I/O-COMPUTABLE DATA STRUCTURES

J. A. Bergstra and J.-J. Ch. Meyer
Department of Computer Science
University of Leiden
Wassenaarseweg 80
Postbus 9512
2300 RA LEIDEN
The Netherlands

According to ADJ [6] a data structure is a many-sorted algebra that satisfies various equations and conditional equations. Several authors [3], [4], [5], [8], [9] have emphasised that visible and non-visible sorts should be distinguished; objects that belong to an invisible sort show a so-called observable behaviour and all of the authors mentioned point out that on these sorts terminal equivalence rather than initial equivalence is the more interesting congruence relation. Objects that have indistinguishable observable behaviour are made equivalent in this terminal congruence.

We have in mind to present the reader with a survey of some terminology and several new definitions, notably that of an I/O-computable data structure. We shall define I/O-equivalence and prove that for each data structure A there is a computable data structure B that is I/O-equivalent to A. This very simple result may prove to be significant. First of all it shows the adequacy of the concept of an I/O-computable data structure, as in principle all such data structures can practically exist. Moreover this fact suggests that both initial and terminal semantics of equations may differ from what actually occurs in an implementation. This means that both should be considered as theoretical concepts that can be used to describe a data structure, given certain purposes.

Certainly this result gives a post facto justification of the restriction to computable algebras that was made in [1] and [2].

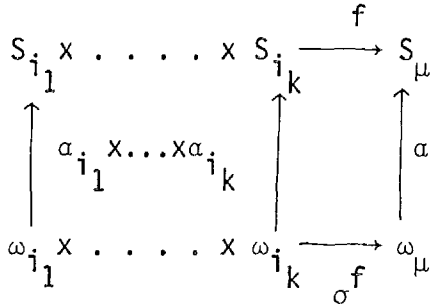
Some technical terms have to be introduced in our discussion now. Let A be an algebra with sorts

$$S_0,\ldots,S_{k-1},S_k,\ldots,S_{\ell-1}\ .$$

We assume A to be finitely generated by its finite signature Σ (i.e. A is minimal). $S_0,\ldots,S_{k-1}$ are the visible sorts, $S_k,\ldots,S_{\ell-1}$ the invisible ones. The terminology visible~invisible being not so pleasant in the long run, we propose to call the visible sorts I/O-sorts and the invisible sorts internal sorts. Elements of I/O-sorts can be seen as tokens on which an equality relation is fixed. Elements of the internal sorts are the true data structures. It is on the internal sorts that investigating the variety of congruence relations that respect the I/O-behaviour (i.e. the word problem on the I/O-sorts) is interesting.

Coming back to the structure A the sorts $S_0,\ldots,S_{k-1}$ are I/O-sorts and $S_k,\ldots,S_{\ell-1}$ are internal sorts. We choose $\omega_0,\ldots,\omega_{\ell-1}$ as disjoint copies of ω, the non-negative integers. A co-ordinatization of A is a tuple of mappings $\alpha_0,\ldots,\alpha_{\ell-1}$ plus a collection of tracking functions σ^f for all $f \in \Sigma$, that satisfy the following conditions:

$\alpha_i : \omega_i \to S_i$ a surjective mapping for each $f \in \Sigma, f: S_{i_1} x \ldots x S_{i_k} \to S_\mu$ the diagram



commutes. Moreover, the functions σ^f are computable.

On each of the ω_i a congruence $\equiv_i^\alpha$ is induced as follows:

$n \equiv_i^\alpha m \Leftrightarrow \alpha_i(n) = \alpha_i(m)$.

Two important definitions can now be stated.

DEFINITION I. A is computable if there is a coordinatization α_i, σ^f $(f \in \Sigma)$ of A such that for each i the congruence relation $\equiv_i^\alpha$ is computable.

DEFINITION II. A is I/O-computable if there is a coordinatization α_i, σ^f $(f \in \Sigma)$ of A such that for each $i < k$ (i.e. for all I/O-sorts) $\equiv_i^\alpha$ is computable.

These definitions are compatible with the definitions in [1] and [2].

We propose that a data structure is an arbitrary I/O-computable minimal algebra. The condition that the I/O-behaviour of a data structure is computable is justified in a deterministic context, as it is inconceivable that any other data structures have an effective implementation (realization).

An algebra A is semicomputable if for all sorts the word problem is semicomputable. A is called co-semicomputable if for all sorts the complement of the word problem is semicomputable. Clearly if A is a data structure these definitions refer to the internal sorts only.

Two data structures A and B are called comparable, according to KAMIN [3] if they have the same signature.

DEFINITION III. Let A,B be comparable data structures. A and B are called I/O-equivalent if for all I/O-sorts S and for all terms t_1^S, t_2^S of sort S:

$$A \models t_1^S = t_2^S \Leftrightarrow B \models t_1^S = t_2^S .$$

I/O-equivalence of A and B implies that both have identical word problems on the I/O-sorts.

A data abstraction K is a collection of I/O-equivalent data structures. $A \in K$ is called an implementation of K.

Given a data structure A we denote with $G_{I/O}(A)$ the set of all closed term identities $t_1^S = t_2^S$ that hold true in A for I/O-sorts S. Obviously our convention implies that $G_{I/O}(A)$ is a decidable set of pairs of terms. Suppose that A satisfies equations (or conditional equations) E. This leads to a natural data abstraction:

$$DA(G_{I/O}(A) \cup E) = \{B \mid B \text{ is I/O-equivalent with A, and } B \models E\}.$$

TERMINAL SEMANTICS.

PROPOSITION I. Let A be a data structure of signature Σ. There exists a data structure B, I/O-equivalent to A with the following property:

If C is I/O-equivalent to A then there is a homomorphism $\varphi: C \to B$.

B is isomorphic to $A/\equiv$ for a congruence $\equiv$ that is usually called the terminal congruence on A.

PROOF. See [4], [5], [7], [8].

T_Σ denotes the term algebra of signature Σ.

PROPOSITION II. The terminal congruence on $A,\equiv$, seen as a congruence on T_Σ, is co-semicomputable.

PROOF. Straightforward.

On the other hand, given $G_{I/O}$ (A) and a finite set of (conditional) equations E the initial algebra of $G_{I/O} \cup E$ is semicomputable, as are all initial algebras of decidable sets of axioms.

In a forthcoming note we will show that a data structure A exists for which $DA(G_{I/O}(A) \cup E)$ has no computable element with E a set of equations such that $T_{\Sigma,E} \cong A$.

Here $T_{\Sigma,E}$ is the initial algebra in $ALG(\Sigma,E)$, or, differently: $T_\Sigma/\equiv_E$ where $\equiv_E$ is the congruence of E-provable equality on T_Σ.

In this case there is no computable congruence between the initial and the terminal one.

COMPUTABLE IMPLEMENTATIONS.

We will next consider the case, where $G_{I/O}(A)$ is given but $E = \emptyset$. The question is: must there exist a computable implementation of $DA(G_{I/O}(A))$? This is indeed the case:

PROPOSITION III. Let A be a data structure then $T_{\Sigma,G_{I/O}(A)}$ is a computable algebra in $DA(G_{I/O}(A))$.

PROOF. Choose for each I/O-sort S a sequence t_i^S such that all elements of S occur as the value of t_i^S for exactly one i. Moreover, the sequence t_i^S must be uniformly given from i. If S happens to be finite in A then the sequence is finite as well.

Introduce a function f^S for each S such that in a computable way f^S: $T_\Sigma^S \to \omega$ and $A \models r = t^S_{f^S(r)}$ for all terms r of sort S.

Now one defines the congruence $\equiv$ on T_Σ as follows:

for an I/O-sort S : $t^S \quad r^S$ iff $A \models t^S \equiv r^S$ ($\Leftrightarrow G_{I/O}(A) \vdash t^S = r^S \Leftrightarrow$ $f^S(t^S) = f^S(r^S)$.

Notice that $f^S(t^S)$ is a unique normal form of t^S under the rewrite rule $t^S \to f^S(t^S)$

For an internal sort S one also introduces a unique notation $F(\tau^S)$ for each term τ^S. $F(\tau^S)$ results from τ^S by replacing each of its subterms t^T of an I/O-sort T by $f^T(t^T)$.

We put $t^S \equiv r^S$ iff $F(\tau^S) = F(r^S)$ (syntactic equality is meant here).

$\equiv$ is a computable congruence relation that extends $G_{I/O}(A)$. In fact $T_\Sigma/\equiv \cong T_{\Sigma,G_{I/O}(A)} \in DA(G_{I/O}(A))$.

CONCLUDING REMARKS.

In [1] and [2] a fruitful format of mathematical problems is as follows:

Given a computable algebra, allow the addition of hidden enrichment functionsand then investigate initial algebra specifications of the enriched algebra.

The present situation rather suggests to start with $G_{I/O}(A)$ for some data structure A of signature Σ which has a specification (Σ,E) with (conditional) equationsE. The problem is now to find another specification (Σ,E_0) (of an algebra B,I/O-equivalent to A) such that E_0 has some additional properties (for instance being sufficiently powerful to prove some partial correctness conditions).

The interesting issue here is that several of such requirements may be incompatible. This is the case with two specifications E_0 and E_1 of I/O-equivalent data structures A and B such that $DA(G_{I/O}(A) + E_0 \cup E_1) = \emptyset$. Of course this can only happen if E_0 or E_1 contain conditional equations that are not preserved under homomorphisms.

Mathematically speaking one introduces a collection $SPEC(G_{I/O}(A))$ of all finite sets of conditional equations E such that $T_{\Sigma,E}$ is I/O-equivalent to A. On this collection there is a partial order $\sqsubseteq$ defined by $E_0 \sqsubseteq E_1$ if all axioms in E_0 can be proved from E_1. In the general case $\sqsubseteq$ will neither show a smallest nor a largest element. This setting seems appropriate for us for the mathematical investigation of specifications not allowing hidden functions.

REFERENCES

[1] Bergstra, J.A. & J.V. Tucker, Algebraic specifications of computable and semicomputable data structures, Mathematical Centre, Department of Computer Science Research Report IW 115, Amsterdam 1979.

[2] Bergstra, J.A. & J.V. Tucker, Equational specifications for computable data types: six hidden functions suffice and other sufficiency bounds, Mathematical Centre, Department of Computer Science Research Report IW 128, Amsterdam 1980.

[3] Botha, K. Institutsbericht, Humboldt Universität zu Berlin [1979].

[4] Broy, M. & M. Wirsing, The theory of recursive functions as an abstract type and its semantical models. Report, Institut für Informatik, Technische Universität München [1980].

[5] Giarattana, F. Gimona & V. Montanari, Observability concepts in abstract data types specification, Proc. of 5th MFCS symposium, LNCS 45 (576-587) 1976.

[6] Goguen, J.A., J.W. Thatcher & E.G. Wagner, An initial algebra approach to the specification, correctness and implementation of abstract data types, in R.T. Yeh (ed.). Current trends in programming methodology IV, Data Structures, Prentice Hall, Engelwood Cliffs, New Jersey, 1978, 80-149.

[7] Hornung, G. & P. Raulefs, Terminal algebra semantics and retractions for abstract data types. Proceedings of ICALP '80, Spr. LNCS 85.

[8] Kamin, S., Final data type specifications: a new data type specification method, 7th POPL Conference, Las Vegas, ACM 1980, 131-138.

[9] Kapur, D. & M.K. Srivas, Expressiveness of the operation set of a data abstraction, in 7th ACM POPL Conference, Las Vegas, ACM 1980.