Annotated bibliography for the tutorial on "Exploring typed language design in Haskell"*

Oleg Kiselyov[1] and Ralf Lämmel[2]

[1] Fleet Numerical Meteorology and Oceanography Center, Monterey, CA

[2] Universität Koblenz-Landau, Software Languages Team, Koblenz, Germany

Draft as of January 13, 2010

Abstract

The tutorial is primarily based on our publication [11] and the OOHaskell draft [10]. The introductory lecture (on the Expression Problem & Co.) also leverages our publication [14]. Besides, the tutorial *takes advantage of* and *relates to* the rich literature on type-level programming (mostly in Haskell), object encoding and OO programming support in functional programming (again, mostly in Haskell, but also see the mentioning of OCaml and ML-ART), and several other subjects of programming-language research, e.g., the Expression Problem, and record calculi.

References

[1] G. Bracha and G. Lindstrom. Modularity Meets Inheritance. In *Proceedings: 4th International Conference on Computer Languages*, pages 282–290. IEEE Computer Society Press, 1992. Available online.

*This text is under construction. We are more than happy to add entries. In fact, we are aware of much more related work (see [10, 11]), but help with adding entries is most welcome. Please send us an email, perhaps even with a proposal for a comment to be included in this annotated bibliography.

The paper presents an advanced record calculus. It actually separates a number of roles otherwise readily merged in actual OO programming languages, and thereby allows for a design-space exploration. For instance, basic record operations, inheritance, and mixins are covered. Our work on heterogenous collections, extensible records, and OOHaskell [11, 10] relates to this line of work in so far that it shows how such calculi and their applications can be readily explored (in fact, embedded) in Haskell—based on type-level programming.

[2] Manuel M. T. Chakravarty, Gabriele Keller, and Simon Peyton Jones. Associated type synonyms. In *ICFP '05: Proceedings of the tenth ACM SIGPLAN international conference on Functional programming*, pages 241–253. ACM, 2005. Available online.

The paper describes a form of indexed type families (see [3] for a companion paper). Type families are a more recent extension to Haskell that provides an alternative to functional dependencies for multi-parameter type classes. In the tutorial, we discuss all alternatives.

[3] Manuel M. T. Chakravarty, Gabriele Keller, Simon Peyton Jones, and Simon Marlow. Associated types with class. In *POPL '05: Proceedings of the 32nd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 1–13. ACM, 2005. Available online.

The paper describes a form of indexed type families (see [2] for a companion paper). Type families are a more recent extension to Haskell that provides an alternative to functional dependencies for multi-parameter type classes. In the tutorial, we discuss all alternatives.

[4] Sigbjorn Finne, Daan Leijen, Erik Meijer, and Simon Peyton Jones. Calling hell from heaven and heaven from hell. In *ICFP '99: Proceedings of the fourth ACM SIGPLAN international conference on Functional programming*, pages 114–125. ACM, 1999. Available online.

[5] Matthew Fluet and Riccardo Pucella. Phantom types and subtyping. *Journal of Functional Programming*, 16(6):751–791, 2006. Available online.

The paper investigates the established technique of phantom types in depth in the context of subtyping. It shows that such phantom types can be used to model subtyping hierarchies.

[6] B.R. Gaster and M.P. Jones. A Polymorphic Type System for Extensible Records and Variants. Technical report NOTTCS-TR-96-3, University of Nottingham, Department of Computer Science, 1996. Available online.

This is a seminal reference in so far that it describes a type-system extension for Haskell that covers extensible records and that has been actually implemented and also used relatively well. Our tutorial (and [11] for that matter) demonstrates how the expressivity (including the aspects of static typing specifically) of extensible records and other related concepts can be achieved by type-level programming.

[7] T. Hallgren. Fun with functional dependencies. In *Joint Winter Meeting of the Departments of Science and Computer Engineering, Chalmers University of Technology and Goteborg University, Varberg, Sweden, Jan. 2001*, 2001. Available online.

Just like [18], this paper is a seminal reference on type-level programming in Haskell. Various, by now classical examples or idioms of type class-based programming appear in the paper. The development is specifically based on the point of view that such type class-based programming enables compile-time computations, and hence triggers a separation of static and dynamic computations. This view is similar to the one of metaprogramming in the context of C++ and its template system.

[8] J. Hughes and J. Sparud. Haskell++: An Object-Oriented Extension of Haskell. In *Proc. of Haskell Workshop, La Jolla, California*, YALE Research Report DCS/RR-1075, 1995.

[9] Didier Rémy Jérôme Vouillon and Jacques Garrigue. The Objective Caml system, release 3.10, Documentation and user's manual, Chapter 3. Objects in Caml, 16 May 2007. `http://caml.inria.fr/pub/docs/manual-ocaml/index.html`.

Part of the OOHaskell [10] effort was focused at the question whether we can achive OCaml's expressivity in Haskell by means of an appropriate extensible record system and other idioms of type-level programming. We have, in fact, covered nearly all the (principled) examples from the OCaml manual in the code distribution of OOHaskell.

[10] Oleg Kiselyov and Ralf Lämmel. Haskell's overlooked object system. Available online, 2005.

The paper is the backbone of the tutorial. The paper shows that the basic tenet of OO and quite advanced expressivity (especially in terms of type-system variations) can be modelled in Haskell—starting from a powerful record calculus.

[11] Oleg Kiselyov, Ralf Lämmel, and Keean Schupke. Strongly typed heterogeneous collections. In *Haskell '04: Proceedings of the 2004 ACM SIGPLAN workshop on Haskell*, pages 96–107. ACM, 2004. Available online.

The paper advances type-level programming in Haskell based on type classes and functional dependencies. That is, the paper provides types for the programming domain of heterogenous collections, and it shows the usefulness of this development for supporting type-safe database access and providing an extensible record system.

[12] Oleg Kiselyov and Chung-chieh Shan. Functional pearl: implicit configurations–or, type classes reflect the values of types. In *Haskell '04: Proceedings of the 2004 ACM SIGPLAN workshop on Haskell*, pages 33–44. ACM, 2004. Available online.

The paper addresses the configurations problem, i.e., the problem of propagating run-time preferences through the program. To this end, it contributes a type-level programming technique which proopagates configurations type-safely through type-level reifications. Part of this technique is a simulation of coherent, local type-class instances.

[13] Oleg Kiselyov, Ken Shan, and Simon Peyton Jones. Fun With Type Functions, 2009. Draft. Available online.

The paper provides a more recent and application-oriented discussion of type-level programming in Haskell. The paper specifically discusses the more recent language constructs for type-level programming with type functions based on (indexed) type families. Various applications of type-level programming are sketched, e.g., memoization and session types.

[14] Ralf Lämmel and Klaus Ostermann. Software extension and integration with type classes. In *GPCE '06: Proceedings of the 5th international conference on Generative programming and component engineering*, pages 161–170. ACM, 2006. Available online.

The paper (like so many others, see, e.g., [28, 30]) discusses the Expression Problem, and other extensibility or software integration problems. The specific contribution of the paper is to clarify the capabilities of Haskell's type classes in addressing these various problems.

[15] Daan Leijen. Extensible records with scoped labels. In *Proceedings of the 2005 Symposium on Trends in Functional Programming (TFP'05)*, September 2005.

[16] Daan Leijen and Erik Meijer. Domain specific embedded compilers. In *PLAN '99: Proceedings of the 2nd conference on Domain-specific languages*, pages 109–122. ACM, 1999. Available online.

This is a seminal reference on language embedding. Like many other papers of that tradition, domain-specific languages are modelled as combinator libraries, i.e., suites of higher-order functions. The approach is particularly interesting for our type-level and functional OO programming endeavour because it uses extensible records, phantom types and various forms of polymorphism for the embedding. We see type-level programming as the natural next step to achieve even more programmability in language embedding.

[17] Andres Löh and Ralf Hinze. Open data types and open functions. In *PPDP '06: Proceedings of the 8th ACM SIGPLAN international conference on Principles and practice of declarative programming*, pages 133–144. ACM, 2006. Available online.

[18] C. McBride. Faking It (Simulating Dependent Types in Haskell). *Journal of Functional Programming*, 12(4–5):375–392, July 2002. Available online.

> The paper is a seminal reference on type-level programming in Haskell. It is profound in so far that it relates such type-class programming to dependent typing. The paper is also most clear in so far that it clarifies the nature of type class-based type-level programming in Haskell—i.e., the use of counterfeits of type-level copies of data. See [7] for another similar development conveived about the same time.

[19] E. Meijer and K. Claessen. The Design and Implementation of Mondrian. In *ACM SIGPLAN Haskell Workshop*. ACM Press, June 1997. Available online.

> Mondrian is a simple Haskell dialect with an object-oriented flavour. To this end, algebraic datatypes and type classes are combined into a simple object-oriented type system with no real subtyping, with completely co-variant type-checking. Such a simple OO type system can be explored in OOHaskell [10] just as much as more as other forms of OO types.

[20] M. Neubauer, P. Thiemann, M. Gasbichler, and M. Sperber. A Functional Notation for Functional Dependencies. In *Proc. 2001 ACM SIGPLAN Haskell Workshop, Firenze, Italy, September 2001*, pages 101–120, 2001. Available online.

> The paper shows an early example of type-level programming with Haskell's type classes, and it also discusses different styles such as the inherent logic programming style suggested by multi-parameter type-classes with functional dependencies and a conceivable, more functional notation. A number of by now classical examples occur in the paper.

[21] Matthias Neubauer, Peter Thiemann, Martin Gasbichler, and Michael Sperber. Functional logic overloading. In *POPL '02: Proceedings of the 29th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 233–244. ACM, 2002. Available online.

The paper shows an early example of type-level programming with Haskell's type classes, and it also proposes an extension for the support of different evaluation strategies. See [20] for a companion paper.

[22] J. Nordlander. Polymorphic subtyping in O'Haskell. *Science of Computer Programming*, 43(2–3):93–127, 2002. Also in the Proceedings of the APPSEM Workshop on Subtyping and Dependent Types in Programming, Ponte de Lima, Portugal, 2000. Available online.

The paper develops an extension of Haskell, O'Haskell, to enable a form of subtyping inspired by OO programming. Such subtyping can be simulated by OOHaskell [10] and various forms of subtyping can be explored (with regard to variance, inheritance, depth of subtyping etc.).

[23] A.T.H. Pang and M.M.T. Chakravarty. Interfacing Haskell with Object-Oriented Languages. In P.W. Trinder and G. Michaelson and R. Pena, editor, *Implementation of Functional Languages, 15th International Workshop, IFL 2003, Edinburgh, UK, September 8-11, 2003, Revised Papers*, volume 3145 of *LNCS*, pages 20–35. Springer, 2004. Available online.

The paper addresses interfacing of OO languages with Haskell. The paper uses multi-parameter type classes in this context. See [27] for a similar paper.

[24] D. Rémy. Programming Objects with ML-ART: An extension to ML with Abstract and Record Types. In M. Hagiya and J.C. Mitchell, editors, *International Symposium on Theoretical Aspects of Computer Software*, number 789 in LNCS, pages 321–346. Springer, 1994. Available online.

ML-ART provides the foundation to the design of OCaml, specifically its type system with object and subtyping capabilities. Our development of OOHaskell can be insightfully

compared with ML-ART. While the expressivity of the final language frameworks is very similar, the underlying expressivity differs in important ways. For instance, ML-ART relies on row polymorphism whereas OOHaskell leverages type functions based on multi-parameter type classes and functional dependencies.

[25] Tim Sheard and Simon Peyton Jones. Template meta-programming for haskell. *SIGPLAN Not.*, 37(12):60–75, 2002. Available online.

Template Haskell supports a form of meta-programming based on program templates, program ASTs, and compile-time computations over those entities. Such meta-programming is orthogonal to type-level programming with type classes, and the applications are largely complementary. That is, while type-level programming serves extra typing, meta-programming serves code generation and transformation.

[26] Mark Shields and Erik Meijer. Type-indexed rows. In *POPL '01: Proceedings of the 28th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 261–275. ACM, 2001. Available online.

The paper develops a type system for type-indexed types (type constructors) such as type-indexed products (relevant, for example, in XML programming), and the dual concept of type-indexed co-products. The tutorial shows (based on [11] and so far unpublished experiments) that such types can be simulated in Haskell (in a number of ways) by means of type-level programming in Haskell.

[27] Mark Shields and Simon L. Peyton Jones. Object-Oriented Style Overloading for Haskell. *ENTCS*, 59(1), 2001. Available online.

The paper describes the mapping of classes (interfaces) and subtyping hierarchies from a language like C# to Haskell. On the side of Haskell, type classes are used. The motivation for the mapping is specifically to enable a foreign-language interface—such as calling an OO library from within Haskell. A general object encoding and full subtyping for objects is not addressed. See [23] for a similar paper.

[28] Mads Torgersen. The Expression Problem Revisited. In *ECOOP 2004 - Object-Oriented Programming, 18th European Conference, Oslo, Norway, June 14-18, 2004, Proceedings*, volume 3086 of *LNCS*, pages 123–143. Springer, 2004. Available online.

The paper describes a number of solutions ("encodings") of the Expression Problem in Java or C# with generics available. It also discusses criteria for assessing solution proposals for the Expression Problem. As shown in [14], type-level programming in Haskell can also solve the Expression Problem, and it is insightful to compare the Java/C# solutions with the Haskell approach.

[29] Philip Wadler. The expression problem. Message to java-genericity electronic mailing list, November 1998. Available online at `http://www.daimi.au.dk/~madst/tool/papers/expression.txt`.

[30] Alessandro Warth, Milan Stanojević, and Todd Millstein. Statically scoped object adaptation with expanders. In *OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*, pages 37–56. ACM, 2006. Available online.

The paper addresses software extensibility in an OO context. It proposes a simple but powerful extension technique, which can also be used to solve the Expression Problem. There are arguably no elements of type-level programming in this work, but it provides a good baseline for work on extensibility problems—be it in Haskell or otherwise.

[31] Stefan Wehr, Ralf Lämmel, and Peter Thiemann. Javagi : Generalized interfaces for java. In *ECOOP 2007 - Object-Oriented Programming, 21st European Conference, Berlin, Germany, July 30 - August 3, 2007, Proceedings*, volume 4609 of *Lecture Notes in Computer Science*, pages 347–372. Springer, 2007. Available online.

Inspired by Haskell's type classes, the paper generalizes Java-like interfaces to achieve much of the type-class expressivity in an OO programming context. Type-level programming style is less of an issue in the paper, also because the approach is

limited to single-parameter type classes. The generalized interfaces are however sufficient to address the Expression Problem.