

Applications and Tools for Equational Reasoning in Haskell

Research related to WG2.11

John O'Donnell
University of Glasgow

13 May 2004

My research projects involve a combination of practical design and formal reasoning, with a variety of application areas in computer systems. The common threads running through all this work are the use of pure functional programming in Haskell as a specification language, and the use of equational reasoning as the primary formal method. This combination is attractive because Haskell lends itself to executable specifications and practical implementations, as well as to formal reasoning. The formality is tightly integrated with the programming.

The areas in which I have used this approach include both software and hardware design. Because of the clean and simple nature of the specifications, the methods are well suited for exposition and teaching as well as for implementation. The applications include

- *Hydra*, a computer hardware description language (CHDL) embedded in Haskell. Many techniques developed by the Hydra project have become widely adopted, including the use of multiple semantics for circuit specifications. Equational reasoning plays two central roles in Hydra. First, it is available to the designer for deriving circuits, proving their correctness, transforming them to improve efficiency under a cost model, and so on. Second, the Hydra software uses a formal program transformation in order to implement its netlist generation facility without violating referential transparency.
- *Parallel programming*. Combinators are used to express families of parallel operations, at varying levels of abstraction. The programmer can express an algorithm abstractly, using a high level set of combinators, and then gradually add the necessary detail to the program in order to make it executable on a real machine.
- *Discrete mathematics in education*. A toolbox implemented in Haskell is being used to help students learning discrete mathematics by giving a more concrete working understanding of the concepts. The software tools allow one to perform calculations on real examples. The approach is based on algorithms specified in Haskell, which allows much more complex and realistic theorems about programs to be proved than is feasible using imperative languages.

- *Algorithms.* The combination of functional specification and equational reasoning has proved fruitful in the investigation of several new algorithms, especially in the areas of data parallelism and digital circuits.

A characteristic shared by these applications is that the user writes a set of Haskell definitions, performs some program transformations on them, and then executes the result. This requires some kind of metaprogramming support. In the past, the transformations were usually done by hand, but some of them are now implemented using Template Haskell.

It would be ideal to have a flexible and high level toolbox for equational reasoning, which could be used for some or all of the applications described above. However, there are some differences between the needs of these applications. Sometimes the transformations are automatic, sometimes they are manual, and sometimes they could be either. In some cases the system is working with plain Haskell programs, while in others it is working in a first order logic whose predicates are expressed in Haskell. Because of these differences, the problem of defining a single set of equational reasoning tools that covers the spectrum of applications is subtle, and requires much further work.