Introduction to Algebraic Specification. Part 2: From Classical View to Foundations of System Specifications

H. EHRIG[1], B. MAHR[1] AND F. OREJAS[2]

[1] *Technical University of Berlin, Fachbereich 20 Informatik, Institut für Software und Theoretische Informatik, Franklinstrasse 28/29, W-1000 Berlin 10, Germany* [2] *Technical University of Catalunya*

Part 1 of this paper concerning algebraic specifications is an informal introduction to formal methods for software development using algebraic techniques. The intention of this second part of the paper is to survey the research field of algebraic specifications developed within the last two decades and the role of this field concerning formal methods in computer science. The aim of this paper is to show that the classical field of algebraic specifications, using equational axioms, equational logic and algebraic data types and varieties, has reached a consolidated status by now. It can be considered as an important kernel of a more general theory which is presently under development, focusing on the foundations of system specifications in general.

Received June 1992

1. INTRODUCTION

Algebraic specification is a formal specification approach that emerged in the mid 1970s in order to provide an abstract and implementation-independent definition of data types. This original idea of using algebraic specifications as a formal specification technique for abstract data types was quite successful. It was extended within the last decade to horizontal and vertical structuring of specifications as required from the software engineering point of view. Today algebraic specification is a well-known formal method for software development. In part 1 of this paper we have given an introduction to algebraic specification from the software engineering point of view which shows how algebraic specification techniques can support the general software development process.[14] In this second part of the introduction to algebraic specification we concentrate more on the mathematical theory, which has been developed from a subfield of universal algebra to a categorical theory for the foundations of system specifications.

Algebraic specification originally concentrated on the use of equational logic as the basic formalism for research. This use has been one of the causes of its success, since the good behaviour of equational logic with respect to a number of semantic constructs has helped in the study of new specification concepts and constructs. Moreover, the abstract categorical techniques used to obtain these results allow a generic study of specification constructs independent of any specific formalism. In this sense, research in algebraic specification departed from equational logic a number of years ago and has concentrated, on the one hand, in extending the concepts and results for more expressive formalisms. On the other hand, the generic study of specification constructs provides the new aim of allowing the possibility of integrating several 'logics' into a single specification framework, in such a way that different aspects of a given system could be specified using the most appropriate formalism. Therefore, one may argue now that the area has broadened so much, with respect to techniques, aims and views, that it is no longer appropriate to call it algebraic specification. As a consequence, to describe all the work going on in this area we advocate using the name 'Foundations of System Specification', where algebraic specifications can be considered to be an important special case.

The rest of the paper is organised as follows. In the following section we present the basic specification concepts using the equational formalism, i.e. the classical view of algebraic specifications. In Sections 3 and 4 we present the main constructions for building specifications and for implementations of specifications. Current work in the foundations of system specification is briefly described in Section 5. Finally a conclusion and references are given in Sections 6 and 7.

2. CLASSICAL VIEW OF ALGEBRAIC SPECIFICATIONS

In this section we review the basic concepts underlying the classical view of algebraic specifications as given in Refs 20 and 12. These concepts have not lost their scientific relevance: the more general ones still provide guidelines for future research and foundation of systems specification, and the more technical ones still stand as a paradigm of mathematical strength in development of new techniques with a higher degree of expressiveness and flexibility.

Information structures, data structures and data types appear in programming in an implicit and in an explicit way. With the advent of languages aiming at higher levels of description in the late 1960s and early 1970s there came the need for a language-independent and more mathematical understanding of these notions. It was observed then that the concepts of abstract algebra, heterogeneity and equational presentation in the fields of universal algebra and equational logic as well as the concept of free construction in algebraic categories can provide an appropriate basis for a theory that helps this understanding. In subsequent years such a theory – the theory of algebraic specification of abstract data types – has been developed. Its basic general and technical concepts are as follows.

2.1 Abstract data types and their specification in general

Information structures, data structures and data types – we will not discuss possible differences here, but simply

speak of data types – are seen as many-sorted algebras, i.e. as collections of data domains and operations on these domains. Abstract data types correspond to classes of algebras. What makes abstract data types abstract and in what sense, is to be modelled by the choice of class. With the basic intention that any member of the class is considered good as a data type, abstraction in this context has two goals, as follows.

- To neglect certain differences of the data types under consideration. This is achieved by admitting with each algebra in the class also those algebras as members whose differences are to be neglected.
- To focus on certain commonalities and invariants of the data types under consideration. This is achieved by defining the class in such a way that all its members share these commonalities and obey these invariants.

An elementary example of an abstract data type is that of binary trees whose nodes are strings over some fixed-size alphabet. Let $TREE(n^*)$ denote the class of algebras modelling this abstract data type. Which algebras exactly belong to this class requires a (formal) specification. But in any case, we would like to neglect differences in the choice of alphabet symbols, the representation of strings and of trees and in the computational aspects of string and tree operations. On the other hand, we would like to imply that all algebras have the same structure, concerning domains and operations, and that its data objects really represent strings and trees.

Specification of abstract data types concerns the definition of classes of algebras. This is distinct from the specification of a single data type, which would mean the definition of a single algebra. Specification of abstract data types is seen to combine two things:

- a formal description *SPEC*, called specification, of commonalities and invariants, to be shared and obeyed by all members of the class;
- a semantic construction SEM, used to define from the specification *SPEC* the class of algebras in such a way that the specification is satisfied and that additional global requirements for the choice of class are fulfilled.

Most formal approaches to specification of abstract data types give specifications in terms of signatures and axioms and use semantic constructions, which ensure that the classes chosen are closed under isomorphism. Differences, however, are found in the form of axioms and in the global requirements. These requirements mainly express the conceptual understanding of abstract data types in general (initial semantics, final semantics, loose semantics, etc.).

2.2 Algebraic specification of abstract data types

In the algebraic approach to specification of abstract data types concepts and results from the fields of universal algebra, equational logic and algebraic categories are used to provide a mathematical formalism for specifications and semantic construction. The theory of algebraic specifications documents the study of this formalism and gives results for its practical use.

An algebraic specification $SPEC = (S, OP, E)$ consists of a signature (S, OP) and equational axioms E. S denotes a set of sort symbols and OP a set of constant and operation declarations. Constant declarations have the form $c: \to s$ for some sort symbols s, and operation declarations have the form $op: s1 \dots sn \to s$ for sort symbols $s1, \dots, sn, s$. A signature is meant to declare the structure of a data type, while the interpretation of a signature by an algebra (S_A, OP_A) defines the extension of a data type. S_A associates with each sort symbol $s \in S$ a domain A_s, with each constant declaration $c: \to s$ a data element $c^A \in A_s$, and with each operation declaration $op: s1 \dots sn \to s$ a function $op^A: A_{s1} \times \dots \times A_{sn} \to A_s$.

Sorted variables and constant and operation declarations allow the formation of sorted terms t which, in a sense, make the linguistic level for the handling of abstract data types. Equational axioms are then defined to be equations of the form $t = t'$ or implications of the form $t1 = t1' \wedge \dots \wedge tn = tn' \to t = t'$.

The following algebraic specification **tree(n*)** can serve as a specification of the abstract data type $TREE(n^*)$ above:

tree(n*) =
sorts *alphabet, string, tree*
opns $a1, \dots, an: \to alphabet$
empty: $\to$ *string*
make: *alphabet* $\to$ *string*
concatenate: *string string* $\to$ *string*
leaf: *string* $\to$ *tree*
compose: *tree string tree* $\to$ *tree*
eqns *concatenate* $(s, empty) = s$
concatenate $(empty, s) = s$
concatenate (*concatenate* $(s1, s2), s3) =$
concatenate $(s1,$ *concatenate* $(s2, s3))$
compose $(t1, s, t2) =$ *compose* $(t2, s, t1)$

There are various forms of semantic construction in the algebraic approach to specification of abstract data types. The most prominent ones are loose semantics and initial semantics.

Loose semantics associates with each specification (S, OP, E) the class $\mathrm{Alg}(S, OP, E)$ of all algebras $A = (S_A, OP_A)$ which satisfy the equational axioms E. Loose semantics is mainly of technical interest. It stems from universal algebra, where classes of algebras with a given signature obeying a given set of equations – so-called varieties or quasi-varieties – are the object of study. For an abstract data type loose semantics has little expressiveness, since one would not accept every member of $\mathrm{Alg}(S, OP, E)$ as a data type to the given specification. The trivial algebra, for example, with singleton domains and trivial operations, shows little correspondence with the intended meaning of **tree(n*)**.

Initial semantics associates with each specification $SPEC = (S, OP, E)$ the class $\mathrm{Init}(S, OP, E)$ of initial $SPEC$-algebras. In this class all algebras are isomorphic and cannot be distinguished by properties expressible in first-order logic. Furthermore, all algebras in this class are generated, i.e. for every data element there is a variable-free term representing it, and typical, i.e. they satisfy a variable-free equation if and only if this equation is satisfied by all algebras in $\mathrm{Alg}(S, OP, E)$. These two properties form the global requirements for the choice of class with this semantic construction. One can show that these two properties uniquely determine the class $\mathrm{Init}(S, OP, E)$. They reflect a certain relationship between an abstract data type and its specification, namely that no data type is accepted which has data objects unreachable by executing operations ('no junc') or which satisfies

elementary properties not required in the specification ('no confusion'). Despite the fact that initial semantics is intuitively very appealing, in its purest form it is often too restrictive. This observation, as well as variants in the understanding of abstract data types, has led to semantic constructions mediating between loose and initial semantics. Among those are final semantics, semantic constructions with constraints and others.

Initial semantics of the algebraic specification **tree(n*)** nicely reflects the intended meaning of the abstract data type $TREE(n^*)$: all its data types have three domains corresponding to alphabet, string and tree; alphabets always have n symbols and the strings in each case are exactly concatenated symbols; trees can in each case be represented by a graph in the usual way without loss of information. Trees, however, are not ordered (due to the last equation), so that their graph representation or their description by a term may be misinterpreted.

A good understanding of initial semantics and also a rather practical tool for 'computing' and 'reasoning' on the level of terms is obtained by picking a particular member of the class Init (S, OP, E), the so-called quotient term algebra Q. This algebra is an example of the so-called 'free construction', predominant in the study of varieties and algebraic categories, and is defined as follows.

The domains of the quotient term algebra Q are sets of classes $[t]$ of terms obtained by factorisation, where terms $t1$ and $t2$ are congruent if the equation $t1 = t2$ is derivable from E. The constants and operations of Q are defined by $c^Q := [c]$ and $op^Q([t_1], \ldots, [t_n]) := [op(t1, \ldots, tn)]$ respectively.

Q is initial and we have Init $(S, OP, E) = \{A \mid A \cong Q\}$. The quotient term algebra nicely reflects the relationship between the linguistic level of terms and the specified abstract data type. It realises a general understanding of semantics as a form of classification and abstraction in a concrete way.

2.3 Algebraic specification and universal algebra

It has already been mentioned that algebraic specification has its conceptual roots in universal algebra, equational logic and algebraic categories. Apart from notational and methodological influences which these mathematical theories have, it is mainly the study of classes of algebras and their presentation and the equational calculus with its proof theory, which forms the mathematical basis for algebraic specification.

Birkhoff, who is known as the founder of universal algebra as a mathematical discipline (see Refs 5 and 7) studied classes of algebras which can be presented, i.e. axiomatised, by sets of equations. He characterised such equational classes by closure properties and proved among other results that a class C of algebras is equational if and only if it contains all products, subalgebras and homomorphic images of its members. This characterisation is of particular interest, since it allows us to show that equational classes are abstract in the sense that they contain with every member all its isomorphic copies, and that they admit free constructions for every set of generators. Equational algebraic specifications can therefore be seen as presentations of equational classes, and initial semantics as the isomorphism class of the free algebra over the empty set of generators. Deduction with algebraic specifications then corresponds, at least in its simplest form, to proofs in the equational calculus.

This correspondence turned out to be very fruitful for the study of algebraic specification and formed the basis of further investigations in the light of the new application to universal algebra. These investigations also made use of category-theoretic notions, namely of categories, functors and transformations,[2,3,23] which proved most appropriate for the study of algebraic implementations and abstraction techniques in specification languages.

3. HORIZONTAL STRUCTURING TECHNIQUES

In this section we introduce formal structuring techniques for algebraic specifications. This allows us to build up larger specifications from smaller units leading to a horizontal structuring of system specifications as discussed in part 1 of our paper.[4,14,30] In order to formalise horizontal structuring concepts we introduce specification morphisms between algebraic specifications leading to the category **SPEC** of algebraic specifications. The category **SPEC** satisfies a number of properties, formulated as main lemmas, leading to several interesting results concerning horizontal structuring techniques including extension union, renaming and parameter passing of parameterised specifications, formulated as main results of this section. For more details of horizontal structuring techniques for basic and parameterised specifications we refer to Ref. 12. Moreover, we would like to mention the concept of algebraic module specifications with explicit import and export interfaces in the sense of modules in software engineering (see Ref. 28), which are studied in detail in Refs 13, 16 and 31.

3.1 Categories of algebraic specifications and SPEC-algebras

In order to define specification morphisms we first consider the special case of signature morphisms. Given two signatures $(S1, OP1)$ and $(S2, OP2)$ in the sense of 2.2 a signature morphism $f: (S1, OP1) \to (S2, OP2)$ consists of a pair $f = (f_S: S1 \to S2, f_{OP}: OP1 \to OP2)$ of compatible functions between sort symbols and (constant and) operation declarations respectively. Compatibility means that $op: s1 \ldots sn \to s$ in $OP1$ implies

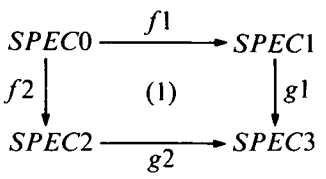
$$f_{OP}(op): f_S(s1) \ldots f_S(sn) \to f_S(s) \text{ in } OP2.$$

A signature morphism $f: (S1, OP1) \to (S2, OP2)$ is called specification morphism $f: SPEC1 \to SPEC2$ with $SPECi = (Si, OPi, Ei)$ $(i = 1, 2)$, if the translated equations $f\#(E1)$ of $E1$ are derivable from $E2$.

The category **SPEC** of algebraic specifications consists of all algebraic specifications $SPEC = (S, OP, E)$ as objects and all specification morphisms as morphisms, with componentwise notion of identities and composition of morphisms. For each specification $SPEC$ there is also a category **Cat(SPEC)** of SPEC-algebras and $SPEC$-homomorphisms on the semantic level and for each specification morphism $f: SPEC1 \to SPEC2$ there is a forgetful functor V_f: **Cat(SPEC2)** $\to$ **Cat(SPEC1)** which assigns to each $SPEC2$-algebra $A2$ the $SPEC1$-algebra $A1$ obtained by restricting $A2$ to $SPEC1$ according to f.

3.1.1 Main lemma for specifications and algebras

(1) The category **SPEC** has pushouts, i.e. for each pair of specification morphisms $f1: SPEC0 \to SPEC1$ and $f2: SPEC0 \to SPEC2$ there are a specification $SPEC3$ and specification morphisms $g1: SPEC1 \to SPEC3$ and $g2: SPEC2 \to SPEC3$ with $f1 \circ g1 = f2 \circ g2$, which satisfy the universal properties of pushouts in a category (see Ref. 12 for more detail).



(2) For each pushout (1) in **SPEC** we have the following amalgamation properties for algebras. Given $SPECi$-algebras Ai for $i = 0, 1, 2$ so that the restriction of $A1$ and $A2$ to $SPEC0$ is equal to $A0$, there is a unique $SPEC3$-algebra $A3$, called amalgamation of $A1$ and $A2$ via $A0$, written

$$A3 = A1 +_{A0} A2,$$

so that the restriction of $A3$ to $SPEC1$ and $SPEC2$ is equal to $A1$ and $A2$ respectively. In more detail the restriction of $A1$ to $SPEC0$, and similar in other cases, is defined by application of the forgetful functor V_{f1}: **Cat(SPEC1)** $\to$ **Cat(SPEC0)** so that the amalgamation property means the following: for all Ai in **Cat(SPECi)** for $i = 0, 1, 2$ with $V_{f1}(A1) = A0 = V_{f2}(A2)$ there is a unique $A3$ in **Cat(SPEC3)** with $V_{g1}(A3) = V_{g2}(A3)$.

3.2 Horizontal structuring techniques for basic specifications

The horizontal structuring techniques introduced in part 1 of this paper for basic specifications can be formalised using the notions of 3.1 above as follows.

3.2.1 Extension

An extension of a specification $SPEC1$ is a specification morphism $f: SPEC1 \to SPEC2$. It is called initial conservative, if the restriction of the initial $SPEC2$-algebra T_{SPEC2} to $SPEC1$ is isomorphic to the initial $SPEC1$-algebra T_{SPEC1}, i.e. $V_f(T_{SPEC2}) \cong T_{SPEC1}$. The extension f of $SPEC1$ is called loose conservative, if for each $SPEC1$-algebra $A1$ there is a $SPEC2$-algebra $A2$, so that $A1$ is the restriction of $A2$, i.e. $V_f(A2) = A1$.

3.2.2 Union

The union of specifications $SPEC1$ and $SPEC2$ with shared specification $SPEC0$ given by specification morphisms $f1: SPEC0 \to SPEC1$ and $f2: SPEC0 \to SPEC2$ is the pushout specification $SPEC3$ (see 3.1.1). On the semantic level each algebra $A3$ of the union specification has a unique decomposition into algebras $A0$, $A1$, and $A2$ of the component specifications, so that $A3$ is the amalgamation of $A1$ and $A2$ via $A0$:

$$A3 = A1 +_{A0} A2.$$

This fact is a direct consequence of the amalgamation property in 3.1.2 taking $A1 = V_{g1}(A3)$, $A2 = V_{g2}(A3)$ and $A0 = V_{f1}(A1) = V_{f2}(A2)$.

3.2.3 Renaming

The renaming of specification $SPEC1$ is a bijective specification morphism $f: SPEC1 \to SPEC2$, i.e. an isomorphism in the category **SPEC** of algebraic specifications. This implies that there is also an inverse morphism $f^{-1}: SPEC2 \to SPEC1$ and that the corresponding categories of $SPEC1$- and $SPEC2$-algebras are isomorphic, i.e.

$$\textbf{Cat(SPEC1)} \cong \textbf{Cat(SPEC2)}$$

so that for each $SPEC1$-algebra $A1$ there is a corresponding $SPEC2$-algebra $A2$ given by $A2 = V_{f^{-1}}(A1)$ with the same structural properties, and vice versa.

3.3 Parameterised specifications

Parameterised specifications in the sense of part 1 of this paper can be formalised and generalised as follows.

(1) A parameterised specification $PSPEC = (PAR, BOD, s)$ consists of a formal parameter specification PAR, a body specification BOD and a specification morphism

$$s: PAR \to BOD.$$

(2) The semantics of a parameterised specification $PSPEC$ is given by a free construction

$$F_s: \textbf{Cat(PAR)} \to \textbf{Cat(BOD)}$$

which assigns to each PAR-algebra P a freely generated BOD-algebra $B = F_s(P)$, which satisfies the universal properties of free constructions,[12] and can be extended uniquely to a functor between the categories **Cat(PAR)** and **Cat(BOD)** of PAR- and BOD-algebras respectively.

(3) The parameterised specification $PSPEC = (PAR, BOD, s)$ is called (internal) correct if for each PAR-algebra P the restriction of the free construction $F_s(P)$ to PAR is isomorphic to P. This means that the free construction does not change the parameter part.

In more detail, $PSPEC$ is internal correct if the free construction F_s is strongly persistent or at least persistent, i.e. $V_s \circ F_s$ is equal or natural isomorphic to the identity on **Cat(PAR)**.

Finally let us note that our notion of semantics of $PSPEC$ corresponds to 'initial semantics' of parameterised specifications as discussed in Ref. 12 and there are also different notions of 'loose semantics' of parameterised specifications in the literature. Moreover, in addition to 'internal correctness' as introduced above, 'model correctness' of parameterised specifications is discussed in Ref. 12.

3.4 Horizontal structuring via parameter passing

According to the discussion in part 1 of this paper, another important structuring technique is given by parameterised specifications in connection with parameter passing, where the result specification can be considered as a composition of the parameterised specification and the actual parameter specification. This kind of compositionality holds not only on the syntactic level of specifications using pushouts, but also on the

semantic level using an important extension property for strongly persistent functors, especially for strongly persistent free constructions.

3.4.1 Main lemma for parameterised specifications and functors

(1) For each parameterised specification $PSPEC = (PAR, BOD, s)$ there is a free construction

$$F_s: \mathbf{Cat(PAR)} \to \mathbf{CAT(BOD)}$$

and hence a semantics of *PSPEC* uniquely defined up to isomorphism (see 3.3.2).

(2) For each pushout (1) in **SPEC** as given in 3.1.1 we have the following extension property for functors. For each strongly persistent functor

$$F: \mathbf{Cat(SPEC0)} \to \mathbf{Cat(SPEC1)}$$

there is a unique strongly persistent functor

$$F^*: \mathbf{Cat(SPEC2)} \to \mathbf{Cat(SPEC3)},$$

called extension of F via $f2$, so that we have

$$F \circ V_{f2} = V_{g1} \circ F^*,$$

where $F^*(A2)$ is given by the following amalgamation

$$F^*(A2) = F(A0) +_{A0} A2 \quad \text{with} \quad A0 = V_{f2}(A2).$$

Moreover, F^* is a free construction, i.e. $F^* \cong F_{g2}$, if F is a free construction, i.e. $F \cong F_{f1}$.

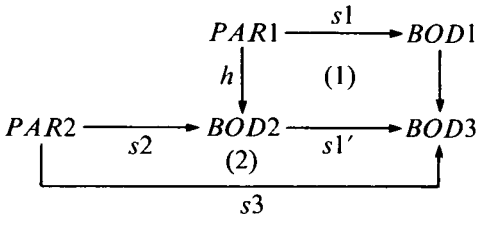
Now we are able to formalise parameter passing of parameterised specifications and to state the main results concerning induced correctness and compositionality. Given parameterised specifications

$$PSPECi = (PARi, BODi, si)(i = 1, 2)$$

and a parameter passing morphism $h: PAR1 \to BOD2$ the result of parameterised parameter passing is the parameterised specification

$$PSPEC3 = (PAR2, BOD3, S3),$$

where $BOD3$ is defined as pushout in (1) and $s3 = s1' \circ s2$ in (2).



If *PSPEC*2 is non-parameterised, i.e. $PAR2 = \varnothing$, the result *PSPEC*3 is also non-parameterised given by $BOD3$. This special case is called standard parameterised passing.

3.4.2 Main results for parameter passing

(1) Induced Correctness. Correctness of the given parameterised specifications *PSPEC*1 and *PSPEC*2 implies correctness of the result *PSPEC*3 of parameterised parameter passing, i.e. F_{s1} and F_{s2} strongly persistent implies strong persistency of F_{s3}.

(2) Compositionality. Given correct parameterised specification *PSPEC*1 and *PSPEC*2 with semantics F_{s1} and F_{s2}, the semantics F_{s3} of the result *PSPEC*3 is given by

$$F_{s3} = F^*_{s1} \circ F_{s2},$$

where F^*_{s1}: **Cat(BOD2)** $\to$ **Cat(BOD3)** is the extension of F_{s1} via h. These main results are essential for stepwise construction of correct software system specifications.

4. IMPLEMENTATIONS OF SPECIFICATIONS

In this section we review the main ideas that underlie the study of the implementation process in the algebraic specification literature. As discussed in part 1 of this paper, this process is always seen as stepwise in the sense that it consists of a sequence of implementation steps:

$$SPEC_0 \to SPEC_1 \to \ldots \to SPEC_n$$

where each step, $SPEC_i \to SPEC_{i+1}$, consists in the refinement of some specification unit of $SPEC_i$. Being more specific, in our framework, we may consider that a refinement step consists of the implementation of the data type specified by such a unit. In the rest of the section, we provide a general notion of implementation, and then we review the conditions that have to be fulfilled in order to infer the correctness of the whole implementation process from the correctness of each step. A more detailed study of the topics of this section (including a more comprehensive list of references) can be found in Ref. 27. For other algebraic techniques of stepwise refinement via transformations we refer to Refs 8 and 9.

4.1 Implementations

Given specifications $SPEC1 = (S1, OP1, E1)$ and $SPEC2 = (S2, OP2, E2)$ we may consider that implementing the data type specified by *SPEC*1 by the data type specified by *SPEC*2 consists in defining the operations (and the data sorts) in $(S1, OP1)$ in terms of the operations (and data sorts) from $(S2, OP2)$, in such a way that the enriched *SPEC*2-algebras 'behave' like the *SPEC*1-algebras. In this sense, syntactically, an implementation would be an enrichment together with a mapping (technically, a signature morphism) relating the sorts and operations from $(S1, OP1)$ with the sorts and operations from the enriched $(S2, OP2)$ signature and, semantically, it would be a construction (associated with the enrichment) together with some kind of behavioural abstraction that relates the *SPEC*1-algebras to the enriched *SPEC*2-algebras.

4.1.1 Definition (syntax of implementations)

Given two specifications *SPEC*1 and *SPEC*2, the syntax of an implementation $I = ((S, OP, E), h)$ of *SPEC*1 by *SPEC*2 consists of (1) an enrichment (S, OP, E) of *SPEC*2 and (2) a signature morphism $h: (S1, OP1) \to (S2, OP2) + (S, OP)$. Now in order to define the semantics of implementations we shall first define the concepts of constructors and abstractors. Constructors are intended to give semantics to the enrichment associated with the implementation while abstractors define which is the class of algebras that 'simulate' or 'reify' a given algebra.

4.1.2 Definition (constructors)

Given two specifications $SPEC1$ and $SPEC2$, with $SPEC1 \subseteq SPEC2$, a constructor k is a mapping from $SPEC1$-algebras to $SPEC2$-algebras. Usually, given an enrichment, its associated constructor is considered to be the free constructor (see Section 3).

4.1.3 Definition (abstractors)

Given a specification $SPEC = (S, OP, E)$, an abstractor α on (S, OP) is a mapping from $SPEC$-algebras to sets of (S, OP)-algebras satisfying:

(*a*) *Reflexibility.* For every $SPEC$-algebra A, $A \in \alpha(A)$
(*b*) *Transitivity.* For every (S, OP)-algebra $A1$, and all $SPEC$-algebras $A2$, $A3$ we have:

$$A1 \in \alpha(A2) \text{ and } A2 \in \alpha(A3) \text{ implies } A1 \in \alpha(A3).$$

Intuitively, given an algebra A, $\alpha(A)$ is the class of all algebras that may be considered adequate realisations of A. For instance, according to Hoare,[21] $A1$ may be in $\alpha(A)$ if there is an abstraction function from $A1$ to A. Nevertheless, other kinds of abstractor have been used in the literature on algebraic specification.[27]

4.1.4 Definition (semantics and correctness of implementations)

The semantics of an implementation $I = ((S, OP, E), h)$ of $SPEC1$ by $SPEC2$ is a pair $(V_h \circ k_I, \alpha_I)$, where k_I is a constructor from $SPEC2$-algebras to $SPEC2 + (S, OP, E)$-algebras, α_I is an abstractor over $(S1, OP1)$ and V_h is the forgetful functor associated with h. The implementation I is correct iff for every $SPEC1$-algebra $A1$ there exists a $SPEC2$-algebra $A2$ such that $V_h \circ k_I(A2) \in \alpha_I(A1)$.

4.1.5 Example (implementation of sets by sequences)

Let $SPEC1$ be the following specification of sets of integers:

$SPEC1 = INTEGER + BOOLEAN +$
sorts *set*
opns *e-set*: $\rightarrow$ *set*
add: *set int* $\rightarrow$ *set*
*_is-in_*: *int set* $\rightarrow$ *bool*
eqns *add*(*add*(*s*, *n*), *n'*) = *add*(*add*(*s*, *n'*), *n*)
add(*add*(*s*, *n*), *n*) = *add*(*s*, *n*)
n is-in e-set = *false*
n is-in add(*S*, *n'*) = (*n eq n'*) *or* (*n is-in S*)

and let $SPEC2$ be the following specification of sequences of integers:

$SPEC = INTEGER + BOOLEAN +$
sorts *seq*
opns *e-seq*: $\rightarrow$ *seq*
app: *seq int* $\rightarrow$ *seq*
head: *seq* $\rightarrow$ *int*
tail: *seq* $\rightarrow$ *seq*
eqns *tail*(*app*(*S*, *X*)) = *S*
head(*app*(*S*, *X*)) = *X*
tail(*e-seq*) = *e-seq*
head(*e-seq*) = 0

Then we may define a correct implementation of sets by sequences by means of the enrichment $(\emptyset, OP, E)$ defined below and the signature morphism h, from $(S1, OP1)$ to $(S2, OP2 + OP)$, mapping the sort *set* into the sort *seq* and the operations *e-set*, *add* and *_is-in_* into *e-seq*, *app* and *_is-in_*, respectively.

$(\emptyset, OP, E) =$ opns *_is-in_*: *int seq* $\rightarrow$ *bool*
n is-in e-seq = *false*
eqns *n is-in app*(*S*, *n'*) = (*n eq n'*) *or* (*n is-in S*)

If we consider that k_I is the free constructor from $SPEC2$-algebras to $SPEC2 + (\emptyset, OP, E)$-algebras then, given the algebra $A2$ of sequences of integers, $k_I(A2)$ is the algebra of sequences of integers including a new operation that, given a sequence S and an integer X, tells if X is an element of S. Now, $V_h(k_I(A2))$ is the same algebra as $k_I(A2)$ but not including the operations head and tail and having renamed the rest of the operations according to h, i.e. $V_h(k_I(A2))$ is an $(S1, OP1)$-algebra. Finally, if $A1$ is the algebra of sets of integers and if we define α_I as in Ref. 21, then $V_h(k_I(A2) \in \alpha_I(A1)$ considering the abstraction function f mapping every sequence $\langle x1, \ldots, xn \rangle$ into the set $\{x1, \ldots, xn\}$.

4.2 Correctness of the implementation process

A key issue in the software development process is that the correctness of the whole process could be established from the correctness of all steps. Moreover, global correctness must be independent of the order in which some development steps are performed. These questions were first studied in Ref. 17, where the software development process was described as a bidimensional process and two properties were identified as essential.

4.2.1 Vertical composition property

If $SPEC$ is a correct implementation of $SPEC'$ and $SPEC'$ is a correct implementation of $SPEC''$, $SPEC$ must be a correct implementation of $SPEC''$.

4.2.2 Horizontal composition property

If $PSPEC1 = (PAR1, BOD1)$ and $PSPEC2 = (PAR2, BOD2)$ are two parameterised specifications such that $PSPEC2$ is a correct implementation of $PSPEC1$ and if $SPEC1$ and $SPEC2$ are two specifications such that $SPEC2$ is a correct implementation of $SPEC1$ and $SPEC1$ is an actual parameter for $PSPEC1$, $SPEC2$ must be an actual parameter for $PSPEC2$ and, additionally, $PSPEC2(SPEC2)$ must be a correct implementation of $PSPEC1(SPEC1)$. These properties have been studied for different kinds of implementation concepts in the literature and summarised in Refs 27 and 29. Recently it has been shown that the fact that these properties are satisfied depends mainly on the choice of

the implementation (i.e. programming) language chosen rather than on the specification formalism used (e.g. see Ref. 27).

5. TOWARDS FOUNDATIONS OF SYSTEM SPECIFICATIONS

In this section we discuss aspects and directions of research for the foundations of system specification in general. Such foundations are meant to provide mathematical techniques for formal specification and reasoning and a mathematical framework for their development and study.

5.1 General needs for the formal specifications of systems

Requirements for software and the main conceptual stages and steps in the software development process, as discussed in part one of this introduction, show the general needs for the formal specification of systems:

- adequate specification formalisms for the various levels of concreteness and for the different kinds of components and aspects found in the system to be specified;
- techniques for the modelling of requirements, design and implementation with formal specifications;
- techniques for integration and coordination of specifications written in different formalisms;
- tools for creation, handling, transformation and prototypical execution of formal specifications;
- techniques for reasoning about specifications and for verification of design and implementation.

Future research on the foundations of system specification has to focus on all these aspects. The theory of specification of abstract data types has laid the ground for this research and has led to guidelines and concepts that are still relevant. Although the understanding of data types about twenty years ago had a rather narrow basis, its general concepts have proved suitable even in a world of systems with heterogeneous and distributed components, with reliance on knowledge and communication and with the need to tackle with openness, multimediality and uncertainty. It is clear, however, that much is left to be done for a theory that provides foundations for system specification. The research for adequate specification formalisms and the development of techniques for integration and coordination of specifications written in different formalisms will here play a major role. This was early observed, and there are already various concepts and results that belong to these topics.

5.2 Increasing expressiveness of specifications

Equational specifications in their purest form soon show limitations in both expressive power and expressiveness. Therefore extensions have been studied that fall into one or more of the following categories.

More general axioms. Axioms including predicates, negation or disjunction as well as universal and existential quantification have been proposed to enhance the expressive power of specifications. But it was shown in Ref. 25 that universal Horn-formulas form, in a certain sense, the most general type of axiom which admits initial semantics. This result as well as the paradigm of logic programming, which is traditionally based on definite Horn-clauses, however, has more or less ended this discussion.

Semantic variants. What is intuitively seen as a data type may not necessarily be modelled by a simple algebra in an adequate way. In order to find an elegant way for error handling, non-termination, recursiveness and programming with functionals, various semantic constructions have been proposed, usually combined with enriched forms of specification, which are appropriate to handle these phenomena. Among those are partial algebras, order-sorted algebras, continuous algebras, higher-order algebras and others. No unified semantic, however, has been worked out, which could cope with the many different cases and choices and avoid or at least control the semantic anomalies which have appeared in several of these concepts.

More elaborate specifications. Signatures and axioms alone cannot well enough specify abstract data types where abstraction is more sophisticated than in the cases of initial and loose semantics. For the specification of abstract data types that are restricted, that express behaviours, or that have components with fixed or initial interpretation and other components that are loose, for example, appropriate additional information is being added to specifications, which controls the semantic construction in the intended way.

More general sorting. Sorting in algebraic specifications has the simplest form possible. More adequate descriptions, support in the process of modelling, better help for avoiding redundancy, and increased expressiveness can be obtained by more general and more powerful forms of sorting. This observation has led to the inclusion of concepts like subsorting, dependent types, higher-order types or polymorphic types into the specification formalism. But, similar to the semantic variants, no uniform approach has been worked out which is fully integrated with the rest of the theory of specifications of abstract data types.

Research and experience from case studies and applications hinted early on at the need for more abstract concepts. With the development of a framework for 'logic independent' formulation of specification constructs and specification languages an important step towards integration and coordination of specifications written in different formalisms has been made.

5.3 Abstraction using specification logics

Higher-level specification constructs can be expressed and semantically founded independent of the basic formalism used for specifications and semantic constructs. In the early 1980s, motivated from the semantics of the specification language CLEAR, the concept of 'institution' was developed.[18] It provides a categorical framework for the relation between signatures, axioms and structures (e.g. algebras). Independently, similar abstraction, however in model-theoretic terms, was used to characterise specification logics allowing for initial semantics.[24] The results in this framework were later reformulated and generalised in the categorical framework of institutions. Recent work on specification techniques,[11,15] is based on the notion of 'specification

logic', which is equivalent to indexed and to fibred categories. Here a specification logic is a category of abstract specifications **ASPEC** together with a functor *Cat* into the quasi-category **CATCAT** of all categories, which is contravariant in **ASPEC**

$$Cat: \mathbf{ASPEC}^{op} \rightarrow \mathbf{CATCAT}.$$

It maps every specification *SPEC* to a category of models **Cat(SPEC)** and every specification morphism $f: SPEC1 \rightarrow SPEC2$ to the forgetful functor $Cat(f): \mathbf{Cat(SPEC2)} \rightarrow \mathbf{Cat(SPEC1)}$. If $Cat(f)$ has a left adjoint, free construction is possible (of which initial semantics is a special case). The work in Ref. 11 shows that all of the structural theory of algebraic specifications can be expressed within this framework of specification logic, where the main lemmas for specifications and algebras in 3.1 and for parameterised specifications and functors in 3.4 are taken as axioms in a specification logic. A framework that builds on institutions but also captures the more logical aspects of deduction is given in Ref. 22 and is called 'general logics'.

Abstraction in these frameworks takes place in two respects. First, they treat the notions of signature, axiom, specification, satisfaction, entailment, structure and others as abstract notions. Second, they define these abstract notions in terms of category theory. Category theory provides a 'functional' approach based on an axiomatic foundation of the notion of function, in categorical terms called 'morphism', with axioms just expressing the monoid structure of function composition (associativity and identity). The variety of formalisms to be captured by these frameworks is therefore very large. Specification constructs and specification languages based on one of these frameworks thus have a universal nature and wide range of applicability. Integration and coordination of specifications, written in different formalisms, can be supported by specification constructs and languages which are formalism-independent.

Another way to provide support is to reflect the different formalisms in a uniform calculus of declarations and a uniform notion of structure, powerful enough to capture the many different features. Such an approach is presently under development (see Ref. 26). It combines the techniques underlying type-theoretic calculi with the notions underlying models of set theories. As a result, all the known type and specification disciplines can be 'rewritten' in a uniform way, and new disciplines can be designed which allow even for self-application and self-reference.

5.4 Specifications and semantic constructions in a uniform framework

An equational specification, for example, contains various kinds of information. Among those are

$\underline{\text{sort}}\ s, \ldots$
$op: s1 \rightarrow s$
$t1 = t2$

All this information may be seen as a type proposition of the form $e: E$ for expressions e and E. We thus obtain in rewritten form

$s: \underline{\text{sort}}$
$op: s1 \rightarrow s$
$(t1 = t2): true$

An interpretation will have to associate with sort an object that is a set that contains the object associated with s as an element. It will interpret $s1 \rightarrow s$ as an object that is a function space that contains the object associated with *op*, which is a function, as an element. It will associate with true an object that is a singleton set with element T and with $(t1 = t2)$ the object T (for 'true'). A model is therefore an $\in$-structure $(M, \in)$ in which M is a set of objects and $\in$ is a binary relation such that all the denotational constraints – 'the object associated with $s1 \rightarrow s$ is a function space of the form' and the like – are true. These denotational constraints can be expressed in $\in$-logic which is first-order logic with a single binary predicate symbol (just like the logic used for axiomatic set theory).

The revision that is done here has two aspects: it reads all information given in a specification as a type proposition with the colon':' to be interpreted as membership; and it views sets, pairs, functions, etc. as objects that are not, but that have an extension which makes them a set, a pair or a function in the same way as these notions have a set-theoretic foundation (for example, a function is a set of pairs and a pair is a set of the form $\{\{a\}, \{a, b\}\}$).

Based on the notion of $\in$-structure, a new set theory can be established that is a refined form of P. Aczel's theory of non-well-founded sets.[1] It allows us to model self-application and gives new models for the λ-calculus, and to model semantic paradoxes, and therefore provides an appropriate basis for truth theories and the study of modalities. For foundations of system specification-type propositions and $\in$-structures as their models provide a most interesting basis for integration and coordination by uniformity and expressive power. In a uniform manner, [declaration] or [type] disciplines can be defined:

A discipline $D = (Expr, C, DCL)$ consists of

- *Expr*, a set of entities, called expressions;
- $C: Expr \rightarrow \in\text{-}Forms(Expr)$, a mapping from *Expr* to the set of $\in$-formulas over the set *Expr*, called denotational constraints;
- *DCL*, a class of sets of type propositions $e: E$ with expressions e and E, called the class of admissible declarations.

Semantics to disciplines is given by interpretation of declarations in $\in$-structures as follows.

For an admissible declaration let d in DCL $Expr(d)$ denote the expressions occurring in d, and $(M, \in)$ be an $\in$-structure, then a function $[\![\ ]\!]: Expr(d) \rightarrow M$ with

- $((M, \in), [\![\ ]\!]) \models C(e)$ for all $e \in Expr(d)$
- $[\![e]\!] \in [\![E]\!]$ for $e: E$ in d defines an interpretation and $(M, \in)$ a model for d with object interpretation $[\![\ ]\!]$.

A revision, as discussed in the example above, now allows us to read traditional declarations in their type-propositional form and thereby to view a particular specification technique as a discipline in the sense defined. In this way some of the limitations and deficiencies of classical specifications are avoided which concern expressive power and formal complexity of both syntax and semantics.

6. CONCLUSION

In this part 2 of an introduction to algebraic specification we have presented some basic elements of the theory of algebraic specifications corresponding to the formal methods for software specification and development as introduced in part 1 of this paper on an informal level.[14]

We have started with the classical view of algebraic specifications in the sense of equational specifications defining classes of algebras which are well known in the mathematical field of universal algebra. Of special importance for the specification of abstract data types is the notion of initial algebras, defining the initial semantics of algebraic specifications. Initiality is one important concept from category theory which plays a fundamental role in the theory of algebraic specifications. Other examples of essential categorical concepts are pushouts and free constructions, which are used in Section 3 for horizontal structuring of algebraic specifications, and categories and functors in general, which are used in Section 5 to define institutions, specification logics and general logics. These notions have been shown to be suitable to generalise and extend algebraic specifications in the classical sense towards a much more general theory suitable for a wide range of practical applications.

In fact, there is now a solid theoretical basis for algebraic specifications and also some very interesting and efficient software support for writing and evaluing formal specifications (see Section 5 in part 1).[14] But we are aware that this is only a first step towards industrial software development using algebraic specification techniques. This task needs further cooperation between academics and industry, as has started already in several ESPRIT projects.

On the other hand, algebraic specifications have provided a significant contribution to the much broader area of foundations of system specifications already. In the last section we have discussed further research topics for this most promising field within computer science.

Acknowledgements

We are grateful to all our colleagues involved in the ESPRIT-projects COMPASS, LOTOSPHERE and PROSPECTRA, and the BMFT-project KORSO, for numerous fruitful discussions on theory and application of algebraic specifications. This scientific environment is the basis for this overview paper. The referees have persuaded us to present the material in two parts, Part 1 for the informalists and Part 2 for those already convinced of formal methods. Finally we would like to thank Helga Barnewitz for excellent typing of the manuscript and diagram drawings. This acknowledgement clearly applies also to Part 1.

REFERENCES

For reasons of space in this section we have preferred to provide a very small list of references that should not be considered representative of the state of the art in the area. References have been included in this list sometimes for historical reasons and at other times because of their generality. For a more complete picture of the work in algebraic specification, the reader is addressed to Ref. 10.

1. P. Aczel, *Non-well-founded Sets*. CSLI Lecture Notes, No. 14, Stanford (1988).
2. J. Adamek, H. Herrlich and G. Strecker, *Abstract and Concrete Categories*. New York: Wiley Interscience (1990).
3. M. A. Arbib and E. G. Manes, *Arrows, Structures and Functors*. Academic Press, New York–San Francisco–London (1975).
4. R. M. Burstall and J. A. Goguen, Putting theories together to make specifications. *Proc. Int. Conf. Artificial Intelligence* (1977).
5. G. Birkhoff, On the structure of abstract algebras. *Proc. Cambridge Philos. Soc.* **31**, 433–454 (1935).
6. M. Bidoit, H.-J. Kreowski, P. Lescanne, F. Orejas and D. Sannella, *Algebraic System Specification and Development: A Survey and Annotated Bibliography*. L[ecture] N[otes in] C[omputer] S[cience] 501. Springer, Heidelberg (1991).
7. G. Birkhoff and J. D. Lipson, Heterogeneous algebras. *Journal of Combinatorial Theory* **8**, 115–133 (1970).
8. CIP Language Group, *The Munich Project CIP*, Vol. 1: *The Wide Spectrum Language CIP-L*. LNCS 183, Springer (1985).
9. CIP Language Group, *The Munich Project CIP*, Vol. 2: *The Transformation System CIP-L*. LNCS 292, Springer (1987).
10. COMPASS ESPRIT Basic Research Working Group No. 3264, *A Comprehensive Algebraic Approach to System Specification and Development*. Final Report, Technical Report University of Bremen 7/91 (1991).
11. H. Ehrig and M. Grosse-Rhode, *Structural Theory of Algebraic Specifications in a Specification Logic*, Part 1: *Functorial Parameterized Specifications*. Technical Report No. 91-23, TU Berlin, FB 20 (1991).
12. H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification 1. Equations and Initial Semantics*. EATCS Monographs on Theoretical Computer Science, Vol. 6, Springer (1985).
13. H. Ehrig and B. Mahr, *Fundamentals of Algebraic Specification 2. Module Specifications and Constraints*. EATCS Monographs on Theoretical Computer Science, Vol. 21, Springer (1990).
14. H. Ehrig, B. Mahr, F. Orejas and I. Classen, Introduction to algebraic specification part 1: formal methods for software development. *The Computer Journal* 35(5), 460–467 (1992).
15. H. Ehrig, P. Pepper and F. Orejas, *On Recent Trends in Algebraic Specification*. Proc. ICALP '89, LNCS 372, pp. 263–289 (1989).
16. W. Fey, *Pragmatics, Concepts, Syntax, Semantics, and Correctness Notions of ACT TWO: An Algebraic Module Specification and Interconnection Language*. PhD thesis TU Berlin (1988); also Technical Report No. 88-26, TU Berlin, FB 20 (1988).
17. J. A. Goguen and R. M. Burstall, *CAT, a System for the Structured Elaboration of Correct Programs from Structured Specifications*. Technical Report CSL-118, Computer Science Laboratory, SRI Int. (1981).
18. J. A. Goguen and R. M. Burstall, *Introducing Institutions*. Proc. Logics of Programming Workshop, Carnegie-Mellon. LNCS 164, Springer, pp. 221–256 (1984).
19. J. V. Guttag and J. J. Horning, *Preliminary Report on the Larch Shared Language*. Technical Report CSL 83-6, Xerox, Palo Alto (1983).
20. J. A. Goguen, J. W. Thatcher and E. G. Wagner, *An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types*. IBM Research Report RC 6487 (1976). Also: *Current Trends in Programming Methodology IV: Data Structuring* (edited R.

Yeh), pp. 80–144. Prentice-Hall, Englewood Cliffs, NJ (1978).
21. C. A. R. Hoare, Proofs of correctness of date representations. *Acta Informatica* **1**, 271–281 (1972).
22. J. Meseguer, General logics. In: *Logic Colloquium '87*, pp. 275–329. Elsevier (1989).
23. S. MacLane, *Categories for the Working Mathematician*. Springer (1972).
24. B. Mahr and J. A. Makowsky, *An Axiomatic Approach to Semantics of Specification Languages, 6*. GI-Fachtagung Theoret. Informatik, Dortmund 1983, LNCS 145, pp. 211–220.
25. B. Mahr and J. A. Makowsky, Characterizing specification languages which admit initial semantics. *Theoretical Computer Science* **31**, 49–60 (1984).
26. B. Mahr, W. Sträter and C. Umbach, *Fundamentals of a Theory of Types and Declarations*. KIT-Report 82, TU Berlin (1990).
27. F. Orejas, M. Navarro and A. Sanchez, Implementation and behaviour equivalence: A survey. To appear in *Proc. 8th Workshop on Alg. Spec.* (Dourdan, 1991), LNCS, Springer.
28. D. C. Parnas, A technique for software module specification with examples. *CACM* **15** 5 330–336 (1972).
29. D. T. Sannella and A. Tarlecki, Toward formal development of programs from algebraic specifications: implementations revisited. Extended abstract in: *Proc. Joint Conf. on Theory and Practice of Software Development, Pisa*. LNCS 249, Springer (1987), pp. 96–110; full version to appear in *Acta Informatica*.
30. J. W. Thatcher, E. G. Wagner and J. B. Wright, Data type specification: parameterization and the power of specification techniques. *10th Symp. Theory of Computing* (1978), pp. 119–132. *Trans. Prog. Languages and Systems* **4**, 711–732 (1982).
31. H. Weber and H. Ehrig, Specification of modular systems. *IEEE Transactions on Software Engineering*, **SE-12** (7), 784–798 (1986).

Journal Review

Informatization in the Public Sector,
Quarterly, Elsevier, ISSN 0925–5052

The editors, Ignace Snellen from Erasmus University, Rotterdam, and John L. King from the University of California, offer this new journal, whose first issue appeared in 1991, as a forum to focus attention on the effects of adopting Information Technology in the public sector. They describe the long history of the use of automation in all functions of public administration as being unknown and unacknowledged. The particular effect on which they wish to concentrate is inevitably that of organisational change; not only on the change that has happened but on the change, even revolution, which they say is looming even larger in the future. Their use of the term 'Informatization' covers not only the technology but also the opinions held by students of information technology.

The central field on which they intend to focus involves the possible structures of public administration and the roles of citizens as individuals and customers of services. They offer a very wide range of issues for debate, ranging from applications of emerging technologies to changing power relations between as well as inside the public and private sectors, including themes concerning experiences in developing information systems. The latter are well covered in existing international journals, some of which are well established.

The editors come from the Netherlands and the USA; their Editorial Board covers Europe, North America and Japan. Academic fields range from politics and public sector studies to information technology. There appear to be few if any representatives from the practitioner sector.

The section 'Information for authors' states that the journal's intention is to publish 'work of high quality of practical and strategic relevance to practitioners' (page 87). There is detailed advice on what and how to contribute and on the refereeing process (the conventional, long-established one). It is not clear whether the authors are seeking for contributions in the form of papers from practitioners.

It should be observed that every contribution to this issue comes from an academic. If the authors wish for a high level of interest from practitioners they must seriously ask themselves how practitioners can participate in making the journal a success. Is it to be only by reading it and applying the lessons from the research? or by sponsoring or supporting research? or could it be by making contributions themselves? The long-drawn-out process of refereeing and waiting, sometimes years, for publication is a feature of research publishing which, while intended to ensure a high quality of paper, militates against contributions from practitioners. With modern methods of electronic information processing, in whose effects in the public sector the editors are interested, it is to be hoped that new ways of communicating more rapidly between academic researchers and practitioners could be found. It is not only the journal publishers who are dragging their feet here, but the whole academic community, which has not yet come up with new methods of obtaining a sufficient number of contributions of relevant quality from practitioners, especially in journals concerned with innovation and with new technologies which develop very quickly.

Papers in the first issue address themes of information technology in the areas of social security in Sweden, experiences in US municipal information systems, social administration in Europe and a revisit to Nolan's stages theory. There is a review of the use of expert systems in social administration and some items of news, entirely concerned with new university programmes in this field.

A review of a new journal can hope to do little more than briefly describe the editors' intentions and evaluate the issue in the context of the field it is entering. This offering will overlap the fields of existing journals. However, the editors are bringing together some well-defined fields to create a new if immense area. Their readership is likely to be international, so they may well find enough readers to stay alive even in the current economic climate. A possible subject for debate might be whether their offering would be ecologically respectable if it had been presented electronically. This new journal, concentrating on the effect of information technology on informing the public sector, should be warmly welcomed.

A. LEEMING
London

Announcement

8–11 NOVEMBER 1992

CIKM-92, First International Conference on **Information and Knowledge Management,** Radisson Hotel, Baltimore, Maryland, USA
Sponsored by ISMM in cooperation with AAAI, SIGART, SIGIR, IEEE

The conference provides an international forum for presentation and discussion of research on information and knowledge management, as well as recent advances on data and knowledge bases.

The focus of the conference includes the following: application of knowledge representation techniques to semantic data modelling; development and management of heterogeneous knowledge bases; automatic acquisition of data and knowledge bases especially from raw text; object-oriented DBMS; optimisation techniques; transaction management; high-performance OLTP systems; security techniques; performance evaluation; hypermedia; unconventional applications; parallel database systems; physical and logical database design; data and knowledge sharing; interchange and inter-operability; cooperation in heterogeneous systems; domain modelling and ontology-building; knowledge discovery in databases; information storage and retrieval and interface technology.

For further information contact:
Dr Yelena Yesha, Computer Science Department, University of Maryland Baltimore County, 5401 Wilkens Avenue, Baltimore, Maryland 21228-5398. Tel: +1 410-455-3000. Fax: +1 410 455-3969. Email: cikm@cs.umbc.edu.