Theorising Data: A History of Abstract Data Types and their Specification

John V Tucker
Department of Computer Science
Swansea University

In the past decade *data* has transformed from an invaluable working term, used in computing and in the sciences,[1] to become a concept whose meaning, scope and significance is expanding and changing many other human domains and enterprises – thanks, not least, to the ubiquitous digital data we generate in everyday life being tractable. The software that holds together modern life collects data intentionally and unintentionally. Software is ideal for monitoring the behavior of people and objects. The appetite for data in the sciences, commercial and public services, and government raises diverse and challenging questions – technical, economic, political, legal, social, and personal – that require diverse theoretical foundations. Data belongs to distinct realms of human activity. Theorising data is work for many disciplines and minds.[2]

In Computer Science, questions and speculations about data are not new.[3] In theorising data we could start by considering what data means in programming and software development. Computer Science has answers to the fundamental questions: *What are data? What are digital data?* The answers are contained in the user-centred programing concept of *abstract data type*. Today, abstract data types are easy to understand and teach; they can be formally modelled using many sorted abstract algebras; they also possess a deep mathematical theory, elegant programming constructs, and useable formal methods for large scale software development.

In this paper I describe the origins of the modern conception and algebraic theory of abstract data types. The theory is based on specifications that axiomatise the operations on data using *equations*, and are equipped with the operational semantics of *term rewriting*; the *many sorted algebras* that satisfy the equations are models of possible implementations of the specifications. The development of the concept and theory is complicated as it blends independent research programmes in (i) programming languages; (ii) programming methodology, and (iii) logical and algebraic semantic methods. The period I have chosen is 1966 to 1985. I begin with Aad van Wijngaarden (1916-1987) on axiomatising numerical data types; I end with the complete classification theory of abstract data types by Jan Bergstra and myself using computability theory, and with the spread of algebraic methods. I will reflect on the technological and scientific motivations and interplay between some of the actors of research programmes (i)-(iii).

---

[1] By sciences I mean physical, biological and social sciences where empirical data are fundamental.

[2] We recognise the intuitive idea of data to be present wherever there are measurements and computation (e.g., in ancient astronomy and accounting). The term data appears in the 17th Century. The present transformation is timelined in https://www.forbes.com/sites/gilpress/2013/05/28/a-very-short-history-of-data-science.

[3] For example, theoretical concerns about natural languages, and data *versus* information *versus* knowledge, arise in thinking about AI in the 1940s.

Programming data types. I will consider the programming methodology literature, as represented by IFIP WG 2.3. E W Dijkstra's (1930-2002) interest in specification started with his writing programs from specifications of un-built machines at the Mathematical Centre, Amsterdam (now CWI) in 1950s. The emphasis on structured programming, specification, and reasoning was influential in making software development a subject for theoretical analysis, one that looked to logic and algebra for ideas and techniques. The treatment of data in C A R Hoare's axiomatic approach to programming language definition of 1969, and data refinement of 1972, helped untie data types from code. David Parnas' work on software engineering in Philips Apeldoorn in 1971 led to his ideas on information hiding, the fundamental role of interfaces, and documentation, which ultimately freed data from implementation via notions of *module*.

An important milestone is Stephen Zilles' independent formal study of data types in 1974-77; he knew about axioms and presentations and the new Birkoff-Lipson paper on heterogenous (= many sorted) universal algebras of 1970. Combined with Barbara Liskov's introduction of modules as a collection of procedures with information hiding, in Venus (1972) and the CLU language (1976), a programmer's way of completely modelling user-centred data types became available. At the same time, there was a study of the representation independent specification of data in J V Guttag's PhD Thesis in 1975. These developments were inspiring to designers of new programming and specification languages; for example, Bjarne Stroustrup's experiments with abstract data types for C (1982-4) which led to C++.

Mathematical theory of data types. But the theory of abstract data types took its mathematical form through the work of the ADJ Group: Jim Thatcher, Eric Wagner, Jesse B Wright and Calvin Elgot at IBM Yorktown Heights, and Joseph Goguen (1941-2006), who wrote many basic papers on initial algebras, equational specifications, parameteristion, errors, etc., starting 1975. Their work on data was a component of a larger scale approach to semantics based upon category theory.[4]

The ADJ work was mathematically clear, rigorous and correct (unlike much other theory). It was the basis for Jan Bergstra and I to work on classifying the scope and limits of algebraic specifications (based on different forms of equation, hidden functions and sorts, initial and final semantics and complete term rewriting systems). The work was scientific, resonating with Birkhoff's 1936 papers on varieties, and Mal'cev's theory of computable, semi-computable and cosemi-computable algebras which could characterize digital data.

Conclusion. The 20 year period 1966-85 was hugely productive for programming. Abstract methods emerged that met the needs of users, and of software developers (e.g., portability and maintenance). For data there emerged a theory that worked: it could model the data in any user's world, and so be of long term technological and scientific use.

[4] The roots of which can be found partially in the early work of Peter Landin on general questions about programming languages; and was notably developed by many (e.g., Rod Burstall and Joe Goguen with their idea of *institutions* of the late 1970s).