Overloading

Elizabeth Labrada Deniz [*1]

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

Abstract

1 Introduction

In this brief survey, we present concept related to overloading and relevant works that formalize overloading with different characteristics.

2 Background

In this section we explain some concepts related to overloading, like polymorphism and the different kind of overloading.

2.1 Polymorphism

- Cardelli and Wegner [5] present a classification of the different types of polyphormism which result relevant to understand overloading and its differences with another kind of polyphormism.
- Parametric polymorphism [5, 7, 12] Concept and example
- Inclusion polymorphism Concept and example
- Overloading [5, 12] Concept and example
- Coercion [5] Concept and example
-

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

2.2 Static Overloading

- Concept and example

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

2.3 Dynamic Overloading

- Concept and example
- Common Lisp Object System(CLSO) [3] is an object-oriented extension to Common Lisp which implements dynamic overloading. It is worth noting that CLSO do not admit any static checking, therefore is dynamically typed.

3 Overloading

In the preset section we report some different works related to the overloading field. We start with type class, a powerful mechanism which allow overloading, combined with parametric polyphormism. Then we present an interesting work that explain overloading in object-oriented programming languages, with static or dynamic overloading.

3.1 Type classes

- Type class in Haskell [12] is a sort of interface that defines some behavior supporting overloading and parametric polymorphism. A type class is defined by Nipkow et al. [8] , as a set of types, which all happen to provide a certain set of functions.
- Example type class class Eq
- All articles revised use Hindley/Damas/Milner system. Every expression which has a type has a most general type which can be inferred automatically [8].
- In the work of Camarão et al. [4] is summarized some of the limitations or requirements of type classes such as: the declaration of an overloaded function requires a class declarations, all the overloaded instances must share a common type pattern and definitions of overloaded symbols must occur in global instance declarations.
- open world assumption where the set of declared instances is considered open to extension (from different modules, introduced at link time).

- Context-dependent overloading, is a form of overloading where the selection of the definition for an application function depends not only on the types of the arguments, but also on the call context. For instance, if we have the expression $m\ e$, we use for the correct selection of the definition of m, the type of e and the information of the context in which $m\ e$ is used.

- Ribeiro and Camarão [10] note that a type system for Haskell allows distinct derivations for ambiguous expressions, which are usually rejected by the type inference algorithm. Also, they remark that the Haskell's open world approach affects the standard definition of ambiguity.

- Coherence establishes a single well-defined meaning for each expression.

- Since the original type inference system is undecidable [7], some additional (syntactical) restrictions had to be imposed to obtain a type system where most general typings can efectively be computed [11].

- Type checking type classes [8].

- A second look at overloading [9].

- Type Inference for Overloading without Restrictions, Declarations or Annotations [4]

3.2 Overloading in Object Oriented Paradigm

- Featherweight Multi Java (FMJ) [1, 2] is an extension of Featherweight Java (FJ) [6] with multimethods.

- All the inherited overloaded methods are copied into the subclass.

- The receiver type of the method invocation has no precedence over the argument types, when the dynamic overloading selection is performed.

- All method invocations are annotated with the type selected during static type checking, in order to choose the best specialized branch during the dynamic overloading method selection. Thus, at run-time it is sound to select only a specialization of the static type.

- A procedure to select the most appropriate branch at run-time using both the dynamic type of the arguments and the annotated static type guarantees that no ambiguity can dynamically occur in well-typed programs.

- In FMJ is used the concept of multi-types, which represents the types of multi-methods. Formally, a multi-types is a set of arrows types

- The approach of [1] propose an straightforward way to obtain static overloading, changing the rule (RInvk) by the rule (RSInvk).

- It is important to note that FMJ, and more general Java, only admit overloaded method invocation, unlike type classes. Type classes allow function calls and function arguments overloaded. The reason for this is that in Java, the functions are not first-class citizen.

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. Bettini, S. Capecchi, and B. Venneri. Featherweight java with multi-methods. In *Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java*, PPPJ '07, pages 83–92, New York, NY, USA, 2007. ACM.

[3] D. G. Bobrow, L. G. DeMichiel, R. P. Gabriel, S. E. Keene, G. Kiczales, and D. A. Moon. Common lisp object system specification. *SIGPLAN Not.*, 23(SI):1–142, Sept. 1988.

[4] C. Camarão and L. Figueiredo. *Type Inference for Overloading without Restrictions, Declarations or Annotations*, pages 37–52. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.

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

[6] A. Igarashi, B. C. Pierce, and P. Wadler. Featherweight java: A minimal core calculus for java and gj. *ACM Trans. Program. Lang. Syst.*, 23(3):396–450, May 2001.

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

[8] 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.

[9] 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.

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

[11] H. Seidl. Haskell overloading is dexptime-complete. *Information Processing Letters*, 52(2):57 – 60, 1994.

[12] 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.