EATCS Bulletin 13, Feb. 1981, pp 45-51

Implicit Specification by Algebraic Domain Equations

H.-D. Ehrich

Abt. Informatik, Univ. Dortmund, PF 500500, D-4600 Dortmund 50

1. INTRODUCTION

Scott's order theoretic approach to data types (SC 71 ff) offers a characteristic specification method: there are given basic types like BOOL, INT, etc., and there is a fixed set of type constructors like sum, product, function space, and powerdomain. Data types are then defined by so-called domain equations that may be recursive, e.g. (cf. SC 72a, SP 77) $N \cong N+1$ (natural numbers), $L \cong (L \times L)+D$ (list structures over D), $F \cong [F \to F]$ (a pure λ-calculus model), $S \cong S \times D+1$ (stacks over D), etc.

In contrast, the algebraic approach to data types has hardly more to offer than explicit specifications: one has to give all sorts, all operations, and all axioms that describe the behaviour of the operations (cf. ADJ 78).

Recently, an algebraic analogon to type constructors, called parameterized data types, has been investigated (BG 77, EH 79, EL 79a+b, BG 80, EKTWW 80a+b). So, in a sense, algebraic data types can be specified by expressions involving parameterized data types applied to actual parameters. The question naturally arises whether a meaningful and useful analogon to recursive domain equations can be established in the algebraic framework. This paper shows that the answer is positive.

The positive answer presented here is inspired by the categorical versions of recursive domain equations (WA 75, LE 76, LS 77, SP 77). Thus, fixpoints of functors play an important role.

2. FUNDAMENTAL NOTIONS

A <u>signature</u> is a quadruple $\Sigma=\langle S,\Omega,arity,sort\rangle$, where S and Ω are sets of <u>sorts</u> and <u>operators</u>, respectively, and $arity:\Omega \longrightarrow S^*$, $sort:\Omega \longrightarrow S$ are mappings. We will write $\Sigma=\langle S,\Omega\rangle$ for short, assuming tacitly the existence of the arity and sort mappings. A <u>signature morphism</u> $f:\Sigma \longrightarrow \Sigma'$ is a pair of mappings $f=\langle f_s:S \longrightarrow S', f_\omega:\Omega \to \Omega'\rangle$ such that $arity(\omega f_\omega)=arity(\omega)f_s$ and $sort(\omega f_\omega)=sort(\omega)f_s$. For convenience, we often omit the index, writing f for f_s or f_ω.

Algebras are interpretations of signatures: a <u>Σ-algebra</u> A is an S-indexed family of sets, $\{s_A\}$, the <u>carrier</u> of A, together with an Ω-indexed family of mappings, $\{\omega_A:arity(\omega)_A \longrightarrow sort(\omega)_A\}$, the <u>operations</u> of A (if $x=s_1 s_2 \ldots s_n \in S^*$, x_A denotes the cartesian product $s_{1,A} \times \ldots \times s_{n,A}$). A <u>$\Sigma$-algebra morphism</u> $\varphi:A \longrightarrow B$ is an S-indexed family of mappings $\varphi_s:s_A \longrightarrow s_B$ such that, for each operator $\omega \in \Omega$ with arity x and sort s, we have $\omega_A \varphi_s = \varphi_x \omega_B$. Here, $\varphi_x=\varphi_{s_1} \times \ldots \times \varphi_{s_n}$ if $x=s_1 \ldots s_n$. The class of all Σ-algebras with all Σ-algebra morphisms forms a category <u>Σ-alg</u>. It is well known that <u>Σ-alg</u> has an <u>initial algebra</u> I_Σ, having a unique morphism to any other algebra in <u>Σ-alg</u>.

If $f:\Sigma\rightarrow\Sigma'$ is a signature morphism, there is a corresponding forgetful functor f-alg:Σ'-alg$\rightarrow\Sigma$-alg sending each Σ'-algebra B to that Σ-algebra A such that $s_A=(sf)_B$ and $\omega_A=(\omega f)_B$.

Let $\Sigma=\langle S,\Omega\rangle$ be a signature. A Σ-equation is a triple $\langle X,\tau_1,\tau_2\rangle$ where X is an S-indexed family of sets (of variables), and τ_1,τ_2 are terms over X and Ω of the same sort, called the sort of the equation. A Σ-algebra A satisfies a Σ-equation $\langle X,\tau_1,\tau_2\rangle$ iff the formula $\forall X: \tau_1=\tau_2$ is true when interpreted in A in the obvious way.

A specification is a pair $\underline{D}=\langle\Sigma,E\rangle$ where Σ is a signature and E is an S-sorted set of Σ-equations. If $\Sigma=\langle S,\Omega\rangle$, we sometimes write $\underline{D}=\langle S,\Omega,E\rangle$ instead of $\underline{D}=\langle\Sigma,E\rangle$. A Σ-algebra satisfies a specification $\underline{D}=\langle\Sigma,E\rangle$ iff it satisfies each equation in E. Σ-algebras satisfying a specification $\underline{D}$ will be called $\underline{D}$-algebras. The full subcategory of Σ-alg consisting of all $\underline{D}$-algebras is denoted by $\underline{D}$-alg . $\underline{D}$-alg, too, has an initial algebra, denoted by I_D .

A specification morphism $f:\underline{D}\rightarrow\underline{D}'$ is a signature morphism $f:\Sigma\rightarrow\Sigma'$ such that the corresponding forgetful functor f-alg:Σ'-alg$\rightarrow\Sigma$-alg sends each $\underline{D}'$-algebra to a $\underline{D}$-algebra. Thus, a specification morphism defines a forgetful functor also denoted by f-alg, but with $\underline{D}'$-alg and $\underline{D}$-alg as domain and range, respectively, obtained by restricting f-alg to $\underline{D}'$-alg. The class of all specifications together with all specification morphisms forms a category denoted by spec. It is not difficult to show that spec has all colimits, i.e. spec is cocomplete.

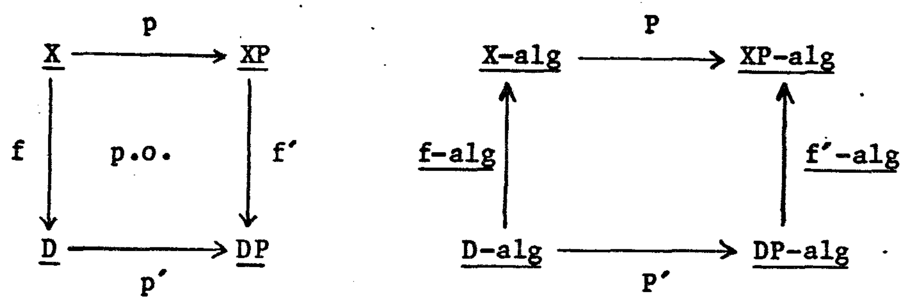
Let $f:\underline{D}\rightarrow\underline{D}'$ be a specification morphism. A functor F:$\underline{D}$-alg $\longrightarrow$ $\underline{D}'$-alg is called strongly persistent with respect to f iff F.f-alg is the identity on $\underline{D}$-alg. The following lemma is proven in EKTWW 80a+b:

Extension lemma: Let the pushout in figure 2.1(a) be given. Let P:$\underline{X}$-alg $\longrightarrow$ $\underline{XP}$-alg be strongly persistent wrt p . Then there is exactly one functor P':$\underline{D}$-alg$\longrightarrow$$\underline{DP}$-alg , called the extension of P via f , that is strongly persistent wrt p' and satisfies f-alg.P = P'.f'-alg .

Associated with a specification morphism $f:\underline{D}\text{—}\underline{D}'$, there is a functor f-free:$\underline{D}$-alg$\longrightarrow$$\underline{D}'$-alg that is left adjoint to f-alg. f-free sends each $\underline{D}$-algebra A to the free $\underline{D}'$-algebra over A (wrt f). Also from EKTWW 80a+b, we have the following:

Extension lemma supplement: If, in addition, P $\cong$ p-free , then we have P' $\cong$ p'-free .

A parameterized specification is an injective spec morphism $p:\underline{X}\rightarrow\underline{XP}$. $\underline{X}$ is the formal parameter of p. A parameterized data type is a pair (p,P) where $p:\underline{X}\rightarrow\underline{XP}$ is a parameterized specification, and P:$\underline{X}$-alg$\longrightarrow$$\underline{XP}$-alg is a functor. (p,P) is called strongly persistent iff P is strongly persistent wrt p. We call p strongly persistent iff (p,p-free) has this property. In the case of strong persistency, parameter passing works as follows. Given $p:\underline{X}\rightarrow\underline{XP}$, an actual parameter for p is a pair $(f,\underline{D})$ where $\underline{D}$ is a specification and $f:\underline{X}\rightarrow\underline{D}$ is a spec morphism. Let (p',f') be the pushout of p and f (cf. fig. 2.2). Let P' be the unique extension of P via f .



(a) (b)

Figure 2.1

Then (p',P') is a strongly persistent parameterized data type, also called the extension of (p,P) via f. Now, each actual parameter D-algebra A is sent to AP'. In this way, (p,P) works as a data type constructor, operating on various actual parameter algebras and preserving their structure.

Example 2.1: The "n-product with constant" $\underline{X}_1\times\ldots\times\underline{X}_n + \underline{1}$ is the embedding $p:\underline{X}\hookrightarrow\underline{XP}$, where $\underline{X}$ and $\underline{XP}$ are defined as follows $(i=1,\ldots,n)$.

X

sorts $X_1,\ldots,X_n$,

ops $\overline{x}_i : \longrightarrow X_i$

$\equiv_i : X_i\times X_i \longrightarrow BOOL$

eqs $\overline{x}_i \equiv_i \overline{x}_i = true$

rest of XP

P

$\overline{p} : \longrightarrow P$

$<\_,\ldots,\_> : X_1\times\ldots\times X_n \longrightarrow P$

$\_(i) : P \longrightarrow X_i$

$\equiv : P\times P \longrightarrow BOOL$

$\overline{p}(i) = \overline{x}_i$

$<x_1,\ldots,x_n>(i) = x_i$

$\overline{p} \equiv \overline{p} = true$

$\overline{p} \equiv <x_1,\ldots,x_n> = false$

$<x_1,\ldots,x_n> \equiv \overline{p} = false$

$<x_1,\ldots,x_n> \equiv <x_1',\ldots,x_n'> = x_1 \equiv x_1' \wedge\ldots\wedge x_n \equiv x_n'$

Consider (p,p-free) for n=2. Let D be a specification of the integers (sort INT) and booleans (sort BOOL). Let $f:\underline{X}\longrightarrow\underline{D}$ be defined by $X_1 \mapsto BOOL$, $X_2 \mapsto INT$, $\overline{x}_1 \mapsto false$, $\overline{x}_2 \mapsto 0$, $\equiv_i \mapsto$ identity on BOOL or INT, respectively, i=1,2 . Then, the initial boolean-integer algebra (D-algebra) is sent to the algebra of (booleean,integer)-pairs with one additional constant, all other operations of XP, and all boolean and integer operations on the components retained. The specification of this algebra is DP, the pushout object of p and f. It is obtained from XP by substituting BOOL for X_1, false for x_1, INT for X_2, 0 for x_2, and the respective identities for $\equiv_i$. Therefore, a suggestive notation for DP is $\underline{BOOL}\times\underline{INT} + \underline{1}$.

3. ALGEBRAIC DOMAIN EQUATIONS

In the order theoretic approach to data types, parameterized data types can be viewed as functors, domain equations as endofunctors, and their solutions as fixpoints of functors (WA 75, LE 76, SP 77).

In the algebraic approach, parameterized data types are essentially functors, too, but most often they are not endofunctors. Typically, the signatures of actual parameter and resultant algebras are different. In order to get endofunctors, we define algebraic domain equations to consist of a parameterized data type and a functor in the reverse direction. We restrict ourselves to free and strongly persistent parameterizations of the form (p,p-free) and to algebraic reverse functors of the form e-alg for some spec morphism e.

Definition 3.1: An algebraic domain equation is a pair of spec morphisms (p,e) , $p,e:\underline{X}\longrightarrow\underline{XP}$, such that p is a strongly persistent parameterized specification.

Let P=p-free, $\bar{P}$=p-alg, and $\bar{E}$=e-alg . There are two endofunctors, namely $P\bar{E}$ on X-alg and $\bar{E}P$ on XP-alg. A fixpoint is an object that is sent to an isomorphic one. It is immediate to see that the fixpoints of $P\bar{E}$ and $\bar{E}P$ are very closely related: A is a fixpoint of $P\bar{E}$ iff AP is a fixpoint of $\bar{E}P$, and vice versa.

For the definition of what we mean by a solution of an algebraic domain equation, we make use of the following result. Let (q,$\underline{Q}$) be the coequalizer in spec of p and e,

$$\underline{X} \underset{e}{\overset{p}{\rightrightarrows}} \underline{XP} \xrightarrow{\ q\ } \underline{Q} \quad ,$$

and let $\bar{Q}$=q-alg .

Theorem 3.2: If B is a fixpoint of $\bar{E}P$, then there is a unique (up to isomorphism) Q-algebra C such that $B=C\bar{Q}$.

It is convenient not to take fixpoints of $P\bar{E}$ or $\bar{E}P$ as solutions, but these uniquely associated Q-algebras.

Definition 3.3: A solution of an algebraic domain equation (p,e) is a Q-algebra C such that $C\bar{Q}$ is a fixpoint of $\bar{E}P$.

The main result can now be stated as follows.

Theorem 3.4: The initial Q-algebra I_Q is a solution of (p,e).

The proof is rather involved and requires some more technical machinery to be developed. It will be published elsewhere. Clearly, I_Q is an initial solution, i.e. it is initial in the full subcategory of Q-alg of all solutions of (p,e). A specification of I_Q is $\underline{Q}$, the coequalizer object of p and e. There is a simple construction for $\underline{Q}$, given p and e, based on the coequalizer construction in set applied to sorts and operators.

Example 3.5: Consider the n-product with constant from example 2.1 for n=1. Let

$$\underline{X} \underset{e}{\overset{p}{\rightrightarrows}} \underline{X} + \underline{1}$$

be defined by taking p as in example 2.1 , and e sending sort X_1 to P, $\bar{x}_1$ to $\bar{p}$, and $\equiv_1$ to $\equiv$. Then the solution is I_Q where Q is the following specification obtained from $\underline{X} + \underline{1}$ by identifying X_1 and P, $\bar{x}_1$ and $\bar{p}$, and $\equiv_1$ and $\equiv$. For convenience, we rename P by N, $\bar{p}$ by 0, <_> by succ, and _(1) by pred.

<u>sorts</u> N

<u>ops</u>
$0: \longrightarrow N$
$succ: N \longrightarrow N$
$pred: N \longrightarrow N$
$\equiv : N \times N \longrightarrow BOOL$

<u>eqs</u>
$$\begin{aligned} pred(0) &= 0 \\ pred(succ(n)) &= n \\ 0 \equiv 0 &= true \\ 0 \equiv succ(n) &= false \\ succ(n) \equiv 0 &= false \\ succ(n) \equiv succ(m) &= n \equiv m \end{aligned}$$

This is a specification of the natural numbers.

<u>Example 3.6</u>: Let the above specification be $\underline{N}$. Let $\underline{X} \times \underline{N} + \underline{1}$ be the parametric specification obtained from $\underline{X}_1 \times \underline{X}_2 + \underline{1}$ (example 2.1) by parameter passing with actual parameter f, where f sends X_1 to X (forgetting the index 1), X_2 to N, x_2 to 0, and $\equiv_2$ to $\equiv$. Then, the algebraic domain equation

$$\underline{X} \underset{e}{\overset{p}{\rightrightarrows}} \underline{X} \times \underline{N} + \underline{1}$$

has stacks as solutions, specified by the following specification (with obvious renamings):

<u>sorts</u> S,N

<u>ops</u>
$empty: \longrightarrow S$
$push : S \times N \longrightarrow S$
$pop : S \longrightarrow S$
$top : S \longrightarrow N$
$\equiv : S \times S \longrightarrow BOOL$
... (ops from N)

<u>eqs</u>
$$\begin{aligned} pop(empty) &= empty \\ top(empty) &= 0 \\ pop(push(s,n)) &= s \\ top(push(s,n)) &= n \end{aligned}$$
... (eqs from N and eqs for $\equiv$)

In a similar way, we get trees with a natural number attached to each node as solutions of the domain equation $\underline{X} = \underline{X} \times \underline{X} \times \underline{N} + \underline{1}$, where $=$ denotes an appropriate pair (p,e) of morphisms, etc.

Our theory so far gives algebraic data types as solutions of algebraic domain equations. It is natural to ask whether we can get parameterized data types as solutions of parameterized algebraic domain equations by a similar method of implicit specification and syntactic solution. This works indeed.The details will be published elsewhere.

5. CONCLUSIONS

The theoretical results presented here provide a sound and consistent semantics for a new algebraic specification method using parameterized specifications and algebraic domain equations. The feasibility and usefulness of this method for the development of specification methods and specification languages should be subject to further study.

Another possible area of application is the algebraic semantics of programming languages. In denotional semantics, domain equations are used extensively to specify the syntactic and semantic domains. Our theory can provide algebraic interpretations for them. There is, however, one difficulty: we get only "finitary" solutions, for example (initial) algebras of finite sets or finite functions. The central semantic domains of environments and states usually are finitary, so there seems to be no problem. It is, however, not quite clear how to cope with cases like procedure parameters. In particular, we cannot obtain a model for λ-calculus with our method, like Scott's reflexive domain (SC 72b). For these and similar cases, an extension of our theory to continuous algebras is necessary. This is subject to further study.

REFERENCES

ADJ 77 Goguen,J.A./Thatcher,J.W./Wagner,E.G./Wright,J.B.: Initial Algebra Semantics and Continuous Algebras. Journal ACM 24, (1977),68-95

ADJ 78 Goguen,J.A./Thatcher,J.W./Wagner,E.G.: An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types. Current Trends in Programming Methodology,Vol IV(R.T.Yeh,ed.). Prentice Hall, Englewood Cliffs 1978, 80-149

BG 77 Burstall,R.M./Goguen,J.A.: Putting Theories Together to Make Specifications. Proc. 5th Int. Joint Conf. on Artificial Intelligence, MIT, Cambridge (Mass.), 1977

BG 80 Burstall,R.M./Goguen,J.A.: The Semantics of CLEAR, a Specification Language. Proc. 1979 Copenhagen Winter School on Abstract Software Specifications (D. Bjorner,ed.). LNCS 86, Springer-Verlag, Berlin 1980, 292-331

EH 79 Ehrich,H.-D.: On the Theory of Specification, Implementation, and Parameterization of Abstract Data Types. Bericht Nr. 82/79, Abteilung Informatik, Univ. Dortmund 1979 (also to appear in Journal ACM)

EL 79a Ehrich,H.-D./Lohberger,V.G.: Parametric Specification of Abstract Data Types, Parameter Substitution, and Graph Replacements. Graphs, Data Structures, Algorithms (M.Nagl/H.-J.Schneider,eds.). Applied Computer Science 13, Hanser Verlag, Muenchen 1979, 169-182

EL 79b Ehrich,H.-D./Lohberger,V.G.: Constructing Specifications of Abstract Data Types by Replacements. Proc. Int. Workshop on Graph Grammars and Their Application to Computer Science and Biology (V.Claus/H.Ehrig/G.Rozenberg, eds.). LNCS 73, Springer-Verlag, Berlin 1979, 180-191

EKTWW 80a Ehrig,H./Kreowski,H.-J./Thatcher,J.W./Wagner,E.G./Wright, J.B.: Parameterized Data Types in Algebraic Specification Languages. Proc. 7th ICALP (J.W.deBakker/J.van Leeuwen, eds.). LNCS 85, Springer-Verlag, Berlin 1980, 157-168

EKTWW 80b Ehrig,H./Kreowski,H.-J./Thatcher,J.W./Wagner,E.G./Wright, J.B.: Parameter Passing in Algebraic Specification Languages. Internal Report, FB 20 TU Berlin, 1980

LE 76 Lehmann,D.J.: Categories for Fixpoint Semantics. Proc. 17th FOCS 1976, 122-126

LS 77 Lehmann,D.J./Smyth,M.B.: Data Types. Proc 18th FOCS 1977, 7-12

SC 71 Scott,D.S.: The Lattice of Flow Diagrams. Proc. Symp. on Semantics of Algorithmic Languages (E.Engeler,ed.).LNM 188, Springer-Verlag, Berlin 1971, 311-372

SC 72a Scott,D.S.: Lattice Theory, Data Types and Semantics. Formal Semantics of Algorithmic Languages (R.Rustin,ed.). Prentice Hall, Englewood Cliffs 1972, 65-106

SC 72b Scott,D.S.: Continuous Lattices. Toposes, Algebraic Geometry and Logic (F.W.Lawvere,ed.). LNM 274, Springer-Verlag, Berlin 1972, 97-136

SC 76 Scott,D.S.: Data Types as Lattices. SIAM Journal of Computing 5(1976), 522-587

SP 77 Smyth,M.B./Plotkin,G.D.: The Category-Theoretic Solution of Recursive Domain Equations. Proc. 18th FOCS 1977, 13-17

WA 75 Wand,M.: On the Recursive Specification of Data Types. Proc. 1st Int. Coll. on Category Theory Applied to Computation and Control (E.G.Manes,ed.). LNCS 25, Springer-Verlag, Berlin 1975, 214-217