

# HaRe: The Haskell Refactorer

Huiqing Li  
H.Li@kent.ac.uk

Claus Reinke  
C.Reinke@kent.ac.uk

Simon Thompson  
S.J.Thompson@kent.ac.uk

Computing Laboratory, University of Kent

**Abstract** Refactoring is the process of improving the design of existing programs without changing their functionality[1]. Its key characteristic is the focus on the structural change, strictly separated from changes in functionality.

The *Refactoring Functional Programs* project group at the University of Kent is building the *HaRe*[2, 3] system to support refactorings for Haskell programs. HaRe supports the full Haskell 98 and is integrated into two standard development environments: Emacs and Vim. HaRe preserves both the comments and the layout of refactored programs. It is built using the *Programatica* front end for Haskell[4] and the *Strafunski* library for generic abstract syntax tree (AST) traversals[5]. The first version of HaRe was released in October 2003, reported in the paper[2], and it supports a dozen of basic structural refactorings including renaming, generalisation, inlining, scope change and a number of others.

More recent work has concentrated on making the tool stronger and more usable in real projects. This includes:

- making all the implemented refactorings module-aware, so that a change will be reflected in all the modules in a project, rather than just in the module where the change is initiated;
- adding various refactorings for the module systems such as moving definitions between modules, cleaning up the imports, etc;
- adding data-type oriented refactorings such as transforming a concrete algebraic data-type to an abstract data type (ADT).

The poster will report on these aspects in more detail, including illustrative examples.

Our current work is focusing on making the structure of HaRe more open so as to allow the users to define their own refactorings in two different ways: the user can either build their own composite refactorings using the existing basic refactorings and the high-level combinators provided by the system, or build their own basic refactorings using the

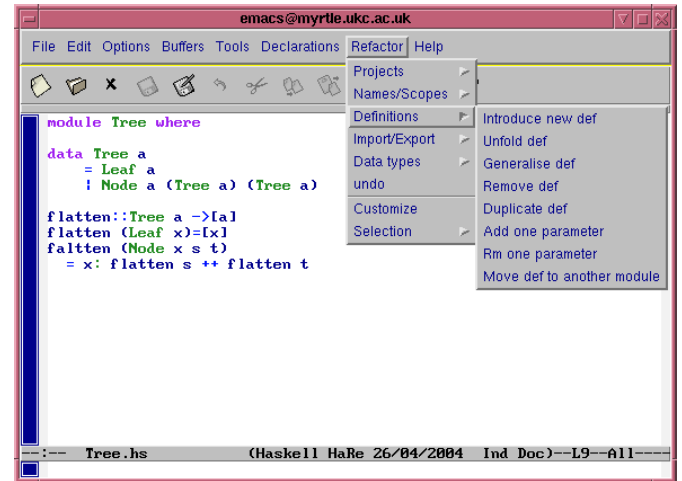


Figure 1: A snapshot of HaRe embedded in Emacs

low-level well-defined API which supports program analysis and AST manipulation. An up-to-date snapshot of HaRe is shown in Figure 1.

## 1. REFERENCES

- [1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.
- [2] H. Li, C. Reinke, and S. Thompson. *Tool support for refactoring functional programs*. In Johan Jeuring, editor, ACM SIGPLAN 2003 Haskell Workshop.
- [3] S. Thompson and C. Reinke. *A case study in refactoring functional programs*. In Roberto Ierusalimsky, Lucilia Figueiredo, and Marcio Tulio Valente, editors, VII Brazilian Symposium on Programming Languages, May 2003.
- [4] PacSoft. *Programatica: Integrating Programming, Properties, and Validation*. <http://www.cse.ogi.edu/PacSoft/projects/programatica/>.
- [5] R. Lämmel and J. Visser. *Generic Programming with Strafunski*. <http://www.cs.vu.nl/Strafunski/>, 1997.