Toward a General Theory of Computation and Specification over Abstract Data Types

J.V. Tucker[1]

Department of Mathematics and Computer Science,
University College of Swansea, Swansea SA2 8PP, Wales
csjvt@pyr.swan.ac.uk

J.I. Zucker[2]

Department of Computer Science and Systems,
McMaster University, Hamilton, Ontario L8S 4K1, Canada
zucker@maccs.dcss.mcmaster.ca

Abstract. We describe an ongoing project, to develop a general theory of computation and specification over classes of structures, modelling abstract data types. Applications include logic programming module development and hardware design for synchronous concurrent algorithms.

Key words and phrases: abstract data type, computation, computability, specification.

1 Introduction

We are developing a theory of computable functions and specifiable relations over classes of many sorted algebras. Such a class K of algebras is a model of the semantics of a module defining an abstract data type. The theory of computation, specification and verification over K has many applications, thanks to the generality of K; those we have studied concern the following subjects: *program specification and verification; logic programming modules and abstract data types;* and *synchronous concurrent algorithms and their application in hardware design.* Research on these apparently disparate applications is part of a programme, based on the algebraic theory of abstract data types, that aims at a general mathematical framework for the formulation and analysis of the many interesting notions of computability, specifiability, and verifiability that exist in different areas of computer science and artificial intelligence.

2 Algebras, Modules and Abstract Data Types

Algebras and modules. A many-sorted algebraic structure A consists of a number of sets of data, operations, and relations on the data. Such a structure can be used to model semantically a concrete implementation of a software module. We concentrate

[1]Research supported by SERC Research Grant GR/F 59070 under the Alvey Programme (IKBS 103).
[2]Research supported by SERC Research Grant GR/F 10606 under the Alvey Programme, and by a grant from the Science & Engineering Research Board of McMaster University.

on *standard structures*, which contain sorts for the natural numbers N and booleans B, together with the standard operations on these. (This is no real restriction, since any structure can be standardised.) A class K of such structures can be used to model semantically the module abstractly as a class of implementations. We assume K is closed under isomorphism. K can then be viewed as an *abstract data type.* Among the properties of algebras that we use to characterise meaningful or useful implementations are: *minimality, initiality* and *computability.* See [**5**] for background material on algebra.

Higher-order computation and logic over a structure A can be represented by applying first order computation and logic to suitable extended structures. For example, we use the following *weak second order* structure over a standard structure A. Add an undefined symbol u_i to each carrier A_i and extend the operations by strictness. This forms an algebra A^{u} with carriers $A_i^{\mathsf{u}} = A_i \cup \{\mathsf{u}_i\}$. We model a finite array over A_i by a pair (ξ, l), where $\xi : \mathrm{N} \to A_i^{\mathsf{u}}$ and $l \in \mathrm{N}$, such that $\xi(n) = \mathsf{u}_i$ for all $n > l$. Let A_i^* be the set of all such pairs. We add these sets to the algebra A^{u} to create carriers of the array algebra A^*. The new constants and operations of A^* are: the everywhere undefined array, evaluating an array at a number address, updating an array by an element at a number address, evaluating the length, and updating the length, of an array.

3 Computability Theory

The mathematical theory is a generalisation of classical computability theory on N to any abstract algebra. Computability theory for algebras originates with work by E Engeler in 1965. For a survey of subsequent work by Friedman, Shepherdson and others, see [**6**] and [**8**, §4.10].

Computable functions on abstract data types. In [**8**] we examined the class $\mathrm{IND}(A)$ of functions over a standard algebra A that are generated from its basic operations by means of *sequential composition; parallel composition; simultaneous primitive recursion; least number (μ) operator.* If simultaneous primitive recursion is replaced by *simultaneous course-of-values recursion*, we obtain the class $\mathrm{COVIND}(A)$ of functions obtained is the class of *course-of-values (cov) inductively definable functions.* In either case, the functions are defined by a *parallel deterministic* model of computation, due to the simultaneous recursion. Hence we also obtain the families of functions $\mathrm{IND}(\mathrm{K})$ and $\mathrm{COVIND}(\mathrm{K})$ over the class K.

There are many models of computation equivalent with these models. This is discussed in detail in [**8**], where it is argued that the ideas of deterministic computation and operational semantics are meaningful and equivalent, and the following thesis is proposed.

Generalisation of the Church-Turing thesis for computation. *Consider a deterministic programming language over an abstract data type* ***dt****. The set of functions on a structure A implementing* ***dt****, that can be programmed in the language, is contained in the set* $\mathrm{COVIND}(A)$*. The class of functions over the class* K *of structures implementing* ***dt****, that can be programmed in the language, uniformly over all structures in* K*, is contained in the family* $\mathrm{COVIND}(\mathrm{K})$*.*

Computation on A^*. For many purposes it is convenient to present the computable functions on A using (simultaneous) primitive recursion on A^* (see §2). This is justified by the following:

THEOREM. *Let f be a function on A. Then*

$$f \in \mathrm{COVIND}(A) \iff f \in \mathrm{IND}(A^*).$$

4 Specification Theory

Paradigms for specification and computation. We use the *relational paradigm* for *specification* and the *functional paradigm* for *computation.* The connection between them is that *we can compute functions that are selection functions for certain relations.*

Given a set A and a relation $R \subseteq A^{n+m}$, a *selection function for R* is a map $f : A^n \to A^m$ such that $\forall x[\exists y R(x,y) \Rightarrow R(x, f(x))]$. In programming theory, a progam *implements a specification* over an abstract data type if it computes some selection function f for the relation R uniformly over a class K of algebras.

Now for a program to be *effectively testable* against a specification $\exists y R(x,y)$, the relation R must be computable. This leads us to study (generalisations of) the notion of r.e. set on algebras.

A relation on A is said to be *(cov) semicomputable* if it is the domain of a partial (cov) computable function on A. A relation is *projective (cov) semicomputable* if it is a *projection* of a (cov) semicomputable relation. Not every set that is projective cov semicomputable is cov semicomputable. However, in the case of *minimal structures*, the notions are equivalent.

Scope and limits of specification. According to the above, the specifiable relations are the projective cov semicomputable ones. What about the different views of specification to be found in say logic programming or the theory of stepwise program development? In [**9**] we formalise Horn clause definability over any structure or class of structures and show that this corresponds with projective cov definablility. Furthermore, from [**8**] we can characterise such sets by their definability in a specification language.

THEOREM. *A relation R on A is projective cov semicomputable over* K *if, and only if, either of the following holds:*
(i) R is definable by Horn clauses on A^*, uniformly over $A \in$ K;
(ii) R is definable by a $\mathbf{\Sigma}_1$ formula over A^* uniformly over $A \in$ K.

These notions are also equivalent to several other characterisations by means of nondeterministic models for specification including *'while'-array with initialisation*; *'while'-array with random assignments*; and older notions such as *search computability* in the sense of Moschovakis. We are working on the following general kind of Church-Turing Thesis for specification to complement that for computation.

Generalisation of the Church-Turing thesis for specification. *Consider a non-deterministic programming or algorithmic specification language over an abstract data type* ***dt****. The set of relations on a structure A implementing* ***dt****, that can be specified in the language, is contained in the set of projective cov semicomputable relations on A. The class of relations over the class* K *of structures implementing* ***dt****, that can be specified in the language, uniformly over all structures in* K*, is contained in the family of relations projective cov semicomputable over* K.

This is a more complex task: for details see [**9**].

Provable totality of selection functions. In [**10**] we investigate selection functions for *existential ($\mathbf{\Sigma}_1^*$) assertions* $\exists y R(x,y)$ over A, for computable predicates R. It is shown that if such an existential assertion is *provable* in a formal system which includes induction over N for $\mathbf{\Sigma}_1^*$ formulae only, then a selection function for it can be found which is *primitive cov computable*, *i.e.*, generated in COVIND(A) *without use of the* μ *operator.*

5 Applications

Logic programming. This work is relevant to the development of the concept of a *logic programming module* that generalises the abstract data type module. There is a close connection between logic programming modules and algebraic specification modules: see [**1**].

Theory of Hardware. A *synchronous concurrent algorithm* (sca) is a network of processors and channels that compute and communicate in parallel, synchronised by a global clock. Such algorithms compute on infinite streams of data. Examples of scas include: *clocked hardware; systolic algorithms; neural nets; cellular automata; and coupled map lattice dynamical systems.* This type of computation is formalised using the course-of-values inductive functions over *stream algebras* (*i.e.*, standard structures which include domains of functions from N to the various carriers). The study of such models of clocked algorithms, and their application, is a substantial task: it aims at a general mathematical theory of computation based on hardware. The special case of unit delays throughout the network corresponds with the use of primitive recursive functions and has been studied intensively in joint work with B C Thompson. Contributions to the theory and to the development of case studies have been made by K Meinke, N A Harman, S M Eker and K Hobley (see [**2**, **3**, **4**, **7**] and the references therein).

Program correctness. Consider program specifications over the class K, and the validity of Hoare formulae $\{p\}S\{q\}$ over K. Our computability theory was developed originally to prove that important sets of program states, such as *weakest preconditions* and *strongest postconditions*, were expressible in a weak second order many sorted logical language. At the heart of this exercise is the theorem (*ii*) in §4 that characterises generalised r.e. sets in the language. The present work is a natural extension of our studies of subjects first encountered in the preparation of our book [**8**].

References

1. J Derrick and J V Tucker, *Logic programming and abstract data types*, in "Proceedings of 1988 UK IT Conference," held under the auspices of the Information Engineering Directorate of the Department of Trade and Industry (DTI), Institute of Electrical Engineers (IEE), 1988, pp. 217-219.
2. N A Harman and J V Tucker, *Clocks, retimings, and the formal specification of a UART*, in "The Fusion of Hardware Design and Verification," Proceedings of IFIP Working Group 10.2 Working Conference (ed. G Milne), North-Holland, 1988, pp. 375-396.
3. K M Hobley, B C Thompson, and J V Tucker, *Specification and verification of synchronous concurrent algorithms: a case study of a convolution algorithm*, in "The Fusion of Hardware Design and Verification," Proceedings of IFIP Working Group 10.2 Working Conference (ed. G Milne), North-Holland, 1988, pp. 347–374.
4. K Meinke and J V Tucker, *Specification and representation of synchronous concurrent algorithms*, in "Concurrency '88," (ed. F H Vogt), Lecture Notes in Computer Science 335, Springer-Verlag, 1988, pp. 163–180.
5. K Meinke and J V Tucker, *Universal algebra*, in "Handbook of Logic in Computer Science," (ed. S Abramsky, D Gabbay, T Maibaum), Oxford University Press (to appear).
6. J C Shepherdson, *Algorithmic procedures, generalised Turing algorithms, and elementary recursion theory*, in "Harvey Friedman's Research on the Foundations of Mathematics," (ed. L A Harrington *et al.*), North-Holland, 1985, pp. 285–308.
7. B C Thompson and J V Tucker, *A parallel deterministic language and its application to synchronous concurrent algorithms*, in "Proceedings of 1988 UK IT Conference, held under the auspices of the Information Engineering Directorate of the Department of Trade and Industry (DTI), Institute of Electrical Engineers (IEE)," 1988, pp. 228–231.
8. J V Tucker and J I Zucker, "Program correctness over abstract data types, with error state semantics," CWI Monograph 6, North Holland, 1988.
9. J V Tucker and J I Zucker, *Horn programs and semicomputable relations on abstract structures*, in "Proceedings of the 16th International Colloquium on Automata, Languages and Programming, Stresa, Italy," (ed. G Ausiello et al.) Lecture Notes in Computer Science 372, Springer-Verlag, 1989, pp. 745–760.
10. J V Tucker and J I Zucker, *Provable computable functions on abstract data types*, Department of Computer Science & Systems, McMaster University, Technical Report 90-02, 1990.