Declarative Multi-paradigm Programming

Michael Hanus

Institut für Informatik, CAU Kiel, D-24098 Kiel, Germany.
mh@informatik.uni-kiel.de

Abstract. This tutorial provides an overview and introduction to declarative programming exploiting multiple paradigms, in particular, functional, logic, and constraint programming. To demonstrate the possibility to support these paradigms within a single programming model, we survey the features of the declarative multi-paradigm language Curry.

1 Overview

Compared to traditional imperative languages, declarative programming languages provide a higher and more abstract level of programming that leads to reliable and maintainable programs. However, there is no distinct "declarative programming" paradigm. Instead, there are various programming paradigms and related languages based on different methods to structure declarative knowledge. *Functional programming* is based on the lambda calculus and provide functions as computational entities. *Logic programming* is based on first-order predicate logic and uses predicates as basic programming entities. *Constraint programming* offers constraint solvers to reason about models described with the help of various constraint structures. Although the motivation to exploit high-level programming is similar in all paradigms, the concrete languages associated to them are quite different. Thus, it is a natural idea to combine these worlds of programming into a single paradigm, and attempts for doing so have a long history. However, the interactions between functional and logic programming features are complex in detail so that the concrete design of such declarative multi-paradigm languages is a non-trivial task. This is demonstrated by a lot of research work on the semantics, operational principles, and implementation of functional logic languages since more than two decades. Fortunately, recent advances in the foundation and implementation of functional logic languages have shown reasonable principles that lead to the design of practically applicable programming languages.

This tutorial provides an overview on the principles of integrated functional logic languages. As a concrete programming language, we survey the declarative multi-paradigm language Curry[1] [13, 20]. It is developed by an international initiative of researchers in this area and intended to provide a common platform for research, teaching, and application of integrated functional logic languages.

[1] http://www.curry-language.org

Details about functional logic programming and Curry can be found in recent surveys [5, 18] and in the language report [20].

The integration of functional and logic programming has various advantages. Beyond the fact that one can use the best features of declarative languages in a single language, like strong typing, higher-order functions, optimal (lazy) evaluation from functional programming, or non-determinism, computing with partial information, and constraint solving from logic programming, there are also clear advantages compared to the individual paradigms. For instance, the combination of lazy evaluation and non-determinism leads to a demand-driven exploration of the search space which is sometimes more efficient and optimal for particular classes of programs [2]. Moreover, non-declarative features, which are regularly used in practical logic programs, can be avoided in functional logic languages, e.g., by functional notation or declarative I/O [22].

The combined features offered by functional logic languages led to new design patterns [3, 6], better abstractions for application programming (e.g., programming with databases [7, 11], GUI programming [14], web programming [15, 16, 19], string parsing [10]), and new techniques to implement programming tools, like partial evaluators [1] or test case generators [12, 21]. In particular, functional patterns, as proposed in [4], exploit non-determinism from logic programming and demand-driven pattern matching from functional programming in order to achieve a powerful executable specification method. For instance, functional patterns have been used for XML processing [17] where it has been shown that specialized logic programming approaches [8, 9] can be implemented with a few lines of code in Curry. Some of these techniques are reviewed in this tutorial.

References

1. M. Alpuente, M. Falaschi, and G. Vidal. Partial evaluation of functional logic programs. *ACM Transactions on Programming Languages and Systems*, 20(4):768–844, 1998.
2. S. Antoy, R. Echahed, and M. Hanus. A needed narrowing strategy. *Journal of the ACM*, 47(4):776–822, 2000.
3. S. Antoy and M. Hanus. Functional logic design patterns. In *Proc. of the 6th International Symposium on Functional and Logic Programming (FLOPS 2002)*, pages 67–87. Springer LNCS 2441, 2002.
4. S. Antoy and M. Hanus. Declarative programming with function patterns. In *Proceedings of the International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'05)*, pages 6–22. Springer LNCS 3901, 2005.
5. S. Antoy and M. Hanus. Functional logic programming. *Communications of the ACM*, 53(4):74–85, 2010.
6. S. Antoy and M. Hanus. New functional logic design patterns. In *Proc. of the 20th International Workshop on Functional and (Constraint) Logic Programming (WFLP 2011)*, pages 19–34. Springer LNCS 6816, 2011.
7. B. Braßel, M. Hanus, and M. Müller. High-level database programming in Curry. In *Proc. of the Tenth International Symposium on Practical Aspects of Declarative Languages (PADL'08)*, pages 316–332. Springer LNCS 4902, 2008.

8. F. Bry and S. Schaffert. A gentle introduction to Xcerpt, a rule-based query and transformation language for XML. In *Proceedings of the International Workshop on Rule Markup Languages for Business Rules on the Semantic Web (RuleML'02)*, 2002.
9. F. Bry and S. Schaffert. Towards a declarative query and transformation language for XML and semistructured data: Simulation unification. In *Proceedings of the International Conference on Logic Programming (ICLP'02)*, pages 255–270. Springer LNCS 2401, 2002.
10. R. Caballero and F.J. López-Fraguas. A functional-logic perspective of parsing. In *Proc. 4th Fuji International Symposium on Functional and Logic Programming (FLOPS'99)*, pages 85–99. Springer LNCS 1722, 1999.
11. S. Fischer. A functional logic database library. In *Proc. of the ACM SIGPLAN 2005 Workshop on Curry and Functional Logic Programming (WCFLP 2005)*, pages 54–59. ACM Press, 2005.
12. S. Fischer and H. Kuchen. Systematic generation of glass-box test cases for functional logic programs. In *Proceedings of the 9th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'07)*, pages 63–74. ACM Press, 2007.
13. M. Hanus. A unified computation model for functional and logic programming. In *Proc. of the 24th ACM Symposium on Principles of Programming Languages (Paris)*, pages 80–93, 1997.
14. M. Hanus. A functional logic programming approach to graphical user interfaces. In *International Workshop on Practical Aspects of Declarative Languages (PADL'00)*, pages 47–62. Springer LNCS 1753, 2000.
15. M. Hanus. High-level server side web scripting in Curry. In *Proc. of the Third International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, pages 76–92. Springer LNCS 1990, 2001.
16. M. Hanus. Type-oriented construction of web user interfaces. In *Proceedings of the 8th ACM SIGPLAN International Conference on Principles and Practice of Declarative Programming (PPDP'06)*, pages 27–38. ACM Press, 2006.
17. M. Hanus. Declarative processing of semistructured web data. In *Technical Communications of the 27th International Conference on Logic Programming*, volume 11, pages 198–208. Leibniz International Proceedings in Informatics (LIPIcs), 2011.
18. M. Hanus. Functional logic programming: From theory to Curry. In *Programming Logics - Essays in Memory of Harald Ganzinger*, pages 123–168. Springer LNCS 7797, 2013.
19. M. Hanus and S. Koschnicke. An ER-based framework for declarative web programming. *Theory and Practice of Logic Programming*, 14(3):269–291, 2014.
20. M. Hanus (ed.). Curry: An integrated functional logic language (vers. 0.8.3). Available at `http://www.curry-language.org`, 2012.
21. C. Runciman, M. Naylor, and F. Lindblad. Smallcheck and lazy smallcheck: automatic exhaustive testing for small values. In *Proc. of the 1st ACM SIGPLAN Symposium on Haskell*, pages 37–48. ACM Press, 2008.
22. P. Wadler. How to declare an imperative. *ACM Computing Surveys*, 29(3):240–263, 1997.