J. Winkowski, editor, Proc. 7th MFCS, LNCS 64, pages 155–164, Berlin, 1978. Springer–Verlag

EXTENSIONS AND IMPLEMENTATIONS OF ABSTRACT DATA TYPE SPECIFICATIONS

H.-D. Ehrich

Abteilung Informatik , Universität Dortmund
Postfach 5oo5oo, 46oo Dortmund 5o, West Germany

ABSTRACT - Equational specifications of abstract data types are related by morphisms to form a category spec of specifications. Extensions are defined to be special morphisms, and weak extensions are introduced as a generalization. On this basis, a conceptually simple but powerful notion of implementation is given. The effects of these concepts on the associated initial algebras are investigated, and it is shown that implementations can be done in multiple levels.

1. Introduction

The theory of abstract data types has recently found a lot of attention [2, 3,5,8-14,24-26] . It is motivated by the promising development of equational specification as a design tool in program construction [6,14,16,27] and by the strong relation to the field of algebraic semantics [1,7,2o,22]. The inherent limitations of the equational specification method [17,18,24] do not seem to be too restrictive for these purposes.

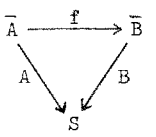
In this paper, we give a conceptually simple but powerful notion of implementation of equational specifications, making it especially easy to treat multiple levels of implementation. The necessary prerequisites are developed in the framework of a category named spec, whose objects are specifications and whose morphisms are certain pairs of functions mapping sorts and operations. Extensions of specifications (cf.[2,1o,11]) are defined to be special morphisms in spec, and a useful generalization called weak extension is introduced. The effect of the concepts on the associated initial algebras is studied throughout.

Implementations are considered here to be relations between specifications rather than between algebras [14] or between specifications and algebras [2,1o,11]. Insofar, our approach is similar to that in [6]; we do not insist, however, that implementations are morphisms, and in that way we get a simplification of both notions. With quite a different motivation, a related concept of "realization" as a relation between algebras has been considered in [21] .

We are not concerned here with the construction of models for specifications. This has been done by ADJ [1,2] with initial algebras, and there is an interesting new approach to solving Scott's domain equations [23] utilizing more general initial constructions [15]. We use, however, initial algebras to demonstrate the effects of our concepts.

2. Specifications

Let set be the category of sets with functions as morphisms. If $S \in |\underline{set}|$, let $\underline{set}_S$ be the comma category of "S-sorted sets" whose objects are functions $A:\overline{A} \to S \in \underline{set}$, S fixed, and morphisms are functions $f:\overline{A} \to \overline{B}$ leaving the sort fixed, i.e. $A = fB$, if $B:\overline{B} \to S \in \underline{set}$ (morphism composition is written from left to right) .



Let S be a given set of sorts, and let S^+ be the set of nonempty words over S. An object $\Omega \in |\underline{set}_{S^+}|$ is called a signature over S, and elements in Ω are called operations. If $x \in S^+$, $s \in S$ and $(\omega \mapsto xs) \in \Omega$, xs is the index of ω , x its domain, and s its codomain.

A signature Ω determines an endofunctor $T_\Omega:\underline{set}_S \to \underline{set}_S$ sending an S-sorted set of variables $X \in |\underline{set}_S|$ to the S-sorted set XT_Ω of all Ω-terms over X. Morphisms are sent to the corresponding variable substitutions. Given $Y \in |\underline{set}_S|$, the elements of an S-sorted set $E \subset YT_\Omega \times YT_\Omega$ are called Ω-equations.

Definition 2.1: A specification is a triple $D = \langle S, \Omega, E \rangle$ where S is a set of sorts, Ω is a signature over S, and E is an S-sorted set of Ω-equations.

With each specification D, its initial algebra init D can be associated uniquely (up to isomorphism). For details of the initial algebra construction and its mathematical background, we refer to the literature [1,2,9,19] . Here we use the following notation: Ω-alg denotes the category of all Ω-algebras and ΩE-alg its full subcategory of all Ω-algebras satisfying the equations E. $T_\Omega^E:\underline{set}_S \to \underline{set}_S$ is the endofunctor sending an S-sorted set of variables $X \in |\underline{set}_S|$ to the equivalence class XT_Ω^E of all Ω-terms in XT_Ω identified by E. Morphisms are sent to the corresponding variable substitutions. Thus, $\emptyset T_\Omega^E$ is the S-sorted carrier set of (the usual construction of) the initial algebra init D .

We give some examples using the notation of Guttag [14]. Thus, $\omega:s_1 \times \ldots \times s_p \to s_{p+1}$ means that $s_1 \ldots s_p s_{p+1}$ is the index of the operation ω. Signatures and equations are separated by horizontal lines.

Example 2.2: Very basic specifications are the following:

D_b true: $\longrightarrow$ bool
false: $\longrightarrow$ bool

D_n 0: $\longrightarrow$ nat
succ: nat $\longrightarrow$ nat

Clearly, initD$_b$ is a two-element set, and init D_n is isomorphic to the set of natural numbers generated by the constant 0 and the successor function.

Example 2.3: D_{nb} is obtained by D_b and D_n adding the following items:

eq : nat $\times$ nat $\longrightarrow$ bool
if-then-else-fi : bool $\times$ nat $\times$ nat $\longrightarrow$ bool

eq(0 , 0) = true
eq(0, succ(n)) = false
eq(succ(n), 0) = false
eq(succ(n),succ(m))=eq(n,m)

if true then n else m fi = n
if false then n else m fi = m

The initial algebra init D_{nb} has init D_b and init D_n as subalgebras, connected by an equality test and a branching operation.

Example 2.4: D_a is obtained from D_{nb} by adding the following items:

new : $\longrightarrow$ array
store : array $\times$ nat $\times$ nat $\longrightarrow$ array
read : array $\times$ nat $\longrightarrow$ nat
undef? : array $\times$ nat $\longrightarrow$ bool

read (new,n) = 0
read(store(a,n,m),p) = if eq(n,p) then m else read(a,p) fi
undef?(new,n) = true
undef?(store(a,n,m),p) = if eq(n,p) then false else undef?(a,p) fi

init D_a behaves like an array with indexes and entries in init D_{nb} .

Example 2.5: D_s is obtained from D_a by adding the following items:

create : $\longrightarrow$ stack
push : stack $\times$ array $\longrightarrow$ stack
pop : stack $\longrightarrow$ stack
top : stack $\longrightarrow$ array

pop(create) = create
pop(push(s,a)) = s
top(create) = new
top(push(s,a)) = a

In the first two equations, usually errors are introduced. We avoid to do so in order to keep the examples small and complete. Equational specification of error handling is treated in [2]. init D_s is a stack whose entries are arrays taken from init D_a .

Relationships between different specifications are generally given by relationships between their sorts, signatures, and equations. Let $D_i=\langle S_i,\Omega_i,E_i\rangle$, i=0,1 , be specifications. The sorts are related by a mapping $h:S_0\longrightarrow S_1$. Let $h^+:S_0^+\longrightarrow S_1^+$ be the string homomorphism given by h. Then we can relate operation symbols by a mapping $g:\overline{\Omega}_0\longrightarrow\overline{\Omega}_1$, where $\overline{\Omega}_i$, i=0,1 , are the

domains of Ω_i , such that $\Omega_o h^+ = g\Omega_1$. Thus, g may be considered to be a morphism $g: \Omega_o h^+ \longrightarrow \Omega_1 \in \underline{set}_{S_1^+}$.

Given h and g , we can define a morphism $X\hat{f} : XT_{\Omega_o} h \longrightarrow XhT_{\Omega_1} \in \underline{set}_{S_1}$, for each $X \in |\underline{set}_{S_o}|$, by (1) $x \mapsto x$ and (2) $[t_1,\ldots,t_p]\omega \mapsto [t_1(X\hat{f}),\ldots,t_p(X\hat{f})](\omega g)$, disregarding the sorts. We thus have a natural transformation

$$\hat{f} : T_{\Omega_o} h \xrightarrow{\cdot} h T_{\Omega_1} \quad .$$

Definition 2.6: The category spec has specifications $D=\langle S,\Omega,E\rangle$ as objects, and its morphisms $f:D_o \longrightarrow D_1$ are pairs of mappings $f=(h,g)$, $h:S_o \longrightarrow S_1$, $g:\overline{\Omega}_o \longrightarrow \overline{\Omega}_1$, such that (1) $\Omega_o h^+ = g\Omega_1$ and (2) init D_1 satisfies the equations $E_o(Y_o\hat{f} \times Y_o\hat{f})$.

Here, Y_o is the variable set underlying E_o, and $E_o(Y_o\hat{f} \times Y_o\hat{f}) = \{(t(Y_o\hat{f}),t'(Y_o\hat{f})) \mid (t,t') \in E_o\}$. Specification morphisms correspond to simple instances of theory morphisms in [6] , and a similar notion has been defined (but not utilized) in [13] .

Each morphism $f \in \underline{spec}$ determines a natural transformation $\hat{f}$, as shown above. Because of property (2) in the above definition, we get another natural transformation by factorization :

$$\tilde{f} : T^{E_o}_{\Omega_o} h \xrightarrow{\cdot} h\, T^{E_1}_{\Omega_1} \quad .$$

3. Extensions

Definition 3.1 : A morphism $f=(h,g) \in \underline{spec}$ is called an embedding iff h and g are injective. f is called an Ω-embedding iff h is bijective and g is injective.

Ω-embeddings are called enrichments in [2,10,11]. It is easily seen that embeddings are monomorphisms in spec, but the converse is not true. In the present paper, we will only consider embeddings, general morphisms are used in [9].

If $f \in \underline{spec}$, let init D_1^o be the subalgebra of init D_1 induced by the sorts $S_o h$ and the operations $(\Omega_o h^+)g$. Clearly, if f is an embedding, there is an algebra in Ω_o-alg that equals init D_1^o modulo renaming of sorts and operations. This algebra will be identified with init D_1^o . The next theorem relates embeddings to morphisms on the associated initial algebras.

Theorem 3.2: Let $f:D_o \longrightarrow D_1$ be an embedding. Then, $\emptyset\tilde{f}$ is an Ω_o-algebra morphism: init $D_o \longrightarrow$ init D_1^o .

The proof is straightforward by the definitions of $\hat{f}$ and $\tilde{f}$. Conversely, if there is an algebra morphism φ : init $D_o \longrightarrow$ init D_1^o , the latter must be of

the same algebraic type as the first one, i.e. injective mappings h,g on the sorts and operations are assumed such that $\Omega_0 h^+ = g\Omega_1$, and clearly f=(h,g) is a morphism in spec.

Let $X_0 \in |\underline{set}_{S_0}|$, $X_1 := X_0 h$, and $T_i := X_i T^{E_i}_{\Omega_i}$ for i=0,1.
Furthermore, let T^0_1 be the subset of all terms in T_1 with a sort in $S_0 h$. Since $T_0(X_0 \tilde{f}) \subset T^0_1$, $X_0 \tilde{f}$ decomposes into $X_0 \tilde{f}^0 : T_0 \to T^0_1$ and the inclusion $T^0_1 \hookrightarrow T_1$.

Definition 3.3: An embedding $f \in$ spec is called full iff $X\tilde{f}^0$ is surjective for each $X \in |\underline{set}_{S_0}|$. f is called s-full iff $\emptyset\tilde{f}^0$ is surjective. f is called true iff $X\tilde{f}^0$ is injective for each X.

True and s-full embeddings correspond to consistent resp. sufficiently complete specifications in [14]. It is immediately verified that, if f and g are (s-)full (true) embeddings, so is fg.

Definition 3.4: A morphism $f \in$ spec is called an (Ω-)extension iff f is an s-full and true (Ω-)embedding. f is called a weak (Ω-)extension iff f is an s-full (Ω-)embedding.

Referring to examples 2.2 to 2.5, the inclusions $D_b \hookrightarrow D_{nb}$, $D_n \hookrightarrow D_{nb}$, $D_b \hookrightarrow D_a$ and $D_a \hookrightarrow D_s$ are easily seen to be extensions. If we add, for example, the equation top(create)=top(push(s,a)) to D_s , we still have a weak extension $D_a \hookrightarrow D_s$. If we drop, however, the equation top(create)=new in D_s , we have no longer a weak extension.

For the following theorem, let $f: D_0 \to D_1$ be an embedding. Then, by theorem 3.2 , $\emptyset\tilde{f}$: init $D_0 \to$ init D^0_1 is an Ω_0-algebra morphism. If f is an Ω-embedding, the notation init $D^0_1 \sim$ init D_1 means that each term in T_1 can be reduced to an equivalent term in T_1 having operations in Ω_0 only (modulo renaming). This is, for example, the case if the operations in $\Omega_1 - \Omega_0$ can be expressed by some terminating recursion schemes using operations in Ω_0 .

Theorem 3.5: (1) If f is a weak extension, $\emptyset\tilde{f}$ is an epimorphism.
(2) If f is an extension, $\emptyset\tilde{f}$ is an isomorphism.
(3) If f is a weak Ω-extension, we have init $D^0_1 \sim$ init D_1 .

The proof is immediate from the definitions. The theorem shows that our notion of extension coincides with that given in [2,10,11] .

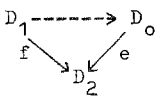
4. Implementations

Roughly speaking, a specification D_1 implements a specification D_0 if the operations in D_0 can be associated with derived operations in D_1 realizing

the behaviour expressed in the equations of D_o. If we add new operation symbols for the derived operations and corresponding defining equations to D_1, we get another specification D_2, and there are obvious morphisms from both D_o and D_1 to D_2.

Definition 4.1: A specification D_1 implements a specification D_o iff there are (1) a specification D_2, (2) an extension $e: D_o \longrightarrow D_2$ and (3) a weak Ω-extension $f: D_1 \longrightarrow D_2$.

If D_1 implements D_o, we use the notation $D_1 \dashrightarrow D_o$. The definition is illustrated by the opposite diagram. Please note that our definition of implementation is general enough to include (1) arbitrary terminating recursion schemes for specifying the derived operations, and (2) identification of (derived) operations that are different in D_1, by allowing f to be weak. This covers cases where one operation in D_o is implemented by different derived operations in D_1 (there are, for example, two different zeroes in one's complement binary integer arithmetic).



Example 4.2: We give a simplified version of Guttag's symbol table [14] (cf. also [10,11]) and implement it by the specification D_s of example 2.5 (stack of array of nat and nat). We assume that identifiers and attributes have already been implemented by nat.

D_{sy} is obtained from D_{nb} by the following extension:

init: $\longrightarrow$ Sytb	add : Sytb × nat × nat $\longrightarrow$ Sytb
begin: Sytb $\longrightarrow$ Sytb	retrieve: Sytb × nat $\longrightarrow$ nat
end: Sytb $\longrightarrow$ Sytb	

end(init) = init	retrieve(init,i) = 0
end(begin(s)) = s	retrieve(begin(s),i) = retrieve (s,i)
end(add(s,i,a))=end(s)	retrieve(add(s,i,a),j)=if eq(i,j) then a else retrieve(s,j) fi

In order to implement D_{sy} by D_s, we specify D_2 as follows: D_2 consists of D_s plus the following operations and equations:

init': $\longrightarrow$ stack	add': stack × nat × nat $\longrightarrow$ stack
begin': stack $\longrightarrow$ stack	retrieve': stack × nat $\longrightarrow$ nat
end': stack $\longrightarrow$ stack	

init'=create	end'(s)=pop(s)
begin'(s)=push(s,new)	add'(s,i,a)=push(pop(s),store(top(s),i,a))
retrieve'(create,i)=0	
retrieve'(push(s,a),i)=if undef?(a,i) then retrieve'(s,i) else read(a,i) fi	

Clearly, D_2 is an Ω-extension (and thus a weak one) of D_s. We define $e=(h,g):D_{sy} \longrightarrow D_2$ by

h: $\underline{\text{Sytb}} \longmapsto \underline{\text{stack}}$, $\sigma \longmapsto \sigma$ for all other sorts σ

g: $\omega \longmapsto \omega'$, where $\omega = \omega'$ for the operations in D_{nb} .

The correctness proof for this implementation consists of showing that e is an extension. The main part is to show that e is a morphism, i.e. that the equations of D_{sy} carry over to valid equations in $\underline{\text{init}}\ D_2$. These validity proofs can easily be drawn from [14].

The next theorem gives the effect of our notion of implementation on the initial algebras.

Theorem 4.3: If D_1 implements D_o, there is an epimorphism $\varphi \in \underline{\Omega_1\text{-alg}}$ and an isomorphism $\psi \in \underline{\Omega_o\text{-alg}}$ with the following properties:

$$\underline{\text{init}}\ D_1 \xrightarrow{\varphi} \underline{\text{init}}\ D_2^1 \sim \underline{\text{init}}\ D_2 \supset \underline{\text{init}}\ D_2^o \xrightarrow{\psi} \underline{\text{init}}\ D_o$$

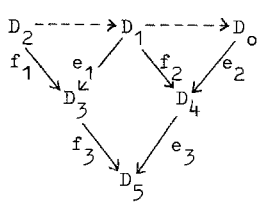
The proof is immediate from the definition of implementation and theorem 3.5. Another simple consequence is:

Theorem 4.4: If $e:D_o \longrightarrow D_1$ is an extension, each implementation of D_1 implements D_o , too .

This follows from the fact that the composition of extensions is again an extension. Our last theorem shows that implementations can be done in multiple levels.

Theorem 4.5: If D_2 implements D_1 and D_1 implements D_o , then D_2 implements D_o .

Proof: Since $D_2 \dashrightarrow D_1$ and $D_1 \dashrightarrow D_o$, there are specifications D_3 and D_4 as well as weak Ω-extensions f_1, f_2 and extensions e_1, e_2 , as shown in the diagram. In order to simplify notation, let these morphisms be inclusions, i.e. $S_o \subset S_4$; $S_1 = S_4$; $S_1 \subset S_3$; $S_2 = S_3$; $\Omega_o, \Omega_1 \subset \Omega_4$; $\Omega_1, \Omega_2 \subset \Omega_3$. Let $\Omega_1 = \Omega_3 \cap \Omega_4$ (if this not the case, we can achieve it by renaming the operation symbols). Now, referring to the diagram, we construct D_5, e_3 and f_3 as follows: $S_5 := S_3$, $\Omega_5 := \Omega_3 \cup \Omega_4$, $E_5 := E_3 \cup E_4$. f_3 and e_3 are defined to be the obvious inclusions. Clearly, f_3 is an Ω-embedding, and e_3 is an embedding. We must show that f_3 is an Ω-extension and e_3 is an extension. The theorem then follows easily from the fact that the composition of (Ω-)extensions is again an (Ω-)extension: D_5 is the specification and $f_1 f_3$, $e_2 e_3$ are the morphisms required by the definition of implementation.



According to definition 3.4 , we have to show that e_3 and f_3 are s-full and that e_3 is true. We give an informal proof.

In order to prove that e_3 is s-full, let t be an arbitrary term in $\emptyset T^4_{\Omega_5}$, i.e. a term in $\emptyset T_{\Omega_5}$ with a sort in S_4 . If t contains operations from Ω_3-Ω_1 , they can be removed by bottom-up reductions in $\underline{\text{init}}\ D_3$ and $\underline{\text{init}}\ D_4$, since e_1 and f_2 are s-full. Thus, t is reduced in init D_5 to a term t' in $\emptyset T_{\Omega_4}$. That f_3 is s-full folows from the symmetry of the situation.

In order to prove that e_3 is true let t,t' be terms in XT_{Ω_4} , for some $X \in |\underline{\text{set}}_{S_4}|$, that are not equivalent in $\underline{\text{init}}\ D_4$. The only way t and t' could be identified in $\underline{\text{init}}\ D_5$ is by equations in E_3 identifying some subterms of t and t'. But this is impossible since e_1 is true. ///

The construction of D_5,e_3,and f_3 in the above proof is an instance of a pushout construction in the category spec. It can be shown that spec is cocomplete [9] , thus pushouts exist in general. Moreover, there are some results how relevant properties of morphisms carry over to opposite sides of a pushout diagram. Using these results, the above theorem is simply proven by letting (e_3,f_3) be the pushout of (e_1,f_2).

5. Conclusions

We have defined a notion of implementation of abstract data type specifications modelling the translation from one (abstract) level of description to another (more concrete) one. Theorem 4.5 shows that the composition of implementations can be carried out effectively. Thus, the correctness of a multi-level implementation follows from the correctness of each single step.

From a practical viewpoint, it is desirable to generalize the notion of implementation by dropping the requirement that e be s-full. Theorem 4.3 then still holds with the modification that Ψ is a monomorphism from $\underline{\text{init}}\ D_0$ to $\underline{\text{init}}\ D^o_2$. We thus allow for some redundancy in the implementation, i.e. not every item in the implementation need have an interpretation. In [9] we show that theorem 4.5 then still holds, but there is no longer an effective construction of the overall implementation unless we allow for partially defined derived operations.

Another important concept in the realm of specification is that of parametrization. Different approaches to this problem can be found in [9,15,24]. As in the case of implementation, we care much about a model-independent notion of parametrization, defined on specifications only. In [9] , a parametric specification is defined to be an inclusion $F \hookrightarrow P$ in spec, where F is the "formal part" of the specification. Parameter assignments are morphisms with source F, and parameter substitution is defined utilizing pushouts in spec .

Acknowledgements

I am very much indepted to G.Dittrich, H.Ehrig, V.Lohberger, W.Merzenich, and A.Poigné for helpful discussions on the subjects of this paper.

References

1. ADJ (J.A.Goguen,J.W.Thatcher,E.G.Wagner,J.B.Wright): Initial algebra semantics and continuous algebras. J.ACM 24(1977), 68-95

2. ADJ (GTWa): An initial algebra approach to the specification, correctness, and implementation of abstract data types. Current trends in progr. methodology IV, ed. by R.Yeh, Prentice Hall, New Jersey

3. ADJ (TWaWr): Specification of abstract data types using conditional axioms (extended abstract). IBM Res. Report RC-6214, 1976

4. Arbib,M.A.-Manes,E.G.: Arrows, structures, and functors. Academic Press, New York 1975

5. Bergstra,J.: What is an abstract data type? Report No. 77-12, Inst. of Appl. Math. and Comp. Sc., Univ. of Leiden, The Netherlands, 1977

6. Burstall, R.M.-Goguen,J.A.: Putting theories together to make specifications. Proc. 5th IJCAI-77, MIT, Cambridge, Mass. 1977

7. Ehrich,H.-D.: Algebraic semantics of type definitions and structured variables. Proc. FCT'77, ed. by M.Karpiński, Lecture Notes in Comp. Sc. 56, Springer-Verlag, Berlin 1977

8. Ehrich,H.-D.: Algebraische Spezifikation von Datenstrukturen. Proc. Workshop "Graphentheoretische Konzepte in der Informatik", ed. by J.Mühlbacher, Hanser-Verlag, München 1977

9. Ehrich,H.-D.: On the theory of specification, parametric specification, and implementation of abstract data types. To be published

10. Ehrig,H.-Kreowski,H.-J.-Padawitz,P.: Some remarks concerning correct specification and implementation of abstract data types. Bericht Nr. 77-13, Techn. Univ. Berlin, FB 20, 1977

11. Ehrig,H.-Kreowski,H.-J.-Padawitz,P.: Stepwise specification and implementation of abstract data types. Internal Report, Techn. Univ. Berlin, FB 20, 1977

12. Giarratana,V.-Gimona,F.-Montanari,U.: Observability concepts in abstract data type specification. Proc. 5th MFCS, ed. by A. Mazurkiewicz, Lecture Notes in Comp. Sc. 45, Springer-Verlag, Berlin 1976, 576-587

13. Goguen,J.A.: Correctness and equivalence of data types. Proc. 1975 Conf. on Algebraic Systems, Lecture Notes in Comp. Sc., Springer-Verlag, Berlin 1975

14. Guttag,J.V.: The specification and application to programming of abstract data types. Tech. Report CSRG-59, Univ. of Toronto, 1975

15. Lehmann,D.J.-Smith,M.B.: Data types. The University of Warwick, Theory of Computation Report No. 19, May 1977

16. Liskov,B.H.-Zilles,S.N.: Specification techniques for data abstractions. IEEE Transactions on Software Engineering, Vol. SE-1 (1975) , 7-19

17. Majster,M.E.: Data types, abstract data types and their specification problem. Report TUM-INFO-7740, Techn. Univ. München, 1977

18. Majster,M.E.: Limits of the algebraic specification of data types. SIGPLAN Notices 12 (1977), 37-42

19. Manes,E.G.: Algebraic theories. Springer-Verlag, New York 1976

20. Mosses,P.: Making denotational semantics less concrete. Proc. Bad Honnef Workshop on Semantics of Programming Languages, Bericht Nr. 41, Abteilung Informatik, Univ. Dortmund, 1976

21. Reusch,B.: Zur Realisierung von Automaten über Algebren. Berichte der GMD Band 38, Bonn 1971

22. Reynolds,J.C.: Semantics of the domain of flow diagrams. J.ACM 24 (1977), 484-503

23. Scott,D.S.: Data types as lattices. SIAM J. of Computing, Vol.5 (1976), 522-587

24. Thatcher,J.W.-Wagner,E.G.-Wright,J.B.: Data type specification: parametrization and the power of specification techniques. (Extended abstract). Internal Report, Yorktown Heights 1977

25. Wand,M.: First-order identities as a defining language. Tech. Report No. 29, Comp. Sc. Dept. Indiana University, Bloomington 1976

26. Wand,M.: Efficient axioms for algebra semantics. Tech. Report No. 42, Comp. Sc. Dept. Indiana University, Bloomington 1975

27. Zilles,S.N.: Algebraic specification of data types. MIT Project MAC, Computation Structures Group Memo 119, 1975