A CASE STUDY OF ABSTRACT IMPLEMENTATIONS AND THEIR CORRECTNESS

H. Ehrig

H.-J. Kreowski

P. Padawitz

ABSTRACT

A new implementation concept for algebraic specification languages supports hierarchical programming mainly because it provides a semantical basis for correctness proofs. "Abstract programs" describe syntactically how data and operations of a lower level data type should represent those of an upper level type. Dependent on these programs a general semantical construction transforms the lower level type into an implementation of the upper level type. The implementation is correct if the result of this construction coincides with the semantics of the upper level type. Therefore this concept involves a clear distinction between the syntactical and the semantical part of an abstract implementation. Although the syntax of such an implementation always supplies a "freely generated" semantics, the concept also admits the use of other (algebraic) models which often ease correctness proofs.

A data type for performing some text analysis is specified and implemented by arrays which are accessed via an efficient hashing technique. Moreover, we give a correctness proof of this implementation that partly refers to correctness criteria introduced in an earlier paper where the whole concept is discussed in more detail.

Address of authors:

TU Berlin, FB Informatik (20), 1000 Berlin 10,

Otto-Suhr-Allee 18/20, Germany (West)

1. INTRODUCTION

For the last five years or so there has been a great effort to develop specification languages with various structuring concepts, e.g. ALPHARD (cf. /WLS 76/), SPECIAL (cf. /RR 77/, /LRS 79/) and OBJ (cf. /GT 78/). Three important goals are achieved by expressing programming tasks in terms of specification languages before writing down the code. First of all one avoids the consideration of special programming environments. Nevertheless specification languages have a precise syntax and thus provide the basis for an unambiguous semantics of specifications. Secondly, they incorporate tools for building up large programs from small pieces both in a horizontal and a vertical manner: Module and data type facilities evolve from the principles of information hiding and data encapsulation (cf. /Par 72/) while the method of stepwise refinement gets a formal basis against which its correct use can be checked. Thirdly, if primitive as well as structuring constructs of specification languages have a formal semantics, then one is able to prove whether properties of the problem to be specified and of its refinement are met by the specification.

Specification languages are based either on logical theories or on "abstract models". Algebraic specifications as introduced by Guttag (cf. /Gut 76/, /GHM 78/) and the ADJ group (cf. /GTW 78/, /Gog 77/) belong to the theory approach since they consist of pure syntax, namely operation symbols and equational axioms. Their semantics results from a general construction built up on that syntax. In the model approach operations are specified by their effect on a pre-defined mathematical object called abstract model or state space. Model approach languages are, for example, the assertion languages ALPHARD and SPECIAL. While ALPHARD provides a fixed set of models, the state space of a SPECIAL program is given by user-defined access operations ("V-functions").
The structuring facilities of specification languages are manifold. The basic construct that comprises a self-contained specification is called "form" in ALPHARD, "module" in SPECIAL and "theory" resp. "object" in the algebraic specification languages CLEAR (cf. /BG 77/) and OBJ. Specifications are composed to build new specifications by the features "extension" (cf. /GTW 78/, /Nou 79/), "type parameterization" resp. "procedure" (cf. /TWW 78/, /BG 77/), "abstract implementation" etc. Abstract implementations may be regarded as the formalization of stepwise refinement that was invented by Dijkstra and Wirth for the structured design of programs (cf. /Dij 72/, /Wir 71/).
A facility for writing abstract implementations is part of theory as well as model approach languages (cf. /GHM 78/ and /WLS 76/, /RL 77/, resp.). While /GHM 78/ does not deal with the semantics of its syntactical constructions, other algebraic approaches to implementations (/GN 78/, /LS 77/) tackle the semantics but do not

consider implementations as a structuring construct of specification languages that has its own syntax. The concept presented in this paper starts from very similar requirements as the approaches mentioned above. But we make the resulting constructions more explicit and avoid conceptual restrictions which are not due to the requirements. Hence a syntactical (or axiomatic), a semantical and a correctness level of implementations are treated separately. The semantics is completely determined by the syntax and a general semantical construction. If this construction results in the data type to be implemented, then the implementation is correct. For the purpose of a correctness proof one may use "abstract model" algebras which are isomorphic to the derived semantics. But these models are not part of the specification language. In /EKMP 80/ we define the composition of implementations and thus pay further attention to the language aspect of our concept.

The syntax of abstract implementations is defined in chapter 2 while chapter 3 deals with semantics and correctness. The whole concept is discussed in full detail in /EKP 79 a,b/. In this paper we stress its practical significance by presenting the correctness proof of an efficient implementation of a histogram data type that counts the occurrences of different strings in a string file. A six-level implementation of a corresponding data type given in /LRS 79/, Vol.III, for illustrating the facilities of SPECIAL has inspired us to regard this example from an algebraic point of view. Chapter 2 contains the syntax of this implementation, and its correctness proof is given in chapter 4.

2. THE SYNTACTICAL LEVEL OF IMPLEMENTATIONS

2.1 PRELIMINARIES

Let S be a set of sorts and Σ be a family of sets $\Sigma_{w,s}$ of operation symbols for all $w \in S^*$ and $s \in S$. $\sigma \in \Sigma_{w,s}$ is written $\sigma: w \longrightarrow s$.

We assume that the reader is familiar with the basic notions of many-sorted universal algebra, particularly with "Σ-term", "Σ-algebra" and "Σ-homomorphism" (see e.g. /GTW 78/).

Let $T_\Sigma(X)$ be the Σ-algebra of Σ-terms with variables in X and A be an arbitrary Σ-algebra. Then any function $a: X \longrightarrow A$ - called assignment - admits a unique Σ-homomorphic extension from $T_\Sigma(X)$ to A that is also denoted by a. $\underline{eval}_A$ is the unique Σ-homomorphism from $T_\Sigma = T_\Sigma(\emptyset)$ to A. Given a family E of binary relations $E_s \subseteq T_\Sigma(X)_s^2$ for all $s \in S$, the pairs of E are called equations, and the triple $SPEC=\langle S,\Sigma,E\rangle$ is a specification. A satisfies E and is a SPEC-algebra if aL=aR for all $(L,R) \in E$ and all $a: X \longrightarrow A$.

The semantics of SPEC is given by the quotient algebra $T_{SPEC} = T_\Sigma / \equiv_E$ where $\equiv_E$ is the least Σ-congruence that contains $\{(aL,aR) \mid a: X \longrightarrow T_\Sigma,\ (L,R) \in E\}$. T_{SPEC} is initial in Alg_{SPEC}, the category of SPEC-algebras (cf. /GTW 78/).

Abstract implementations in the sense of /EKP 79 a,b/ are defined as follows. We confine the definition to canonical implementations (/EKP 79/, 5.3), but we additionally admit "hidden" operations.

2.2 DEFINITION

Let SPEC0=SPEC+<SO,ΣO,EO> and SPEC1=SPEC+<S1,Σ1,E1> be two specifications with a common subspecification SPEC=<S,Σ,E> (+ denotes the componentwise disjoint union) A weak implementation IMPL=(ΣSORT,EOP,ΣHID,EHID) consists of operations ΣSORT and ΣHID, called sorts implementing operations resp. hidden operations, and of equations EOP and EHID, called operations implementing equations resp. hidden equations, such that

1. the range sorts of all $\sigma\in\Sigma$SORT belong to SO,
2. SORTIMPL=SPEC1+<SO,ΣSORT,$\emptyset$> and OPIMPL=(SORTIMPL+<$\emptyset$,ΣHID,EHID>)+<$\emptyset$,ΣO,EOP>are specifications, called sort implementation resp. operation implementation level.

□

2.3 REMARKS

Sorts implementing operations are domain constructors which combine SPEC1-data to build up SPEC0-data. ΣSORT may be partly identified with the syntactical devices "mappings" in SPECIAL and "representation" in ALPHARD and /GHM 78/. Operations implementing equations can be considered as programs that implement the SPEC0-operations, especially if EOP represents recursive definitions of ΣO-operations on (Σ+Σ1+ΣSORT)-terms. These definitions make use of hidden operations ΣHID which are specified in EHID. EHID+EOP corresponds to "programs" in /GHM 78/, "abstract programs" in SPECIAL and to the "implementation" part of ALPHARD specifications. Note that ALPHARD as well as the "derivor" approach to implementations (cf. /GTW 78/, /GN 78/) do not allow recursive definitions of SPEC0-operations. □

2.4 EXAMPLE

Each of the following specifications histogram, tup_m and array implicitly shares a specification bool of truth values TRUE and FALSE and boolean operations such that TRUE$\neq$FALSE and contains for all sorts s a conditional

IF-THEN-ELSE:bool s s$\longrightarrow$ s with equations

IF TRUE THEN x ELSE y = x

and IF FALSE THEN x ELSE y = y.

Let specifications nat and string of natural numbers resp. strings be given with successor SUCC, equality predicates EQ? and the empty string ε.

We want to implement a data type of histograms which provides an operation that for each string returns the number of its occurrences in a file. A simple specification of such a data type is the specification of string files enriched by an operation that counts equal entries. But the linear structure of a file implies that histograms specified as string files may be distinct even if the operation for counting string occurrences returns the same values. Thus all implementations of histograms would be forced to maintain unnecessary distinctions of data. This fact is also a formal consequence of correctness criterium II for implementations (see chapter 3). Especially, the implementation given in /LRS 79/ (see above) which we shall describe algebraically would not be an implementation of such an enriched file specification. Hence we add a commutativity axiom for strings to this specification and thus identify all files which are permutations of the same set of string occurrences. Therefore the specification of histograms corresponds to that of multisets (or bags) of strings together with a counting operation HOWMANY:

```
histogram = string + nat +
    sorts: hist
    opns: ∅: ⟶ hist
          INSERT: hist string ⟶ hist
          HOWMANY: hist string ⟶ nat
    eqns: INSERT(INSERT(h,w),v) = INSERT(INSERT(h,v),w)
          HOWMANY(∅,w) = 0
          HOWMANY(INSERT(h,w),v) = IF EQ?(w,v)
                                   THEN SUCC(HOWMANY(h,v))
                                   ELSE HOWMANY(h,v)
```

A histogram is implemented by an array a1 of strings and an array a2 of natural numbers as follows: a2 contains the number of occurrences of a string w at the same position where w is located in a1. The arrays are unbounded and initialized with ε resp. 0.

```
array(string) = string + nat +
    sorts: array1
    opns: NEW: ⟶ array1
          ASSIGN: array1 nat string ⟶ array1
          -[-]: array1 nat ⟶ string
    eqns: ASSIGN(NEW,n,ε) = NEW
          ASSIGN(ASSIGN(a,n,x),m,y) = IF EQ?(n,m)
                                      THEN ASSIGN(a,m,y)
                                      ELSE ASSIGN(ASSIGN(a,m,y),n,x)
          NEW[n] = ε
          ASSIGN(a,n,x)[m] = IF EQ?(n,m) THEN x ELSE a[m]
```

array(nat) is the same as array(string) except that string, array1, string and ε are replaced by nat, array2, nat and O, respectively. Instead of array(nat) and array(string) one may specify array(item) where item comprises only those properties of the parameter that are necessary for specifying arrays. For a formal treatment of type parameterization see /TWW 78/.

Access to the arrays is performed as follows: A hash function supplies for each string w a directory location that contains the array index where the search for w or an empty slot for w should start. The hash function values are assumed to range from O to m-1 so that the directory is specified as an m-tuple of natural numbers:

tup_m(nat) = nat +

sorts: nat_m, tup

opns: - MOD m: nat $\longrightarrow$ nat_m

[-,...,-] : nat^m $\longrightarrow$ tup

ENTRY: tup nat_m $\longrightarrow$ nat

CHANGE: tup nat_m nat $\longrightarrow$ tup

eqns: $SUCC^m$(i) MOD m = i MOD m

ENTRY([x1,..,xm], k MOD m) = x(k+1) for all $0 \leqslant k < m$

CHANGE([x1,..,xm], k MOD m, x) = [x1,...,xk,x,x(k+2),...,xm] for all $0 \leqslant k < m$

Hence, the implementation of histogram combines three specifications:

SPEC1 = tup_m(nat) + array(string) + array(nat)

implements SPECO = histogram by

sorts implementing operations:

TRIPLE: tup array1 array2 $\longrightarrow$ hist

hidden operations:

HASH: string $\longrightarrow$ nat_m

SEARCHSLOT: array1 nat $\longrightarrow$ nat

SEARCHHIT: array1 string nat $\longrightarrow$ nat

LOC: tup array1 string $\longrightarrow$ nat

INCREASE: array2 nat $\longrightarrow$ array2

hidden equations:

E(HASH) (equations for HASH)

```
SEARCHSLOT(a,n) = IF EQ?(a[n],ε)
                  THEN n
                  ELSE SEARCHSLOT(a,SUCC(n))
SEARCHHIT(a,w,n) = IF EQ?(a[n],w) OR EQ?(a[n],ε)
                   THEN n
                   ELSE SEARCHHIT(a,SUCC(n))
```

```
LOC(t,a,w) = SEARCHHIT(a,ENTRY(t,HASH(w)),w)
INCREASE(a,n) = ASSIGN(a,n,SUCC(a[n]))
operations implementing equations:
    ∅ = TRIPLE([0,...,0],NEW,NEW)
    INSERT(TRIPLE(t,a1,a2),w) =
        = IF EQ?(ENTRY(t,HASH(w)),0)
          THEN TRIPLE(CHANGE(t,HASH(w),SEARCHSLOT(a1,SUCC(0))),
                      ASSIGN(a1,SEARCHSLOT(a1,SUCC(0)),w),
                      ASSIGN(a2,SEARCHSLOT(a1,SUCC(0)),SUCC(0)))
          ELSE IF EQ?(a1[LOC(t,a1,w)],ε)
               THEN TRIPLE(t,ASSIGN(a1,LOC(t,a1,w),w),
                             ASSIGN(a2,LOC(t,a1,w),SUCC(0)))
               ELSE TRIPLE(t,a1,INCREASE(a2,LOC(t,a1,w)))
    HOWMANY(TRIPLE(t,a1,a2),w) = IF EQ?(ENTRY(t,HASH(w)),0)
                                 THEN 0
                                 ELSE a2[LOC(t,a1,w)]
```

HASH may be considered as a parameter such that SORTIMPL +<∅,HASH,E(HASH)> is an enrichment of SORTIMPL (see chapter 3). The common subspecification of SPEC1 and SPEC0 is given by SPEC = string + nat.

3. SEMANTICS AND CORRECTNESS OF IMPLEMENTATIONS

3.1 DEFINITION

Given a weak implementation IMPL=(ΣSORT,EOP,ΣHID,EHID) of SPEC0 by SPEC1, the semantical construction SEM_{IMPL} is the composition (to be applied from right to left)

$$SEM_{IMPL} = IDENTIFICATION \circ RESTRICTION \circ SYNTHESIS$$

such that

$$SYNTHESIS(T_{SPEC1}) = T_{OPIMPL},$$

$$RESTRICTION(T_{OPIMPL}) = REP_{IMPL} := eval(T_{\Sigma+\Sigma 0})$$

where eval is the unique $\Sigma 0$-homomorphism from $T_{\Sigma+\Sigma 0}$ to T_{OPIMPL},

$$IDENTIFICATION(REP_{IMPL}) = REP_{IMPL}/\equiv_{E+E0}.$$

$SEM_{IMPL}(T_{SPEC1})$ is called the semantics of IMPL.

IMPL is correct and thus an implementation if

I. OPIMPL is an enrichment of SORTIMPL,

i.e. $T_{SORTIMPL}$ and T_{OPIMPL} are $(\Sigma+\Sigma 1+\Sigma SORT)$-isomorphic, and

II. IMPL is RI-correct, i.e. $SEM_{IMPL}(T_{SPEC1})$ is $(\Sigma+\Sigma 0)$-isomorphic to T_{SPEC0}.

3.2 REMARKS

SYNTHESIS extends the implementing data type T_{SPEC1} by the data and operations that are to be implemented. Correctness in the sense of /EKP 79 a,b/ also requires type protection, i.e. that T_{SPEC1} and $T_{SORTIMPL}$ are $(\Sigma+\Sigma 1)$-isomorphic. But here we have restricted weak implementations to canonical ones (cf. /EKP 79b/, 5.3) so that type protection is always guaranteed (/EKP 79 b/, Lemma 5.1).

RESTRICTION extracts all data from the OPIMPL-semantics that are generable exclusively by $(\Sigma+\Sigma O)$-operations. IDENTIFICATION identifies all data of REP_{IMPL} which are semantically equivalent with respect to SPECO.

OPIMPL being an enrichment of SORTIMPL means that the operation implementation level preserves the semantics of the sort implementation level.

The "RI" of RI-correctness reflects the order of application of RESTRICTION and IDENTIFICATION. Goguen and Nourani (cf. (GTW 78/, (GN 78/) apply their corresponding constructions vice versa and take the result to be isomorphic to $REP_{IMPL}/\equiv_{E+EO}$. But IR-correctness has been proved to be stronger than RI-correctness by /EKP 79b/, Example 5.7.

If the common subspecification SPEC of SPECO and SPEC1 (cf. 2.2) is "protected" by SPECO and SPEC1, i.e. T_{SPEC} is Σ-isomorphic to T_{SPECO} and T_{SPEC1}, then T_{SPEC} and $SEM_{IMPL}(T_{SPEC1})$ are Σ-isomorphic, too (/EKP 79a/, 3.11). □

Before proving the correctness of our histogram implementation in chapter 4 we state some conditions equivalent to 3.1 I. resp. 3.1 II. which will be shown to hold for our example.

First we give a characterization of enrichments.

Let $SPEC=\langle S,\Sigma,E\rangle$ and $SPEC'=\langle S,\Sigma',E'\rangle$ be two specifications such that $\Sigma\subseteq\Sigma'$ and $E\subseteq E'$. Then we have a unique Σ-homomorphism h from T_{SPEC} to $T_{SPEC'}$. Moreover, h is defined by the following commutative diagram where inc is the inclusion of terms and nat, nat' are natural homomorphisms:

$$\begin{tikzcd} T_{\Sigma} \arrow[r, "inc"] \arrow[d, "nat"'] \arrow[dr, phantom, "(1)"] & T_{\Sigma'} \arrow[d, "nat'"] \\ T_{SPEC} \arrow[r, "h"'] & T_{SPEC'} \end{tikzcd}$$

SPEC' is called an enrichment of SPEC if h is bijective.

3.3 LEMMA (correctness criterium I)

SPEC' is an enrichment of SPEC iff one of the following conditions holds true:

1. For all $t'\in T_{\Sigma'}$ there is $t\in T_{\Sigma}$ such that $t\equiv_{E'}t'$, and for all $t1,t2\in T_{\Sigma}$ $t1\equiv_{E'}t2$ implies $t1\equiv_{E}t2$.
2. T_{SPEC} can be extended to a SPEC'-algebra and h to a Σ'-homomorphism.

Proof: Diagram (1) implies that 1. is equivalent to bijectivity of h.
Let A be a SPEC'-extension of T_{SPEC} such that h is Σ'-compatible. Then there is a unique Σ'-homomorphism $g: T_{SPEC'} \longrightarrow A$. Since T_{SPEC} and $T_{SPEC'}$ are initial in Alg_{SPEC} and $Alg_{SPEC'}$, respectively, (2) and (3) below are commuting diagrams that consist of Σ- and Σ'-homomorphisms, respectively. (id and id' are identities.)

$$\begin{tikzcd}
T_{SPEC} \arrow[r, "h"] \arrow[d, "id"'] \arrow[dr, phantom, "(2)", pos=0.25] & T_{SPEC'} \arrow[dl, "g"] \arrow[d, "id'"] \\
A \arrow[r, "h"'] \arrow[ur, phantom, "(3)", pos=0.75] & T_{SPEC'}
\end{tikzcd}$$

Hence h is bijective.
On the other hand, if h is bijective, then we immediately obtain a SPEC'-extension of T_{SPEC} such that h becomes Σ'-homomorphic.
Thus 2. is equivalent to bijectivity of h. □

3.4 REMARKS

Condition 3.3.1 which was already given in /EKP 78/ may be considered as an "operational" enrichment characterization because it refers exclusively to transformations of terms via the congruence relations $\equiv_E$ and $\equiv_{E'}$. Although the congruence between two terms is undecidable in general, sufficient conditions for 3.3.1 which can be verified automatically are about to be investigated. The research on term rewriting systems (see e.g. /KB 70/, /Ros 73/, /Huet 77/, /Der 79/) has influenced the formulation of "syntactical" conditions that imply 3.3.1 (cf. /GH 78/, /Mus 78/, /EKP 78/, /EKP 80/, /Pad 80/). Instead of verifying such syntactical conditions in order to prove correctness criterium I for our histogram implementation we directly show the "semantical" enrichment characterization 3.3.2. More precisely, T_{SPEC} will be replaced by another SPEC-algebra A that is Σ-isomorphic to T_{SPEC} and h by the unique Σ-homomorphism h_A from A to $T_{SPEC'}$ that is defined by

$$\begin{tikzcd}
T_{\Sigma} \arrow[r, "inc"] \arrow[d, "eval_A"'] \arrow[dr, phantom, "(4)"] & T_{\Sigma'} \arrow[d, "nat'"] \\
A \arrow[r, "h_A"'] & T_{SPEC'}
\end{tikzcd}$$

Hence h_A is $(\Sigma'-\Sigma)$-compatible iff for all $n \in \mathbb{N}$, $s1,\dots,sn,s \in S$, $\sigma: s1\dots sn \longrightarrow s$ in $\Sigma'-\Sigma$ and all $ti \in T_{\Sigma,si}$, $1 \leq i \leq n$, we have $h_A(\sigma_A(t1_A,\dots,tn_A)) = [\sigma(t1,\dots,tn)]_{E'}$, (5)
where $t_A = eval_A(t)$.

A SPEC-algebra A that is isomorphic to T_{SPEC} may be called an <u>abstract model</u> of SPEC. It was shown elsewhere that $A \cong T_{SPEC}$ iff $eval_A$ is bijective on some set of Σ-terms which contains a representative of each equivalence class in T_{SPEC}.

Finally, let us point out that the second part of 3.3.1 as well as the first part of 3.3.2 are both equivalent to the injectivity of h. Therefore the first part of 3.3.1 together with the first part of 3.3.2 is also an enrichment characterization. □

Given a weak implementation IMPL of SPECO by SPEC1 (cf. 2.2), there is an unique ΣO-homomorphism l from T_{SPECO} to $REP_{IMPL}/\equiv_{E+EO}$ that is defined by the following diagram where eval' is the restriction of eval to its image REP_{IMPL} and nat, nat' are natural homomorphisms (cf. 3.1):

$$\begin{tikzcd} T_{\Sigma+\Sigma O} \arrow[r, "eval'"] \arrow[d, "nat"'] \arrow[dr, phantom, "(6)"] & REP_{IMPL} \arrow[d, "nat'"] \\ T_{SPECO} \arrow[r, "l"'] & REP_{IMPL}/\equiv_{E+EO} \end{tikzcd}$$

Since eval' and nat are surjective, l is surjective, too.
The following characterization of RI-correctness is also given in /EKP 79 a,b/ (Theorem 4.3 resp. 5.5).

3.5 <u>LEMMA (correctness criterium II)</u>

IMPL is RI-correct iff one of the following conditions holds true:

1. For all $t1,t2 \in T_{\Sigma+\Sigma O}$ $t1 \equiv_{\bar{E}} t2$ implies $t1 \equiv_{E+EO} t2$ where $\bar{E}$=E+E1+EHID+EOP.
2. There is a $(\Sigma+\Sigma O)$-homomorphism $rep: REP_{IMPL} \longrightarrow T_{SPECO}$.

<u>Proof:</u> 1. and 2. are equivalent because $(\Sigma+\Sigma O)$-compatibility of rep implies $rep \circ eval' = nat$ and, vice versa, if rep is a function that satisfies $rep \circ eval' = nat$, then rep is $(\Sigma+\Sigma O)$-homomorphic.

If l is injective, then rep exists by the well-known diagonal fill-in lemma (cf. /AM 75/). On the other hand, since T_{SPECO} satisfies E+EO, rep induces a $(\Sigma+\Sigma O)$-homomorphism $rep': REP_{IMPL}/\equiv_{E+EO} \longrightarrow T_{SPECO}$. By initiality of T_{SPECO} in Alg_{SPECO} we have $rep' \circ l = id$. Hence l is injective. □

3.6 <u>REMARKS</u>

Our remark in 3.4 concerning the operational enrichment characterization 3.3.1 also applies to 3.5.1.

The homomorphism rep in 3.5.2 is mostly called <u>abstraction function</u>.

rep guarantees a structure-preserving representation of T_{SPECO}. Moreover, rep is always surjective because rep∘eval'=nat (cf. diagram (6)) so that the representation is complete. The abstraction function is central to all implementation concepts. It is called representation function in ALPHARD and mapping function in SPECIAL and is sometimes given by an "equality interpretation" (cf. /GHM 78/) that would be a $(\Sigma+\Sigma O)$-congruence on REP_{IMPL} in our approach.
Note that rep is only defined on those OPIMPL-data which are generated by $(\Sigma+\Sigma O)$-operations. □

4. THE HISTOGRAM IMPLEMENTATION IS CORRECT

In this chapter we present the correctness proof for our histogram implementation IMPL given in Example 2.4. We provide abstract models A and B for SPEC1 resp. SPECO and show that OPIMPL is an enrichment of SORTIMPL and that IMPL is RI-correct using Lemma 3.3 and 3.5, respectively.

Let SPEC1 = $\underline{tup}_m(\underline{nat}) + \underline{array}(\underline{string}) + \underline{array}(\underline{nat})$.

The abstract model that makes precise what we imagined when writing SPEC1 is given by the following SPEC1-algebra A. The carrier sets of A are

$A_{\underline{string}} = Z^*$ for some alphabet Z,
$A_{\underline{nat}} = \mathbb{N}$, $A_{\underline{nat}_m} = \{0,\ldots,m-1\}$, $A_{\underline{tup}} = \mathbb{N}^m$,
$A_{\underline{array1}} = \{f\colon \mathbb{N} \longrightarrow Z^* \mid f(n) = \varepsilon$ for all but finitely many $n \in \mathbb{N}\}$,
$A_{\underline{array2}} = \{f\colon \mathbb{N} \longrightarrow \mathbb{N} \mid f(n) = 0$ for all but finitely many $n \in \mathbb{N}\}$.

All operations of $\underline{tup}_m$ have obvious meanings in A, and the $\underline{array}$-operations are defined as follows:
For all $n \in \mathbb{N}$, $f \in A_{\underline{array1}}$ resp. $f \in A_{\underline{array2}}$ and $x \in Z^*$ resp. $x \in \mathbb{N}$ we have

$NEW_A(n) = \varepsilon$ resp. $NEW_A(n) = 0$,
$ASSIGN_A(f,n,x) = \lambda i.$ $\underline{if}$ $i=n$ $\underline{then}$ x $\underline{else}$ $f(n)$,
$f[n]_A = f(n)$.

The proof of $A \cong T_{SPEC1}$ is left to the reader (cf. Remark 3.3.3).
A represents the sort implementation of $\underline{histogram}$ by

$A_{\underline{hist}} = A_{\underline{tup}} \times A_{\underline{array1}} \times A_{\underline{array2}}$ and
$TRIPLE_A(t,f,g) = (t,f,g)$.

In order to show that OPIMPL is an enrichment of SORTIMPL we want to apply Lemma 3.3 to SPEC=SORTIMPL and SPEC'=OPIMPL and therefore define the operations of $\Sigma HID+\Sigma O$ on A as follows.
We assumed that SORTIMPL+<∅,HASH,E(HASH)> is an enrichment of SORTIMPL where E(HASH) is the subset of EHID that specifies HASH. Hence, by Lemma 3.3, HASH can be defined on A such that A satisfies E(HASH) and h_A is compatible with HASH.

For all $f \in A_{\underline{array1}}$, $w \in Z^*$ and all $n \in \mathbb{N}$

$SEARCHSLOT_A(f,n) = \min\{i \in \mathbb{N} \mid i \geq n, f(i) = \varepsilon\}$ and

$SEARCHHIT_A(f,w,n) = \min\{i \in \mathbb{N} \mid i \geq n, f(i) = w \text{ or } f(i) = \varepsilon\}$.

Since each of the operations LOC, INCREASE, $\emptyset$, INSERT and HOWMANY is implemented as a derived operation (cf. /EKP 78/, 2.5), it is simply defined on A by interpreting the right side of its respective (EHID+EOP)-equation in A. For example, for all $g \in A_{\underline{array}}$ and all $n \in \mathbb{N}$

$INCREASE_A(g,n) = \lambda i.\ \underline{if}\ i=n\ \underline{then}\ g(i)+1\ \underline{else}\ g(i)$.

Clearly, this extension of A to a (ΣHID+ΣO)-algebra satisfies EHID+EOP. Hence A is an OPIMPL-algebra.

It remains to show that h_A (cf. 3.4) is compatible with ΣHID+ΣO. Of course, this holds true for the derived operations of ΣHID+ΣO, while compatibility with σ=SEARCHSLOT (and, analogously, with SEARCHHIT) is proved as follows:

For all $t1 \in T_{\Sigma+\Sigma1,\underline{array1}}$ and all $t2 \in T_{\Sigma,\underline{nat}}$ let

$$n(t1,t2) = \sigma_A(t1_A,t2_A)-t2_A.$$

We show 3.4(5) by induction on $n(t1,t2)$. If $n(t1,t2) = 0$, then $\sigma_A(t1_A,t2_A)=t2_A$ and $t1_A(t2_A)=\varepsilon$ by definition of σ_A. Since $eval_A$ is ($\Sigma+\Sigma1$)-homomorphic, $t1[t2]_A=\varepsilon_A$. Thus $t1[t2] \equiv_{E+E1} \varepsilon$ so that $EQ?(t1[t2],\varepsilon) \equiv_{E+E1} TRUE$. Hence $\sigma(t1,t2) \equiv_{E+E1+EHID} t2$, and we get

$h_A(\sigma_A(t1_A,t2_A))=h_A(t2_A)=[t2]_{E'}=[\sigma(t1,t2)]_{E'}$, by 3.4(4) where E'=E+E1+EHID+EOP.

If $n(t1,t2)>0$, then $\sigma_A(t1_A,t2_A)=\sigma_A(t1_A,SUCC(t2)_A)$ and $t1_A(t2_A) \neq \varepsilon$. Therefore $t1[t2] \not\equiv_{E+E1} \varepsilon$ so that $EQ?(t1[t2],\varepsilon) \equiv_{E+E1} FALSE$. Hence $\sigma(t1,t2) \equiv_{E+E1+EHID} \sigma(t1,SUCC(t2))$.

$SUCC(t2)_A=t2_A+1$ implies $n(t1,SUCC(t2))<n(t1,t2)$. Thus we obtain $h_A(\sigma_A(t1_A,t2_A))=h_A(\sigma_A(t1_A,SUCC(t2)_A))=[\sigma(t1,SUCC(t2))]_{E'}=[\sigma(t1,t2)]_{E'}$, by induction hypothesis.

As we have already seen, the semantics of SPECO=histogram may be represented by the multisets of strings. The following abstract model B for histogram describes such multisets by their characteristic functions:

$B_{\underline{string}} = Z^*$ (see above), $B_{\underline{nat}} = \mathbb{N}$,

$B_{\underline{hist}} = \{b: Z^* \longrightarrow \mathbb{N} \mid b(w)=0 \text{ for all but finitely many } w \in Z^*\}$.

The operations of histogram are defined accordingly.

Let A' be the subalgebra of A that consists of all ($\Sigma+\Sigma$O)-generable elements of A. Then $A' \cong REP_{IMPL}$ (cf. 3.1). In order to get a well-defined abstraction function $rep: A' \longrightarrow B$ one must show that for all $(t,f,g) \in A'_{\underline{hist}}$ f is injective up to ε, i.e. $f(i)=f(j)$ implies $i=j$ or $f(i)=\varepsilon$. But this property follows from the fact that (t,f,g) is generated by ($\Sigma+\Sigma$O)-operations. Therefore rep is given by

$rep(x) = x \quad$ for all $x \in Z^* \cup \mathbb{N}$

$rep(t,f,g) = \lambda w.\ \underline{if}\ f(i) = w\ \underline{then}\ g(i)\ \underline{else}\ 0 \quad$ for all $(t,f,g) \in A'_{\underline{hist}}$.
The proof that rep is $(\Sigma+\Sigma O)$-homomorphic is rather tedious but straightforward and thus omitted here.

Hence, by Lemma 3.5, our histogram implementation is RI-correct, and the correctness proof of Example 2.4 is finished.

□

The abstract models A and B for SPEC1 resp. SPEC0 may be replaced by canonical term algebras as introduced in /GTW 78/ and further investigated in /Nou 79/. The utility of canonical term algebras in correctness proofs for implementations has been demonstrated in /Pad 79/ at an implementation of stacks by array-pointer pairs. The proofs that A satisfies EHID+EOP and that rep is $(\Sigma+\Sigma O)$-homomorphic were done by structural inductions and term replacements.

REFERENCES

/AM 75/ Arbib, M.A., Manes, E.G.: Arrows, Structures, and Functors, Academic Press, New York, 1975

/BG 77/ Burstall, R.M., Goguen, J.A.: Putting Theories together to Make Specifications, Proc. Int. Conf. Artificial Intelligence, Boston, 1977

/Der 79/ Dershowitz, N.: Orderings for Term-Rewriting Systems, Proc. 20th IEEE Symp. on FOCS, 1979, 123-131

/Dij 72/ Dijkstra, E.W.: Notes on Structured Programming, in: Structured Programming, C.A.R. Hoare, Ed., Academic Press, New York, 1972

/EKMP 80/ Ehrig, H., Kreowski, H.-J., Mahr, B., Padawitz, P.: Compound Algebraic Implementations: An Approach to Stepwise Refinement of Software Systems, 1980,Bericht Nr.80-4,TU Berlin,FB 20,1980

/EKP 78/ Ehrig, H., Kreowski, H.-J., Padawitz, P.: Stepwise Specification and Implementation of Abstract Data Types, Proc. 5th ICALP, Udine 1978, Springer Lect. Not. in Comp. Sci. 62, 205-226

/EKP 79a/ --: Algebraische Implementierung abstrakter Datentypen, Forschungsbericht Nr. 79-3, TU Berlin, FB 20, 1979

/EKP 79b/ --: Algebraic Implementation of Abstract Data Types: Concept, Syntax, Semantics and Correctness, 1979, accepted for ICALP 80

/EKP 80/ --: Completeness in Algebraic Specifications, to appear in Bull. EATCS, No. 11 , 1980

/GH 78/ Guttag, J.V., Horning, J.J.: The Algebraic Specification of Abstract Data Types, Acta Informatica 10, 1978, 27-52

/GHM 78/ Guttag, J.V., Horowitz, E., Musser, D.R.: Abstract Data Types and Software Validation, Comm. ACM, Vol. 21, No. 12, 1978, 1048-1063

/GN 78/ Goguen, J.A., Nourani, F.: Some Algebraic Techniques for Proving Correctness of Data Type Implementation, Extended Abstra(Comp. Sci. Dept., UCLA, Los Angeles, 1978

/GT 78/ Goguen, J.A., Tardo, J.J.: An Introduction to OBJ: A Language for Writing and Testing Formal Algebraic Specifications, Techn. Report, Univ. of California at LA, 1978

/GTW 78/ Goguen, J.A., Thatcher, J.W., Wagner, E.G.: An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types, in: Current Trends in Programming Methodology, IV: Data Structuring (R.Yeh Ed.), Prentice Hall, New Jersey, 1978, 80-144

/Gut 76/ Guttag, J.V.: Abstract Data Types and the Development of Data Structures, Supplement to Proc. Conf. on Data Abstraction, Definition, and Structure, SIGPLAN Notices 8, March 1976

/Huet 77/ Huet, G.: Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems, Proc. 18th Symp. on FOCS, 1977, 30-45

/KB 70/ Knuth, D., Bendix, P.: Simple Word Problems in Universal Algebras, in: Computational Problems in Abstract Algebra, J.Leech, Ed., Pergamon Press, Oxford 1970, 263-297

/LRS 79/ Levitt, K.N., Robinson, L., Silverberg, B.A.: The HDM Handbook, SRI International, Menlo Park, 1979

/LS 77/ Lehmann, D.H., Smyth, M.B.: Data Types, Univ. of Warwick, Dept. of Comp. Sci., Report No. 19, 1977, and Proc. 18th IEEE Symp. on Found. of Computing, Providence, R.I., Nov. 1977, 7-12

/Mus 78/ Musser, D.R.: A Data Type Verification System Based on Rewrite Rules, Univ. of Southern California, ISI Report, 1978

/Nou 79/ Nourani, F.: Constructive Extension and Implementation of Abstract Data Types and Algorithms, Ph.D.Thesis, University of California at LA, 1979

/Pad 79/ Padawitz, P.: Proving the Correctness of Implementations by Exclusive Use of Term Algebras, Forschungsbericht Nr. 79-8, TU Berlin, FB 20, 1979

/Pad 80/ --: New Results on Completeness and Consistency of Abstract Data Types, 1980, submitted to 5th Conf. on Automated Deduction

/Par 72/ Parnas, D.L.: A Technique for Module Specification with Examples, Comm. ACM, Vol. 15, No. 5, 1972, 330-336

/RL 77/ Robinson, L., Levitt, K.N.: Proof Techniques for Hierarchically Structured Programs, Comm. ACM, Vol. 20, No. 4, 271-283

/Ros 73/ Rosen, B.K.: Tree-Manipulating Systems and Church-Rosser Theorems, Journal ACM, Vol. 20, No. 1, 1973, 160-187

/RR 77/ Roubine, O, Robinson, L.: SPECIAL Reference Manual, 3rd Edition, SRI Report No. CSG-45, Menlo Park, 1977

/TWW 78/ Thatcher, J.W., Wagner, E.G., Wright, J.B.: Data Type Specification: Parameterization and the Power of Specification Techniques, Proc. 10. SIGACT Symp. on Theory of Computing, San Diego 1978, 119-132

/Wir 71/ Wirth, N.: Program Development by Stepwise Refinement, Comm. ACM, Vol. 14, No. 4, 1971, 221-227

/WLS 76/ Wulf, A., London, R.L., Shaw, M.: Abstraction and Verification in ALPHARD: Introduction to Language and Methodology, Techn. Report, Carnegie-Mellon Univ., 1976