Chasing Bottoms — a Case Study in Program Verification in the Presence of Partial and Infinite Values

Nils Anders Danielsson Patrik Jansson*

February 13, 2004

Abstract

This work is a case study in program verification: We have written a simple parser and a corresponding pretty-printer in a lazy functional programming language with lifted pairs and functions (Haskell). A natural aim is to prove that the programs are, in some sense, each other's inverses. The presence of partial and infinite values in the domains makes this exercise interesting. We have tackled the problem in different ways, and this is a report on the merits of those approaches. More specifically, we first describe a method for testing properties of programs in the presence of partial and infinite values. By testing before proving we avoid wasting time trying to prove statements that are not valid. Then we prove that the programs that we have written are in fact (more or less) inverses using first fixpoint induction and then the approximation lemma.

Infinite values are commonly used in (lazy) functional programs, often to improve modularity [Hug89]. Partial values are seldom used explicitly, but they are still present in all non-trivial Haskell programs because of non-termination, pattern match failures, calls to the *error* function etc. Unfortunately, articles that prove properties about functional programs often ignore details related to partial and infinite values.

This work is a case study where we explore how one can go about testing and proving properties even in the presence of partial and infinite values. We use random testing and two proof methods, fixpoint induction and the approximation lemma, both described in Gibbons' and Hutton's tutorial Proof methods for corecursive programs [GH04].

The programs that our case study revolves around are a simple pretty-printer and a corresponding parser. In Jansson and Jeuring's Polytypic data

*This work is partially funded by the Swedish Foundation for Strategic Research as part of the research programme "Cover — Combining Verification Methods in Software Development".

conversion programs [JJ02] several more complex pretty-printers and parsers are defined and proved correct for total, finite input. Here we use cut down versions of those programs but prove a stronger statement. The programming language used for all programs and properties is Haskell [PJ03], a lazy, pure functional language with lifted pairs and functions.

Before starting to prove something it is often useful to test the properties. That way one can avoid spending time trying to prove something which is not true anyway. However, testing partial and infinite values can be tricky. We combine two (already known) techniques for doing that. Infinite values can be tested with the aid of the approximation lemma, and for partial values we make use of some Haskell extensions (implemented in the Haskell compiler GHC).

The method used for testing turns out to work well, with some obvious limitations related to non-termination. The proofs are mostly straightforward, but rather lengthy and tedious compared to those proofs that only take finite, total values into account. A tentative conclusion of this experiment is that to be able to scale up the proof methods used to larger programs, we have to settle for approximative (conservative) results, unless large parts of the proof process can be automated.

References

[GH04] Jeremy Gibbons and Graham Hutton. Proof methods for corecursive programs. To be submitted for publication, available online at `http://www.cs.nott.ac.uk/~gmh/bib.html`, February 2004.

[Hug89] John Hughes. Why functional programming matters. *Computer Journal*, 32(2):98–107, 1989.

[JJ02] Patrik Jansson and Johan Jeuring. Polytypic data conversion programs. *Science of Computer Programming*, 43(1):35–75, 2002.

[PJ03] Simon Peyton Jones, editor. *Haskell 98 Language and Libraries, The Revised Report.* Cambridge University Press, 2003.