ABSTRACT OBJECTS AS ABSTRACT DATA TYPES REVISITED

by Alexander Ollongren

Department of Mathematics and Computer Science
Leiden University

PREFACE

The present paper is a revision of an article written by H. Gerstmann and the author and published in the Proceedings of the 1979 Copenhagen Winter School 'Abstract Software Specifications', Springer Lecture Notes in Computer Science No. 86. The main part of the paper is taken *verbatim* from the mentioned article (permission from the publisher has been requested); the last section is written anew.

I am happy that the compilers of the *Liber Amicorum* dedicated to Jaco de Bakker have decided to include the present paper, because it gives me the opportunity to discuss, albeit briefly, an issue which was considered in the seventies important in the field of the semantics of programming languages. The question which raised much attention at the time was whether the semantics should be defined in an operational manner or by different means. In the view of the school based in Vienna to which P. Lucas, K. Walk and many others belonged, the issue could be settled by defining a suitable formal universal interpreter. At the time I was much attracted to the idea and tried to contribute to it by formalizing the underlying so-called *objects*.

The interpreter was to be universal in the sense that it could give an interpretation to any program in any high-level programming language of the procedural type. In modern terminology the set of objects served as the basic data type of the interpreter. PL/I was given a semantics in this way [1], and in the seventies the Vienna group showed the usefulness of the method by formally defining a number of semantics and semantic concepts.

There was a rivalling school of thought, the Oxford school, led by ideas of Chr. Strachey, which set the scene for what is now called the denotational semantics of programming languages. When I first met Jaco de Bakker it was by no means clear which school would emerge as the most promising and I remember him saying that the future would tell. (In hindsight it is clear that both approaches were promising: the denotational method is well-founded and widely used today, the original idea of the universal interpreter defined in the Vienna Definition Language (VDL) led to the establishment of the Vienna Development Method (VDM), equally well-known). Since then Jaco wrote the magnificent volume on the semantics of programming languages [10], which is based on yet another concept: the set-theoretic approach.

By the end of the seventies the operational method of the Vienna school was under criticism from theoreticians who considered the basic objects in VDL 'not really abstract'. This was clearly an open invitation for one to try to counter the point and I presented at the mentioned Winter School an axiomatic treatment of the same - what could be more abstract? H. Gerstmann gave a lecture too, presenting an algebraic treatment. At the meeting we decided to cooperate and contribute to the proceedings with just one paper: the main sections are reproduced here. The last section is revised because I wish to connect with modern developments in VDM.

Finally I must remark here that B. Nordström [11] a few years later gave yet another foundation of the abstract objects in the regime of constructive mathematics, thus definitely settling the account in favor of the abstract nature of the datatype

and so of the universal interpreter. The complexity of the device, however, remains a drawback as far as application is concerned.

INTRODUCTION

In the Vienna Definition Language (VDL) [1,2,3] 'abstract objects' are provided to describe the state space of symbolic machines as well as the phrase structure of programs. Since the language is used as a meta language to define the operational semantics of programming languages, some authors [4] refuse to consider abstract objects as being abstract. However, whether semantics are denotational or operational is just a matter of the mapping from the syntactic to the semantic domain, and not of the meta language in which the mapping is described.

As a matter of fact, the authentic reference [1] makes a clear distinction between abstract objects and their representations, and in a paper by H. ZEMANEK [5] isomorphic representations are discussed. In pursuit of this design philosophy P. LUCAS introduced the notion of a 'software device', which seems to be the earliest instance of an abstract data type. His paper presented at the Second Courant Computer Symposium in 1970 [6] contains an equational presentation of a stack and an implementation for it.

The purpose of the present paper is to show that abstract objects are indeed abstract data types in the strict algebraic form of [4]. To reveal the structural properties of the data type Abstract Object, it is derived from the data type Set in a sequence of refinements by means of enrichment and functors [7]. This way of proceeding does not only yield the original VDL objects, but also its advanced descendants in form of new objects [8]. The paper concludes with an outline of how this abstract data type can be used as model for parts of META IV, the meta language in the Vienna Development Method (VDM) [9].

ALGEBRAIC PROVISIONS

A specification of an abstract data type consists of a set of sorts S, a Set of operation symbols Σ, and a set of equations E between the operation symbols. A Σ-algebra contains for each sort S a carrier

$$A_s$$

and for each operator symbol of type

$$s1s2 \ldots sn \rightarrow s$$

a function

$$\sigma_A : A_{s1} \times A_{s2} \times \ldots \times A_{sn} \rightarrow A_s \ .$$

The class of all Σ-algebras together with their homomorphisms can be regarded as a category. In this category the word algebra $T(\Sigma)$ of well-formed terms built up from the operator symbols is initial, whereas in the subcategory of Σ-algebras that satisfy the set of equations this property is taken over by the quotient algebra $T(\Sigma E)$ with respect to the congruence relation generated by E.

The class of all initial algebras in the subcategory defines the abstract data type of the given specification. Within this class the subclass of all word algebras, with

$T(\Sigma E)$ as its representative, yields the abstract syntax, and any other initial algebra is a model of the data type.

During stepwise specification by enrichment, new operation symbols on the given sorts and equations between them are added to the specification of the data type such that existing congruence classes are maintained. If new sorts are required too, a functor can be defined between the respective categories which assures that each model of the extended specification is also a model of the original one.

STEPWISE SPECIFICATION OF DATA TYPE ABSTRACT OBJECT

Starting from a specification of Set the data type Abstract Object is developed in a chain of refinements. The reader is assumed to have a model of Set at his disposal which enables him to verify the following equational presentation:

Data type: Set
Sorts: el, set, bool

Operation Symbols
insert: el set → set
delete: el set → set
has: el set → bool
ø: → set

Equations

insert (e,insert(e,s))	=	insert(e,s)	
insert(e1,insert(e2,s))	=	insert(e2,insert(e1,s))	e1≠e2
delete(e,delete(e,s))	=	delete(e,s)	
delete(e1,delete(e2,s))	=	delete(e2,delete(e1,s))	e1≠e2
insert(e,delete(e,s))	=	insert(e,s)	
delete(e,insert(e,s))	=	delete (e,s)	
insert(e1,delete(e2,s))	=	delete(e2,insert(e1,s))	e1≠e2
delete(e,ø,)	=	ø	
has(e,insert(e,s))	=	true	
has(e,delete(e,s))	=	false	
has(e1,insert(e2,s))	=	has(e1,s)	e1≠e2
has(e1,delete(e2,s))	=	has(e1,s)	e1≠e2 .

The first step of refinement introduces assumptions about the elements. Each element is again a set, prefixed by a name to identify it.

el = name set.

However, elements are not just considered Cartesian products, which would leave the axioms essentially unchanged. Since the first component is to act as a name, the occurrence of equal names requires special treatment. In this case the second component is overwritten by the insert and forgotten by the delete operation. This yields the

Data Type: Set of named Sets
Sorts: name, set, bool

Operations
insert: name set set → set
delete: name set set → set

has: name set set → bool
Ø: → set

Equations

insert (n,s1,insert(n,s2,s))	=	insert(n,s1,s)	
insert (n1,s1,insert(n2,s2,s))	=	insert(n2,s2,insert(n1,s1,s))	n1≠n2
delete(n,s1,delete(n,s2,s))	=	delete(n,s1,s)	
delete(n1,s1,delete(n2,s2,s))	=	delete(n2,s2,delete(n1,s1,s))	n1≠n2
insert(n,s1,delete(n,s2,s))	=	insert(n,s1,s)	
delete(n,s1,insert(n,s2,s))	=	delete(n,s1,s)	
insert(n1,s1,delete(n2,s2,s))	=	delete(n2,s2,insert(n1,s1,s))	n1≠n2
delete(n,s,Ø)	=	Ø	
has(n,s1,insert(n,s1,s))	=	true	
has(n,s1,delete(n,s1,s))	=	false	
has(n1,s1,insert(n2,s2,s))	=	has(n1,s1,s)	n1≠n2
has(n1,s1,delete(n2,s2,s))	=	has(n1,s1,s)	n1≠n2

In order to show that the refined version is still a model of Set, a functor

F: Set of named Sets → Set

is introduced by the mappings

F(name set)	=	name
F(set)	=	set
F(bool)	=	bool

for objects and

F(insert(n,s1,s))	=	insert(n,s)
F(delete(n,s1,s))	=	delete(n,s)
F(has(n,s1,s))	=	has(n,s)
F(Ø)	=	Ø

for operations. With its help the original equations can be recovered from the refined ones, for instance

insert(n,s1,insert(n,s2,s)) = insert(n,s1,s)
F
------>
insert(n,insert(n,s)) = insert(n,s)

or

(n1≠n2) ⇒
insert(n1,s1,insert(n2,s2,s)) = insert(n2,s2,insert(n1,s1,s))
F
----->
(n1≠n2) ⇒
insert(n1,insert(n2,s))= insert(n2,insert(n1,s)) .

An inspection of the axioms discloses a striking symmetry between the operations of insertion and deletion. This property suggests the definition of one operation in terms of the other. Since, due to the inductive definition of set, the empty set can be used in an element, it is not wrong to try

delete(n,s1,s) = insert(n,Ø,s).

It is easy to see that with this definition the axioms containing delete become special cases of the other ones and can be removed. The remaining equations are:

insert(n,s1,insert(n,s2,s))	=	insert(n,s1,s)	
insert(n1,s1,insert(n2,s2,s))	=	insert(n2,s2,insert(n1,s1,s))	n1≠n2
insert(n,Ø,Ø)	=	Ø	
has(n,s1,insert(n,s1,s))	=	true	
has(n,s1,insert(n,Ø,s))	=	false	
has(n1,s1,insert(n2,s2,s))	=	has(n1,s1,s)	n1≠n2 .

To obtain abstract objects, the derived operation

select: name set → set

defined by

select(n,s) = s1 if has(n,s1,s) = true
Ø otherwise

is introduced instead of has. Substitution into the equations for has yields the corresponding equations for select

select(n,insert(n,s1,s)) = s1

select(n1,insert(n2,s2)) = select(n1,s) n1≠n2.

The final step includes a transition to the more familiar notation

name	←	sel(ector)
set	←	obj(ect)
empty set (Ø)	←	null object (Ω)
insert(x,y,y')	←	mk(y',x,y)
select(x,y)	←	*s*l(x,y)

and the specification of a basis from which the objects are constructed. Within sort obj a set of generators el, called elementary objects, is assumed, which, combined with selectors, yield objects. Additional axioms extend the operations to elementary objects. The result of all these measures is the

Data Type: Abstract Object
Sorts: obj, sel

Operation Symbols

mk : obj sel obj	→	obj
*s*l: sel obj	→	obj
:	→	obj

Equations:

mk(mk(o,s,o1),s,o2)	=	mk(o,s,o2)	sort (o)≠el v o1≠Ω
mk(mk(e,s,o1),s,Ω)	=	Ω	sort(e) = el
mk(mk(o,s1,o1),s2,o2)	=	mk(mk(o,s2,o2),s1,o1)	s1≠s2
mk(Ω,s,Ω)	=	Ω	
mk(e,s,Ω)	=	e	sort(e)=el
*s*l(s,mk(o,s,o1))	=	o1	
*s*l(s1,mk(o,s2,o2))	=	*s*l(s1,o)	s1≠s2 .

Actually this is not the specification of a data type but of a class of data types. The presentation contains the two basis types sel and el which still must be specified to obtain a specific data type within the class. Any data object within the class is called a new object in reference [8].

The original VDL objects [1] appear as a special case in which sel is specified as a monoid and el as a set of atoms. Monoid multiplication is identified with functional composition so that the additional axioms

$$sl(s1*s2,o) = sl(s1,sl(s2,o))$$
$$mk(o,s1*s2,o1) = mk(o,s2,mk(sl(s2,o),s1,o1))$$

hold. The consequences of these additional axioms will be discussed after a model for abstract objects is available in the next section.

STANDARD MODEL FOR ABSTRACT OBJECTS

To show the consistency of the final equations, a model is presented. The model is based on the assumptions that models for the basis types have already been given and therefore applies to any data type within the class of abstract objects. For this reason it is called standard model.

Before the model can be defined the syntax must be specified. The syntax of the abstract data type is defined by the word algebra T with

Carriers: T_{obj} , T_{sel}

Operations

$mk_T : T_{obj} \times T_{sel} \times T_{obj} \rightarrow T_{obj}$

$sl_T : T_{sel} \times T_{obj} \rightarrow T_{obj}$

$\Omega_T : \rightarrow T_{obj}$.

A term t in T_{obj} is generated from the basis terms T_{sel} and T_{el} by means of the productions:

$$t \leftarrow \Omega \mid e \qquad e \in T_{el},\ s \in T_{sel},\ t \in T_{obj}$$
$$t \leftarrow sl(s,t) \mid mk(t,s,t) \quad .$$

The terms t are considered as words 't' on which the operations are defined by

$$mk('t','s','t2') = 'mk(t1,s,t2)'$$
$$sl('s','t') = 'sl(s,t)'.$$

The axioms induce a partition of the set of words into disjoint classes of equivalent words. For each class there is a unique representative: sl can be eliminated from a term applying the axioms for this function. Any nested mk-term can be reduced by the other axioms to an equivalent one in which all s are distinct and obey a certain order. Thus any class can either be represented by 'Ω', 'e', or by a word of the form

$$'mk(mk(...mk(t_0,s_1,t_1), \ldots , s_{n-1},t_{n-1}),s_n,t_n)'$$

in which t_0 is an elementary or null object, any other t again a term of the same kind, and

$$s_i \neq s_k \text{ for } i \neq k.$$

The specification of abstract objects in a chain of refinements starting from Set indicates that there must exist a set theoretic model. Instead of pairs, however, the notion of mappings as provided in META IV [9] is used because the associated operations are moreadequate for the present purpose.

The model is an algebra

$$0 = (0_{obj}, 0_{sel}; mk_0, sl_0, \Omega_0)$$

with

Carriers: 0_{obj}, 0_{sel}

and

Operations

$mk_0 : 0_{obj} \times 0_{sel} \times 0_{obj} \rightarrow 0_{obj}$

$sl_0 : \quad 0_{sel} \times 0_{obj} \rightarrow 0_{obj}$

$\Omega_0 : \quad \rightarrow 0_{obj}$.

Objects are mappings in the domain

$$0_{obj} =_m (0_{sel} \rightarrow (0_{gen} \mid 0_{obj}))$$

with 0_{sel} and 0_{el} assumed to be specified and the generic set is

$$0_{gen} = 0_{el} \cup \{\Omega_0\} \quad .$$

Partial mappings

$$[s \rightarrow o(s) \mid s \in Sel] \qquad Sel \subset 0_{sel}$$

are completed by the convention

$$[s \rightarrow \text{if } s \in Sel \text{ then } o(s) \text{ else } [\,] \mid s \in 0_{sel}] \quad .$$

[ ] denotes the empty mapping, which is also used to represent Ω_0. The other operations are defined by the expressions

$mk_0(o,s,o1) = (o \in 0_{el} \wedge o1 = [\,] \rightarrow o,$

$o \in 0_{el} \wedge o1 \neq [\,] \rightarrow [s \rightarrow o1],$

$o \notin 0_{el} \rightarrow o + [s \rightarrow o1])$

$sl_0(s,o) = (o \in 0_{el} \rightarrow [\,],$

$o \notin 0_{el} \rightarrow o(s)).$

Since for the basis types the existence of models is assumed, there is already an isomorphism between them and the respective terms in T:

$h('e') = e \qquad 'e' \in T_{el},\ e \in 0_{el}$

$h('s') = s \qquad 's' \in T_{sel},\ s \in 0_{sel}$.

Next the nullary operations must be associated

$h('\Omega') = [\ ]$.

This partial mapping can now be uniquely extended to the classes of equivalent words in T: For each representative

$'mk(mk(\ldots\ mk(t_0,s_1,t_1),\ \ldots\ s_{n-1},t_{n-1}),\ s_n,t_n)'$

of a class its image is defined recursively by

$h('mk(t_A,s,t_B)') = h('t_A') \cup [h('s') \rightarrow h('t_B')]$.

It is not difficult to show that the axioms are satisfied by the model, for instance

$sl(s,mk(o,s,o1)) = o1$ for $o \notin 0_{el}$:

$sl_0(s,mk_0(o,s,o1)) = sl_0(s,o + [s \rightarrow o1])$

$= (o + [s \rightarrow o1])(s) = o1.$

To show the initiality of T, let

$A = (A_{obj},\ A_{sel}\ ;\ M,\ S,\ 0)$

be any algebra in the subcategory of algebras satisfying the axioms and

$u: 0 \rightarrow A$

uniquely defined by

$u(e) = a \qquad e \in 0_{el},\ a \in A_{el}$

$u(s) = r \qquad s \in 0_{sel},\ r \in A_{sel}$

$u([\ ]) = 0$

$u([si \rightarrow oi \mid si \in Sel]) = M(u([si \rightarrow oi \mid si \in Sel \backslash \{sk\}]),u(sk),u(ok))$.

Again the conditions for a homomorphism are only proved for a typical case. For

$si \in Sel$

$u(S(si,[sj \rightarrow oj \mid sj \in Sel])) = u(oi) =$ (by the first axiom for sl)

$S(u(si),M(u([sj \rightarrow oj \mid sj \in Sel \backslash \{si\}]),u(si,u(oi))) =$

$S(u(si),u([sj \rightarrow oj \mid sj \in Sel]))$

$u(M([sj \rightarrow oj \mid sj \in Sel],si,o')) =$

$u([sj \rightarrow oj \mid sj \in Sel \backslash \{si\}] \cup [si \rightarrow o']) =$

$M(u([sj \rightarrow oj \mid sj \in Sel \backslash \{si\}]), u(si,u(o')) =$ (by the first axiom for mk)

$M(M(u([sj \rightarrow oj \mid sj \in Sel \backslash \{si\}),u(si),u(oi)),u(si),u(o')) =$

$M(u([sj \rightarrow oj \mid sj \in Sel]),u(si),u(o'))$.

To obtain VDL objects within the standard model 0_{sel} must be specified as monoid and the equations

$$o + [s1*s2 \rightarrow o12] = o + [s2 \rightarrow o(s2) + [s1 \rightarrow o12]]$$

$$o(s1*s2) = o(s2)(s1)$$

added. For the null object the first equation becomes

$$[s1*s2 \rightarrow o12] = [s2 \rightarrow [s1 \rightarrow o12]]$$

so that in any object with components

$[s1 \rightarrow o1]$, $[s2 \rightarrow o2]$, and $[s1*s2 \rightarrow o12]$

the identity

$$o2 = [s1 \rightarrow o12]$$

holds. If the equations are dropped new objects over a selector monoid are created in which the objects o1, o2, and o12 can be chosen arbitrarily.

On the other hand, paths through an object can be equated by imposing additional axioms on the monoid. Let

$$0_{sel} = (\{0,1\}; *)$$

be a monoid with two selectors satisfying the axiom

$$1^i 0^k = 0^k 1^i$$

In this case a two dimensional array A is obtained, in which an element A(i,k) can be retrieved along any path equivalent to the path $1^i 0^k$.

SOME FURTHER REMARKS

In the previous section de VDM meta language META IV (cf. the introduction) is briefly used to represent abstract objects as mappings. In the present section the attention is directed to the problem of providing interpretations for some parts of the meta language in the abstract data type developed.

To start with the set of elementary objects obviously can be represented by the basis of the data type, i.e. generators of the sort el. Further classes of trees are interpreted as classes of new objects (multi-level arrays) with some appropriate set of selectors, i.e. the sort sel. This allows an interpretation of an expression in the abstract syntax like

$$A :: s1: B1 \quad s2 : B2 \ \ldots \ sn : Bn \qquad n > 0$$

as the class of multi-level arrays characterized by

$$a \in A \Leftrightarrow s_i(si,a) \in Bi \qquad i=1,2, \ldots ,n.$$

The symbol :: can be read as 'is composed of' ([12]), p. 122). An axiomatic treatment of (generalized) multi-level arrays is to be found in [13].

Elements of the class can be created under the META IV conventions by so-called make-functions. Let Bi (i = 1,2, ...,n) be given sets, possibly of trees. Then the function mk-A of type

$$mk\text{-}A : B1 \times B2 \times \ldots \times Bn \rightarrow A$$

evaluates (for a given sequence of selectors s1, s2, ..., sn) to an element for which the following property holds:

$$s_i(si,mk\text{-}A\ (b1, \ldots, bn)) = bi \qquad i = 1,2 \ldots, n.$$

This property is the same one as the characterizing property of the expression we started out with. META IV allows to write instead of the last equality:

$$si(mk\text{-}A(b1, \ldots ,bn)) = bi$$

so that the selector si takes the role of a function of type $A \rightarrow Bi$ (i = 1,2,...,n).

Within the abstract data type the make-functions admit the following interpretation:

$$mk\text{-}A\ (b1, \ldots , bn) = a \quad \Leftrightarrow \quad a \in A \wedge a =$$

$$mk^n(^n\Omega,s1,b1),s2,b2),\ldots)sn,bn)$$

with

$$mk^n(^no,s1,b1),s2,b2)\ldots),sn,bn) =$$

$$mk^{n-1}(^{n-1}mk(o,s1,b1),s2,b2),\ldots),sn,bn)$$

and

$$mk^0(o,) = o .$$

From this we see that mk-A can be expressed as a lambda form:

$$mk\text{-}A = \lambda\ (b1,b2,\ldots,bn).\ \ mk^n(^n\Omega\ ,s1,b1),s2,b2),\ldots),sn,bn) .$$

As before the sequence of selectors s1,...,sn is supposed to be given.

REFERENCES

[1] LUCAS, P. and WALK, K.: On the Formal Description of PL/1, Annual Review of Automatci Programming 6, 3(1969)

[2] WEGNER, P.: The Vienna Definition Language, ACM Comp. Surveys 4, 1(1972), 5-63

[3] OLLONGREN, A.: Definition of programming languages by interpreting automata, Acad. Press 1974.

[4] GOGUEN, J.A. et al.: Abstract Data Types as initial Algebras and the Correctness of Data Representations, SIGGRAPH Notices (May 1975), 89-93

[5] ZEMANEK, H.: Abstrakte Objekte, Elektronische Rechenanlagen 10, 5(1968), 208-217

[6] LUCAS, P.: On the Semantics of Programming Languages and oftware Devices, in Formal Semantics of Progr. Lang., edited by R. Rustin, Prentice Hall 1972, 41-57

[7] EHRIG, H. et al.: Stepwise Specification and Implementation of Abstract Data Types, in Automata, Languages and Programming, edited by G. Ausiello and C. Boehm, Lect. Notes in Comp. Sc. Vol. 62, Springer-Verlag 1978, 205-226

[8] GOEMAN, H.J.M. et al.: Axiomatiek van Datastructuren, MC Syllabus 37 Colloquium Capita Datastructuren (1978), 85-98

[9] BJöRNER, D. and JONES, C.B., The Vienna Development Method: The Meta-Language, Lect. Notes in Comp. Sc. Vol. 61, Springer-Verlag 1978

[10] DE BAKKER, J.W.: Mathematical Theory of Program Correctness, Prentice Hall, 1980

[11] NORDSTRöM, B. : Multilevel functions in Martin-Löf's type theory, Lect. Notes in Comp. Sc. Vol.217, 206-221, Springer Verlag 1986

[12] JONES, C.B.: Systematic Software Development Using VDM, Prentice Hall, 1986.

[13] BERGSTRA, J.A. et al.: Axioms for Multilevel Objects, Annales Societas Mathematicae Polonae Series IV: Fundamenta Informaticae, Vol. III,2, 171-180, (1979)