Algebraic Specification of Objects

Our methodology for the abstract specification of (classes of) objects parallels traditional algebraic ADT specifications in many ways. We believe that our approach clarifies the essential difference between data abstraction and object abstraction. It stresses the connection of the behavior of objects with sequences of messages while maintaining the essence of state as an overt but completely abstract entity. While we abstract away from details about state, we do not regard it as a hidden sort as in [GM87] — the presence or absence of state is not transparent. Effective integration of the state concept into formal specification enhances an intuitively natural match of the abstraction with implementation details. The idea of viewing state as a hidden argument captures intuition better since its connection with the visible operations is concealed but not completely severed.

The approach we suggest begins with categorizing the methods according to their dependence and effect on the internal state of an object and making those categories an explicit part of the specification. Specifications of the state transition functions are associated with the side-effect inducing methods and provide an abstract description of each of them and the states. Once the state behavior is established, descriptions of the state-dependent methods can be pursued. Because of the role of states in overall behavior, our specifications are intended to distinguish objects only if they have observable behavior which differs. Thus we prefer the final algebra interpretation [GH78, Kam83, Wan79] over the initial algebra view [GTW78]. Of course, the state-independent methods constitute an ordinary ADT and can be specified by well established means.

We wish to describe object behavior and hence seek to avoid differentiation of objects based on their internal structure. Hence two states are equivalent (indistinguishable) provided that if we start with two objects, one in each of these states, then for every sequence of messages ending with a selector message, both objects return the same result — that is, the two objects are indistinguishable by any external means. In the ADT literature this is known as the “final algebra” view.

Since our approach results in an ADT-style description of a collection of functions, ADT analysis concepts generally apply. For instance, “sufficient completeness” for terms whose primary operation is a selector remains a property of interest. The primary difference to be accounted for is with the State domain which is represented by sequences of side-effect inducing messages. For the model of an AOC specification we take an “abstract machine” view [MG85, GM87]. The states are the indistinguishability classes of the state representations occurring in our specification, with transitions given by the collection of next-state functions. Any implementation equivalent to the minimal state machine is regarded as acceptable.

Example — Histogram Objects.
This example is a familiar device — the histogram, a commonly used aggregation of values that may be inspected, graphed, etc. For illustration we assume methods of the class with informal descriptions as follows:
empty — the initial, empty histogram,
tally(v) — add value v to the receiving histogram,
high — return the greatest value in the histogram,
low — return the least value in the histogram,
average — return the average of the values in the histogram,
sampleSize — return the number of observations in the histogram,
frequency(v) — return the number of times value v appears in the histogram.

class Histogram
Signatures — state-dependent operations — visible operations have State as a hidden argument
tally: Value □ Histogram
•$□_{\text{tally}}$: State, Value □ State (hidden function)
high: □□ Value
low: □□ Value
average: □□ Value
sampleSize: □□ Natural
frequency: Value □ Natural

State-independent operations
empty: □□ Histogram

The results of sending messages to a Histogram object are specified by the collection of equations below. For the 'tally' message we want the result to be the receiving object in its new state. This is implicitly indicated by providing no equation for the 'tally' operation itself and interpreting this omission as the signal for this common option. The relevant equation is that provided for the associated state transition function. Equations use conditional expressions as is typical in ADT descriptions. In addition to the TOI (Histogram), there are two pre-defined classes (ADTs actually) assumed in this specification. Natural is thought of as the familiar natural numbers, and Value is assumed to provide a domain with suitable numeric operations. Details of these classes are omitted.

Equations — for each s□State and v,w□Value
$□_{\text{tally}}$(s,v) = s^'tally'(v)
high(s^'tally'(v)) = if s=empty then v else max(v, high(s))
low(s^'tally'(v)) = if s ∫ 'empty' then v else min(v, low(s))
sampleSize(empty) = 0
sampleSize(s^'tally'(v)) = if s=empty then 1 else 1 + sampleSize(s)
frequency('empty', w) = 0
frequency(s^'tally'(v), w) = if v = w then 1 + frequency(s, w) else frequency(s, w)
average(s^'tally'(v)) = if s = 'empty' then v
                                        else (v + sampleSize(s)*average(s)) / (1 + sampleSize(s))

Note that the state of a Histogram object is assumed to be determined by the sequence of all the side-effect inducing messages it has received since its creation. This message history representation of states consists of a sequence of message identifiers and their arguments (including their state information). Comparisons of states written as "∫" denote state equivalence. Actually the above equations are the "ok-equations". For brevity the error cases such as sending the 'high' message to the 'empty' Histogram are omitted.

References
[BLM91] A. Brogi, E. Lamma & P. Mello, "Objects in a logic programming framework", First Russian Conf. on Logic Programming A. Voronkov (ed.), Springer-Verlag, Lect. Notes in AI, V. 592, 1991, 102-113.

[Bre91] R. Breu, Algebraic Specification Techniques in Object Oriented Programming Environments, Springer-Verlag, Lect. Notes in Comp. Sci. V. 562, 1991, 228 pp.

[Bud91] T. Budd, An Introduction to Object-Oriented Programming, Addison-Wesley, 1991, 399 pp.

[Coo90] W. R. Cook, “Object-oriented programming versus abstract data types”, Foundations of Object-Oriented Languages (J. W. de Bakker, W. P. de Roever & G. Rozenberg, eds.), Springer-Verlag, Lect. Notes in Comp. Sci. V. 489, 1990, 151-178.

[DKR91] R. Duke, P. King, G. Rose & G. Smith, “The Object-Z specification language, version 1”, Tech Rpt. 91-1, Software Verification Research Centre, Univ. of Queensland, Australia, May 1991, 61 pp.

[EGS92] H. Ehrig, M. Gogolla & A. Sernadas, “Objects and their specification”, Eighth Workshop on Abstract Data Types (M. Bidoit & C. Choppy, eds.), Springer-Verlag, Lect. Notes in Comp. Sci., V. 655, 1992, 40-66.

[EM85] H. Ehrig & B. Mahr, Fundamentals of Algebraic Specification 1, Springer-Verlag, 1985, 321 pp.

[Fle95] A. C. Fleck, “Algebraic specifications of objects”, Math. Modelling & Sci. Comput. 6 (1996), also presented at Tenth Inter. Conf. on Mathematical Modelling and Scientific Computing, Boston MA, 1995.

[Gog78] J. A. Goguen, “Abstract errors for abstract data types”, Formal Descriptions of Programming Concepts (E. J. Neuhold, ed.), North-Holland, 1978, 491-522.

[GM82] J. A. Goguen & J. Meseguer, “Universal realization, persistent interconnection and implementation of abstract modules”, Proc. 9th Inter. Conf. on Automata, Languages and Programming (A. Poigne' & D. Rydeheard, eds.), Springer-Verlag, Lect. Notes in Comp. Sci., V. 140, 1982, 313-333.

[GM87] J. A. Goguen & J. Meseguer, "Unifying functional, object-oriented and relational programming", Research Directions in Object-Oriented Programming (B. Shriver & P. Wegner, eds.), MIT Press, 1987, 417-477.

[GM92] J. A. Goguen & J. Meseguer, “Order-sorted algebra I: equational deduction for multiple inheritance, overloading, exceptions and partial operations”, Theor. Comput. Science 105,2(1992), 217-274.

[GTW78] J. A. Goguen, J. W. Thatcher & E. G. Wagner, “An initial algebra approach to the specification, correctness, and implementation of abstract data types”, Current Trends in Programming Methodology, Vol. IV: Data Structuring (R. T. Yeh, ed.), Prentice-Hall, 1978, 80-149.

[GR89] A. Goldberg & D. Robson, Smalltalk 80: the language, Addison-Wesley, 1989, 585 pp.

[GH78] J. V. Guttag & J. J. Horning, “The algebraic specification of abstract data types”, Acta Informatica 10(1978), 27-52.

[Kam83] S. Kamin, “Final data types and their specification”, ACM Trans. Prog. Lang. & Sys. 5,1(1983), 97-123.

[MG85] J. Meseguer & J. A. Goguen, “Initiality, induction, and computability”, in Algebraic Methods in Semantics (M. Nivat & J. C. Reynolds, eds.), Cambridge University Press, 1985, 458-541.

[Mey88] B. Meyer, Object-oriented Software Construction, Prentice Hall, 1988, 534 pp.

[SA89] G. Smolka & H. Ait-Kaci, “Inheritance hierarchies: semantics and unification”, J. Symbolic Computation 7(1989), 343-370.

[SB86] M. Stefik & D. G. Bobrow, “Object-oriented programming: themes and variations”, The AI Magazine, 1986, 40-62.

[Wan79] M. Wand, “Final algebra semantics and data type extensions”, J. Comput. & Sys. Sci. 19,1(1979), 27-44.

[Weg90] P. Wegner, “Concepts and paradigms of object-oriented programming”, OOPS Messenger 1,1(Aug. 1990), 7-87.

[Wie91] R. J. Wieringa, “A formalization of objects using equational dynamic logic”, DOOD'91 (C. Delobel, M. Kifer & Y. Masunga, eds.), Springer-Verlag, Lect. Notes in Comp. Sci., V. 566, 1991, 431-452.