Overloading

Elizabeth Labrada Deniz *[1]

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

Abstract

1 Introduction

Explicar Overloadin, ejemplos, para que sirve. Hablar de overl $\mathsf{wellformed}(\{\overline{\overline{S} \to S}\})$

2 Concepts

2.1 Explicit substitution

2.2 Local Type Inference

2.3 Polymorphism

2.4 Static Overloading

2.5 Dynamic Overloading

2.6 Polymorphism

3 Ad-hoc Polymorphism

3.1 Type classes

3.2 Featherweight Java with dynamic and static overloading

Featherweight Multi Java (FMJ) [1, 2] is an extension of Featherweight Java (FJ) [3] with multi-methods. FJ is a basic version of Java, which focuses on the following set of features: class definitions, object creation, method invocation, field access, inheritance, subtyping and method recursion through this. For the resolution overloading:

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

Syntax :			
L	::=	class C extends C $\{\overline{C}\ \overline{f};\ K;\ \overline{M}\}$	classes
K	::=	$C(\overline{C}\ \overline{f})\{\mathsf{super}(\overline{f});\ \mathsf{this}.\overline{f} = \overline{f}\}$	constructors
M	::=	$C\ m\ (\overline{C}\ \overline{x})\{\mathsf{return}\ e;\}$	methods
e	::=	$x \mid e.f \mid e.m(\overline{e}) \mid \mathsf{new}\ C(\overline{e})$	expressions
v	::=	$\mathsf{new}\ C(\overline{v})$	values

Figure 1: Syntax of FMJ.

$$\frac{\Gamma \vdash e : C}{a} \quad \text{(TInvk)}$$

$$\frac{a}{a} \quad \text{(TMethod)}$$

$$\frac{a}{a} \quad \text{(TClass)}$$

Figure 2: Typing rules for FMJ.

- 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.

the authors point out that semantics of overloading and inheritance is rather clean if it is interpreted through a copy semantics of inheritance, whereby all the inherited overloaded methods are intended to be directly copied into the subclass (apart for those explicitly redefined by the subclass itself)

, and the branch selection is symmetric: during dynamic overloading selection the receiver type of the method invocation has no precedence over the argument types (differently from the encapsulated multimethods of [14] and [13]). Symmetry of branch selection is tightly related to copy semantics: if the receiver of a method invocation had the precedence over the parameters, we

would search for the best matching branch only in the class of the receiver (or in the first superclass that defines some branch in case the class of the receiver does not define branches), i.e., without inspecting also the superclasses branches (which could provide a more specialized version). Thus, copy semantics would not be implemented

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] 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.