

# Modular Monadic Semantics for Aspect-Oriented Programming

Garrin Kimmell

July 15, 2004

This research examines the use of modular monadic semantics to specify the semantics of aspect oriented languages.

Aspect-oriented programming [5] is a paradigm where different facets of a system's behavior are programmed independently. Aspect languages then provide a mechanism for “weaving” the different behavior into a complete program.

A principal criticism of the paradigm is the difficulty in reasoning about the interaction of different aspects in a woven system. The aspect language is expressed in terms of the object language, eliminating the distinction between the aspect meta-language and the object language.

There have been several formulations of semantic definitions of AOP constructs. Wand [15] gave an early dynamic semantics for AOP. Lammel provided an alternative semantics based on method-call interception and superimposition. Later work extended by separated static and dynamic aspect semantics, yielding significant efficiency gains [12, 14].

Modular monadic semantics [2, 11, 13] was first introduced as a method for independently specifying denotations for various language constructs. This work focused on specification of modular interpreters; follow-on work examined the usefulness of modular monadic semantics for building and proving correctness of compilers [10, 3].

While the early work focused on independent specification of language constructs, it still coupled the language constructs by requiring them to be members of the same abstract syntax data type. A solution to this problem models algebraic data types as least fixed-points of functors. Each language element is represented as a single functor. This model allows language elements to be specified completely independently as *algebras* over a functor [4, 1, 6, 7, 8, 4]. An interpretation is then simply the folding of the collection of algebras for the language.

Our work builds on the idea of semantics as functors. In contrast to the work above, which defines an algebra as a simple function over a functor, we represent an algebra as an explicitly as a data type with function valued members [9]. Doing this allows us to dynamically update the algebras.

The use of updatable algebras facilitate the treatment of the aspect weaving process formally as a meta-programming activity. Traditional language constructs are specified as updatable algebras. Aspect language semantics are specified as *transformations* on those semantic algebras.

By using modular monadic semantics with updatable algebras, we expect to realize several benefits. Composability allows experimentation with various AOP semantic definitions. Additionally, updatable algebras provide a clear separation between the aspect meta-language semantics and the object language semantics, simplifying reasoning about a composed system. Finally, monadic semantics provides a convenient mechanism for staging compilers, allowing the separation of the dynamic and static semantics of aspect languages.

## References

- [1] Luc Duponcheel. Using catamorphisms, subtypes and monad transformers for writing modular functional interpreters.
- [2] David A. Espinosa. *Semantic Lego*. PhD thesis, Columbia University, 1995.

- [3] William L. Harrison and Samuel N. Kamin. Metacomputation-based compiler architecture. In *Mathematics of Program Construction*, pages 213–229, 2000.
- [4] Graham Hutton. Fold and unfold for program semantics. In *Proceedings of the third ACM SIGPLAN international conference on Functional programming*, pages 280–288. ACM Press, 1998.
- [5] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Longtier, and John Irwin. Aspect-oriented programming. In Mehmet Akşit and Satoshi Matsuoka, editors, *Proceedings European Conference on Object-Oriented Programming*, volume 1241, pages 220–242. Springer-Verlag, Berlin, Heidelberg, and New York, 1997.
- [6] Jose E. Labra Gayo, Juan M. Cueva L., M. Candida Luengo D., and Agustin Cernuda del Rio. Reusable monadic semantics of object oriented programming languages, June 2002.
- [7] Jose E. Labra Gayo, M. C. Luengo Díez, J. M. Cueva Lovelle, and A. Cernuda del Río. LPS: A language prototyping system using modular monadic semantics. In M. van der Brand and D. Parigot, editors, *Proceedings 1st Workshop on Language Descriptions, Tools and Applications, LDTA'01, Genova, Italy, 7 Apr 2001*, volume 44(2). Elsevier, Amsterdam, 2001.
- [8] Jose E. Labra Gayo, M. C. Luengo Díez, J. M. Cueva Lovelle, and A. Cernuda del Río. Reusable monadic semantics of logic programs with arithmetic predicates. In *Proceedings 2001 APPIA-GULP-PRODE Joint Conf. on Declarative Programming, AGP'01, Évora, Portugal, 26–28 Sept. 2001*, pages 31–45. Dept. of Informatics, Univ. of Évora, 2001.
- [9] Ralf Lammel, Joost Visser, and Jan Kort. Dealing with large bananas. In Johan Jeuring, editor, *Workshop on Generic Programming*, 2000.
- [10] Sheng Liang and Paul Hudak. Modular denotational semantics for compiler construction. In *Programming Languages and Systems – ESOP'96, Proc. 6th European Symposium on Programming, Linköping*, volume 1058, pages 219–234. Springer-Verlag, 1996.
- [11] Sheng Liang, Paul Hudak, and Mark Jones. Monad transformers and modular interpreters. In ACM, editor, *Conference record of POPL '95, 22nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages: San Francisco, California, January 22–25, 1995*, pages 333–343, New York, NY, USA, 1995. ACM Press.
- [12] Damien Sereni and Oege de Moor. Static analysis of aspects. In *Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 30–39. ACM Press, 2003.
- [13] Guy L. Steele, Jr. Building interpreters by composing monads. In ACM, editor, *Conference record of POPL '94, 21st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages: Portland, Oregon, January 17–21, 1994*, pages 472–492, New York, NY, USA, 1994. ACM Press.
- [14] David B. Tucker and Shriram Krishnamurthi. Pointcuts and advice in higher-order languages. In *Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 158–167. ACM Press, 2003.
- [15] Mitchell Wand, Gregor Kiczales, and Chris Dutchyn. A semantics for advice and dynamic join points in aspect-oriented programming. In Gary T. Leavens and Ron Cytron, editors, *FOAL 2002 Proceedings: Foundations of Aspect-Oriented Languages Workshop at AOSD 2002*, number 02-06 in Technical Report, pages 1–8. Department of Computer Science, Iowa State University, April 2002.