Position Paper: Constructing Abstract Algebraic Models for Object-Oriented Languages with Aliasing

Phillip M. Yelland,
Sun Microsystems Laboratories,
901 San Antonio Rd, MS UMTV29-117,
Palo Alto, CA 94303,
USA.
phillip.yelland@sun.com

Statement

In this submission, I'd like to take the opportunity presented by the workshop to reprise some of the material that originated in my Ph.D. work [5]. Though almost eight years old, the results of this research are still fairly obscure — an upshot of my own indolence (the thesis itself was never published) and of a general indifference that (with a few honorable exceptions) prevailed at the time of original writing.

I began the research with the intention of devising models for object-oriented languages based on the algebraic semantics popularized by the "ADJ group" in the late '70's and early '80's [1]. Since algebraic semantics was devised with the intent of supporting specification languages and formal verification, my hope was that constructing such models for object-oriented languages would legitimate the application of Hoare-style datatype verification methods to OOL's. (You might recall that a number of researchers in OOL's — particularly those associated with Eiffel — advocated the use of such methods [3].)

Unfortunately, in attempting to come up with such models, I had terrible problems dealing with the subject of this workshop — the chains of reference intrinsic to conventional object-oriented programming languages. Many of my early essays collapsed into complex jumbles, with a profusion of low-level details needed to cope with references. These models only sanctioned proof methods so complicated and turgid as to be almost useless.

Eventually, I realized I needed "abstract" denotational models of OOL's. Such models would ascribe a representation to a collection of objects (or equivalently, classes) bereft of all internal details that were inaccessible to objects outside of that collection. (As you'd no doubt suspect, this notion of abstractness is a weakening of Plotkin's definition of "full abstraction" [4].) Abstract algebraic models yield proof methods that are "the best one can do", in as far as they address only the complexities that arise from the characteristics of the subject language itself, without introducing extraneous complications.

I began by devising an abstract model for the language that Hoare used in presenting his seminal work on data type verification [2]. Though it's apparently not widely appreciated, Hoare very wisely excluded any possibility of aliasing involving objects in his language (all assignment and parameter-passing is accomplished by copying). Naturally enough, the abstract model gives rise to a proof method that coincides precisely with that Hoare proposed.

I found that once aliasing is introduced, however, abstract models (and their associated proof methods) become considerably more convoluted. Problems arise not only from the updating of shared objects described in the workshop rubric, but from the cycles of reference all to easy to contrive in conventional object-oriented languages. Cycles, for example, allow an object to be observed in an inconsistent state, if an operation invokes another operation on a subsidiary object that has a reference back to the original. Even worse, a cycle can lead to the non-termination of a program in an OOL in a way that's not at all evident from an inspection of the code.

The results of my thesis work serve to motivate the construction of mechanisms for controlling aliasing in object-oriented languages. For they establish that without such mechanisms, formal (or indeed informal but rigorous) verification methods must deal with significant complexity that originates not in the formal notation or proof methods used but in the programming languages themselves. Hoare's elegant (if questionably practical) presentation shows that by reigning in the effects of aliasing, we can produce object-oriented programming systems which are significantly easier to reason about and which therefore comprise much more propitious bases for the construction of reliable software.

References

[1] Goguen, J., Thatcher, J., Wagner, E., An initial algebra approach to the specification, correctness and implementation of abstract datatypes. In Yeh, R. (ed), *Current Trends in Programming Methodology (Vol. IV: Data Structuring)*, Prentice-Hall, 1978.

[2] Hoare, C. A. R., Proof of correctness of data representations. *Acta Informatica*, 1, 1972.

[3] Meyer, B., *Object-Oriented Software Construction*. Prentice-Hall, 1988.

[4] Plotkin, G., LCF considered as a programming language. *Theoretical Computer Science*, 5, 1977.

[5] Yelland, P., *Models of Modularity: A Study of Object-Oriented Programming*. Ph.D. Dissertation, Cambridge University, Cambridge, U.K., 1991.