Measuring and Visualising Haskell Programs

Chris Ryder*
Simon Thompson

Software engineering techniques such as refactoring are growing in popularity. The Haskell community has tools such as `HaRe`[1] to facilitate this activity, however achieving the most effective use of refactoring tools requires careful selection of the particular refactorings to use and of the location to apply them.

Software measurement techniques, also known as software metrics, can offer ways to analyse and quantify various attributes of program code, which in-turn can help indicate unusual characteristics in a program and therefore might be used to select and target refactorings.

There are many established software metrics for imperative and object-oriented languages. such as the pathcount metric which measures complexity of control flow and testing. Some of these metrics can be directly applied to Haskell, but they may ignore common features of functional languages such as pattern matching and higher order functions.

This work investigates a selection of metrics designed to measure attributes of Haskell programs which are not addressed by standard imperative or object-oriented metrics. Examining the change history of a number of Haskell programs has shown correlation between high metric values and increased refactoring or bug fixing changes.

A library of metrics and other ancillary functions has been implemented in Haskell, allowing programmers to produce their own metrics or combine existing ones. To further support the exploration of Haskell programs using metrics, a selection of visualisation systems has been produced.

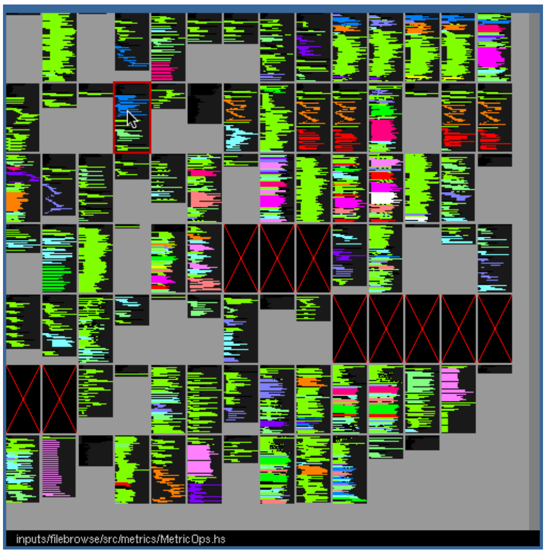

Figure 1: A tool for browsing multiple source files.

An example of a visualisation system included with the library is shown in Figure 1, which shows a collection of source files as rectangles. Each rectangle shows a small "pixel" representation of the text in the corresponding file, colour coded by the value of a pathcount metric. Green areas have the lowest values and red areas have the highest. Rectangles containing a red cross indicate files which failed to parse. This visualisation system allows one to quickly examine a large number of source files to identify sections of program code with unusual behaviour and can also help other tasks, such as finding duplicate code.

*cr24@kent.ac.uk

[1] `http://www.cs.kent.ac.uk/projects/refactor-fp/`