Under consideration for publication in J. Functional Programming

Interactive Programming in Agda – Objects and Graphical User Interfaces

ANDREAS ABEL
Department of Computer Science and Engineering, Gothenburg University, Sweden
STEPHAN ADELSBERGER
Department of Information Systems and Operations,
Vienna University of Economics, Austria
ANTON SETZER
Department of Computer Science, Swansea University, Swansea SA2 8PP, UK
(*e-mail:* andreas.abel@gu.se, sadelsbe@wu.ac.at, a.g.setzer@swan.ac.uk)

Abstract

We develop a methodology for writing interactive and object-based programs (in the sense of Wegner) in dependently-typed functional programming languages. The methodology is implemented in the ooAgda library. ooAgda provides a syntax similar to the one used in object-oriented programming languages, thanks to Agda's copattern matching facility. The library allows for the development of graphical user interfaces (GUIs), including the use of action listeners.

Our notion of interactive programs is based on the IO monad defined by Hancock and Setzer, which is a coinductive data type. We use a *sized* coinductive type which allows us to write corecursive programs in a modular way. Objects are server-side interactive programs that respond to method calls by giving answers and changing their state. We introduce two kinds of objects: simple objects and IO objects. Methods in simple objects are pure, while method calls in IO objects allow for interactions before returning their result. Our approach also allows us to extend interfaces and objects by additional methods.

We refine our approach to state-dependent interactive programs and objects through which we can avoid exceptions. For example, with a state-dependent stack object, we can statically disable the pop method for empty stacks. As an example, we develop the implementation of recursive functions using a safe stack. Using a coinductive notion of object bisimilarity, we verify basic correctness properties of stack objects and show the equivalence of different stack implementations. Finally, we give a proof of concept that our interaction model allows to write GUI programs in a natural way: we present a simple drawing program, and a program which allows to move a small spaceship using a button.

Note. We recommend printing this paper in color.

1 Introduction

Functional programming is based on the idea of reduction of expressions. This is a good notion for writing batch programs which take a fixed number of inputs to compute a fixed number of outputs. Interactive programs, however, do not fit directly into this paradigm, since they are programs which over time accept a possibly infinite number of inputs and respond in sequence with a possibly infinite number of outputs. There are several ways to overcome this. In functional programming, the main method currently used is Moggi's