Algool – Object-Orientation with Algebraic Specification

Anya Helene Bagge
Department of Informatics, University of Bergen, Norway
anya@ii.uib.no

ACM Reference format:
Anya Helene Bagge. 2017. Algool – Object-Orientation with Algebraic Specification. In *Proceedings of the -2th Workshop on New Object-Oriented Languages, Vancouver, Canada, October 2017 (NOOL'17),* 1 pages.
https://doi.org/10.475/123_4

Abstract: This extended abstract discusses a few challenges and opportunities in combining object-orientation with program specification, with a brief look at specification techniques and a highly experimental OO language, Algool.

Informal formality: Although formal specification of software is primarily used for *high integrity* use-cases, some technique have found "informal" uses in more mainstream software development – e.g., precondition checks and property-based testing.

Knowledge is power, also when programming: the more your tools know about the (intended) meaning of your program, the more help they can give you. Formal verification is an extreme; more mundane examples include automated testing (e.g., QuickCheck), bug checking (e.g, static type checking; nullness checking; FindBugs), optimisation (e.g., algebraic simplification), security analysis (e.g., taintedness analysis, for instance), as well as refactoring and automated transformation.

Specification Techniques: *Design by contract* with pre- and post-conditions are commonly used, and built into Eiffel, for instance (also available as extensions for other languages). However, for modular software with generics and interfaces, *algebraic specification* (specifying relationships between operations) is more expressive and flexible [1] – although some adaptions must be made to deal with updatable object state.

Algool: Algool is our highly experimental platform for object-oriented programming with integrated (primarily algebraic) specification. In Algool, types are interfaces, with objects constructed from classes impementing one or more interfaces. The only slight novelty here is that the interfaces describe not only the syntax of methods (signatures), but also include algebraic *properties* that specify behaviour. More complex specifications can be built from simpler parts by composing interfaces.

Let's have a look at two particular challenges when combining OO and algebraic specification:

State change: Object updates are problematic in a setting with references and aliasing, since even if we can determine that change is happening, we may not be able to reason statically about which objects are changed (e.g., do *a* and *b* have a reference to a shared object, so that a change to *a* causes a change to *b*?).

For Algool, we require the programmer to declare which objects are changed (indicated with an exclamation mark on both declaration and call site, e.g., "pos.move!(dx,dy)"), and we restrict references and aliasing. A full *ownership type system* [2] can be used to resolve this issue; for now, we rely on restrictions, plus a system of *handles* that can be used together with containers to update a contained object.

Dynamic Dispatch: Dynamic dispatch showcases an interesting opportunity and challenge: Some things are statically known about the method call (the object's type, the declaration of the method), but some things are not known (the particular class of the object at run-time, the definition of the method). In Algool, we know more than we'd know in e.g., Java, since Algool interfaces also specify behaviour. A Java programmer *may* make reasonable assumptions about what happens, based on documentation and the assumption that the substitution principle applies – an Algool programmer will know that any object implementing a particular interface will have methods with at least the specified properties.

However, at runtime we may know *more* properties. For example, for a *Collection*, the *add* method gives no guarantee about the order of objects, but if the collection is a *SortedList* we'd known that the list is still sorted after *add*. Thus, to make full use of our specifications, we'll need reasoning tools also at runtime (perhaps also with just-in-time code transformations).

Status: Algool is still experimental, both in terms of design and implementation.

References

[1] A.H. Bagge and M. Haveraaen. 2014. Specification of Generic APIs, or Why Algebraic May Be Better than Pre/Post. In *HILT'14*. ACM, New York, NY, USA. DOI: http://dx.doi.org/10.1145/2663171.2663183

[2] D.G. Clarke, J.M. Potter, and J. Noble. 1998. Ownership Types for Flexible Alias Protection. In *OOPSLA '98 (OOPSLA '98)*. ACM, New York, NY, USA, 48–64. DOI: http://dx.doi.org/10.1145/286936.286947

NOOL'17, October 2017, Vancouver, Canada
2017. ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4