Algebraic Specification goes multimedia – A few tentative steps*

Renate Klempien-Hinrichs and Hans-Jörg Kreowski

Department of Computer Science, University of Bremen
P.O. Box 33 04 40, D-28334 Bremen (Germany)
Email: {rena, kreo}@informatik.uni-bremen.de

Algebraic specification is the subject of one of the courses within the computer science studies at the University of Bremen. The course will start on October 15, 2001 and run for 15 weeks at four hours per week until February, 2002. It will be read by the second author and assisted by the first. 40 to 60 participants are expected.

Background

Algebraic specification is a graduate course in the area of theoretical computer science. In the undergraduate phase, every computer science student must attend and pass two courses on theoretical computer science that introduce fundamentals of computability, complexity, formal languages, and automata. In the graduate phase, at least one course and an oral examination must be passed. Subject of the exam is one of the subareas of theoretical computer science including computability and complexity theory, formal language theory, theory of programming, and special subareas like theory of concurrency and theory of artificial intelligence. In practice, the exam focusses usually on the contents of one of the courses in a subarea. The additional course must belong to a different subarea. The students can choose from a wide spectrum of subjects. As a rule, at least one course is offered in each subarea per academic year.

Algebraic specification belongs to the theory of programming and is offered about every second year. The second author read algebraic specification for the first time in 1979 and about a dozen times since then. From the very beginning, the content was based on the seminal survey paper of the ADJ group (see Goguen, Thatcher, and Wagner [GTW78]) and has not changed very much over the years. Also from the beginning, there were class room notes consisting mainly of a German translation of the ADJ survey, supplemented by more examples and explanations. Meanwhile, one encounters a good number of textbooks on algebraic specification (see, e.g., [Kla83, EM85, EGL89, HKB89, HL89, LEW96]). Each has its own merits, but in the first chapters they coincide very

* This activity is partially supported by the *Bundesministerium für Bildung und Forschung* (BMBF) within the project MMISS (Multimedia Instruction in Safe Systems).

much with the ADJ survey so that there is no need to change the content of the introductory course.

But this time, there will be a major difference. While reading algebraic specification, we will think about and try to provide some multimedia support. The aim is not to come up with a final conception, but we will introduce some first tentative ideas, see how they work, and discuss them with the participating students.

Contents

A good part of computer science concerns data processing. How can it be done systematically, efficiently, correctly? What can and should be done?

As the term "data processing" indicates, the main ingredients are on the one hand data of various sorts, from simple ones like bits, digits, and letters to complex structured ones like sequences, trees, networks, etc., and on the other hand means to operate on them. Data must be generated, changed, consumed, analyzed, compared, etc. In other words, the entities of interest in computer science, like algorithms, data types, and whole data-processing systems, are known to be algebras and models in mathematics. This observation motivates exploiting the theory of universal algebra (or, likewise, mathematical logic) and looking for syntactic and semantic concepts for a better support and understanding of computing and programming. Initiated by researchers like Guttag and Horning, Liskov and Zilles and the ADJ-group in the 70s, this line of research led to the flourishing area of algebraic specification (see, e.g., the state-of-the-art report [AKK99] for an up-to-date and quite complete survey).

A first syntactic feature is the *signature*, allowing to declare sorts and operations, with domain and codomain sorts for each operation. Semantically, one may associate a data domain with each sort and a (functional) operation on the respective domains with each declared operation. This defines a *SIG-algebra* if *SIG* is the signature at hand. Moreover, *SIG* gives rise to a recursive generation of *SIG-terms* as (formal) applications of declared operations to *SIG*-terms and sorted *variables*, chosen as one likes. Given a *SIG*-algebra A, a term can be *evaluated* in A, yielding a unique value, if each operation is replaced by the corresponding operation in A and each variable is assigned a value. It turns out that the term generation process leads to a *SIG-term algebra* $T_{SIG}(X)$ for each choice of variables X and that the evaluation is the only homomorphism from $T_{SIG}(X)$ into A for each assignment of values to X. The latter means that $T_{SIG}(X)$ is the *free* *SIG*-algebra over X and that $T_{SIG} = T_{SIG}(\emptyset)$, the term algebra without variables, is the *initial* *SIG*-algebra.

T_{SIG} is a proper candidate to formalize the idea of abstract data types on the level of signatures. The principle behind T_{SIG} is that all data are accessible by repeated application of operations, and that such applications yield different values if the applied operations are different or some of their arguments are different. Unfortunately, many examples show that only a very few data types, like the natural numbers with 0 and successor or the strings over some alphabet with empty string and left (or right) addition of symbols to strings, are of this nature. In most cases, different terms may yield the same value, as Boolean expressions yield either TRUE or FALSE.

To cover the phenomenon of equivalent terms, one allows to specify *equations* by pairs of terms on the syntactic level. A *SIG*-algebra *satisfies* such an equation if its two terms evaluate equally for each variable assignment. Hence, a *specification SPEC* consists of a signature *SIG* and a set of equations, and a *SPEC-algebra* is a *SIG*-algebra satisfying all equations in the set. Nicely enough, a set of equations gives rise to a recursive construction of a congruence relation on $T_{SIG}(X)$ (for each X) that is obtained by substitution of variables by terms and equivalence closure. This congruence relation contains exactly all equations that are satisfied by all *SPEC*-algebras, and if one factors T_{SIG} through the respective congruence, one gets the *initial SPEC*-algebra T_{SPEC} as a formal equivalent to an abstract data type.

The *quotient term algebra* equals the desired data type (up to isomorphism) in many examples like integers, strings, sets, arrays, etc. These correctness results establish an excellent justification of the choice of the initial algebras as semantics of algebraic specifications. Given some *SPEC*-algebra A, there is always a unique homomorphism from T_{SPEC} to A. If A represents certain requirements, then this homomorphism often helps to show that T_{SPEC} fulfils the requirements and *SPEC* is *correct* with respect to them.

Further topics of the course concern the structured design of algebraic specification including parameterization and stepwise refinement. The course closes with an outlook on the practical application of algebraic specification by using algebraic specification languages and their tools.

In our experience, computer science students have no problem to grasp the syntactic features of algebraic specification and the intuition of their semantics. But most of them find it difficult to see and believe that the mathematical formalization meets the intuition. They have even more difficulty with using the term congruence to prove correctness results. This is the reason why we look for multimedia support, as it may help students to better understand the semantic issues of algebraic specification.

Multimedia support

First of all, one should be aware that even traditional (computer science) courses have a multimedia dimension. Graduate courses in theoretical computer science in Bremen, for example, combine lectures and working sessions. The blackboard is used to develop the basic ideas, concepts, results and proofs. Transparencies are used to summarize the attained state of affairs. Posters may be used for the same purpose to present central issues visibly. There are classroom notes and written exercises. And so on. In other words, if we speak about multimedia support, we mean the additional use of digital media like the Internet. We are not aiming at a total and revolutionary change of teaching, but a moderate evolution of the didactic concepts by means of digital media.

In more detail, we consider and plan the following measures. As a first step, a course website is installed that provides the classroom notes, exercises, and any further information of interest for easy reading and printing. It may also be a good idea to provide the written material in such a way that the students can annotate it on screen with their own comments. Moreover, the website may offer an on-line forum for commenting, criticizing, asking questions, and discussing all matters of concern. These are didactic measures that are independent of the content, can be realized for most academic courses in a similar way, and are somewhat straightforward.

More interesting is the question how a better understanding of algebraic specification can be supported by computerized media. The key seems to be that the initial semantics of an algebraic specification is based on two recursive processes: term generation and congruence closure. Hence a simulation of these processes should help. One option is the use of existing tools for term rewriting, theorem proving, or interpretation of algebraic specification languages that evaluate terms or equations within the congruence generated by the specific equations. We will try to use the CASL tools for this purpose (see [CoF01]) because CASL is a kind of standard algebraic specification language. Unfortunately, all existing evaluators aim at efficiency by employing a deterministic depth-first evaluation strategy while the congruence is more of a non-deterministic and breadth-first nature. Another option to overcome this discrepancy would be the development of a special tutorial system that visualizes and animates term generation and congruence closure in the way they are defined. For automata theory, a very nice example of a tutorial system is JFLAP [JFL01], a tool which allows to graphically create and simulate several versions of automata (cf. also the last column in EATCS Bulletin No. 74). But we are not at all sure that we should and can undertake such an effort.

Concluding remarks

We have sketched the very first ideas how a course on algebraic specification may be supported by multimedia. Admittedly, some of the considerations are still very vague. Things will be much clearer in half a year, i.e. after the experiment. We have reported on our plans in this early stage because we hope that some readers can point out similar experiments, from which we can learn, or tutorial systems that work well and serve the purpose of illustrating the behaviour of theoretical features.

References

[AKK99] Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner. *Algebraic Foundations of Systems Specification*. Springer, 1999.

[CoF01] CoFI tools. `http://www.tzi.de/cofi/Tools`, Sept. 14, 2001.

[EGL89] H.-D. Ehrich, M. Gogolla, and U. Lipeck, editors. *Algebraische Spezifikation abstrakter Datentypen*. Teubner-Verlag, Stuttgart, 1989.

[EM85] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[GTW78] Joseph A. Goguen, James W. Thatcher, and Eric G. Wagner. An initial algebra approach to the specification, correctness, and implementation of abstract data types. In Raymond Yeh, editor, *Current Trends in Programming Methodology, IV*, pages 80–149. Prentice-Hall, 1978. Also as Report RC 6487, IBM T.J. Watson Research Center, Yorktown Heights, 1976.

[HKB89] J. Heering, P. Klint, and J.A. Bergstra. *Algebraic Specification*. ACM Press Fontier Series, 1989.

[HL89] Ivo Van Horebeek and Johan Lewi. *Algebraic Specifications in Software Engineering*. Springer, 1989.

[JFL01] JFLAP 3.1. `http://www.cs.duke.edu/~rodger/tools/jflap`, Sept. 14, 2001.

[Kla83] H.A. Klaeren. *Algebraische Spezifikation*. Springer, 1983.

[LEW96] J. Loeckx, H.-D. Ehrich, and B. Wolf. *Specification of Abstract Data Types*. Wiley, 1996.