

Ph.D. Dissertation Summary: Type-Oriented Logic Meta-Programming

Kris De Volder

March 21, 2000

1 Thesis

The dissertation shows that undecidable and ambiguous type systems have useful applications and should therefore be considered viable options for future statically typed object-oriented¹ languages.

Traditionally, type systems have been included into programming languages for the following reasons [MMMP90]:

documentation Static types improve the readability of programs.

robustness Detection of “type errors”.

efficiency Optimizations based on static type knowledge.

The aforementioned uses are passive in the sense that types have mostly a descriptive nature. This makes them great as documentation. The type checker verifies whether the description is a consistent one and this increases robustness by eliminating type errors. Optimizers also make use of the descriptive nature of types to perform optimizations.

We want to draw attention in this dissertation to the potential of “actively” using static types, writing real programs that manipulate types: consult them, construct them, or make decisions regarding interfaces or internal representations of classes. In short, we want to draw attention to the potential of active manipulation of types by real “type programs”, expressed with a real, Turing-complete, type-programming language. It is in this active type manipulation that lies the currently unharvested potential which makes undecidable and ambiguous type systems potentially useful.

2 Dissertation

To prove our thesis we implemented a system which offers a full-fledged logic programming language with which types can be constructed, consulted and implemented. This approach is highly reminiscent of and inspired by the flavor of instance declarations for type and constructor classes [Kae88, WB89, Jon93, Jon94, OWW95] in the functional languages Gofer [Jon95] and Haskell [PH⁺97].

Because of limited resources we have been forced to take a less than perfect approach. The system we implemented is not a real type system in the true sense of the word since it does no type checking of its own. Type systems for object-oriented languages being very complicated artifacts both theoretically and implementation wise, this was not a feasible option to us. Instead, our system generates Java code and leaves type checking to the Java compiler. Potential type errors are therefore only detected in as far as the generated output code violates the Java typing rules. Clearly this is not what one wants for a production level environment. However, since the added potential of having a Turing-complete type language is in active type manipulation rather than in the type-checking aspect of the system, this is sufficient to prove our point.

The dissertation provides several examples that illustrate the usefulness of having a real, Turing complete, type-programming language. These examples show how type-oriented meta programming complements the expressiveness of the base programming language. Amongst others, we discuss how type-oriented logic meta programming is potentially useful to support *aspect-oriented programming* [MLTK97].

¹This statement can probably be made about static type systems in general. This dissertation however focuses on object-oriented languages.

References

- [Jon93] Mark P. Jones. *GOFER 2.28 Release Notes*. Departement of Computer Science, Yale University, februari 1993.
- [Jon94] Mark P. Jones. A theory of qualified types. *Science of Computer Programming*, 22(3):231–256, June 1994.
- [Jon95] M. P. Jones. *Gofer*. CS, Yale, August 1995.
- [Kae88] Stefan Kaes. Parametric overloading in polymorphic programming languages. In H. Ganzinger, editor, *Proceedings of the European Symposium on Programming*, volume 300 of *Lecture Notes in Computer Science*, pages 131–144. Springer Verlag, 1988.
- [MLTK97] K. Mens, C. Lopez, B. Tekinerdogan, and G. Kiczales. Aspect-oriented programming. In Jan Bosch and Stuart Mitchell, editors, *ECOOP 97 Workshop Reader*, Lecture Notes in Computer Science, pages 483–496. Springer Verlag, 1997.
- [MMMP90] Ole Lehrmann Madsen, Boris Magnusson, and Birger Møller-Pedersen. Strong Typing of Object-Oriented Languages Revisited. In *Proceedings of the OOPSLA/ECOOP '90 Conference on Object-oriented Programming Systems, Languages and Applications*, pages 140–150, October 1990. Published as ACM SIGPLAN Notices, volume 25, number 10.
- [OWW95] Martin Odersky, Philip Wadler, and Martin 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, La Jolla, California, June 25–28, 1995. ACM SIGPLAN/SIGARCH and IFIP WG2.8, ACM Press.
- [PH⁺97] John Peterson, Kevin Hammond, et al. Report on the programming language haskell, a non-strict purely-functional programming language, version 1.4. Technical report, Yale University, April 1997.
- [WB89] P. Wadler and S. Blott. How to make *ad-hoc* polymorphism less *ad hoc*. In *16th ACM Symposium on Principles of Programming Languages*, pages 60–76, 1989.