Annales Univ. Sci. Budapest., Sect. Comp. **8** (1987) 109-118

IMPLEMENTATION OF ABSTRACT DATA TYPES WITH CORRECTNESS PROOF

LÁSZLÓ VARGA

Dedicated to Professor I. Kátai on his 50^{th} birthday

Abstract. In this paper a correct implementation of abstract data types is defined. Abstract data types are given by algebraic specification. It is shown that a correct implementation satisfies the semantic equations of the given abstract data type. On the other hand we show that if an implementation satisfies the semantic equations of an abstract data type then it is a correct implementation.

1. Introduction

Nowadays the use of data abstraction in programming is generally accepted. Recently many specification methods for data abstraction have been proposed [2, 3, 4, 5]. Among them algebraic specification is frequently used since it does not depend on an abstract representation. An introduction to the method can be found in [1]. In this paper algebraic specification is used for describing abstract data types.

We assume that a set of abstract objects is given together with a group of abstract data type operations where the semantics are defined by algebraic axioms.

We assume we are also given a set of concrete objects that represent the abstract objects and a group of concrete operations that implement the abstract operations in question.

For a given abstract data type we define the notion of correct representation and the notion of correct implementation in Section 2.

We prove that a concrete data type which is a correct representation and implementation of a given abstract data type has the same semantics as the abstract data type in question.

The next result of this paper is that if operations of a concrete data type with correct representations satisfy the semantic axioms of the given abstract data type, then the concrete implementation is correct.

The results can be used in a data type specification methodology as it is demonstrated by an example in Section 3.

2. Theorems about correct representations and implementations

It is known that a data type can be regarded as a pair $\alpha = (A, F)$, where $A = \{A_1, A_2, \ldots, A_m\}$, A_i $(i = 1, 2, \ldots, m)$ is a recursively enumerable set, $F = \{f_0, f_1, \ldots, f_n\}$, and

$$f_i = A_{i_1} \times A_{i_2} \times \ldots \times A_{i_k} \to A_{i_0}$$

is a partial mapping function.

The set A_1 is a set of objects of the type to be defined and it is called the *type of interest*. The set A_1 is constructive, which means that all elements of it can be built by applying the operations f_i only.

The operations of a minimal subset of F, sufficient for producing all elements of A_1 are called *constructors or constructor functions*. Among the constructor functions there is at least one constant function. In our case let f_0 be a constant function.

Among the sets A_i there could be sets which are formal parameters of the data type. The parameters could be replaced by the types of interest of other data types as actual parameters. A data type with formal parameters is called a *parameterized data type.*

In the case of a parameterized data type the function f_i has the form

$$f_i(a,p),$$

where $a \in A_1$, $p = (p_1, p_2, \ldots, p_k)$, $p_i \in A_{i_j}$, $i \in \{2,3,...,k\}$, $i_j \in \{1,2,\ldots,m\}$.

The meaning of the operations $f_0, f_1, \ldots, f_n$ could be given by different specification methods. Among them *algebraic specification* gives the semantics by equations usually in the following form:

$$f_s(f_c(a,p),q) = h(f_i, f_j, \ldots, f_l, a, p, q),$$

where f_c is a constructor, f_s is a non-constructor and h is a partial mapping function constructed by $f_i, f_j, \ldots, f_l$, $i,j,\ldots,l \in \{0,1,\ldots,n\}$.

EXAMPLE.

$$\begin{aligned} f_s(f_0,p) &= f_0, \\ f_s(f_c(a,p),q) &= f_c(f_s(a,q),p), \\ f_s(f_c(a,p),q) &= p. \end{aligned}$$

DEFINITION 2.1. Let $d_a = (A,F)$ be a parameterized data type. The set C_1 is a correct *representation* of the set A_1 if there is a mapping function $\varphi : C_1 \to A_1$, where

$$(\forall a \in A_1)\ (\exists c \in C_1)\ (a = \varphi(C)).$$

Lemma 2.1. *Let the data types* $d_a = (A, F)$, $d_c = (C, G)$ *be given, where* $F = \{f_0, f_1, \ldots, f_k, \ldots, f_n\}$, $G = \{g_0, g_1, \ldots, g_k, \ldots, g_n\}$, $A = \{A_1, P_1, \ldots, P_n\}$, $C = \{C_1, P_1, \ldots, P_n\}$ *and the operations* $f_0, f_1, \ldots, f_k, g_0, g_1, \ldots, g_k$ *are constructors. If the mapping function* φ *is given by the equations*

$$f_0 = \varphi(g_0),$$

$$(\forall c \in C_1)\ (\forall i,\ 1 \leq i \leq k)\ (f_i(\varphi(c), p) = \varphi(g_i(c, p))),$$

then C_1 *is a correct representation of* A_1.

Proof. It is proven by induction on the number of constructor functions in $a \in A_1$.

If $a = f_0$ then there exists a $c = g_0 \in C_1$ for which $a = \varphi(c)$.

Let $a = f_c(a', p)$, where the lemma holds for a'. Then $a' = \varphi(c')$ and

$$a = f_c(a', p) = f_c(\varphi(c'), p) = \varphi(g_c(c', p)) = \varphi(c)$$

which verifies the lemma. □

DEFINITION 2.2. The data type $d_c = \{C, G\}$, $C = \{C_1, P\}$, $G = \{g_0, g_1, \ldots, g_k, \ldots, g_n\}$ is a *correct implementation* of the data type $d_a = (A, F)$, $A = \{A_1, P\}$, $F = \{f_0, f_1, \ldots, f_k, \ldots, f_n)$ if there is a mapping function $\varphi : C_1 \to A_1$ which satisfies the equations:

$$f_0 = \varphi(g_0),$$

$$(\forall c \in C)\ (\forall i,\ 1 \leq i \leq n)\ (f_i(\varphi(c), p) = \varphi(g_i(c, p))).$$

REMARK. If $f_i : A \times P \to P$ then

$$f_i(\varphi(c), p) = g_i(c, p)$$

holds instead of the equation

$$f_i(\varphi(c), p) = \varphi(g_i(c, p)).$$

AXIOM 2.1. Let $d_a = (A, F)$ be a data type. Then

$$a_1 = a_2 \equiv (a_1 = f_0 \wedge a_2 = f_0) \vee (a_1 \neq f_0 \wedge a_2 \neq f_0) \wedge$$
$$(\forall f_s)(\forall p \in P)(f_s(a_1, p) = f_s(a_2, p)),$$

where $a_1, a_2 \in A_1$ and $\forall f_s$ means the all non-constructors if F .

Theorem 2.1. *Let $d_c = (C, G)$ be a correct implementation of $d_a = (A, F)$. Let the semantics of d_a be given by equations of the form:*

$$f_s(f_c(a,p),q) = h(f_i, f_j, \ldots, f_l, a, p, q).$$

If $\varphi(c_1) = \varphi(c_2) \Rightarrow c_1 = c_2,$

and $\varphi(h(g_i, g_j, \ldots, g_l, c, p, q)) = h(f_i, f_j, \ldots, f_l, \varphi(c), p, q)$

or $h(g_i, g_j, \ldots, g_l, c, p, q) = h(f_i, f_j, \ldots, f_l, \varphi(c), p, q),$

then $g_s(g_c(c,p),q) = h(g_i, g_j, \ldots, g_l, c, p, q),$

i.e. the semantics of d_c is equal to the semantics of d_a.

Proof. a/ Let $c_1 = g_s(g_c(c,p),q)$ and $c_2 = h(g_i, g_j, \ldots, g_l, c, p, q)$. Then from Definition 2.2. we have $\varphi(c_1) = f_s(f_c(\varphi(c),p),q)$, but $\varphi(c_2) = h(f_i, f_j, \ldots, f_l, \varphi(c), p, q)$ and $\varphi(c_1) = \varphi(c_2) \Rightarrow c_1 = c_2$.

b/ Let $p_1 = g_s(g_c(c,p),q)$ and $p_2 = h(g_i, g_j, \ldots, g_l, c, p, q)$. Then $p_2 = h(f_i, f_j, \ldots, f_l, \varphi(c), p, q) = f_s(f_c(\varphi(c),p),q) = g_s(g_c(c,p),q) = p_1$. □

REMARK. If $h(f_i, f_j, \ldots, f_k, a, p, q) = a$ or $= f_i(f_j(a,q),p)$, then obviously $\varphi(h(g_i, g_j, \ldots, g_c, c, p, q)) = h(f_i, f_j, \ldots f_l, \varphi(c), p, q)$.

Theorem 2.2. *Let the data types*

$$d_a = (\{A_1, P\}, \{f_0, f_1, \ldots, f_k, \ldots, f_n\}),$$
$$d_c = (\{C_1, P\}, \{g_0, g_1, \ldots, g_k, \ldots, g_n\})$$

be given by the same semantics:

$$f_s(f_c(a,p,q)) = h(f_i, f_j, \ldots, f_c, a, p, q),$$
$$g_s(g_c(c,p),q) = h(g_i, g_j, \ldots, g_l, c, p, q).$$

If the mapping function $\varphi : C_1 \to A_1$ is defined by the equations

$$f_0 = \varphi(g_0),$$
$$(\forall c \in C_1)\ (\forall i,\ 1 \le i \le k)\ (f_i(\varphi(c),p) = \varphi(g_i(c,p))),$$

where $f_0, f_1, \ldots, f_k$ and $g_0, g_1, \ldots, g_k$ *are constructors in* F *and* G *respectively, furthermore*

$$\varphi(h(g_i, g_j, \ldots, g_l, c, p, q)) = h(f_i, f_j, \ldots, f_l, \varphi(c), p, q)$$

or

$$h(g_i, g_j, \ldots, g_l, c, p, q) = h(f_i, f_j, \ldots, f_l, \varphi(c), p, q),$$

then d_c *is a correct implementation of* d_a.

Proof. To prove the theorem we have to show that

$$(\forall c \in C_1)\ (\forall i,\ k < i \leq n) \quad (f_i(\varphi(c), p) = \varphi(g_i(c, p))).$$

Let $c = g_0$, then

$$\varphi(g_s(g_0, q)) = \varphi(h(g_i, g_j, \ldots, g_l, g_0, q)) =$$
$$h(f_i, f_j, \ldots, f_l, \varphi(g_0), q) = f_s(f_0, q).$$

Let now $c = g_c(c', p)$, where g_c is a constructor, then

$$\varphi(g_s(c, q)) = \varphi(g_s(g_c(c', p), q)) = \varphi(h(g_i, g_j, \ldots, g_l, c', p, q)) =$$
$$h(f_i, f_j, \ldots, f_l, \varphi(c'), p, q) = f_s(f_c(\varphi(c'), p), q) =$$
$$f_s(\varphi(g_c(c', p)), q) = f_s(\varphi(c), q). \square$$

3. An example

Let the data type $d_i = (integer, \{zero, succ, prec\})$ be given by the usual semantics:

$A1$: $prec(zero) = zero,$

$A2$: $prec(succ(i)) = i,$

and the data type

$$d_v = (\{vector, index, elem\}, \{new, put, access\})$$

be given by the semantic equations

$A3$: $access(new, i) = new - elem,$

$A4$: $access(put(v, i, e), j) =$ if $i = j$ then e else $access(v, j)$.

Here the type vector has two formal parameters: index and elem, and we write $vector(index, elem)$.

Using these data types we will give a representation and implementation for the data type $bag(elem)$.

type $bag(elem)$:

$d_a = (\{bag, elem\}, \{empty, insert, delete, many\})$

where $empty :\rightarrow bag;$

$insert : bag \times elem \rightarrow bag,$

$delete : bag \times elem \rightarrow bag,$

$many : bag \times elem \rightarrow integer.$

Semantic equations:

$S1$: $delete(empty, e) = empty,$

$S2$: $delete(insert(b, e_1), e_2) =$ if $e_1 = e_2$ then b else $insert(delete(b, e_2), e_1),$

$S3$: $many(empty, e) = zero,$

$S4$: $many(insert(b, e_1), e_2) =$ if $e_1 = e_2$ then $succ(many((b, e_2))$ else $many(b, e_2).$

Equality axiom for two vectors:

E : $v_1 = v_2 \equiv (v_1 = new \wedge v_2 = new)\ (\forall e \in elem)$ $(access(v_1, e) = access(v_2, e)).$

A possible representation of the type bag could be

$$\varphi : vector(elem, integer) \rightarrow bag(elem),$$

and then we have the following implementation:

$$empty_c = new,$$
$$insert_c(v, e) = put(v, e, succ(access(v, e))),$$
$$delete_c(v, e) = put(v, e, prec(access(v, e))),$$
$$many_c(v, e) = access(v, e).$$

Theorem 3.1. *If the mapping function φ is given by the equations*

$$empty = \varphi(new),$$
$$insert(\varphi(v), e) = \varphi(put(v, e, succ(access(v, e)))),$$

then

$$d_c = (\{vector, elem, integer\},\ \{empty_c, insert_c, delete_c, many_c\})$$

is a correct implementation of d_a.

Proof. According to Theorem 2.2. we have to show that $empty_c$, $insert_c$, $delete_c$, $many_c$ satisfy the semantic equations S1, S2, S3, S4.

$S1$: $$put(new, e, prec(access(new, e))) = new.$$

Using axioms A2 and A3 with *new-elem* = *zero* and the fact $put(new, e, zero) = new$ we get S1.

$S2$: $put(put(v, e_1, succ(access(v, e_1))),\ e_2,$
$prec(access(put(v, e_1, succ(access(v, e_1))), e_2))) =$
$\underline{\text{if}}\ e_1 = e_2\ \underline{\text{then}}\ v\ \underline{\text{else}}\ put(put(v, e_2, prec(access(v, e_2))), e_1,$
$succ(access(put(v, e_2, prec(access(v, e_2))), e_1))).$

Substituting both sides of equations S2 according to the equality axiom E we get the result immediately.

$S3$: $$access(new, e) = zero.$$

This is axiom A3.

$S4$: $$access(put(v, e_1, succ(access(v, e_1))), e_2) =$$
$$\underline{\text{if}}\ e_1 = e_2\ \underline{\text{then}}\ succ(access(v, e_2))\ \underline{\text{else}}\ access(v, e_2).$$

It could be get from the axiom A4 directly. □

References

[1] BERZTISS, A. T. and THATTE, S., Specification and implementation of abstract data types. In: *Advances in Computers*, Vol. 22 (Ed. H. C. Yovits). Academic Press, New York–London, 1983, 296-353.

[2] BURSTALL, R. M. and GOGUEN, J. A., Putting theories together to make specifications. In: *Proceedings 1977 IJCA*. MIT, Cambridge MA, 1977, 1045-1058.

[3] GREITER, G., A formal method to define data types. *SIGPLAN Notices* **10** (1984) 22-31.

[4] GUTTAG, J. V. and HORNING, J. J., The algebraic specification of abstract data types. *Acta Informatica* **9** (1978) 27-52.

[5] LISKOV, B. H. and ZILLES, S. N., Specification techniques for data abstraction. *IEEE Trans. on Software Eng.*, **SE-1** 1 (1975) 7-19.

(Received December 29, 1987)

LÁSZLÓ VARGA
Dept. of General Computer Science
Eötvös Loránd University
H–1088, Budapest, Múzeum krt. 6–8.

HUNGARY