

# How to make *ad-hoc* polymorphism less *ad hoc*

Philip Wadler and Stephen Blott  
University of Glasgow\*

October 1988

## Abstract

This paper presents *type classes*, a new approach to *ad-hoc* polymorphism. Type classes permit overloading of arithmetic operators such as multiplication, and generalise the “eqtype variables” of Standard ML. Type classes extend the Hindley/Milner polymorphic type system, and provide a new approach to issues that arise in object-oriented programming, bounded type quantification, and abstract data types. This paper provides an informal introduction to type classes, and defines them formally by means of type inference rules.

## 1 Introduction

Strachey chose the adjectives *ad-hoc* and *parametric* to distinguish two varieties of *polymorphism* [Str67].

*Ad-hoc* polymorphism occurs when a function is defined over several different types, acting in a different way for each type. A typical example is overloaded multiplication: the same symbol may be used to denote multiplication of integers (as in `3*3`) and multiplication of floating point values (as in `3.14*3.14`).

Parametric polymorphism occurs when a function is defined over a range of types, acting in the same way for each type. A typical example is the `length` function, which acts in the same way on a list of

integers and a list of floating point numbers.

One widely accepted approach to parametric polymorphism is the Hindley/Milner type system [Hin69, Mil78, DM82], which is used in Standard ML [HMM86, Mil87], Miranda<sup>1</sup>[Tur85], and other languages. On the other hand, there is no widely accepted approach to *ad-hoc* polymorphism, and so its name is doubly appropriate.

This paper presents *type classes*, which extend the Hindley/Milner type system to include certain kinds of overloading, and thus bring together the two sorts of polymorphism that Strachey separated.

The type system presented here is a generalisation of the Hindley/Milner type system. As in that system, type declarations can be inferred, so explicit type declarations for functions are not required. During the inference process, it is possible to translate a program using type classes to an equivalent program that does not use overloading. The translated programs are typable in the (ungeneralised) Hindley/Milner type system.

The body of this paper gives an informal introduction to type classes and the translation rules, while an appendix gives formal rules for typing and translation, in the form of inference rules (as in [DM82]). The translation rules provide a semantics for type classes. They also provide one possible implementation technique: if desired, the new system could be added to an existing language with Hindley/Milner types simply by writing a pre-processor.

Two places where the issues of *ad-hoc* polymorphism arise are the definition of operators for arithmetic and equality. Below we examine the approaches to these three problems adopted by Standard ML and Miranda; not only do the approaches differ between the two languages, they also differ within a single language. But as we shall see, type classes provide a uniform mechanism that can address these problems.

---

\*Authors' address: Department of Computing Science, University of Glasgow, Glasgow G12 8QQ, Scotland. Electronic mail: `wadler, blott@cs.glasgow.ac.uk`.

Published in: *16'th ACM Symposium on Principles of Programming Languages*, Austin, Texas, January 1989.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

---

<sup>1</sup>Miranda is a trademark of Research Software Limited.