7th International Conference on Applied Informatics
Eger, Hungary, 2007.

Static rules for variable scoping in Erlang*

László Lövei, Zoltán Horváth, Tamás Kozsik, Roland Király

Department of Programming Languages and Compilers
Faculty of Informatics
Eötvös Loránd University, Budapest

Abstract

Erlang/OTP is a functional programming environment designed for building concurrent and distributed fault-tolerant systems with soft real-time characteristics (like telecommunication systems). The core Erlang language consists of simple functional constructs extended with message passing to handle concurrency, and OTP is a set of design principles and libraries that supports building fault-tolerant systems. The language has a very strong dynamic nature that partly comes from concurrency and partly from dynamic language features.

Refactoring is a programming technique for improving the design of a program without changing its behaviour. In other words, you clean up your code but do not change what it does. Refactoring may precede a program modification or extension, preparing the program for the modification, or may be used after finishing the work in order to bring the program into a nicer shape. The transformations of refactoring can be used for optimisation as well.

Most of the refactorings Erlang are concerned about variables in some way. Erlang programs mainly consist of expressions, and expressions can use and define variables almost anywhere. The meaning of variables in an expression depends on its context, so the relation of the variables and the context of the expression must be maintained during a refactoring, otherwise the meaning of the modified program text would differ from the original.

The relation of variables and expressions is defined in terms of visibility rules. An expression can only use visible variables, and the visibility of variables begins with their creation, often in an expression. Every refactoring that works with variables or expressions must be able to determine the exact visibility region of every variable.

The exact semantical rules defining variable visibility are given in [8], using input and output contexts for every language construct, which is hard to follow and not really helpful in defining the conditions of a refactoring. We have created a more compact definition which is suitable for verifying refactoring conditions.

*Supported by GVOP-3.3.3-2004-07-0005/3.0 ELTE IKKK and Ericsson Hungary

References

[1] M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 1999.

[2] Martin Fowler's refactoring site. http://www.refactoring.com/.

[3] H. Li, C. Reinke, and S. Thompson. Tool support for refactoring functional programs. *Haskell Workshop: Proceedings of the ACM SIGPLAN workshop on Haskell, Uppsala, Sweden*, pages 27–38, 2003.

[4] R. Szabó-Nacsa, P. Diviánszky, and Z. Horváth. Prototype environment for refactoring Clean programs. In *The Fourth Conference of PhD Students in Computer Science (CSCS 2004), Szeged, Hungary, July 1–4*, 2004.

[5] P. Diviánszky, R. Szabó-Nacsa, and Z. Horváth. Refactoring via database representation. In L. Csőke, P. Olajos, P. Szigetváry, and T. Tómács, editors, *The Sixth International Conference on Applied Informatics (ICAI 2004), Eger, Hungary*, volume 1, page 129.

[6] J. Armstrong, R. Virding, M. Williams, and C. Wikstrom. *Concurrent Programming in Erlang*. Prentice Hall, 1996.

[7] J. Armstrong. *Making reliable distributed systems in the presence of software errors*. PhD thesis, The Royal Institute of Technology, Stockholm, Sweden, December 2003.

[8] J. Barklund and R. Virding. *Erlang Reference Manual*, 1999. Available from http://www.erlang.org/download/erl_spec47.ps.gz.