NON-TERMINATION, IMPLICIT DEFINITIONS AND ABSTRACT DATA TYPES*

by

T.S.E. Maibaum

Computer Science Dept.
University of Waterloo
Waterloo, Ontario N2L 3G1
CANADA

Research Report CS-79-26
June 1979

*This work was supported by a grant from the National Sciences and Engineering Research Council of Canada.

Abstract

Based on the observation that non-termination of procedures and the use of implicitly defined constructs (recursion, iteration) in implementations are not taken into account in the algebraic theory of abstract data types, we propose an extension of this theory to take these factors into account. The extension is based on the concept of continuous algebras and on recent developments (partially) generalising the theory of abstract data types to this setting. The nature of the extension is such that there is a simple and automatic transformation from a conventional specification to the corresponding "continuous" specification. Thus the conventional theory need not be abandoned but may be included in the generalisation in a straightforward manner.

Introduction

Much work has been done in recent years to develop a mathematical theory of data types [8,2,3,9,10,11,12]. The main aim of all these methods has been the abstract or representation independent characterisation of data types. Probably the most effective approach has been the algebraic one as exemplified in [2,11,9,10].

In parallel, much work has been done on the semantics of programming languages using the algebraic approach [1,4,6,7,19,20]. One main factor divides these subject areas as far as the uses of algebra are concerned. This is the use of "normal" algebras in the study of abstract data types and the use of continuous algebras in the study of semantics. In [15,16] we have tried to make the point that a theory of data types using conventional algebras is not good enough since it is not possible to characterise some data types (e.g. data types with sharing and/or circularities) using conventional algebras. Even if it is possible, the characterisation may not be the most elegant (see, for example, the treatment of referencing in [13]).

Our aim in this section is to show that a theory of continuous data types is needed for a completely different reason. We can illustrate our points through a simple example. Suppose that we have defined the data type sequence of integers which has amongst its operations the test isintinseq (to test a sequence to see if it contains a specified integer) and insert (which inserts an integer at the end of a specified sequence). Now consider the operation

insert(n,s).

This is quite normal and makes sense. Now replace n by the expression f(x) where f is a procedure which returns an integer. Thus we have

insert(f(x),s).

This still makes sense only if f terminates with an integer value. If f does not terminate, what is the value of the insert operation? It is impossible to tell from the normal specification since "undefined" is not a formal value of the data type. It is not an error value since "undefined" is not a value in the normal sense. It seems clear that one way of overcoming this problem (perhaps the only way using algebras) is to introduce "undefined" as a formal object in the algebras and thus consider continuous algebras.

Now consider the operation isintinseq. Suppose that we implement the type using linked lists (i.e., pointers). How will we implement the operation isintinseq? The natural implementation is

```
operation isintinseq (n: integer; s: sequence): boolean;

    begin
        m := first(s);
        while   endofseq(s) do
            if m=n then isintinseq := true
            else m := next(s)
        end
```

The only operations we have used are integer or linked list operations. However, the "meaning" of this program as an expression is an *infinite* expression (obtained by unfolding the while loop). (See [1,7,6,19,20])

The usual concept of implementation (as in [2,12]) requires that isintinseq be implemented in terms of a finite expression over the integer and linked list operations as only such expressions define derived operations over conventional algebras. However, as the example above points out, it is not the natural way to define implementations of operations and in fact it may not even be possible in all cases (e.g. where recursion is required).

On the other hand, the infinite expression above does belong to a well defined continuous algebra. Thus again continuous algebras seem to offer a possible solution. We confine ourselves in this report to showing how the conventional algebraic theory of data types can be transformed into a continuous theory in a straightforward manner. We refer the reader to [13,15,16] for a more general attempt to use continuous algebras to define data types.

Mathematical Preliminaries

Let $\Sigma = \{\Sigma_{w,s}\}_{<w,s>\in S^*\times S}$ be a <u>many-sorted alphabet</u> sorted by the sorting set S. A symbol $f \in \Sigma_{w,s}$ is said to be of <u>type</u> $<w,s>$, <u>arity</u> w, <u>sort</u> s, and <u>rank</u> $|w|$ (where $|w|$ is the length of the string $w \in S^*$). If $w = \lambda$, f is said to be a <u>constant</u> or <u>nullary</u> symbol of sort s. A <u>Σ-algebra</u> A_Σ is a family of sets $\{A_s\}_{s\in S}$ together with an assignment of operations to symbols in Σ so that $f \in \Sigma_{w,s}$ is assigned an operation $f_A\colon A_{s_1}\times\ldots\times A_{s_n}\to A_s$ for $w = s_1\ldots s_n$. (We denote $A_{s_1}\times\ldots\times A_{s_n}$ by A^w.)

Given Σ-algebras A_Σ and B_Σ, a (Σ)-<u>homomorphism</u> $h\colon A_\Sigma\to B_\Sigma$ is a family of mappings $h = \{h_s\}_{s\in S}$ such that for $f \in \Sigma_{w,s}$ and $a_i \in A_{s_i}$, where $w = s_1\ldots s_n$ and $1 \le i \le n$, we have

$$h(f_A(a_1,\ldots,a_n)) = f_B(h(a_1),\ldots,h(a_n)).$$

(Note that for convenience we have dropped subscripts from the h's.) Homomorphisms which are injective, surjective, or bijective are called <u>monomorphisms</u>, <u>epimorphisms</u>, and <u>isomorphisms</u>, respectively. An algebra A_Σ is said to be <u>initial</u> in a class $\mathcal{C}$ of Σ-algebras if $A_\Sigma\in\mathcal{C}$ and if for each $B_\Sigma\in\mathcal{C}$ there is a unique homomorphism $h\colon A_\Sigma\to B_\Sigma$.

<u>Theorem</u>: The class of all Σ-algebras has an initial algebra denoted T_Σ. We can think of T_Σ as the algebra of (finite) expressions over the alphabet Σ.

<u>Example</u>: Our standard example throughout this report will be the data type "stack of natural numbers". We use the following alphabet (using the

notation of [2] to specify our data type):

$$
\begin{array}{rl}
S = \{\underline{st}, \underline{nat}\}; & \\
\text{zero:} & \to \underline{nat} \\
\text{succ:} & \underline{nat} \to \underline{nat} \\
\Lambda: & \to \underline{st} \\
\text{push:} & \underline{nat} \times \underline{st} \to \underline{st} \\
\text{pop:} & \underline{st} \to \underline{st} \\
\text{top:} & \underline{st} \to \underline{nat} \\
\text{error}_{nat}: & \to \underline{nat} \\
\text{error}_{st}: & \to \underline{st}
\end{array}
$$

□

Let $X = \{X_s\}$ by any family of sets. We define the set of <u>expressions generated</u> by the <u>variables</u> X as follows:

(i) $X_s \subseteq T_\Sigma(X)_s$ for each $s \in S$;

(ii) If $f \in \Sigma_{w,s}$, $w = s_1 \ldots s_n$ and $t_i \in T_\Sigma(X)_{s_i}$ for $1 \le i \le n$, then $ft_1 \ldots t_n \in T_\Sigma(X)_s$.

We can make $T_\Sigma(X)$ into a Σ-algebra by defining $f_{T_\Sigma(X)}(t_1,\ldots,t_n) = ft_1\ldots t_n$. If each $X_s = \phi$, then $T_\Sigma(\{\phi\}_{s\in S})$ is isomorphic to T_Σ.

Let A_Σ be an Σ-algebra and $a\colon X \to A$ an <u>assignment</u> of values to variables. Then we have the result that a extends uniquely to a homomorphism $\bar{a}\colon T_\Sigma(X) \to A_\Sigma$ so that $\bar{a}$ agrees with a on X. (For a proof see [1]).

Let $X_w = \{x_{1,s_1},\ldots,x_{n,s_n}\}$ for $w = s_1 \ldots s_n$. We denote by $T_\Sigma(X_w)$ the algebra $T_\Sigma(\{x_{i,s_i} \mid 1 \le i \le n,\ s_i = s\}_{s\in S})$. The use of X_w will be seen below. For further discussion and examples, see [2,18].

Given a Σ-algebra A_Σ, a (Σ-)congruence q over A_Σ is a family $q = \{q_s\}_{s\in S}$ of equivalence relations with the following substitution property. If $f \in \Sigma_{w,s}$ and $a_i, b_i \in A_{s_i}$ so that $a_i q_{s_i} b_i$ for $1 \leq i \leq n$, then

$$f_A(a_1,\ldots,a_n) q_s f_A(b_1,\ldots,b_n).$$

Denote by A_Σ/q the algebra whose carrier of sort s is $\{[a] \mid a \in A_s\}$ where [a] is the congruence class of a. The operations are defined by $f_{A_\Sigma/q}([a_1],\ldots,[a_n]) = [f_A(a_1,\ldots,a_n)]$. The substitution property above guarantees the consistency of this definition. We call A_Σ/q the quotient of A_Σ by q.

A (Σ-)equation is a pair $<\ell,r>$ (written $\ell=r$) for $\ell,r \in T_\Sigma(X_w)$. An algebra A_Σ is said to satisfy $\ell=r$ if for all assignments $a: X_w \to A$, $\bar{a}(\ell) = \bar{a}(r)$. A_Σ is said to satisfy a set of equations ε if it satisfies each equation in ε separately. It is well known that a set of equations ε generates a least congruence q_ε on a Σ-algebra A_Σ (and so guarantees that A_Σ/q_ε satisfies ε). If we denote by $\underline{Alg}_{\Sigma,\varepsilon}$ the class of algebras satisfying ε, then we have the following important result.

Theorem: T_Σ/q_ε (denoted $T_{\Sigma,\varepsilon}$ in the sequel) is initial in $\underline{Alg}_{\Sigma,\varepsilon}$.

□

Example: Given the alphabet for stacks of natural numbers defined above, we consider the following equations ε_{st} to define the data type:

$$\begin{aligned}
top(push(n,s)) &= n \\
top(push(n,s)) &= s \\
top(\Lambda) &= error_{nat} \\
pop(\Lambda) &= error_{st}
\end{aligned}$$

n and s are variables of sort nat and st respectively.

□

A partially ordered set (poset) is a pair $(D, \leq_D)$ (often denoted just be D) where D is a set and $\leq_D$ is a partial order on D. D is strict if D has a minimal element (denoted by $\perp_D$). A set $D' \subseteq D$ is directed if every pair of elements d,d' in D' has an upper bound in D'. D is a complete partial order (cpo) if D is strict and each directed subset of D has a least upper bound (ℓub) in D. If $\{d_i\}_{i\in I}$ is a directed set in D, we denote by $\bigsqcup d_i$ the ℓub of the set. If D,D' are posets and $f: D \to D'$, then f is continuous if $f(\bigsqcup d_i) = \bigsqcup f(d_i)$ (assuming that both $\bigsqcup d_i$ and $\bigsqcup f(d_i)$ exist).

An algebra A_Σ is continuous if each A_s is a cpo and if each operation is continuous. A homomorphism of continuous algebras $h: A_\Sigma \to B_\Sigma$ is continuous if each h_s is. Denote by $\underline{CAlg}_\Sigma$ the class of continuous Σ-algebras together with continuous homomorphisms between them.

Theorem: $\underline{CAlg}_\Sigma$ has an initial algebra CT_Σ.

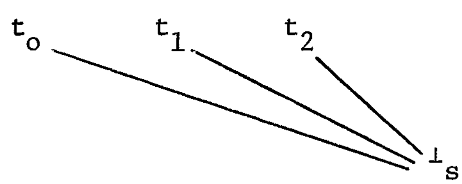
□

We can think of CT_Σ as the algebra of (finite and infinite) partially specified expressions over Σ. The least element (denoted by $\perp_s$) of $CT_{\Sigma,s}$ is the completely unspecified expression of sort s. $t \leq t'$ if t' is obtained by replacing some unspecified subexpressions of t by some "specified" expressions. If $\Sigma(\perp)$ denotes the alphabet obtained from Σ by adding $\perp_s$ to

$\Sigma_{\lambda,s}$ for each $s\epsilon S$, then the algebra of completely specified (finite and infinite) expressions over $\Sigma(\perp)$ is isomorphic to CT_Σ. The order relation on this algebra is the least order consistent with $\perp_s \le t$ for each t and if $t_i \le t_i'$ for $1 \le i \le n$ then $ft_1 \ldots t_n \le ft_1' \ldots t_n'$. We define $CT_\Sigma(X_w)$ analogously to $T_\Sigma(X_w)$.

Construction of Continuous Data Types

Suppose we have specified a data type over the alphabet Σ using the equations ε. We now want to contend with the kinds or problems discussed in the introduction. The "natural" procedure is to construct a continuous algebra from $T_{\Sigma,\varepsilon}$ by making the set $(T_{\Sigma,\varepsilon})_s$ into a "flat" (discrete) cpo ([4]) as follows: Let $\perp_s$ be a new data structure such that $\perp_s \leq t$ for all $t \in (T_{\Sigma,\varepsilon})_s$ and otherwise the elements of $(T_{\Sigma,\varepsilon})_s$ are incomparable. We can graphically illustrate this partial order $(T_{\Sigma,\varepsilon}^{\perp})_s$ as follows where comparable elements are connected by edges with "smaller" elements below larger elements in the diagram.



We can now make the family of sets $T_{\Sigma,\varepsilon}^{\perp}$ into a continuous algebra by defining for each $f \in \Sigma_{w,s}$ and each $1 \leq i \leq n$

$$f_{T_{\Sigma,\varepsilon}^{\perp}}(x_1,\ldots,x_{i-1},\perp_{s_i},x_{i+1},\ldots,x_n) = \perp_s$$

for any x_j in $(T_{\Sigma,\varepsilon}^{\perp})_{s_j}$, $j \neq i$. That this definition makes $T_{\Sigma,\varepsilon}^{\perp}$ into a continuous Σ-algebra is readily verified (see [4]).

However, we have now lost all the power of the theory of abstract data types since $T_{\Sigma,\varepsilon}^{\perp}$ is no longer a simple quotient algebra and we cannot in fact be sure that $T_{\Sigma,\varepsilon}^{\perp}$ even satisfies ε. For example if we have a binary operation + and the axiom $+xy = y$, then

$$+_{T_{\Sigma,\varepsilon}^{\perp}}(\perp,[t]) = \perp$$

whereas the right hand side of the equation is [t] for t in $T_{\Sigma,\varepsilon}$.

Example: We can make our stack example into a continuous algebra by introducing $\perp_{nat}$ and $\perp_{st}$ as values in $T_{\Sigma,\varepsilon}^{\perp}$ and extending the operations as follows:

$$\begin{aligned} top(\perp_{st}) &= \perp_{nat} \\ pop(\perp_{st}) &= \perp_{st} \\ push(\perp_{nat},s) &= push(n,\perp_{st}) = \perp_{st} \\ succ(\perp_{nat}) &= \perp_{nat} \end{aligned}$$

□

Even if $T_{\Sigma,\varepsilon}^{\perp}$ satisfies ε, it is not in general initial as a $\Sigma(\perp)$-algebra satisfying ε. This is of course a great pity since we no longer have a simple "handle" on this algebra. The results in [2,12] on proofs of correctness of specifications and implementations are no longer applicable. So the question now arises: Do we start all over again with a new theory of continuous data types or can we somehow salvage the situation.

Some recent developments in the theory of continuous data types (see [15]) turn out to be quite useful for our development. Suppose that ε is a set of equations and q_ε is the least congruence on CT_Σ generated by ε. It is shown that in general CT_Σ/q_ε is not initial in $\underline{CAlg}_{\Sigma,\varepsilon}$ (the class of continuous Σ-algebras satisfying ε together with continuous homomorphisms between them). In fact it is not even always possible to partially order the congruence class of CT_Σ/q_ε in a way which is consistent with the order on

CT_Σ. As a first step in getting around this problem, we define a continuous congruence q over a continuous algebra A_Σ to be a congruence with the following continuity property. If $\{a_i\}_{i\in I}$, $\{b_i\}_{i\in I}$ are two directed sets in some A_s so that for all $i\in I$, $a_i q_s b_i$, then $(\sqcup a_i) q_s (\sqcup b_i)$. In other words, if the elements of two directed sets are pairwise congruent, then so are the ℓub's. In [15] it is shown that a set of equations ε generates a least continuous congruence on a continuous Σ-algebra A_Σ. In fact, the class of continuous congruences form a lattice.

As the next step in our development, we will introduce a generalisation of canonical term algebras in the theory of abstract data types. A canonical term algebra for a given data type $T_{\Sigma,\varepsilon}$ is an algebra C_Σ such that

(i) $C_s \subseteq T_{\Sigma,s}$ for each $s\in S$;

and (ii) $ft_1 \ldots t_n \in C_s$ implies t_i in C_{s_i} for $w = s_1 \ldots s_n$, $1 \le i \le n$ and moreover

$$f_C(t_1, \ldots, t_n) = ft_1 \ldots t_n.$$

Canonical term algebras are useful because each congruence class of $T_{\Sigma,\varepsilon}$ is represented by a canonical representative and the operations of C_Σ preserve canonical terms. The usefulness of canonical term algebras is guaranteed by the fact that C_Σ is isomorphic to $T_{\Sigma,\varepsilon}$ and so C_Σ is initial in $\underline{Alg}_{\Sigma,\varepsilon}$. Moreover, a canonical term algebra always exists for each data type ([2]).

In the case of continuous algebras, the matter is again not so simple. However, the following partial generalisation was developed in [15]. Let q be a continuous congruence over CT_Σ and suppose there exists a function

$$nf: \; CT_\Sigma \to CT_\Sigma$$

such that

(i) $[t_1] = [t_2] \Rightarrow nf(t_1) = nf(t_2)$;

(ii) $[t] = [nf(t)]$;

(iii) nf is continuous.

nf is called a normaliser for q. We then have the following important result.

Theorem: If q_ε is a continuous congruence generated by the equations ε on CT_Σ and a normaliser for q_ε exists, then CT_Σ/q_ε is initial in $\underline{CAlg}_{\Sigma,\varepsilon}$.

□

Now, the image of CT_Σ under nf can be made into a "normal term algebra", generalising the concept of canonical term algebra. In fact, if a normal term algebra exists for CT_Σ/q_ε, then a normaliser exists ([15]). Also note that the existence of a normaliser guarantees initiality of CT_Σ/q_ε whereas we saw that this was not in general true.

Now suppose that ε is the set of equations specifying some abstract data type. Denote by ε' the set of equations obtained from ε as follows: If $\ell = r$ is in ε, for $\ell, r \in T_\Sigma(X_w)_s$, then put into ε' the equation

$$\ell = r \text{ if } x_{1,s_1} \neq \bot_{s_1} \wedge \ldots \wedge x_{n,s_n} \neq \bot_{s_n}$$

where $w = s_1 \ldots s_n$. Thus we place into ε' equations which are conditioned by requiring that none of the variables be given values which are $\bot$. (In [2] it is shown how conditioned equations can be transformed into an equivalent set of normal equations.) The failure of our first attempt resulted partly from the fact that we did not condition our equations. Let $\varepsilon'(\perp)$ be the set of equations obtained by adding to ε' for each $f \in \Sigma_{w,s}$ and each $1 \le i \le n$ (where $w = s_1 \ldots s_n$) the equation

$$f(x_{1,s_1}, \ldots, x_{i-1,s_{i-1}}, \perp_{s_i}, x_{i+1,s_{i+1}}, \ldots, x_{n,s_n}) = \perp_s.$$

We call such equations <u>strictness axioms</u> as they specify that if any argument of an operation is "undefined", then the result of the operation is "undefined".

<u>Lemma</u>: Given $q_{\varepsilon'(\perp)}$ as defined above, if t is in $CT_{\Sigma(\perp)} - T_{\Sigma(\perp)}$ but does not contain any occurrences of $\perp$, then t is congruent to $\perp$. (i.e. if t is an infinite expression then t is congruent to $\perp$.)

<u>Proof</u>: It is well known that there exists a directed set $\{t_i\}_{i \in I}$ such that each t_i is finite and $\bigsqcup t_i = t$. Moreover, each t_i contains occurrences of $\perp$. Thus each t_i is congruent to $\perp$. Thus we have two directed sets $\{t_i\}_{i \in I}$ and $\{\perp\}_{i \in I}$ whose elements are pairwise congruent. Thus $\bigsqcup t_i$ is congruent to $\bigsqcup \perp = \perp$ demonstrating the result.

$\square$

<u>Theorem</u>: $CT_{\Sigma(\perp)}/q_{\varepsilon'(\perp)}$ is initial in $\underline{CAlg}_{\Sigma(\perp),\varepsilon'(\perp)}$.

<u>Proof</u>: If we can demonstrate the existence of a normaliser for $q_{\varepsilon'(\perp)}$, then our result is proved. For $t \in T_{\Sigma,s}$, let cf(t) be the canonical form of t in the sense of [2]. (cf is in fact the unique homomorphism from T_Σ to C_Σ where C_Σ is the canonical term algebra.) Now define nf: $CT_{\Sigma(\perp)} \to CT_{\Sigma(\perp)}$ by:

$$\text{(i)} \qquad nf(\perp_s) = \perp_s \text{ for each } s\in S;$$

$$\text{(ii)} \qquad nf(t) = \begin{cases} cf(t) & \text{if } t\in T_\Sigma \\ \perp & \text{if } t\in T_{\Sigma(\perp)} - T_\Sigma; \end{cases}$$

$$\text{(iii)} \qquad nf(t) = \perp_s \text{ (for } t\in CT_{\Sigma(\perp),s}) \text{ otherwise.}$$

Now we must demonstrate the properties of normalisers. Firstly, we must show $[t] = [t']$ implies $nf(t) = nf(t')$. If $t = \perp$, then t' is in $CT_{\Sigma(\perp)} - T_\Sigma$ (since all infinite expressions and all expressions containing $\perp$ are congruent to $\perp$ and no others are). Thus $nf(t) = nf(\perp) = \perp = nf(t')$ by definition. If t is in $CT_{\Sigma(\perp)} - T_\Sigma$, then t' is in $CT_{\Sigma(\perp)} - T_\Sigma$ and so $nf(t) = \perp = nf(t')$ by definition. If t is in T_Σ, then t' is in T_Σ and the result follows from properties of cf.

Secondly, we must show $[t] = [nf(t)]$. If t is in $CT_{\Sigma(\perp)} - T_\Sigma$, then $[t] = [\perp]$ by the above lemma and the strictness axioms. But $nf(t) = \perp$ and so $[t] = [\perp] = [nf(t)]$. If $t\in T_\Sigma$, then the result follows from properties of cf.

Finally, we must show nf is continuous. Let $\{t_i\}_{i\in I}$ be directed and let $t = \sqcup t_i$. If t is in $CT_{\Sigma(\perp)} - T_\Sigma$, then $nf(t) = \perp$ and for each $i\in I$ we have t_i in $CT_{\Sigma(\perp)} - T_\Sigma$ and so $nf(t_i) = \perp$. Hence $\sqcup nf(t_i) = \perp$ and so $nf(\sqcup t_i) = \perp = \sqcup nf(t_i)$. If t is in T_Σ, then $t = t_j$ for some $j\in I$. In fact no other t_i can be in T (since $t_i \leq t_j$) and so all other t_i are congruent to $\perp$. Thus $nf(t) = cf(t)$ and for all other $t_i \neq t$ we have $nf(t_i) = \perp$. Thus $nf(\sqcup t_i) = cf(t) = \sqcup nf(t_i)$ since $\{nf(t_i)\}_{i\in I}$ is directed with $cf(t)$ as ℓub.

Having demonstrated the existence of a normaliser, we have our result.

□

Example: Applying the above construction to our stack example, we get the following:

$\Sigma(\perp)$ is Σ together with:

$$\perp_{nat} : \to \underline{nat}$$

$$\perp_{st} : \to \underline{st}.$$

$\varepsilon'_{st}(\perp)$ is:

$top(push(n,s)) = n$ if $n \neq \perp_{nat} \wedge s \neq \perp_{st}$

$pop(push(n,s)) = s$ if $n \neq \perp_{nat} \wedge s \neq \perp_{st}$

$$\left.\begin{aligned} top(\Lambda) &= error_{nat} \\ pop(\Lambda) &= error_{st} \end{aligned}\right\} \text{No conditions exist since there are no variables in the equations.}$$

$$\left.\begin{aligned} top(\perp_{st}) &= \perp_{nat} \\ pop(\perp_{st}) &= \perp_{st} \\ push(\perp_{nat},s) &= \perp_{st} \\ push(n,\perp_{st}) &= \perp_{st} \end{aligned}\right\} \text{Strictness axioms.}$$

□

Finally, we demonstrate the usefulness of our first construction (of $T_{\Sigma,\varepsilon}^{\perp}$).

Theorem: $T_{\Sigma,\varepsilon}^{\perp}$ is initial in $\underline{CAlg}_{\Sigma(\perp),\varepsilon'(\perp)}$.

Proof: We can demonstrate this result by showing that $T_{\Sigma,\varepsilon}^{\perp}$ ($= T$) and $CT_{\Sigma(\perp)}/q_{\varepsilon'(\perp)}$ ($= C$) are isomorphic as $\Sigma(\perp)$ algebras. (Note that $T_{\Sigma,\varepsilon}^{\perp}$ can easily be made into a $\Sigma(\perp)$ algebra by having the symbol $\perp_s$ denote the value $\perp_s$ introduced in the definition of this algebra.) Let $[t]_\varepsilon$ denote the q_ε-congruence class of t for t in T_Σ. Let $[t]_{\varepsilon'(\perp)}$ denote the $q_{\varepsilon'(\perp)}$-congruence class of t in $CT_{\Sigma(\perp)}$. Define $h: T \to C$ by $h([t]_\varepsilon) = [t]_{\varepsilon'(\perp)}$

and $h(\perp_s) = [\perp_s]_{\varepsilon'(\perp)}$. We must verify that h is a homomorphism. So suppose $f \in \Sigma_{w,s}$ and r_i are in T for $w = s_1 \ldots s_n$ and $1 \leq i \leq n$. Then, if $r_i \neq \perp$ for any i, then

$$h(f_T(r_1,\ldots,r_n)) = h(f_{T_{\Sigma,\varepsilon}}([t_1]_\varepsilon,\ldots,[t_n]_\varepsilon))$$

- for some t_i in T_Σ, $1 \leq i \leq n$

$$= h([ft_1\ldots t_n]_\varepsilon)$$

- by definition of $f_{T_{\Sigma,\varepsilon}}$

$$= [ft_1\ldots t_n]_{\varepsilon'(\perp)}$$

- by definition of h

$$= f_{\mathcal{C}}([t_1]_{\varepsilon'(\perp)},\ldots,[t_n]_{\varepsilon'(\perp)})$$

- by definition of $f_{\mathcal{C}}$

$$= f_{\mathcal{C}}(h([t_1]_\varepsilon),\ldots,h([t_n]_\varepsilon))$$

- by definition of h.

If one of the $r_i = \perp$, then

$$h(f_T(n_1,\ldots,r_n)) = h(\perp) \text{ - by definition of h}$$

$$= \perp$$

$$= f_{\mathcal{C}}(h(r_1),\ldots,h(r_{i-1}),\perp,h(r_{i+1}),\ldots,h(r_n))$$

- since $h(r_i) = \perp$ and

the fact that $f_{\mathcal{C}}$ satisfies the strictness axioms

$$= f_{\mathcal{C}}(h(r_1),\ldots,h(r_n)).$$

Thus h is a homomorphism and we must now verify that it is continuous. The only directed sets in $T_{\Sigma,\varepsilon}^{\perp}$ are either of the form $\{\perp\}$ or $\{\perp,[t]_\varepsilon\}$ or

$\{[t]_\varepsilon\}$ for some t in T_Σ. The least upper bounds are $\perp$, $[t]_\varepsilon$, and $[t]_\varepsilon$ respectively. In the first case

$$\begin{aligned} h(\sqcup\perp) &= h(\perp) \\ &= \perp \\ &= \sqcup h(\perp). \end{aligned}$$

In the second case (using $\sqcup$ as a binary operator),

$$\begin{aligned} h(\perp\sqcup[t]_\varepsilon) &= h([t]_\varepsilon) \\ &= \perp\sqcup h([t]_\varepsilon) \\ &= h(\perp)\sqcup h([t]_\varepsilon). \end{aligned}$$

The final case yields

$$\begin{aligned} h(\sqcup[t]_\varepsilon) &= h([t]_\varepsilon) \\ &= \sqcup h([t]_\varepsilon) \end{aligned}$$

We must now show that h is onto. Let $[t]_{\varepsilon'(\perp)}$ be in $\mathcal{C}$. Then if t is in T_Σ, then $h([t]_\varepsilon) = [t]_{\varepsilon'(\perp)}$. If t is not in T_Σ, then $[t]_{\varepsilon'(\perp)}$ contains $\perp$ and so $h(\perp) = [t]_{\varepsilon'(\perp)}$. Thus h is onto. Finally to show that h is one to one, let r, r' be in $T_{\Sigma,\varepsilon}^{\perp}$ so that $r \neq r'$. If neither $r = \perp$ nor $r' = \perp$, then $r = [t]_\varepsilon, r' = [t']_\varepsilon$ for some t, t' in T_Σ. If $h(r) = [t]_{\varepsilon'(\perp)} = [t']_{\varepsilon'(\perp)} = h(r')$, then either t and t' are congruent (under $\varepsilon'(\perp)$) because of the equations $\varepsilon'(\perp)$ or because of the continuity property of $\varepsilon'(\perp)$.

The former case is not possible since $[t]_\varepsilon \neq [t']_\varepsilon$ and $\bot \notin [t]_{\varepsilon'(\bot)}$. On the other hand, if there are directed sets $\{t_i\}_{i\in I}$ and $\{t_i'\}_{i\in I}$ which are pairwise congruent (under $\varepsilon'(\bot)$) and so that $[\bigsqcup t_i]_{\varepsilon'(\bot)} = [\bigsqcup t_i']_{\varepsilon'(\bot)}$, then clearly some $t_i = t$ and some $t_j' = t'$. But then we can show that this requires t and t' to be congruent (under $\varepsilon'(\bot)$) simply because of the equations. This is a contradiction.

Now suppose that $r = \bot$ and $r' \neq \bot$ (and so $r' = [t']_\varepsilon$ for t' in T_Σ). Then $h(r) = [\bot]_{\varepsilon'(\bot)}$ and if $h(r) = h(r')$ we must have $[\bot]_{\varepsilon'(\bot)} = [t']_{\varepsilon'(\bot)}$ But then $\bot \in [t']_{\varepsilon'(\bot)}$ and so t' is not in T_Σ, again a contradiction. Having exhausted all possible cases, we have shown that $T_{\Sigma,\varepsilon}^{\bot}$ is isomorphic to $CT_{\Sigma(\bot)}/q_{\varepsilon'(\bot)}$.

$\square$

This result is interesting because it justifies the past practice of specifiying abstract data types as unordered (non-continuous) algebras and then taking such computational problems as non-termination into account by using a standard construction. As seen above, this standard construction consists of making the algebra $T_{\Sigma,\varepsilon}$ into a continuous algebra $T_{\Sigma,\varepsilon}^{\bot}$ by making the carriers into flat cpo's (by adding $\bot$) and extending the operations of $T_{\Sigma,\varepsilon}$ to define strict (and continuous) functions on $T_{\Sigma,\varepsilon}^{\bot}$. Moreover, the proof techniques developed for the study of abstract data type specifications and implementations can still be used. This is justified as follows. Let $A_{\Sigma(\bot)}^{\bot}$ denote the continuous $\Sigma(\bot)$-algebra obtained from the Σ-algebra A_Σ by making each A_s into a flat cpo (by adding the new element $\bot_s$) and extending operations of A_Σ to strict (and automatically continuous) operations over $A_s^{\bot}$.

<u>Theorem</u>: $A^{\perp}_{\Sigma(\perp)}$ and $B^{\perp}_{\Sigma(\perp)}$ are isomorphic as $\Sigma(\perp)$ algebras if and only if A_{Σ} and B_{Σ} are isomorphic as Σ-algebras.

$\square$

Conclusions

We have attempted to show how the conventional algebraic theory of data types can be fitted into the setting of continuous algebras. It turned out that the "obvious" construction of making the carrier of a given algebra into a flat cpo and extending the operations to be strict (continuous) ones over these flat cpo's resulted in an algebra initial in the class of continuous algebras satisfying (a slightly modified) specification. Thus the extremely important property of initiality has been retained.

The practical lesson to be learned from all this is that data types can continue to be specified in the normal or conventional setting (if this is possible or convenient) but when it comes to proving properties of programs using the type or defining representations for the type, we have to use the extended (continuous) specification.

Another possibility is to abandon the algebraic method and resort to theories based on (first order) logic as this fits in more easily with conventional methods of proof of program correctness. It also allows us (by resorting to the theory of definitions) to get around the problem of implicit definitions (e.g. by recursion or iteration) and thus deal with the problem of implementations. (See [5]) This is more in the spirit of the work done by Hoare [8].

References

1. ADJ - J.A. Goguen, J.W. Thatcher, E.G. Wagner, J.B. Wright: Initial Algebra Semantics and Continuous Algebras, JACM, Vol. 24, No. 1, pp.68-95, 1977.

2. ADJ - J.A. Goguen, J.W. Thatcher, E.G. Wagner, J.B. Wright: An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types in "Current Trends in Programming Methodolody, Vol. 4", ed. R.T. Yeh, Prentice Hall, 1978.

3. ADJ - J.W. Thatcher, E.G. Wagner, J.B. Wright: Data Type Specification: Parameterization and the Power of Specification Techniques, Proc. of 10th SIGACT Symposium on Theory of Computing, 1978.

4. G. Berry, B. Courcelle: Program Equivalence and Canonical Forms in Stable Discrete Interpretations, Proc. of 3rd Colloquium on Automata, Languages and Programming, Edinburgh, University of Edinburgh Press, 1976.

5. R.L. de Carvalho, T.S.E. Maibaum, T.H.C. Pequeno, A.A. Pereda Borquez, P.A.S. Veloso: A Model - Theoretic Approach to the Semantics of Data Types and Structures, in preparation.

6. B. Courcelle: On Recursive Equations Having a Unique Solution, Proc. of 19th Annual Symposium on Foundations of Computer Science, Ann Arbor, 1978.

7. B. Courcelle, M. Nivat: Algebraic Families of Interpretations, Proc. 17th Annual Symposium on Foundations of Computer Science, Houston, 1976.

8. C.A.R. Hoare: Proof of Correctness of Data Representations, Acta Informatica, Vol. 1, No. 1, pp. 271-281, 1972.

9. J.A. Goguen: Abstract Errors for Abstract Data Types, Proc. of IFIP Working Conference on Formal Description of Programming Concepts, North Holland, 1977.

10. J.A. Goguen: Some Design Principles and Theory for OBJ-0, A Language to Express and Execute Algebraic Specifications of Programs, Proc. of International Conference on Mathematical Studies in Information Processing, Kyoto, pp. 429-475, 1978.

11. J.V. Guttag: Abstract Data Types and the Development of Data Structures, CACM, Vol. 20, No. 6, pp. 396-404, 1977.

12. J.V. Guttag, E. Horowitz, D.R. Musser: Abstract Data Types and Software Validation, CACM, Vol. 21, No. 12, pp. 1048-1064, 1978.

13. M.R. Levy: Data Types with Sharing and Circularity, Ph.D. Thesis, Department of Computer Science, University of Waterloo, 1978. (Also Technical Report CS-78-26)

14. M.R. Levy: Verification of Programs with Data Referencing, Proc. of 3^{me} Colloque International sur la Programmation, Dunod, pp. 411-426, 1978.

15. M.R. Levy, T.S.E. Maibaum: Continuous Data Types, submitted for publication.

16. M.R. Levy, T.S.E. Maibaum: Data Types with Sharing and Circularity, in preparation.

17. B.H. Liskov, S.N. Zilles: Specification Techniques for Data Abstractions, IEEE, TSE, SE-1, No. 1, pp. 7-18, 1975.

18. T.S.E. Maibaum - A generalized approach to formal languages: JCSS 8 (1973), 409-439.

19. T.S.E. Maibaum: The Semantics of a Simple Non-deterministic Language, Proc. of 3^{e} Colloque International sur la Programmation, Dunod, pp. 158-171, 1978.

20. T.S.E. Maibaum: The Semantics of Nondeterminism, submitted for publication.

21. S.N. Zilles: Algebraic Specification of Data Types, Project MAC Progress Report 11, MIT, pp. 28-52, 1974.