Overloading

Elizabeth Labrada Deniz *[1]

[1] *Computer Science Department (DCC), University of Chile, Chile*

Abstract

1 Introduction

In this research we are interested in the study of overloading. This feature of the programming languages, is a kind of polyphormism included in a lot of language like: Scala, C++, Java, Haskell, and many others. The classification of the different types of polyphormism presented by Cardelli and Wegner [2], is relevant to understand overloading and their differences with another kind of polyphormism(Figure 1).

On the one hand, universally polymorphic functions typically work on an infinite number of types, where the types share a common structure. Parametric polymorphism occurs when a function defined over a range of types has a single implemetation, acting in the same way for each type [2, 3, 8]. The identity function is one of the simplest examples of parametric polymorphic function, where for any type, the behavior is the same. Inclusion polymorphism is used to model subtypes and inheritance, of the object-oriented paradigm.

On the other hand, ad-hoc polymorphism [2, 8], is obtained when a function is defined over several different types and may behave in unrelated ways for each type. Overloading and coercion polymorphism are classified as the two major subcategories of ad-hoc polymorphism, according to Figure 1. In general, we are in present of overloading when the same variable name is used to denote different functions, then the context is essential to decide which function is selected by a particular instance of the name. An example of the overloading function is +, since it is applicable to both integer and real arguments. Additionally, a coercion [2] is a semantic operation which is needed to convert an argument to the type expected by a function, in a situation which would otherwise result in a type error. The function + is also an example of coercion, if it is defined only for real addition, and integer arguments are always coerced to corresponding reals.

*Funded by grant CONICYT, CONICYT-PCHA/Doctorado Nacional/2015-63140148

$$Polyphormism = \begin{cases} universal & \begin{cases} parametric \\ inclusion \end{cases} \\ ad\ hoc & \begin{cases} overloading \\ coercion \end{cases} \end{cases}$$

Figure 1: Varieties of polyphormism.

Commonly, overloading is integrated in a programming language with another kind of polymorphism. For instance, a type class in Haskell [5, 6, 7, 8] is a sort of interface that defines some behavior and it includes overloading and parametric polymorphism. As well, another languages like Java [1], C++, C# and others, implement overloading with inclusion and parametric polymorphism, with inheritance and generic. In this paper, we have the goal of designing a calculus for overloading without another types of polymorphism, with the aim of studying this mechanism alone.

We want to develop a calculus for overloading with the purpose of, in future researchs, to understand its meaning in the gradual types world. The literature reviewed shows that, mostly, given a program, overloading is eliminated by giving different names to the different functions or overloaded variables are tagged [1, 2, 3, 4, 5, 6, 7, 8].

2 Concepts

2.1 Explicit substitution

2.2 Local Type Inference

2.3 Polymorphism

3 Ad-hoc Polymorphism

3.1 Type classes

3.2 Featherweight Java with dynamic and static overloading

References

[1] M. Aleksy, V. Amaral, R. Gitzel, J. Power, J. Waldron, L. Bettini, S. Capecchi, and B. Venneri. Featherweight java with dynamic and static overloading.

Science of Computer Programming, 74(5):261 – 278, 2009.

[2] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *Computing Surveys*, 17(4):471–522, 1986.

[3] S. L. Kilpatrick. Ad hoc: Overloading and language design. Master's thesis, University of Texas at Austin, 2010.

[4] S. P. J. Mark Shields. Object-oriented style overloading for haskell. September 2001.

[5] T. Nipkow and C. Prehofer. Type checking type classes. In *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '93, pages 409–418, New York, NY, USA, 1993. ACM.

[6] M. Odersky, P. Wadler, and M. Wehr. A second look at overloading. In *Proceedings of the Seventh International Conference on Functional Programming Languages and Computer Architecture*, FPCA '95, pages 135–146, New York, NY, USA, 1995. ACM.

[7] R. Ribeiro and C. Camarão. Ambiguity and context-dependent overloading. *Journal of the Brazilian Computer Society*, 19(3):313–324, 2013.

[8] P. Wadler and S. Blott. How to make ad-hoc polymorphism less ad hoc. In *Proceedings of the 16th ACM Symposium on Principles of Programming Languages (POPL 89)*, pages 60–76, Austin, TX, USA, Jan. 1989.