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

\[ \text{insert}(n,s). \]
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

$$\text{insert}(f(x), s).$$

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 not endofseq(s) do
    if $m=n$ then isintinseq := true
    else $m := \text{next}(s)$
  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 $\Sigma = \{ \Sigma_{w,s} \}_{w,s \in S^\ast \times S}$ be a many-sorted alphabet sorted by the sorting set $S$. A symbol $f \in \Sigma_{w,s}$ is said to be of type $\langle w, s \rangle$, arity $w$, sort $s$, and rank $|w|$ (where $|w|$ is the length of the string $w \in S^\ast$). If $w = \lambda$, $f$ is said to be a constant or nullary symbol of sort $s$. A $\Sigma$-algebra $A_\Sigma$ is a family of sets $\{ A_s \}_{s \in S}$ together with an assignment of operations to symbols in $\Sigma$ so that $f \in \Sigma_{w,s}$ is assigned an operation $f^A_{s_1 \ldots s_n} : A_{s_1} \times \ldots \times A_{s_n} \to A_s$ for $w = s_1 \ldots s_n$. (We denote $A_{s_1} \times \ldots \times A_{s_n}$ by $A^w$.)

Given $\Sigma$-algebras $A_\Sigma$ and $B_\Sigma$, a $(\Sigma)$-homomorphism $h : A_\Sigma \to B_\Sigma$ 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 $1 \leq i \leq n$, 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 monomorphisms, epimorphisms, and isomorphisms, respectively. An algebra $A_\Sigma$ is said to be initial in a class $C$ of $\Sigma$-algebras if $A_\Sigma \in C$ and if for each $B_\Sigma \in C$ there is a unique homomorphism $h : A_\Sigma \to B_\Sigma$.

Theorem: The class of all $\Sigma$-algebras has an initial algebra denoted $T_\Sigma$. We can think of $T_\Sigma$ as the algebra of (finite) expressions over the alphabet $\Sigma$.

Example: 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):

\[
S = \{st, nat\};
\]
\[
\text{zero: } \quad \rightarrow \text{nat}
\]
\[
\text{succ: } \quad \text{nat} \rightarrow \text{nat}
\]
\[
\Lambda: \quad \rightarrow \text{st}
\]
\[
\text{push: } \quad \text{nat} \times \text{st} \rightarrow \text{st}
\]
\[
\text{pop: } \quad \text{st} \rightarrow \text{st}
\]
\[
\text{top: } \quad \text{st} \rightarrow \text{nat}
\]
\[
\text{error : } \quad \rightarrow \text{nat}
\]
\[
\text{error : } \quad \rightarrow \text{st}
\]

Let \(X = \{X_s\}_s\) be any family of sets. We define the set of \textit{expressions} generated by the \textit{variables} \(X\) 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 \(1 \leq i \leq n\), then \(ft_1 \ldots t_n \in T_\Sigma(X)_s\).

We can make \(T_\Sigma(X)\) into a \(\Sigma\)-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 \(T_\Sigma\).

Let \(A_\Sigma\) be an \(\Sigma\)-algebra and \(a: X \rightarrow A\) an \textit{assignment} of values to variables. Then we have the result that \(a\) extends uniquely to a homomorphism \(\tilde{a}: T_\Sigma(X) \rightarrow A_\Sigma\) so that \(\tilde{a}\) agrees with \(a\) on \(X\). (For a proof see [1]).

Let \(X_w = \{x_{s_i} | 1 \leq i \leq n, s_i = s\}_{s \in S}\). We denote by \(T_\Sigma(X_w)\) the algebra \(T_\Sigma(\{x_{s_i} | 1 \leq i \leq n, s_i = s\}_{s \in S})\). The use of \(X_w\) will be seen below. For further discussion and examples, see [2,18].
Given a $\Sigma$-algebra $A_\Sigma$, a ($\Sigma$-)congruence $q$ over $A_\Sigma$ is a family $q = \{q_s\}_{s \in S}$ of equivalence relations with the following substitution property. If $f \in \Sigma_w$, $a_i, b_i \in A_{s_i}$ so that $a_i q_s b_i$ 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 $A_\Sigma/q$ 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 $A_\Sigma/q$ the quotient of $A_\Sigma$ by $q$.

A ($\Sigma$-)equation is a pair $\langle \ell, r \rangle$ (written $\ell=r$) for $\ell, r \in T_\Sigma(\Sigma_w)$. An algebra $A_\Sigma$ is said to satisfy $\ell=r$ if for all assignments $\bar{a}: \Sigma_w \rightarrow A_\Sigma$,

$$\bar{a}(\ell) = \bar{a}(r).$$

$A_\Sigma$ is said to satisfy a set of equations $\varepsilon$ if it satisfies each equation in $\varepsilon$ separately. It is well known that a set of equations $\varepsilon$ generates a least congruence $q_\varepsilon$ on a $\Sigma$-algebra $A_\Sigma$ (and so guarantees that $A_\Sigma/q_\varepsilon$ satisfies $\varepsilon$). If we denote by $\text{Alg}_{\Sigma, \varepsilon}$ the class of algebras satisfying $\varepsilon$, then we have the following important result.

**Theorem:** $T_{\Sigma}/q_\varepsilon$ (denoted $T_{\Sigma, \varepsilon}$ in the sequel) is initial in $\text{Alg}_{\Sigma, \varepsilon}$.

**Example:** Given the alphabet for stacks of natural numbers defined above, we consider the following equations $\varepsilon_{st}$ to define the data type:
\[\text{top}(\text{push}(n,s)) = n\]
\[\text{top}(\text{push}(n,s)) = s\]
\[\text{top}(\Lambda) = \text{error}_{\text{nat}}\]
\[\text{pop}(\Lambda) = \text{error}_{\text{st}}\]

$n$ and $s$ are variables of sort $\text{nat}$ and $\text{st}$ 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 $\bot_D$). 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 ($\sqcup$ub) in $D$. If \(\{d_i\}_{i \in I}\) is a directed set in $D$, we denote by $\sqcup d_i$ the $\sqcup$ub of the set. If $D,D'$ are posets and $f : D \to D'$, then $f$ is continuous if $f(\sqcup d_i) = \sqcup f(d_i)$ (assuming that both $\sqcup d_i$ and $\sqcup f(d_i)$ exist).

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

**Theorem:** $\text{CAlg}_\Sigma^s$ has an initial algebra $CT_\Sigma^s$.

We can think of $CT_\Sigma^s$ as the algebra of (finite and infinite) partially specified expressions over $\Sigma$. The least element (denoted by $\bot_s$) of $CT_\Sigma^s,s$ is the completely unspecified expression of sort $s$. $t \preceq t'$ if $t'$ is obtained by replacing some unspecified subexpressions of $t$ by some "specified" expressions. If $\Sigma(\bot)$ denotes the alphabet obtained from $\Sigma$ by adding $\bot^s$ to
for each $s \in S$, then the algebra of completely specified (finite and infinite) expressions over $\Sigma(i)$ is isomorphic to $CT_\Sigma$. The order relation on this algebra is the least order consistent with $i \preceq t$ for each $t$ and if $t_i \preceq t'_i$ for $1 \leq i \leq n$ then $ft_1 \ldots t_n \preceq ft'_1 \ldots t'_n$. 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 \( \Sigma \) using the equations \( \varepsilon \). 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 \( (T_{\Sigma,\varepsilon})_s \) 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 \( T_{\Sigma,\varepsilon} \) into a continuous algebra by defining for each \( f \in \Sigma_{w,s} \) and each \( 1 \leq i \leq n \)

\[
f_{T_{\Sigma,\varepsilon}}(x_1, \ldots, x_{i-1}, \perp_s, x_{i+1}, \ldots, x_n) = \perp_s
\]

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

However, we have now lost all the power of the theory of abstract data types since \( T_{\Sigma,\varepsilon} \) is no longer a simple quotient algebra and we cannot in fact be sure that \( T_{\Sigma,\varepsilon} \) even satisfies \( \varepsilon \). For example if we have a binary operation \( + \) and the axiom \( +xy = y \), then
\[ +_T,^\dagger \ (i, [t]) = 1 \]

whereas the right hand side of the equation is \([t]\) for \(t \in T, ^\dagger\).

**Example:** We can make our stack example into a continuous algebra by introducing \(i_{\text{nat}}\) and \(i_{\text{st}}\) as values in \(T, ^\dagger\) and extending the operations as follows:

- \(\text{top}(i_{\text{st}}) = i_{\text{nat}}\)
- \(\text{pop}(i_{\text{st}}) = i_{\text{st}}\)
- \(\text{push}(i_{\text{nat}}, s) = \text{push}(n, i_{\text{st}}) = i_{\text{st}}\)
- \(\text{succ}(i_{\text{nat}}) = i_{\text{nat}}\)

Even if \(T, ^\dagger\) satisfies \(\varepsilon\), it is not in general initial as a \(\Sigma(i)\)-algebra satisfying \(\varepsilon\). 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 \(\varepsilon\) is a set of equations and \(q_\varepsilon\) is the least congruence on \(CT, ^\dagger\) generated by \(\varepsilon\). It is shown that in general \(CT, ^\dagger / q_\varepsilon\) is not initial in \(\text{CALg}, ^\dagger, ^{\varepsilon}\) (the class of continuous \(\Sigma\)-algebras satisfying \(\varepsilon\) together with continuous homomorphisms between them). In fact it is not even always possible to partially order the congruence class of \(CT, ^\dagger / q_\varepsilon\) in a way which is consistent with the order on
CT\_\Sigma. As a first step in getting around this problem, we define a 
continuous congruence \( q \) over a continuous algebra \( A_\Sigma \) 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 \( A_\Sigma \) so that for all \( i \in I, a_i q b_i \), then \( (\bigwedge a_i) q (\bigwedge b_i) \). In other
words, if the elements of two directed sets are pairwise congruent, then so
are the \( \wedge \)ub's. In [15] it is shown that a set of equations \( \varepsilon \) generates a
least continuous congruence on a continuous \( \Sigma \)-algebra \( A_\Sigma \). In fact, the
class of continuous congruences form a lattice.

As the next step in our development, we will introduce a generali-
sation 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 \( C_\Sigma \) such that

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

and

(ii) if \( t_1 ... t_n \in C_s \) implies \( t_i \) in \( C_{s_i} \) for

\[ w = s_1 ... s_n, 1 \leq i \leq n \] and moreover

\[ f_C(t_1, ..., t_n) = f_{t_1} ... t_n. \]

Canonical term algebras are useful because each congruence class of \( T_\Sigma, \varepsilon \)
is represented by a canonical representative and the operations of \( C_\Sigma \)
preserve canonical terms. The usefulness of canonical term algebras is
guaranteed by the fact that \( C_\Sigma \) is isomorphic to \( T_\Sigma, \varepsilon \) and so \( C_\Sigma \) is initial in
\( \text{Alg}_{\Sigma, \varepsilon} \). 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 \( CT_\Sigma \) and suppose there exists a
function
\[ \text{nf: } \mathsf{CT}_\Sigma \rightarrow \mathsf{CT}_\Sigma \]
such that

(i) \[ [t_1] = [t_2] \Rightarrow \text{nf}(t_1) = \text{nf}(t_2); \]

(ii) \[ [t] = [\text{nf}(t)]; \]

(iii) \text{nf is continuous.}

\text{nf is called a normaliser for q. We then have the following important result.}

\textbf{Theorem:} If \( q_\epsilon \) is a continuous congruence generated by the equations \( \epsilon \) on \( \mathsf{CT}_\Sigma \) and a normaliser for \( q_\epsilon \) exists, then \( \mathsf{CT}_\Sigma / q_\epsilon \) is initial in \( \mathsf{CALG}_{\epsilon, \Sigma} \).

Now, the image of \( \mathsf{CT}_\Sigma \) under \text{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 \( \mathsf{CT}_\Sigma / q_\epsilon \), then a normaliser exists ([15]). Also note that the existence of a normaliser guarantees initiality of \( \mathsf{CT}_\Sigma / q_\epsilon \) whereas we saw that this was not in general true.

Now suppose that \( \epsilon \) is the set of equations specifying some abstract data type. Denote by \( \epsilon' \) the set of equations obtained from \( \epsilon \) as follows:

If \( \lambda = \tau \) is in \( \epsilon \), for \( \lambda, \tau \in T_\Sigma(X)^s_w \), then put into \( \epsilon' \) the equation

\[ \lambda = \tau \text{ if } x_1, s_1 \neq s_1 \land \ldots \land x_n, s_n \neq s_n \]

where \( w = s_1 \ldots s_n \). Thus we place into \( \epsilon' \) equations which are conditioned by requiring that none of the variables be given values which are \( \perp \). (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 \( \varepsilon'(1) \) be the set of equations obtained by adding to \( \varepsilon' \) for each \( f \in \Sigma \) and each \( 1 \leq i \leq n \) (where \( w = s_1 \ldots s_n \)) the equation

\[
f(x_1, s_1, \ldots, x_{i-1}, s_{i-1}, x_i, s_{i+1}, x_{i+1}, \ldots, x_n, s_n) = s.\]

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

**Lemma:** Given \( q_{\varepsilon'}(1) \) as defined above, if \( t \) is in \( CT_{\Sigma}(1) \rightarrow T_{\Sigma}(1) \) but does not contain any occurrences of \( 1 \), then \( t \) is congruent to \( 1 \). (i.e. if \( t \) is an infinite expression then \( t \) is congruent to \( 1 \).)

**Proof:** It is well known that there exists a directed set \( \{ t_i \}_{i \in I} \) such that each \( t_i \) is finite and \( \bigcup t_i = t \). Moreover, each \( t_i \) contains occurrences of \( 1 \).

Thus each \( t_i \) is congruent to \( 1 \). Thus we have two directed sets \( \{ t_i \}_{i \in I} \) and \( \{ 1 \}_{i \in I} \) whose elements are pairwise congruent. Thus \( \bigcup t_i \) is congruent to \( \bigcup 1 = 1 \) demonstrating the result.

**Theorem:** \( CT_{\Sigma}(1)/q_{\varepsilon'}(1) \) is initial in \( \text{CAAlg}_{\Sigma}(1),\varepsilon'(1) \).

**Proof:** If we can demonstrate the existence of a normaliser for \( q_{\varepsilon'}(1) \), 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 \( T_{\Sigma} \) to \( C_{\Sigma} \) where \( C_{\Sigma} \) is the canonical term algebra.) Now define \( nf: CT_{\Sigma}(1) \rightarrow CT_{\Sigma}(1) \) by:
\[(i) \quad \text{nf}(1_s) = 1_s \text{ for each } s \in S;\]
\[(ii) \quad \text{nf}(t) = \begin{cases} 
\text{cf}(t) & \text{if } t \in T \Sigma \\
1 & \text{if } t \in T \Sigma \setminus T \Sigma 
\end{cases};\]
\[(iii) \quad \text{nf}(t) = 1_s \text{ (for } t \in CT \Sigma(l), s) \text{ otherwise.}\]

Now we must demonstrate the properties of normalisers. Firstly, we must show \([t] = [t']\) implies \(\text{nf}(t) = \text{nf}(t')\). If \(t = 1\), then \(t'\) is in \(CT \Sigma(l) \setminus T \Sigma\) (since all infinite expressions and all expressions containing \(1\) are congruent to \(1\) and no others are). Thus \(\text{nf}(t) = \text{nf}(1) = 1 = \text{nf}(t')\) by definition. If \(t\) is in \(CT \Sigma(l) \setminus T \Sigma\), then \(t'\) is in \(CT \Sigma(l) \setminus T \Sigma\) and so \(\text{nf}(t) = 1 = \text{nf}(t')\) by definition. If \(t\) is in \(T \Sigma\), then \(t'\) is in \(T \Sigma\) and the result follows from properties of \(\text{cf}\).

Secondly, we must show \([t] = [\text{nf}(t)]\). If \(t\) is in \(CT \Sigma(l) \setminus T \Sigma\), then \([t] = [1]\) by the above lemma and the strictness axioms. But \(\text{nf}(t) = 1\) and so \([t] = [1] = [\text{nf}(t)]\). If \(t \in T \Sigma\), then the result follows from properties of \(\text{cf}\).

Finally, we must show \(\text{nf}\) is continuous. Let \(\{t_i\}_{i \in I}\) be directed and let \(t = \bigsqcup t_i\). If \(t\) is in \(CT \Sigma(l) \setminus T \Sigma\), then \(\text{nf}(t) = 1\) and for each \(i \in I\) we have \(t_i\) in \(CT \Sigma(l) \setminus T \Sigma\) and so \(\text{nf}(t_i) = 1\). Hence \(\text{nf}(\bigsqcup t_i) = 1\) and so \(\text{nf}(\bigsqcup t_i) = 1 = \bigcup \text{nf}(t_i)\). If \(t\) is in \(T \Sigma\), 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 \(1\). Thus \(\text{nf}(t) = \text{cf}(t)\) and for all other \(t_i \# t\) we have \(\text{nf}(t_i) = 1\). Thus \(\text{nf}(\bigsqcup t_i) = \text{cf}(t) = \bigcup \text{nf}(t_i)\) since \(\{\text{nf}(t_i)\}_{i \in I}\) is directed with \(\text{cf}(t)\) 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:

$\Sigma(\bot)$ is $\Sigma$ together with:

$\bot_{\text{nat}} : \rightarrow \text{nat}$

$\bot_{\text{st}} : \rightarrow \text{st}$.

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

\[
\begin{align*}
top(\text{push}(n,s)) &= n \text{ if } n \neq \bot_{\text{nat}} \land s \neq \bot_{\text{st}} \\
pop(\text{push}(n,s)) &= s \text{ if } n \neq \bot_{\text{nat}} \land s \neq \bot_{\text{st}} \\
top(\Lambda) &= \text{error}_{\text{nat}} \\
pop(\Lambda) &= \text{error}_{\text{st}} \\
\text{top}(\bot_{\text{st}}') &= \bot_{\text{nat}} \\
pop(\bot_{\text{st}}') &= \bot_{\text{st}} \\
push(\bot_{\text{nat}}, s) &= \bot_{\text{st}} \\
push(n, \bot_{\text{st}}') &= \bot_{\text{st}}
\end{align*}
\]

No conditions exist since there are no variables in the equations.

$\text{top}(\bot_{\text{st}}') = \bot_{\text{nat}}$

$\text{pop}(\bot_{\text{st}}') = \bot_{\text{st}}$

$\text{push}(\bot_{\text{nat}}, s) = \bot_{\text{st}}$

$\text{push}(n, \bot_{\text{st}}') = \bot_{\text{st}}$

Strictness axioms.

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

Theorem: $T_{\Sigma, \varepsilon}$ is initial in $\text{CAlg}_{\Sigma(\bot), \varepsilon'(\bot)}$.

Proof: We can demonstrate this result by showing that $T_{\Sigma, \varepsilon}$ (= $T$) and $CT_{\Sigma(\bot)}/q_{\varepsilon'(\bot)}(= C)$ are isomorphic as $\Sigma(\bot)$ algebras. (Note that $T_{\Sigma, \varepsilon}$ can easily be made into a $\Sigma(\bot)$ algebra by having the symbol $\bot_{\text{st}}$ denote the value $\bot_{\text{st}}$ introduced in the definition of this algebra.) Let $[t]_{\varepsilon}$ denote the $q_{\varepsilon}$-congruence class of $t$ for $t$ in $T_{\Sigma}$. Let $[t]_{\varepsilon'(\bot)}$ denote the $q_{\varepsilon'(\bot)}$-congruence class of $t$ in $CT_{\Sigma(\bot)}$. Define $h: T \rightarrow C$ by $h([t]_{\varepsilon}) = [t]_{\varepsilon'(\bot)}$.
and $h(\mathbf{s}) = [\mathbf{s}]_\varepsilon'(\mathbf{1})$. We must verify that $h$ is a homomorphism. So suppose $f \in \Sigma_{w_1, w_2}$ and $r_i$ are in $T$ for $w = s_1 \ldots s_n$ and $1 \leq i \leq n$. Then, if $r_i \neq \mathbf{1}$ for any $i$, then

$$h(f_T(r_1, \ldots, r_n)) = h(f_T, \varepsilon)\quad \left([t_1]_\varepsilon, \ldots, [t_n]_\varepsilon\right)$$

- for some $t_i$ in $T_{\varepsilon}$, $1 \leq i \leq n$

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

- by definition of $f_T, \varepsilon$

$$= [ft_1 \ldots t_n]_\varepsilon'(\mathbf{1})$$

- by definition of $h$

$$= f_C([t_1]_\varepsilon'(\mathbf{1}), \ldots, [t_n]_\varepsilon'(\mathbf{1}))$$

- by definition of $f_C$

$$= f_C(h([t_1]_\varepsilon), \ldots, h([t_n]_\varepsilon))$$

- by definition of $h$.

If one of the $r_i = \mathbf{1}$, then

$$h(f_T(n_1, \ldots, r_n)) = h(\mathbf{1}) - \text{by definition of } h$$

$$= \mathbf{1}$$

$$= f_C(h(r_1), \ldots, h(r_{i-1}), \mathbf{1}, h(r_{i+1}), \ldots, h(r_n))$$

- since $h(r_i) = \mathbf{1}$ and

the fact that $f_C$ satisfies the strictness axioms

$$= f_C(h(r_1), \ldots, h(r_n)).$$

Thus $h$ is a homomorphism and we must now verify that it is continuous. The only directed sets in $T_{\varepsilon, \mathbf{1}}$ are either of the form $\{\mathbf{1}\}$ or $\{\mathbf{1}, [t]_\varepsilon\}$ or
\{[t]_\varepsilon\} for some t in T_\varepsilon$. The least upper bounds are \(\bot\), \([t]_\varepsilon\), and \([t]_\varepsilon\) respectively. In the first case

\[
\begin{align*}
h(\bot\bot) &= h(\bot) \\
&= \bot \\
&= \bot h(\bot).
\end{align*}
\]

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

\[
\begin{align*}
h(\bot\bot[t]_\varepsilon) &= h([t]_\varepsilon) \\
&= \bot h([t]_\varepsilon) \\
&= h(\bot)\bot h([t]_\varepsilon).
\end{align*}
\]

The final case yields

\[
\begin{align*}
h(\bot[t]_\varepsilon) &= h([t]_\varepsilon) \\
&= \bot h([t]_\varepsilon)
\end{align*}
\]

We must now show that \(h\) is onto. Let \([t]_\varepsilon'(\bot)\) be in \(C\). Then if \(t\) is in \(T_\varepsilon\), then \(h([t]_\varepsilon) = [t]_\varepsilon'(\bot)\). If \(t\) is not in \(T_\varepsilon\), then \([t]_\varepsilon'(\bot)\) contains \(\bot\) and so \(h(\bot) = [t]_\varepsilon'(\bot)\). Thus \(h\) is onto. Finally to show that \(h\) is one to one, let \(r\), \(r'\) be in \(T_\varepsilon^\bot\) so that \(r \neq r'\). If neither \(r = \bot\) nor \(r' = \bot\), then \(r = [t]_\varepsilon r' = [t']_\varepsilon\) for some \(t, t'\) in \(T_\varepsilon\). If \(h(r) = [t]_\varepsilon'(\bot) = [t']_\varepsilon'(\bot) = h(r')\), then either \(t\) and \(t'\) are congruent (under \(\varepsilon'(\bot)\)) because of the equations \(\varepsilon'(\bot)\) or because of the continuity property of \(\varepsilon'(\bot)\).
The former case is not possible since \([t] \neq [t']\) and \(1 \neq [t]_{\varepsilon '(\lambda)}\). 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 \(\varepsilon '(\lambda)\)) and so that \([\sqcup t_i]_{\varepsilon '(\lambda)} = [\sqcup t'_i]_{\varepsilon '(\lambda)}\), 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 \(\varepsilon '(\lambda)\)) simply because of the equations. This is a contradiction.

Now suppose that \(r = \lambda\) and \(r' \neq \lambda\) (and so \(r' = [t']_{\varepsilon}\) for \(t'\) in \(T^1_{\Sigma}\)). Then \(h(r) = [1]_{\varepsilon ',\lambda}\) and if \(h(r) = h(r')\) we must have \([1]_{\varepsilon ',\lambda} = [t']_{\varepsilon '(\lambda)}\).

But then \(1 \in [t']_{\varepsilon ',\lambda}\) and so \(t'\) is not in \(T^1_{\Sigma}\), again a contradiction.

Having exhausted all possible cases, we have shown that \(T^1_{\Sigma,\varepsilon}\) is isomorphic to \(CT^1_{\Sigma(\lambda)}/q_{\varepsilon '}(\lambda)\).

\[\square\]

This result is interesting because it justifies the past practice of specifying 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 \(T^1_{\Sigma,\varepsilon}\) into a continuous algebra \(T^1_{\Sigma,\varepsilon}\) by making the carriers into flat cpo's (by adding \(\lambda\)) and extending the operations of \(T^1_{\Sigma,\varepsilon}\) to define strict (and continuous) functions on \(T^1_{\Sigma,\varepsilon}\). 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 \(A^1_{\Sigma(\lambda)}\) denote the continuous \(\Sigma(\lambda)\)-algebra obtained from the \(\Sigma\)-algebra \(A_{\Sigma}\) by making each \(A_{\Sigma}\) into a flat cpo (by adding the new element \(1_{\Sigma}\)) and extending operations of \(A_{\Sigma}\) to strict (and automatically continuous) operations over \(A^1_{\Sigma}\).
Theorem: \( A_{\Sigma}^{\perp} \) and \( B_{\Sigma}^{\perp} \) are isomorphic as \( \Sigma(1) \) algebras if and only if \( A_{\Sigma} \) and \( B_{\Sigma} \) are isomorphic as \( \Sigma \)-algebras.
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


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


