A Comparative Study of State Emulation in Functional Programming Languages

William Brick

October 24, 2005

Abstract

The purpose of this paper is to examine the methods of emulating state in functional languages. We will examine the languages J, Scheme, and Haskell; the latter two will be representative of the Lisp and ML families, respectively. In specific, we will provide example code for state emulation in terms of object-orientation and compare this to object-oriented programming without use of state.

1 Introduction

In this paper we will examine the state emulation in J, Scheme, and Haskell. The motivation behind this choice of languages is that we might obtain examples of widely different techniques. Each of these languages takes a very different approach to state emulation, and these three approaches are employed in a wide variety of languages. For example, J uses a special locale to create an entire object system, whereas Scheme encapsulates a state change inside of lexical closures and Haskell uses a special monad referred to as a "state transform monad". Each of these techniques will be examined in depth, and will be used to implement a simplistic semaphore object. We will also discuss a purely functional implementation of this object, evaluating the utility and efficiency of each implementation.
2 State Emulation in J
2.1 Explanation
2.2 Code
2.3 Evaluation
2.4 Comparison
3 State Emulation in Scheme
3.1 Explanation
3.2 Code
3.3 Evaluation
3.4 Comparison
4 State Emulation in Haskell
4.1 Explanation
4.2 Code
4.3 Evaluation
4.4 Comparison
5 Comparative Evaluation
6 Conclusion