for the languages of the future.