Available online at www.sciencedirect.com
SCIENCE DIRECT®
Electronic Notes in Theoretical Computer Science
ELSEVIER Electronic Notes in Theoretical Computer Science 136 (2005) 57–102
www.elsevier.com/locate/entcs

Programming Examples Needing Polymorphic Recursion

J. J. Hallett[1]

Department of Computer Science
Boston University
Boston, USA

A. J. Kfoury[2]

Department of Computer Science
Boston University
Boston, USA

Abstract

Inferring types for polymorphic recursive function definitions (abbreviated to *polymorphic recursion*) is a recurring topic on the mailing lists of popular typed programming languages. This is despite the fact that type inference for polymorphic recursion using ∀-types has been proved undecidable. This report presents several programming examples involving polymorphic recursion and determines their typability under various type systems, including the Hindley-Milner system, an intersection-type system, and extensions of these two. The goal of this report is to show that many of these examples are typable using a system of intersection types as an alternative form of polymorphism. By accomplishing this, we hope to lay the foundation for future research into a decidable intersection-type inference algorithm.
We do not provide a comprehensive survey of type systems appropriate for polymorphic recursion, with or without type annotations inserted in the source language. Rather, we focus on examples for which types may be inferred without type annotations, with an emphasis on systems of intersection-types.

Keywords: polymorphic recursion, intersection types, finitary polymorphism , examples

[1] Email: jhalllett@cs.bu.edu
[2] Email: kfoury@cs.bu.edu

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2005.06.014

1 Introduction

Background and Motivation

Type inference in the presence of polymorphic recursion using ∀-types (the familiar "type schemes" of SML) is undecidable
[10,11,4]. Attempts to work around this limitation include explicit type annotations by the user [8] and user-tunable iteration limits [17]. However, both of these approaches require the programmer to be actively engaged in the type checking process, thereby defeating the goal of automatic type inference and transparent type checking. There is also an implementation of SML that allows the user to switch between the standard type system (which is restricted to monomorphic recursion) and a type system augmented with polymorphic recursion using ∀-types, in an attempt to prove that "hard" examples of polymorphic recursion do not arise in practice [1]. Yet, practical examples of programs requiring polymorphic recursion continually appear in discussions on the mailing lists of programming languages such as SML, Haskell, and OCaml.

Contribution of the Report

This document attempts to lay the foundation for further research into the typability of implicit polymorphic recursion by discussing several examples which fail to type under the standard type system of SML – also called the Hindley-Milner system. The examples are written (mostly) in SML syntax (one example is presented in Haskell syntax) and are accompanied by the corresponding error found by the SML/NJ type checker. A few of the examples are also shown in Haskell syntax with its corresponding GHC error message for the side purpose of comparing the error reporting of the SML/NJ and GHC compilers.

We also discuss examples which remain untypable using the Hindley-Milner system augmented with polymorphic recursion with ∀-types – also called the Milner-Mycroft system – but are typable using an intersection-type system. These examples support the use of intersection types as an alternative to ∀-types to represent polymorphism.

In addition, we elucidate the need for what we call "infinite-width" intersection types by examples. However, we do not extend our standard (finite-width) intersection type system in this way, because we do not know a straightforward extension of the standard system and developing one is beyond the scope of this report. Consequently, we resort to polymorphic recursion with ∀-types for these examples; i.e., we present examples which are not typable