Equivariant Syntax and Semantics (Abstract of Invited Talk) *

Andrew M. Pitts

University of Cambridge Computer Laboratory, Cambridge CB3 0FD, UK
Andrew.Pitts@cl.cam.ac.uk

Abstract. The notion of *symmetry* in mathematical structures is a powerful tool in many branches of mathematics. The talk presents an application of this notion to programming language theory.

Algebraic Syntax and Semantics

Since the 1970s at least, universal algebra has played an important role in programming language theory (see [17] for a survey). For example, from the point of view of 'algebraic semantics', a programming language is specified by a signature of sorts and typed function symbols. Then its abstract syntax, i.e. the set of well-formed parse trees of the language, is given by the *initial algebra* for this signature; and a denotational semantics for the language is given by the unique homomorphism of algebras from the initial algebra to some algebra of meanings. This algebraic viewpoint has useful computational consequences that one sees directly in the development of term-rewriting systems [1] and in the notion of user-declared datatype occurring in functional programming languages such as ML [19] or Haskell [22]. The initial algebra property also has useful logical consequences, since it gives rise to principles of structural recursion and induction that are fundamental for proving properties of the programming language. The fact that these principles can be automatically generated from the language's signature has facilitated incorporating them into general-purpose systems for machine-assisted reasoning (cf. the inductive sets package of the HOL system [12], or Coq's inductive types [4]).

In fact if one dips into the 1985 volume edited by Nivat and Reynolds [20] representing the state of the art of algebraic semantics when it graduated from the novel to the routine, one sees that this attractively simple framework is only adequate for rather simple languages. For example, to see denotational semantics for languages containing recursive features as just algebra homomorphisms, one must mix universal algebra with *domain theory* [13] and consider the (interesting!) complications of 'continuous algebras'. And it is only recently, with the work of Plotkin and Turi reported in [26], that (certain kinds of) *structural operational semantics* [25] have been fitted convincingly into the algebraic framework.

* Research funded by UK EPSRC grant GR/R07615 and Microsoft Research Ltd.

Getting in a Bind

In my opinion, the biggest defect of the algebraic approach to syntax and semantics in its traditional form is that it cannot cope convincingly with programming language constructs that involve *binding*. By the latter I mean constructs where the names of certain entities (such as program variables, or function parameters) can be changed, consistently and subject to freshness constraints within a certain textual scope, without changing the meaning of the program. Such 'statically scoped' binding constructs are very common. Any semantics of them has to identify programs that only differ up to bound names—α-*equivalent* programs, as one says. Furthermore, many transformations on the syntax of such programs (such as ones involving capture-avoiding substitution) are only really meaningful up to α-equivalence. Therefore one would really prefer to work with a representation of syntax that abstracts away from differences caused by named variants. In the conventional algebraic approach one has to define a suitable notion of α-equivalence language-by-language, and then quotient the algebra of parse trees by it. In doing so, one looses the initial algebra property and hence looses automatically generated principles of structural recursion/induction that apply directly to parse trees modulo α-equivalence.

Because issues to do with binders and α-equivalence are so prevalent, tiresome and easy to get wrong, it is highly desirable to have a mathematical foundation for syntax involving binders that enables one to generalise the initial algebra view of syntax from parse trees to parse trees modulo α-equivalence. Ever since Church [3], formal systems for *total functions*, i.e. various forms of typed λ-calculus, have been used to model variable-binding operations; see [18] for an excellent recent survey of this so-called *higher-order abstract syntax* and related notions. However, this approach does not provide the simple generalisation of the initial algebra viewpoint one would like. True, it is possible to develop structural induction/recursion principles for higher-order abstract syntax (see [6], [14] and [15] for example), but the logical niceties are rather subtle, involving one or more layers of carefully constrained metalanguage in which syntax representation, and reasoning about it, have to take place. It reminds me a bit of the situation for non-standard analysis, whose logical subtleties hindered widespread take-up by 'users'.

A Fresh Approach

In fact it is possible to have a mathematical foundation for syntax involving binders which directly generalises the classical initial algebra view of syntax from parse trees to parse trees modulo α-equivalence. The papers by Fiore, Plotkin and Turi [7] and Gabbay and myself [10, 11] present two different solutions within the general framework of *(pre)sheaf toposes* [16] (although the connection with sheaf theory is not to the fore in the second work, which chooses to present a set-theoretical version of its model). The work by Fiore et al provides a very nice, syntax-independent and algebraic treatment of the notion of *nameless*, or *de Bruijn terms* [5]; whereas that by Gabbay and myself hinges upon

a syntax-independent characterisation of *freshness* of bound names in terms of permutative renaming. Both approaches can characterise parse trees modulo α-equivalence as initial algebras. But as for the work on higher-order abstract syntax mentioned above, workers in 'applied semantics' will ask to what extent the logical (and in this case category- and set-theoretic) subtleties hinder the take-up of these ideas in, say, systems for machine-assisted reasoning about syntax and semantics, or in logic- and functional-programming languages for metaprogramming?

In the talk I will try to give a positive answer to this question. I will show that the model of abstract syntax with binders in [10, 11] can be formulated as quite a simple and (I hope) appealing generalisation of the classical initial algebra treatment of syntax without binders; and that this generalisation has good logical and computational properties. The presentation is based on the notion of *nominal set* from [23]. The whole approach stems from the somewhat surprising observation that all of the concepts we need (α-equivalence, freshness, name-binding, ...) can be—and, I would claim, should be—defined purely in terms of the operation of *swapping* pairs of names. Thus the mathematical framework takes into account certain symmetries on objects induced by name-swapping; and the the notion *equivariance*, i.e. of invariance for these symmetries, plays a prominent role in the theory.

The talk will introduce nominal sets and their use for modelling abstract syntax with binders. I will survey some of the logical and computational consequences of this 'equivariant syntax and semantics' that have been produced so far: Gabbay's packages for FM-set theory and higher order logic [8, 9] in Isabelle [21]; a functional programming language incorporating our notion of binding and freshness [24] currently being implemented by Shinwell; a 'nominal' version of first-order equational logic, unification and term-rewriting (work in progress by Gabbay, Urban and myself); domain theory in nominal sets (work in progress by Shinwell); the use of name swapping and quantification over fresh names in a spatial logic for concurrent processes by Caires and Cardelli [2].

References

[1] F. Baader and T. Nipkow. *Term Rewriting and All That.* Cambridge University Press, 1998.

[2] L. Caires and L. Cardelli. A spatial logic for concurrency (part I). In N. Kobayashi and B. C. Pierce, editors, *Theoretical Aspects of Computer Software, 4th International Symposium, TACS 2001, Sendai, Japan, October 29-31, 2001, Proceedings*, volume 2215 of *Lecture Notes in Computer Science*, pages 1–38. Springer-Verlag, Berlin, 2001.

[3] A. Church. A formulation of the simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.

[4] The Coq proof assistant. Institut National de Recherche en Informatique et en Automatique, France, ⟨`coq.inria.fr`⟩.

[5] N. G. de Bruijn. Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem. *Indag. Math.*, 34:381–392, 1972.

[6] J. Despeyroux, F. Pfenning, and C. Schürmann. Primitive recursion for higher-order abstract syntax. In *Typed Lambda Calculus and Applications, 3rd International Conference*, volume 1210 of *Lecture Notes in Computer Science*, pages 147–163. Springer-Verlag, Berlin, 1997.

[7] M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In *14th Annual Symposium on Logic in Computer Science*, pages 193–202. IEEE Computer Society Press, Washington, 1999.

[8] M. J. Gabbay. *A Theory of Inductive Definitions with α-Equivalence: Semantics, Implementation, Programming Language*. PhD thesis, University of Cambridge, 2000.

[9] M. J. Gabbay. FM-HOL, a higher-order theory of names. In *Thirty Five years of Automath, Heriot-Watt University, Edinburgh*. Informal proceedings, 2002.

[10] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224. IEEE Computer Society Press, Washington, 1999.

[11] M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax with variable binding. *Formal Aspects of Computing*. Special issue in honour of Rod Burstall. To appear.

[12] M. J. C. Gordon and T. F. Melham. *Introduction to HOL. A theorem proving environment for higher order logic*. Cambridge University Press, 1993.

[13] C. A. Gunter and D. S. Scott. Semantic domains. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B, pages 633–674. North-Holland, 1990.

[14] M. Hofmann. Semantical analysis of higher-order abstract syntax. In *14th Annual Symposium on Logic in Computer Science*, pages 204–213. IEEE Computer Society Press, Washington, 1999.

[15] F. Honsell, M. Miculan, and I. Scagnetto. An Axiomatic Approach to Metareasoning on Nominal Algebras in HOAS. In *28th International Colloquium on Automata, Languages and Programming, ICALP 2001, Crete, Greece, July 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 963–978. Springer-Verlag, Heidelberg, 2001.

[16] S. MacLane and I. Moerdijk. *Sheaves in Geometry and Logic. A First Introduction to Topos Theory*. Springer-Verlag, New York, 1992.

[17] K. Meinke and J. V. Tucker. Universal algebra. In S. Abramsky, D. M. Gabbay, and T. S. E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1, pages 189–411. Oxford University Press, 1992.

[18] D. Miller. Abstract syntax for variable binders: An overview. In John Lloyd et al, editors, *Computational Logic - CL 2000 First International Conference London, UK, July 24-28, 2000 Proceedings*, volume 1861 of *Lecture Notes in Artificial Intelligence*, pages 239–253. Springer-Verlag, 2000.

[19] R. Milner, M. Tofte, R. Harper, and D. MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.

[20] M. Nivat and J. C. Reynolds, editors. *Algebraic Methods in Semantics*. Cambridge University Press, 1985.

[21] L. C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, 1994.

[22] S. L. Peyton Jones et al, editors. *Report on the Programming Language Haskell 98. A Non-strict Purely Functional Language*. February 1999. Available from `<www.haskell.org>`.

[23] A. M. Pitts. Nominal logic, a first order theory of names and binding. Submitted for publication (a preliminary version appeared in the *Proceedings of the 4th International Symposium on Theoretical Aspects of Computer Software* (TACS 2001), LNCS 2215, Springer-Verlag, 2001, pp 219–242), March 2002.

[24] A. M. Pitts and M. J. Gabbay. A metalanguage for programming with bound names modulo renaming. In R. Backhouse and J. N. Oliveira, editors, *Mathematics of Program Construction. 5th International Conference, MPC2000, Ponte de Lima, Portugal, July 2000. Proceedings*, volume 1837 of *Lecture Notes in Computer Science*, pages 230–255. Springer-Verlag, Heidelberg, 2000.

[25] G. D. Plotkin. A structural approach to operational semantics. Technical Report DAIMI FN-19, Aarhus University, 1981.

[26] D. Turi and G. D. Plotkin. Towards a mathematical operational semantics. In *12th Annual Symposium on Logic in Computer Science*, pages 280–291. IEEE Computer Society Press, Washington, 1997.