Principal Type Inference for GHC-Style Multi-parameter Type Classes

Martin Sulzmann[1], Tom Schrijvers[2,⋆], and Peter J. Stuckey[3]

[1] School of Computing, National University of Singapore
S16 Level 5, 3 Science Drive 2, Singapore 117543
sulzmann@comp.nus.edu.sg
[2] Department of Computer Science
Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
tom.schrijvers@cs.kuleuven.be
[3] NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne, Vic. 3010, Australia
pjs@cs.mu.oz.au

Abstract. We observe that the combination of multi-parameter type classes with existential types and type annotations leads to a loss of principal types and undecidability of type inference. This may be a surprising fact for users of these popular features. We conduct a concise investigation of the problem and are able to give a type inference procedure which, if successful, computes principal types under the conditions imposed by the Glasgow Haskell Compiler (GHC). Our results provide new insights on how to perform type inference for advanced type extensions.

1 Introduction

Type systems are important building tools in the design of programming languages. They are typically specified in terms of a set of typing rules which are formulated in natural deduction style. The standard approach towards establishing type soundness is to show that any well-typed program cannot go wrong at run-time. Hence, one of the first tasks of a compiler is to verify whether a program is well-typed or not.

The trouble is that typing rules are often not syntax-directed. Also, we often have a choice of which types to assign to variables unless we demand that the programmer supplies the compiler with this information. However, using the programming language may then become impractical. What we need is a type inference algorithm which automatically checks whether a program is well-typed and as a side-effect assigns types to program text.

⋆ Research Assistant of the fund for Scientific Research - Flanders (Belgium)(F.W.O. - Vlaanderen).

N. Kobayashi (Ed.): APLAS 2006, LNCS 4279, pp. 26–43, 2006.

For programming languages based on the Hindley/Milner system [19] we can typically verify that type inference is *complete* and the inferred type is *principal* [1]. Completeness guarantees that if the program is well-typed type inference will infer a type for the program whereas principality guarantees that any type possibly given to the program can be derived from the inferred type.

Here, we ask the question whether this happy situation continues in the case of multi-parameter type classes (MPTCs) [13], a popular extension of the Hindley/Milner system and available as part of Haskell [21] implementations such as GHC [5] and HUGS [9]. GHC and HUGS also support (boxed) existential types [17] and type annotations [20].[1] It is the combination of all these features that make MPTCs so popular among programmers.

In this paper, we make the following contributions:

- We answer the above question negatively. We show that the combination of MPTCs with type annotations and existential types does not enjoy principal types and type inference is undecidable in general (Section 2).
- However, under the GHC [5] multi-parameter type class conditions, we can give a procedure where every inferred type is principal among all types (Section 4).

We omit proofs for brevity, sketches can be found in [26].

To the best of our knowledge, we are the first to point out precisely the problem behind type inference for MPTCs. Previous work [3] only reports the loss of principal types but does not provide many clues about how to tackle the inference problem.

We have written this introduction as if Haskell (GHC and HUGS) is the only language (systems) that supports MPTCs. Type classes are also supported in a number of other languages such as Mercury [7,10], HAL [2] and Clean [22]. However, as far as we know there is no formal description of multi-parameter type classes and the combination with existential types and type annotations. From now on, we will use MPTCs to refer to the system that combines all these features. For example, Läufer [16] only considered the combination of single-parameter type classes and existential types. The only formal description available is our own previous work [27] where we introduce the more general system of extended algebraic data types (EADTs). Notice that in [27] we discuss type checking but not type inference.

In the next section, we give a cursory introduction to MPTCs as supported in GHC based on a simple example. We refer to [13] for further examples and background material on MPTCs.

2 Multi-parameter Type Classes

Example. We use MPTCs for the implementation of a stack ADT.

[1] For the purposes of this paper, we will use the term "type inference" to refer to type inference and checking in the presence of type annotations.