SEKI-PROJEKT

SEKI
MEMO

Fachbereich Informatik
Universität Kaiserslautern
Postfach 3049
D-6750 Kaiserslautern 1, W. Germany

An Approach to Parameterized Continuous Data Types

Gerd Krützer

Memo SEKI-83-11

An Approach To Parameterized Continuous Data Types

G. Krützer

Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
D-6750 Kaiserslautern
West Germany

Abstract

In this paper we show one possible way to combine Scott's recursive domain equations with the algebraic approach to data types developped by the ADJ-group. This combination is such that we use so called (recursive) algebraic domain equations to define parameterized abstract data types in the sense of ADJ. Then this concept is extended to specifications which possess continuous algebras as their models leading to the specification of parameterized continuous data types.

Keywords:

Abstract data types, domain equations, parameterization, continuous data types.

Kaiserslautern, Mai 1983

List of Contents

An Approach To Parameterized Continuous Data Types

0. Introduction

In this paper I try to point out one possible way to combine two approaches to abstract data types (adt's), namely SCOTT's domains and the initial algebra approach of the ADJ - group.
(References: [ADJ 76], [ADJ 77], [ADJ 82], [Rau 79], [Sco 72], [Sco 76])

Scott has developped an order-theoretic approach to data types. Basically a data type is a domain (usually a complete partial order (cpo) or a lattice). There are basic types like integers, characters etc. (viewed as flat cpo's) and a method of constructing new domains from given domains by using the so-called domain-constructors '+' (sum), '×' (product) and '→' (function space). Operations on data types are required to be continuous (preserve limits of certain directed sets).
One of the most important tools in constructing domains are the so-called recursive domain equations. A recursive domain equation has the form

$$D \cong \underline{K}(D)$$

where D is the domain to be defined and $\underline{K}(D)$ means that this side of the equation consists of (previously) defined domains combined by the constructors '+', '×', '→' and of D itself.
The solution of this equation is the so-called inverse-limit D_∞. Generally D_∞ is a cpo of (infinite) retraction sequences.

Let $P_1,\ldots,P_n$ denote the previously defined domains used in $\underline{K}(D)$ and let $\underline{K}'(D)$ denote the combination of $P_1,\ldots,P_n$ as given by K(D) without D. (e.g. if you have the equation $D = \underline{K}(D)$ with $\underline{K}(D) \cong P_1 \times P_2 + D$ then $\underline{K}'(D)$ is $P_1 \times P_2$).
Now proceed in the following way to solve $D = \underline{K}(D)$:
(1) Define a sequence of domains $D_0, D_1,\ldots$ by
  (i) $D_0 := \underline{K}'(D)$
  (ii) $D_{k+1} := \underline{K}'(D_k)$.

(2) Define a sequence of pairs of continuous mappings (retractions) (i_k, j_k) where

$$i_k : D_k \to D_{k+1},\ j_k : D_{k+1} \to D_k \text{ by}$$

(i) $\forall\, d \varepsilon D_k\ j_k(i_k(d)) = d$

(ii) $\forall d \varepsilon D_{k+1}\ i_k(j_k(d)) \leq d$

(3) Define D_∞ by

$$D_\infty := \{\langle d_k \mid k \geq 0 \ \&\ d_k \varepsilon D_k \rangle \mid d_k = j(d_{k+1})\}.$$

Then D_∞ solves $D \cong \underline{K}(D)$ ('$\cong$' means isomorphy!)
What is important here is that we have a 'clear' way of constructing new types from old ones and the possibility to define data types implicitly (recursively) by means of recursive domain equations.
However it is not a trivial task to find the right retraction sequences for a given domain equation (or for a system of domain equations). And what is more we have got no way of determining a priori (via axioms for example) what the operations on our types should do; that is specifying what they are intended to do.

Here we shall get help from the algebraic approach to adt's namely the initial algebra - approach which has been developped by the ADJ group.
(References: [ADJ 76], [ADJ 77], [ADJ 82])

The algebraic approach provides a means to treat formally (mathematically) the relation between a problem and its solution. In ADJ's view an adt consists of two parts namely a syntactical part to describe the 'form' of the intended data type (the specification) and a semantical level of models (algebras) which satisfy the specified form.

The signature $\Sigma = \langle S, \Sigma \rangle$ with sort - set S and operation - set Σ determines the form of the intended data type.

A heterogenous algebra A is a model of that signature if it has carrier sets A_s for each $s \in S$ and operations $\sigma_A : A_{s1} \times \ldots \times A_{sn} \to A_s$ for each $\sigma \in \Sigma_{s1 \ldots sn, s}$.

As a unique model we take the term-algebra $T\Sigma$ and state that the adt defined by Σ is the isomorphy-class of $T\Sigma$.

$T\Sigma$ is the initial algebra in the class (category) of all Σ-algebras ($\underline{\underline{Alg}}_\Sigma$).

(initial means that there is a unique Σ-algebra homomorphism $h_A : T\Sigma \to A$ for all Σ-algebras A).

Usually operations on data types interact with each other. Therefore ADJ introduce equations (axioms) on the syntactical level to express the (intended) interaction between various operations (e.g. their behaviour).

So we come to the notion of specification. Let E be a set of equations where each equation is a pair $\langle L,R \rangle$ ($L,R \in T\Sigma(X)$) → the free term-algebra on the set X of generators) where L, R are terms of $T\Sigma(X)$ which have the same sort.

Then a specification is a triple $Spec := \langle S, \Sigma, E \rangle$.

The so-defined adt is the quotient-term-algebra $T\Sigma/_{\equiv E}$ (where $\equiv_E$ denotes the congruence-relation defined by the set of equations).

Again $T\Sigma/_{\equiv E}$ is initial in the class (category) of all Σ-algebras whose operations satisfy the equations (for all substitutions of values for variables). This class is denoted by $\underline{\underline{Alg}}_{\Sigma,E}$.

Now there is an important feature in defining new adt's, namely the concept of parameterization. The parameterization-concept will be discussed more explicitly in the following chapter. Thus I'll only give a rough overview. In the sequel let the class (category) of specifications be denoted by $\underline{\underline{Spec}}$.

Let X and D belong to $\underline{\underline{Spec}}$. Then X is the parameter-specification and D the target-specification.

Then a parameterized data type (pdt) or data-type-constructor is a functor $P: \underline{\underline{Alg}}_X \to \underline{\underline{Alg}}_D$ which takes each X-algebra A to a D-algebra B such that B is the free D-algebra generated by A.

Furthermore P is such that for a suitable forgetful-functor $U: \underline{\underline{Alg}}_D \rightarrow \underline{\underline{Alg}}_X$ (each D-algebra is taken to it's X-reduct) the following assumption holds:

$$\forall A \in |\underline{\underline{Alg}}_X| . |U \circ P|(A) \cong A$$

With this definition of a data-type-constructor we have given a unique meaning to pdt's.

Obviously we have no possibility to define adt's in the sense of ADJ implicitly via recursive domain equations.

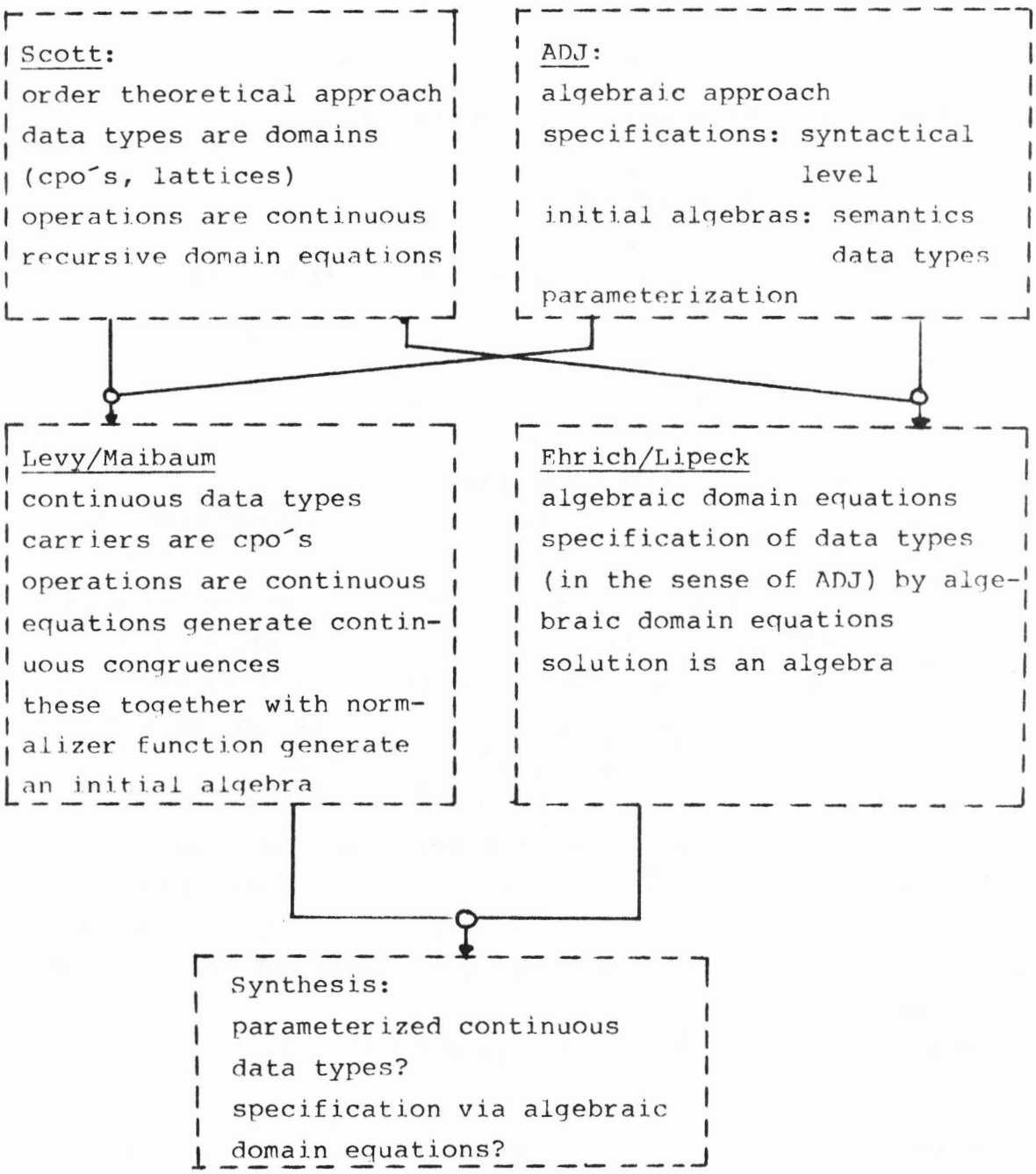
Then we may ask: Is it possible to combine the approaches of Scott and ADJ in a way that we get advantages from both?

The plan will be following:

- We show how Ehrich and Lipeck [E/L 81] solve algebraic domain equations for data types without an order-theoretical structure (simple adt's).
- Then we take the data types suggested by ADJ and enrich them with an order-theoretical structure. Levy - Maibaum [L/M 82] show how to get continuous data types in this sense. (The corresponding carrier sets of the algebra are cpo's and the operations are required to be continuous.)

In the last chapter we show one possible way to come to parameterized continuous data types defined by algebraic domain equations.

The whole plan is not yet worked out, so we give merely a rough overview pointed out by figure 1.


Scott:
order theoretical approach
data types are domains
(cpo´s, lattices)
operations are continuous
recursive domain equations
ADJ:
algebraic approach
specifications: syntactical
level
initial algebras: semantics
data types
parameterization
Levy/Maibaum
continuous data types
carriers are cpo´s
operations are continuous
equations generate contin-
uous congruences
these together with norm-
alizer function generate
an initial algebra
Ehrich/Lipeck
algebraic domain equations
specification of data types
(in the sense of ADJ) by alge-
braic domain equations
solution is an algebra
Synthesis:
parameterized continuous
data types?
specification via algebraic
domain equations?

Chapter I
Specifications, ADT´s and Parameterization

I.1 Specification and ADT´s

Following the well known results of the ADJ - group we take a specification Spec to be a triple

$$Spec = \langle S, \Sigma, E \rangle$$

where Spec is the set of sorts

Σ is the set of operations

and E is the set of equations (axioms).

The pair $\underline{\Sigma} := \langle S, \Sigma \rangle$ is called the signature of this specification.

The signature belongs to a syntactical level in the sense that it determines the form of the specified data type. Each algebra A which is supposed to be a model for this specification must have a carrier-set A_s for each sort $s \in S$ and an operation $\sigma: A_{s1} \times \ldots \times A_{sn} \to A_s$ for each $\sigma \in \Sigma_{s1,\ldots,sn,s}$.

The set E of equations consists of pairs $\langle L,R \rangle$ where L,R are terms built from operations (of the signature) and variables (from an S - sorted variable - set X). In the categorical view we want to choose a category whose objects are specifications. But we still need to say which morphisms should connect various specifications.

Thus first we say what a signature morphism is.

I.1.1 Definition

Let $\underline{\Sigma} = \langle S, \Sigma \rangle$ and $\underline{\Sigma}' = \langle S', \Sigma' \rangle$ be two signatures.

Then a signature-morphism is a pair $f = \langle f_{sort}, f_{op} \rangle$ with

(i) a sort-mapping $f_{sort}: S \to S'$

and

(ii) an operation mapping $f_{op}: \Sigma \to \Sigma'$

such that if $\sigma \in \Sigma_{s1,\ldots,sn,s}$ then

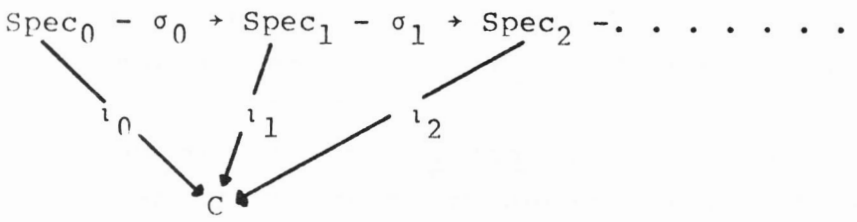
$$f_{op}(\sigma) \in \Sigma'_{fsort(s1),\ldots,fsort(sn),fsort(s)}$$

With this definition we can build the category of signatures as objects and signature morphisms as morphisms. We denote this category by Sig.
We are now able to define specification- morphisms which in a sense take care of a proper translation of equations from one specification to another. So we come along with the following

I.1.2 Definition

Let Spec = $\langle S,\Sigma,E\rangle$ and Spec´ = $\langle S',\Sigma',E'\rangle$ be specifications.
Then a specification-morphism
$f : Spec \to Spec'$
is a signature-morphism $f=\langle f_{sort}, f_{op}\rangle$ such that
$\forall e \in E.f(e) \in E'$

In this sense a specification-morphism corresponds to the theory-morphism as in [B/G 80].
So we get the category with specifications as objects and specification-morphisms as morphisms. This category is a very important one and as already pointed out, we denote it by Spec.
Now some technical results about the category of specifications, which we shall use later on. These results can be found in the paper of Ehrich and Lipeck [E/L 81] and thus detailed proofs will be omitted.
Most of these results have to do with the cocompleteness-property of Spec. Cocompleteness means that for any family $(Spec_i \mid i \in I)$ of specifications in Spec there is a unique object C in Spec and a family of Spec-morphisms ι_i: $Spec_i \to C$ such that the following diagram commutes



Informally speaking this means that whenever we have specifications $Spec_0$, $Spec_1$,... connected by specification morphisms σ_i: $Spec_i \to Spec_{i+1}$ we can determine a unique specification C in which all specifications $Spec_i$ can be 'embedded' (without loss of 'information') by specification morphisms ι_i: $Spec_i \to C$. Furthermore this 'embedding' respects the connection between the specifications $Spec_i$, $Spec_{i+1}$ due to the commutativity property of the diagram above.
This means $\forall i \in I.\ \iota_{i+1} \circ \sigma_i = \iota_i$
The importance of the cocompleteness of <u>Spec</u> lies in the fact that for $(Spec_i \mid i \in I)$ there is a unique specification C which in a sense 'containes' all the structure carried by $(Spec_i \mid i \in I)$.
Now the results:

<u>I.1.3 Theorem</u> [F/L 81]

<u>Spec</u> has coequalizers.
This means that for any pair of Spec-morphisms
f,g: Spec → Spec' there exists a unique specification C and a unique morphism h: Spec' → C with $h \circ f = h \circ g$.
C and h are such that for any specification C' and morphism h': Spec' → C' with $h' \circ f = h' \circ g$ there exists a unique morphism r: C → C' such that $h' = r \circ h$.

<u>I.1.4 Thoerem</u> [F/L 81]

<u>Spec</u> has coproducts.
This means that for any family $(Spec_i \mid i \in I)$ of specifications there exists a unique object C and a family of (coproduct-

injections) ι_k: $Spec_k \to C$. These are such that for any object D and any family $(d_k: Spec_k \to D \mid k \in K)$ there always exists a unique $\underline{\underline{Spec}}$-morphism h: $C \to D$ such that $\forall k \in I.\ h \circ \iota_k = d_k$.

I.1.5 Theorem

$\underline{\underline{Spec}}$ is cocomplete.
This is a consequence of the preceeding two theorems.

For further discussion it is useful to show the construction of coequalizers and coproducts in $\underline{\underline{Spec}}$.
The coproduct of two specifications Spec = $\langle S,\Sigma,E\rangle$ and Spec´ = $\langle S',\Sigma',E'\rangle$ is simply the triple built from the disjoint union of the components of Spec and Spec´. We denote it by Spec + Spec´ := $\langle S+S',\Sigma+\Sigma',E+E'\rangle$ (´+´ means disjoint union).
Given the two specifications Spec and Spec´ as above. Furthermore let f,g: Spec $\to$ Spec´ be specification-morphisms. Then the coequalizer of f and g is built in the following way:
First take $\underline{R}(S')$ to be the least equivalence relation on S´ generated by the set $\{\langle f_{sort}(s), g_{sort}(s)\rangle \mid s \in S\}$
Then take $\underline{R}(\Sigma')$ to be the least equivalence relation generated by the set $\{\langle f_{op}(\sigma), g_{op}(\sigma)\rangle \mid \sigma \in \Sigma\}$ which respects $\underline{R}(S')$. This means: By $\underline{R}(S')$ the set S´ is divided into equivalence classes. Then give each equivalence class a unique new name. This leads to a new sort-set S". Let $\sigma \in \Sigma_{s1,\ldots,sn,s}$. Then the sorts $s_1,\ldots,s_n,s$ are mapped to the new sorts (in S") denoted by $[f_{sort}(s_1)],\ldots, [f_{sort}(s_n)],[f_{sort}(s)]$ for the respective equivalence classes of $\underline{R}(S')$.)
We had already $\sigma \in \Sigma_{s1,\ldots,sn,s}$. The corresponding (coequalizer) operation falls into the equivalence class $[f_{op}(\sigma)]$ belonging to the new operation set $\Sigma'[f_{sort}(s_1)],\ldots,[f_{sort}(s_n)],[f_{sort}(s)]$.
To get the new operation-set Σ" we have to
rename the equivalence classes generated by $\underline{R}(\Sigma')$ with unique new operation names. The equation-set E´ is then renamed according to the previous renaming of sort- and operation-sets.

The coequalizer morphism h (in I.1.3) is then simply defined by

(i) sort-mapping h_{sort}

$\forall s' \in S'.\ h_{sort}(s') := [s']$

(ii) operation-mapping

$\forall \sigma' \in \Sigma'_{s1',\ldots,sn',s'}.\ h_{op}(\sigma') := [\sigma']$

Now one can easily verify $h \circ f = h \circ g$.

I.2 Algebras: Models for Specifications

Specifications are the syntactical description for adt's. They determine in a sense the 'form' of adt's. On the semantical level we have to consider models for specifications. Here the ADJ-group uses heterogeneous algebras. We shall shortly review their interpretation by giving the most important definitions and theorems (without proof) as far as we need them here. For more detailed information consult [ADJ 76], [ADJ 77], [ADJ 82].
Let Spec = $\langle S, \Sigma, E \rangle$ be a specification with signature $\underline{\Sigma} = \langle S, \Sigma \rangle$.

I.2.1 Definition

A Σ-Algebra A is given by:

(i) a set A_s for each sort $s \in S$
($\bigcup_{s \in S} A_s$ is called the carrier-set)

(ii) a mapping $\sigma_A: A_{s1} \times A_{s2} \times \ldots \times A_{sn} \to A_s$ for each operation $\sigma \in \Sigma_{s1,\ldots,sn,s}$

I.2.2 Definition

Let A, B be Σ - algebras (the respective carriers will be denoted by the name of the algebras!). A Σ- algebra-homomorphism is a mapping $h: A \to B$ such that

$\forall \sigma \in \Sigma_{s1,\ldots,sn,s}\ \forall a_1,\ldots,a_n \in A_{s1} \times \ldots \times A_{sn}.$

$h(\sigma_A(a_1,\ldots,a_n)) = \sigma_B(h(a_1),\ldots,h(a_n))$

The category with Σ-algebras as objects and Σ-algebra-homomorphisms as morphisms is denoted $\underline{Alg}_\Sigma$.

I.2.3 Definition

A Σ-term is defined by

(i) each constant $c \in \Sigma_{\lambda,s}$ is a Σ-term (of sort s)
(λ denotes the empty word!)

(ii) Let $t_1,\ldots,t_n$ be Σ-terms of sorts $s_1,\ldots,s_n$ and $\sigma \in \Sigma_{s1,\ldots,sn,s}$. Then $\sigma(t_1,\ldots,t_n)$ is a Σ-term of sort s.

Σ-algebras can be interrelated with certain structure-preserving mappings, called Σ-algebra-homomorphisms.

But Σ-terms may not always give what we need. Thus we introduce terms with variables.

Let $X := \bigcup_{s \in S} X_s$ be a countable set of variables.

Then we define Σ-terms which (eventually) contain variables from X. These terms will be denoted $\Sigma(X)$-terms.

I.2.4 Definition

The following terms are considered to be $\Sigma(X)$-terms.

(i) Each Σ-term is a $\Sigma(X)$-term

(ii) Each variable $x \in X_s$ is a $\Sigma(X)$-term of sort s.

(iii) Let $t_1,\ldots,t_n$ be Σ-terms of sorts $s_1,\ldots,s_n$ and $\sigma \in \Sigma_{s1,\ldots,sn,s}$. Then $\sigma(t_1,\ldots,t_n)$ is a $\Sigma(X)$-term.

Terms can be used to construct carrier-elements of so-called term-algebras. These term-algebras have an interesting property which makes them adequate candidates for unique models for specifications: They are initial in $\underline{\underline{Alg}}_\Sigma$.

I.2.5 Definition

A Σ-algebra A is initial in $\underline{\underline{Alg}}_\Sigma$, if for each Σ-algebra B there exists a unique Σ-algebra-homomorphism h_B: A → B.

Now we have to give a description of the term-algebra determined by Σ.

I.2.6 Definition

The term-algebra $T\Sigma$ determined by the signature Σ is defined by the following conditions:

(i) The carrier set for a sort $s \in S$ is the set of Σ-terms of sort s.

(ii) Let $\sigma \in \Sigma_{s1,\ldots,sn,s}$ and $t_1,\ldots,t_n$ be Σ-terms of sorts $s_1,\ldots,s_n,s$. Then $\sigma_{T\Sigma}$ is defined by

$$\sigma_{T\Sigma}(t_1,\ldots,t_n) := \sigma(t_1,\ldots,t_n).$$

I.2.7 Theorem

$T\Sigma$ is initial in $\underline{Alg}_\Sigma$.

The initial (term-) algebra $T\Sigma$ is fully determined by the signature Σ given in a respective specification. It is in a sense the most 'universal' Σ-algebra and is therefore often called 'the anarchic term-algebra'.
The mere advantage we get from $T\Sigma$ is that it is defined solely by the signature of a specification. But it turns out that $T\Sigma$ is 'too univeral' namely if we want to say something about the properties of the operations. This is very important if we want to specify how the various operations given in the specification work together. This means that we would eventually have to identify certain terms in $T\Sigma$. For example in a specification NAT (for natural numbers) with successor operation **suc** and predecessor operation **pred** we should identify such terms as **pred**(**suc**(t)) and t for each term t in $T\Sigma$. But this is an 'extra-property' which is not given by the signature above, we need additional axioms or equations to specify that we want to identify terms which should have the same meaning in a respective model. Furthermore as $T\Sigma$ is the most 'universal' Σ-algebra it considers each pair of terms to be distinct. This means that the use of $T\Sigma$ above doesn't lead to identification of terms. We have to impose additional structure on $T\Sigma$ to get what we want. We get the additional structure by dividing $T\Sigma$ into congruence-classes

generated by the equation set E in a specification Spec = $\langle S,\Sigma,E\rangle$. The congruence-relation generated by E will be denoted by $\equiv_E$ (the term 'congruence' means that this relation respects the operations). The algebra we get by dividing $T\Sigma$ into congruence classes is the 'quotient-algebra' $T\Sigma/_{\equiv E}$. The following sequence of definitions and theorems will describe the construction of $T\Sigma/_{\equiv E}$.

I.2.7 Definition

Let $\Sigma(X)$ be the signature with variables from definition I.2.4. Then the algebra $T\Sigma(X)$ is defined by the following conditions:

(i) $T\Sigma(X)_S$ is the set of all $\Sigma(X)$-terms as in definition I.2.4 (for each sort $s \in S$).

(ii) Let $t_1,\ldots,t_n$ be terms from $T\Sigma(X)_{s(1)},\ldots,T\Sigma(X)_{s(n)}$ and $\sigma \in \Sigma_{sl,\ldots,sn,s}$.
Then
$$\sigma_{T\Sigma(X)}(t_1,\ldots,t_n) := \sigma(t_1,\ldots,t_n)$$

According to this definition $T\Sigma(X)$ is a Σ-algebra namely the free Σ-algebra generated by the set X.
Terms of $T\Sigma(X)$ can be evaluated in a Σ-algebra A if we assign to each variable an element of A.

I.2.8 Definition

Let $T\Sigma(X)$ be the free Σ-algebra generated by X and A be a Σ-algebra.
Then an assignment from elements of A to X is a mapping
$$\Theta: X \to A$$
with $\Theta := (\Theta_S: X_S \to A_S \mid s \in S)$

Such an assignment determines the so-called evaluation-mapping for $\Sigma(X)$-terms in a Σ-algebra A.

I.2.9 Theorem

Let Θ be an assignment and A be a Σ-algebra.
Let the evaluation-mapping $\bar{\Theta}: T_{\Sigma(X)} \to A$ with
$\bar{\Theta} = (\bar{\Theta}_s : T\Sigma(X)_s \to A_s \mid s \in S)$ be defined by
(i) $\forall x \in X_s.\ \bar{\Theta}_s(x) = \Theta_s(x)$
(ii) Let $t_1,\ldots,t_n$ be elements of $T\Sigma(X)_{s1},\ldots,T\Sigma(X)_{sn}$ and
$\sigma \in \Sigma_{s1,\ldots,sn,s}$. Then
$\bar{\Theta}(\sigma(t_1,\ldots,t_n)) := \sigma_A(\bar{\Theta}_{s1}(t_1),\ldots,\bar{\Theta}_{sn}(t_n))$.
Then $\bar{\Theta}: T\Sigma(X) \to A$ is a Σ-homomorphism.

This theorem shows that interpretations of terms fit into the algebraic framework, because they are Σ-algebra homomorphisms. Now we must say what equations are considered to be and what it means to say: an equation is satisfied in an algebra.

I.2.10 Definition

A Σ-equation is a pair $E=\langle L,R\rangle$ with $L,R \in T\Sigma(X)_s$ for a sort $s \in S$.

I.2.11 Definition

Let $E = \langle L,R\rangle$ be a Σ-equation of sort $s \in S$ and A be a Σ-algebra. Then A satisfies E if for all assignments $\Theta: X \to A$ the evaluation $\bar{\Theta}: T\Sigma(X) \to A$ gives
$\bar{\Theta}_S(L) = \bar{\Theta}_S(R)$.

This definition means that an equation is valid in an algebra, if all interpretations of left- and right-hand sides of E in A have the same value as result.
Let A be a Σ-algebra. Then a congruence relation $\equiv$ on A is a family $\equiv := (\equiv_s \subseteq A_s \times A_s \mid s \in S)$ such that each $\equiv_s$ is an equivalence on A_s and respects the operations in the sense that if
$\sigma \in \Sigma_{s1,\ldots,sn,s}$ and $a_1,\ldots,a_n \in A_{s1} \times \ldots \times A_{sn}$ then
$\sigma_A([a_1],\ldots,[a_n]) = [\sigma_A(a_1,\ldots,a_n)]$.

([a] denotes the equivalence-class of a).
Let Spec = $\langle S,\Sigma,E\rangle$ be a specification. Then we define a congruence relation on $T\Sigma$ by using the equations E and assignments θ: $X \to T\Sigma$ (X is the variable set of E) in the following way:

I.2.12 Definition

The congruence relation on $T\Sigma$ generated by E is a family
$\equiv_E = (\equiv_{E,s} \subseteq T\Sigma_s \times T\Sigma_s \mid s \in S)$ of least congruences defined by $\equiv_{E,s}$.

(i) Let $E = \langle L,R\rangle$ be an equation in E of sort S. Then
$$\bar{\theta}(L) \equiv_{E,s} \bar{\theta}(R)$$

(ii) Let $\sigma \in \Sigma_{s1,\ldots,sn,s}$ and $t_i, t_i' \in T\Sigma_{si}$ $(i=1,\ldots,n)$ with $t_i \equiv_{E,si} t_i'$.
Then
$$\sigma(t_1,\ldots,t_n) \equiv_{E,s} \sigma(t_1',\ldots,t_n')$$

(iii) $\forall t \in T\Sigma_s.\ t \equiv_{E,s} t$

(iv) $\forall t,t' \in T\Sigma_s.\ t \equiv_{E,s} t' \Rightarrow t' \equiv_{E,s} t$

(v) $\forall t,t',t'' \in T\Sigma_s.\ (t \equiv_{E,s} t' \,\&\, t' \equiv_{E,s} t'') \Rightarrow t \equiv_{E,s} t''$

The congruence-classes $[t]$ for $t \in T\Sigma_s$ are built by
$$[t]_{\equiv E,s} := \{t' \in T\Sigma_s \mid t \equiv_{E,s} t'\}$$
Then we can build the quotient-term algebra $T\Sigma/_{\equiv E}$ in the following way:

I.2.13 Definition

Let Spec = $\langle S,\Sigma,E\rangle$ be a specification. Then the quotient-term-algebra $T\Sigma/_{\equiv E}$ is defined by

(i) For each $s \in S$ the carrier-set $T\Sigma/_{\equiv E,s}$ is the set
$$T\Sigma/_{\equiv E,s} := \{[t]_{\equiv E} \mid t \in T\Sigma_s\}$$

(ii) Let $\sigma \in \Sigma_{s1,\ldots,sn,s}$ and $t_i \in T\Sigma_{si}$ $(i = 1,\ldots,n)$.
Then
$$\sigma_{T\Sigma/\equiv E}([t_1],\ldots,[t_n]) := [\sigma(t_1,\ldots,t_n)]$$

As we have already seen $T\Sigma$ plays an important role in the

category of Σ-algebras $\underline{\underline{Alg}}_\Sigma$: it is the initial object in this category and it is uniquely determined (up to isomorphism).
$T\Sigma/_{\equiv E}$ has an analoguous role in the category $\underline{\underline{Alg}}_{\Sigma,E}$. The objects in $\underline{\underline{Alg}}_{\Sigma,E}$ are the Σ-algebras which satisfy the equations E. The morphisms are Σ-algebra-homomorphisms h: A → B (A, B are algebras in $\underline{\underline{Alg}}_{\Sigma,E}$) which satisfy

$$h(\sigma_A([a_1],\ldots,[a_n])) = \sigma_B([h(a_1)],\ldots,[h(a_n)])$$

I.2.14 Theorem

Let Spec = $\langle S,\Sigma,E\rangle$ be a specification and $T\Sigma/_{\equiv E}$ the quotient-termalgebra defined by Spec.
Then $T\Sigma/_{\equiv E}$ is initial in $\underline{\underline{Alg}}_{\Sigma,E}$ (and is uniquley determined up to isomorphism!).
So we are prepared to say what an adt is considered to be in the ADJ-philosophy:

I.2.15 Definition

Let Spec = $\langle S,\Sigma,E\rangle$ be a specification.
Then by the abstract data type specified by Spec we mean the isomorphism-class of the quotient-termalgebra $T\Sigma/_{\equiv E}$.

I.3 Parameterized Specifications and Parameterized Data Types

We now show how 'new' data types can be constructed from 'old' ones in the ADJ-approach by using the 'parameterization-technique. On the syntactical level parameterization means that we start with a formal parameter specification X and 'embed' it into a resulting specification D via an injective Spec-morphim p: X → D.
The formal parameter has very little structure such that there is a (eventually) large class of specifications in Spec which will fit this structure and can therefore serve as actual

syntactical parameters. Parameterization means that one specification is built from one or more parameter-specifications by eventually extending the structure provided by the parameters with new sorts , new operations and new equations. This is so far the syntactical view.

On the semantical level parameterization means transformation of algebras of one category into algebras of another (resultant) category together with transformation of algebra-homomorphisms. This should be done in such a way that the structure of the parameter-algebra will not be lost. This means that by a certain 'reduction' of the resultant algebra we get an algebra that has the same structure as the parameter-algebra. The transformation of 'old' structures (category of parameter algebras) into 'new' (extended) structures (category of parameterized algebras) will be carried out by functors (according to the category-theoretical viewpoint used in the ADJ-approach). Analogously the 'reduction' will be carried out by so-called forgetful functors. These functors 'forget' in a sense all of the additional structure of the resultant algebras and 'concentrate' only on the structure of the 'old' parameter algebra.

According to the philosophy that an adt should be uniquely determined the resultant (parameterized) algebra is the 'free'-extension of the parameter algebra. 'Free extension' means that the elements of the 'old' carriers A_s (for the parameter algebra) become by transformation (with the respective functor) elements of the new carrier $B_{p(s)}$ (if B is the resultant algebra and $p: X \to D$ the parameterized specification belonging to the transformation).

The following definitions and theorems formalize the above ideas. The results are taken from [E/L 81].

Remark: In the sequel if $p: X \to D$ is a Spec-morphism, then

$$p(s) := p_{sort}(s) \ (s \in S) \text{ and } p(\sigma) := p_{op}(\sigma) \ (\sigma \in \Sigma_{s1,..,sn\ s}).$$

I.3.1 Definition

A parameterized specification is an injective $\underline{\underline{Spec}}$-morphism

$p: X \to D$

X is called the formal parameter of p.

In the sequel we use a special kind of parameterized specifications, namely (strongly) persistent specifications. This property is mainly connected with the transformation (of data types) specified by the parameterized specifications. The transformation is expressed by using certain functors between categories of algebras. So we introduce here the functors with which we are concerned in parameterization namely forgetful and (strongly) persistent functors.

Remark: If $e: X \to D$ is a $\underline{\underline{Spec}}$-morphism then the respective persistent functor belonging to e will be denoted by the (upper case letter) E and the respective forgetful functor will be denoted by $\bar{E}$.

I.3.2 Definition

Let $e: X \to D$ be a $\underline{\underline{Spec}}$-morphism and B be a D-algebra. Furthermore let sig(X),sig(D) and sorts(X),sorts(D) denote the signatures and sorts of the specifications X and D.

Then the forgetful functor $\bar{E}: \underline{\underline{Alg}}_D \to \underline{\underline{Alg}}_X$ sends each D-algebra B to the X-algebra A defined by

(i) $\forall s \in sorts(X).\ A_s := B_{e(s)}$

(ii) Each operation $\sigma \in sig(X)_{s1,\ldots,sn,s}$ is defined by the image-operation under e.

$\sigma_A: A_{s1} \times \ldots \times A_{sn} \to A_s$ is defined to be the operation

$e_{op}(\sigma)_B: B_{e(s1)} \times \ldots \times B_{e(sn)} \to B_{e(s)}$

The algebra A is called the sig(X)-reduct of B.

(iii) Let B, B' be D-algebras and $h: B \to B'$ be a D-algebra-homomorphism.

Let A, A' be the respective sig(X)-reducts of B and B'

defined by $\bar{F}$.
Then $\bar{F}$ transforms h into an X-algebra-homomorphism
$q: A \to A'$ by

$$\forall s \in sorts(X).\ q_s := h_{e(s)}$$

In the following discussion we denote the object part of a category $\underline{C}$ by $|\underline{C}|$ and the morphism part by $/\underline{C}/$. If $A,B \in |\underline{C}|$ then $\underline{C}(A,B)$ denotes the set of all morphisms from A to B.
Furthermore if $\underline{C}$, $\underline{D}$ are categories and $F: \underline{C} \to \underline{D}$ is a functor then we shall denote its object part by $|F|: |\underline{C}| \to |\underline{D}|$ and its morphism part by $/F/: /\underline{C}/ \to /\underline{D}/$.
Now we turn to the definition of persistent functors between categories of algebras. Persistent functors are used to construct parameterized data types from parameter data-types. They perform this transformation in such a way that they 'remember' the structure of the parameter-algebra. The structure of the 'remembered' algebra can then be rediscovered by application of a forgetful functor.

I.3.3 Definition

Let $p,e: X \to D$ be $\underline{Spec}$-morphisms.
Then a persistent functor P (determined by p) is a functor

$$P: \underline{Alg}_X \to \underline{Alg}_D$$

such that

$$\forall A \in |\underline{Alg}_X|.\ |\bar{F} \circ P|(A) \cong A$$

('$\cong$' means isomorphy and '$\circ$' denotes the composition of functors)
P is strongly persistent if

$$\forall A \in |\underline{Alg}_X|.\ |\bar{F} \circ P|(A) = A$$

Now we can see what it means to say that a persistent functor 'remembers' the structure of its argument (or parameter)-algebra namely

$$|\bar{F} \circ P|(A) \cong A$$

or

$$|\bar{F} \circ P|(A) = A.$$

We see that we can always rediscover the structure of the argument and thus no relevant ´information´ is lost by application of a persistent functor.
What is left for the moment is to give the respective working definitions for parameterized data types (pdt´s) and the semantics of a parameterized specification.
For short: a pdt or data-type constructor consists namely of a persistent functor and forgetful functor. The standard-semantics of a parameterized specification is given by a persistent functor P_{free}: $\underline{\underline{Alg}}_X \to \underline{\underline{Alg}}_D$ which transforms each X-algebra into its free extension and by the forgetful functor $\bar{P}$: $\underline{\underline{Alg}}_D \to \underline{\underline{Alg}}_X$.
Constructing the free extension of an X-algebra A by application of a persistent functor P_{free}: $\underline{\underline{Alg}}_X \to \underline{\underline{Alg}}_D$ with $|P_{free}|(A) := A'$ means:

The old carriers A_s ($s \in sorts(X)$) are bijectively transformed to the ´new´ carriers $A'_{p(s)}$ (neither new elements are added to A_s in $A'_{p(s)}$ nor ´old´ elements are mapped onto the same image).

and

new carriers, new operations and new equations are eventually added in A´.

These are the key ideas contained in the following sequence of definitions and theorems.

I.3.4 Definition

As a working definition for pdt´s we choose
A parameterized data type consists of a parameterized specification

$$p: X \to D$$

and the (strongly) persistent functor P_{free}: $\underline{\underline{Alg}}_X \to \underline{\underline{Alg}}_D$ that takes each X-algebra A to its free extension over A with respect to p such that

$$|\bar{P} \circ P_{free}|(A) \cong A \quad (|\bar{P} \circ P_{free}|(A) = A)$$

I.3.5 Definition

Let p: X → D be a parameterized specification.
Then by the standard-semantics of p we mean the pair (p, P_{free}).

I.3.6 Definition

Let p X → D be a parameterized specification.
The p is called (strongly) persistent if its underlying standard-semantics has this property.

We shall proceed by clarifying parameter-passing in Spec and by building instances of pdt's. The rest of the chapter contains some category-theoretical results on the interrelationship of Spec and the category of Spec-models, the algebras, which are contained in Cat (category of categories with functors as morphisms).
As a parameterized specification is intended to be used as a method of systematically constructing new specifications from old ones we have to indicate what parameter passing means.
In general we bind an actual (syntactical) parameter to the formal parameter in p: X → D by a morphism f: X → A and then complete the resulting diagram

```
X  --p--> D
f↓
A
```

by giving it a unique meaning as the pushout of the diagram

```
X  --p--> D
f↓        ↓f'
A  --p'-> B
```

(where f'op = p'of)

I.3.7 Definition

Let p: X → D be a parameterized specification.
Then an actual syntactical parameter is a pair (f,A) such that

f: X → A is a Spec-morphism.

I.3.8 Definition

Let (f,A) be an actual syntactical parameter for p: X → D. Then the result of parameter passing (A for X) is the pushout of the diagram

$$\begin{array}{ccc} X & \xrightarrow{p} & D \\ f\downarrow & & \downarrow f' \\ A & \xrightarrow{p'} & D' \end{array}$$

The preceding definition means that the result of parameter-passing is determined by the equivalence of

(i) embedding X into D and then replacing the formal part in D by the actual parameter A via using f

and

(ii) substituting A for X via f and then embedding A into D (by changing D to D').

Again by the requirement that the meaning of parameter- passing should be the pushout of the above diagram we know that the result is uniquely determined. The parameter-passing-mechanism on the syntactical level corresponds in a sense to the following mechanism on the semantical level.

Let (f,A) be an actual parameter for the parameterized specification p: X → D.

Then by using the standard-semantics (p,P_{free}) we indicated that each X-algebra will be transformed into a D-algebra (free-extension) by using the (strongly) persistent functor P_{free}. Now we take an A-algebra and then transform it into a D'-algebra by using the standard-semantics of p': A → D', namely (p',P'_{free}) which sends the actual A-algebra to its free extension in $Alg_{D'}$.

I.3.9 Definition

Let p: X → D be a parameterized specification with actual

parameter (f,$\underline{A}$). Let the result of parameter passing be the pushout of the following diagramm:

$$\begin{array}{ccc} X & \xrightarrow{\underline{P}} & D \\ f\downarrow & & \downarrow f' \\ \underline{A} & \xrightarrow{\underline{P'}} & D' \end{array}$$

Then an actual parameter for (p,P_{free}) is a triple $(f,\underline{A},A)$ where $(f,\underline{A})$ is the actual parameter for p and A is an $\underline{A}$-algebra. The result of parameter-passing is indicated to be the commutativity of the following diagram.

$$\begin{array}{ccc} \underline{Alg}_X & \xrightarrow{\underline{Pfree}} & \underline{Alg}_D \\ \bar{F}\uparrow & & \uparrow\bar{F}' \\ \underline{Alg}_{\underline{A}} & \xrightarrow{\underline{P'free}} & \underline{Alg}_{D'} \end{array}$$

Remark: If P_{free} is (strongly) persistent then P'-free is. (technical result in [E/L 81])

We adopt the following convention for the structure of <u>formal-parameter-specifications</u>.

I.3.10 Convention

Let p: X → D be a (simple) parameterized specification.
Then the formal parameter X is one of the following specifications (alternatives are enclosed in brackets { }):

<u>sorts</u>: X

<u>opns</u>: $\{\bar{x}: \to X\}$ (X contains eventually a constant)

$\equiv_X: X\times X \to BOOL$

<u>eqns</u>: $x \equiv_X x = true$

$x \equiv_X x' = x' \equiv_X x$

$((x \equiv_X x' \;\&\; x' \equiv_X x'') \Rightarrow x \equiv_X x'') = true$

Furthermore we shall give each parameterized specification a <u>unique name</u>. This will help us to identify various specifications by their names and to use them as parameters in other parameterized specifications. Thus if p: X → D is a parameterized specification with simple parameter X we identify the resulting

specification by the (unique) name P. The equation

$P = \underline{K}(X_1,...X_n)$ where $\underline{K}$ is a constructor (that is a combination of the parameters $X_1,...,X_n$ via operations such as '+', '×')

means, that we name the resulting specification by P. The formal parameter will be denoted by the keyword formal.
We make the following convention:

I.3.11 Convention

Let $p\ \ X \to D$ be a parameterized specification.
Then the equation $P = K(X)$ denotes the following specification:

sorts: P ...
  formal X
opns: $\{\bar{p}: \to P\}$
  $\equiv_p$: $P \times P \to BOOL$
  . other .
  . opera- .
  . tions .
  formal $\{\bar{x}: \to x\}$
  $\equiv_x$: $X \times X \to BOOL$
eqns: $p \equiv_p p = true$
  $p \equiv_p p' = p' \equiv_p p$
  $((p \equiv_p p' \ \&\ p' \equiv_p p'') \Rightarrow p \equiv_p p'') = true$
  . other .
  . equations .
  formal $x \equiv_x x$
  $x \equiv_x x' = x' \equiv_x x$
  $((x \equiv_x x' \ \&\ x' \equiv_x x'') \Rightarrow x \equiv_x x'') = true$

In the case of more than one formal parameter e.g.
$P = \underline{K}(X_1,...,X_n)$ this concept can be easily extended by using a tuple of parameterized specifications $(p_i: X_i \to D \mid i=1,...,n)$ and then defining one Spec-morphism by this triple.
What we still need is to extend the parameter-passing concept in the case when the parameters are themselves parameterized specifications for parameterized data types. In this case the following definitions will be used:

I.3.11 Definition

Let $p: X \to D$ be a parameterized specification with actual parameter $(f,\underline{A})$. Let $\hat{p}: Y \to \underline{A}$ be a parameterized specification. Let the result of passing $(f,\underline{A})$ for X be the pushout of the following diagram

$$\begin{array}{ccc} X & \xrightarrow{p} & D \\ f\downarrow & & \downarrow f' \\ Y \xrightarrow{\hat{p}} \underline{A} & \xrightarrow{p'} & D' \end{array}$$

Then the result of passing $\hat{p}: Y \to A$ for X via f is given by $\underline{p' \circ \hat{p}}$.

I.3.12 Definition

Let (p,P_{free}) be (strongly) persistent standard-semantics for $p: X \to D$ and $(f,\underline{A},A)$ be an actual parameter. Furthermore let $(\hat{p},\hat{P}_{free})$ be a (strongly) persistent standard-semantics for $\hat{p}: Y \to \underline{A}$.
Then the result of parameter passing is the (strongly) persistent parameterized data type given by

$(p' \circ \hat{p},\ P'_{free} \circ \hat{P}_{free})$.

Now we give some examples for parameterized specifications using the conventions I.3.10 and I.3.11.

Example 1

$$P = X_1 \times X_2 \times \ldots \times X_n \quad (n \in \mathbf{N})$$

sorts: P, BOOL
  formal $X_1,\ldots,X_n$
opns: $\equiv_p: P \times P \to BOOL$
  $\langle \ldots \rangle: X_1 \times \ldots \times X_n \to P$
  $[i]: P \to X_i \quad (i = 1,\ldots,n)$
  formal $\equiv_{X_i}: X_i \times X_i \to BOOL \quad (i = 1,\ldots,n)$
eqns: $p \equiv_p p = true$ — $\equiv_p$ is an
  $p \equiv_p p' = p' \equiv_p p$ — equivalence

$((p \equiv_p p' \;\&\; p' \equiv_p p'') \Rightarrow p \equiv_p p'') = true$ relation on P

$[i]\langle x_1,\ldots,x_n\rangle = x_i \quad (i = 1,\ldots,n)$

$\langle x_1,\ldots,x_n\rangle \equiv_p \langle x_1',\ldots,x_n'\rangle = (x_1 \equiv_{x1} x_1' \;\&\ldots\&\; x_n \equiv_{xn} x_n')$

formal $x_i \equiv_{xi} x_i$

$x_i \equiv_{xi} x_i' \; x_i = x_i' \equiv x_i \qquad i = 1,\ldots,n$

$((x_i \equiv_{x'i} \;\&\; x_i' \equiv_{xi} x_i'') \Rightarrow x_i \equiv_{xi} x_i'') = true$

Example 2

$P = X_1 \times X_2 \times \ldots \times X_n + 1$ (n-fold product with constant)

sorts: P, BOOL

formal $X_1,\ldots,X_n$

opns: $\bar{p}: \rightarrow P$

$\equiv_p: P \times P \rightarrow BOOL$

$\langle\ldots\rangle: X_1 \times \ldots \times X_n \rightarrow P$

$[i]: P \rightarrow X_i \quad (i = 1,\ldots,n)$

formal $\bar{x}_i: \rightarrow X_i \qquad (i = 1,\ldots,n)$

$\equiv_{Xi}: X_i \times X_i \rightarrow BOOL$

eqns: $p \equiv_p p = true$

$p \equiv_p p' = p' \equiv_p p$

$((p \equiv_p p' \;\&\; p' \equiv_p p'') \Rightarrow p \equiv_p p'') = true$

$\langle x_1,\ldots,x_n\rangle \equiv_p \bar{p} = false$

$\bar{p} \equiv_p \langle x_1,\ldots,x_n\rangle = false$

$[i]\,\bar{p} = \bar{x}_i$

formal $x_i \equiv_{xi} X_i = true$

$x_i \equiv_{xi} x_i' = x_i' \equiv_{xi} x_i \quad (i = 1,\ldots,n)$

$((x_i \equiv_{xi} x_i' \;\&\; x_i' \equiv_{xi} x_i'') \Rightarrow x_i \equiv_{xi} x_i'') = true$

Remark

In the following sections we shall omit the 'equivalence'-part of the specification that is the operations '$\equiv$' and the equations stating the equivalence property of '$\equiv$'.

Now some examples for parameter passing.

We use the following specification NAT for ´natural numbers´:

sorts: NAT, BOOL
opns: $0: \rightarrow NAT$
$suc: NAT \rightarrow NAT$
$\equiv_{NAT}: NAT \times NAT \rightarrow BOOL$
eqns: $0 \equiv_{NAT} 0 = true$
$0 \equiv_{NAT} suc(n) = false$
$suc(n) \equiv_{NAT} 0 = false$
$suc(n) \equiv_{NAT} SUC(N´) = n \equiv_{NAT} n´$

(This is a quite redundant specification of NAT but it is useful in showing parameter-passing!)

Let´s turn to our next example:

We want to build $Q = NAT \times NAT$.

So we take the specification from Example 1 with

$P = X_1 \times X_2$ and the two Spec-morphisms p_1, p_2 characterized by:

$p_1: X_1 \rightarrow NAT, \equiv_{X1} \rightarrow \equiv_{NAT}, \bar{x}_1 \rightarrow 0$

$p_2: X_2 \rightarrow NAT, \equiv_{x2} \rightarrow \equiv_{NAT}, \bar{X}_2 \rightarrow 0$

We take the tuple $p = \langle p_1, p_2 \rangle$ to lead to the parameterized specification

$p: X_1 \times X_2 \rightarrow NAT \times NAT$

The resulting specification is given by:

sorts: Q, NAT, BOOL
opns: $\bar{q}: \rightarrow Q$
$\equiv_Q: Q \times Q \rightarrow BOOL$
$\langle\rangle: NAT \times NAT \rightarrow Q$
$[1]: Q \rightarrow NAT$
$[2]: Q \rightarrow NAT$
. NAT-opns .
eqns: $\bar{q} \equiv_Q \bar{q}$
$[1](\bar{q}) = 0$
$[2](\bar{q}) = 0$
$\langle n_1, n_1´\rangle \equiv_Q \langle n_2, n_2´\rangle = (n_1 \equiv_{NAT} n_2 \;\&\; n_1´ \equiv_{NAT} n_2´)$

. .

. equations for NAT .

. .

We can now use this specification for building

$Q \times Q \triangleq (NAT \times NAT) \times (NAT \times NAT)$

$Q + 1 = (NAT \times NAT) + 1$

etc.

II. Algebraic Domain Equations

We have indicated in the introduction what it means to specify a Scott-like data type (domain) via a recursive domain equation of the form

$$D \cong \underline{K}(D).$$

Here the solution was the inverse limit of a retraction-sequence. Scott's method of constructing new data types from old ones by using the domain constructors '+', '×', '→' and recursive domain equations is in a sense quite comfortable.

But on the other hand we should be able to write down formally how we intend to solve a given problem by using the names of the used data (sorts), and the operations which work on these operations given by the axioms.

The next step is then the construction of mathematical models (universal algebras) for specifications and formalizing ways for constructing new data types from old ones (e.g. parameterization). These requirements are satisfied by the ADJ-approach as we have seen in the previous sections. Thus in following our aim to construct parameterized continuous data types by certain 'recursive algebraic domain equations' we treat domain equations in an algebraic framework. In this section we shall use 'simple' data types without any order-theoretical structure.

First we start with an introduction to recursive domain equations from a category theoretical viewpoint.

We analyse Scott's recursive domain equations. Suppose $\underline{1}$ is the domain with one non-bottom element $\underline{1} = \{\bot, e\}$ with $\bot \leq e$.

Now we look at the domain equation

$$D \cong D + \underline{1}$$

To solve this equation, we need the following sequence of domains

$$D_0 := \underline{1}$$

$$D_{i+1} := D_i + \underline{1}$$

and use retraction sequences $((i_k, j_k) \mid k \in \mathbf{N})$ to construct the solution $D_\infty \overset{(\psi, \phi)}{\cong} D_\infty + \underline{1}$.

But what means the term 'solution' category-theoretically?

As we have suggested in the introduction the above equation $D \cong D + \underline{1}$ can be transformed into the equation $D \cong \underline{K}(D)$ where $\underline{K}$ was called a 'constructor'. In the current example the case is obvious, we have:

$\underline{K}(D) := D + \underline{1}$.

The first observation is that $\underline{K}$ 'maps' in a sense domains to other domains, namely each domain D is mapped to $D + \underline{1}$. But what is more $\underline{K}$ maps each continuous function $f: D \to D'$ to a continuous function $\underline{K}(f): \underline{K}(D) \to \underline{K}(D')$ such that the following diagram commutes

$$\begin{array}{ccc} D & \xrightarrow{\ f\ } & D' \\ {\scriptstyle i_0}\downarrow & & \downarrow{\scriptstyle i_0'} \\ \underline{K}(D) & \xrightarrow{\ \underline{K}(f)\ } & \underline{K}(D') \end{array}$$

where (i_0, j_0) and (i_0', j_0') are retractions with

$$D := D_0 \xrightarrow{\ (i_0, j_0)\ } D_1 =: \underline{K}(D)$$
$$D' := D_0' \xrightarrow{\ (i_0', j_0')\ } D_1' =: \underline{K}(D')$$

What have we gained by these observations?

Let $\underline{\underline{Cpo}}$ be the category with cpo's as objects and strict continuous functions between cpo's as morphisms. (A continuous function $f: D \to D'$ is strict if $f(\bot_D) = \bot_{D'}$).Then the recursive domain equation

$D \cong \underline{K}(D)$ leads to a functor

$\underline{K}: \underline{\underline{Cpo}} \to \underline{\underline{Cpo}}$.

Obviously $\underline{K}$ is an endofunctor (source- and target-category are the same!).

Then by 'solution' of $D \cong \underline{K}(D)$ we mean the initial fixedpoint of $\underline{K}$. This unique fixedpoint is the inverse limit D_∞ (described in the introduction).

But what is more: data types are domains in the theory of Scott. Then the equation $D \cong \underline{K}(D)$ describes in a sense a parameterized data type (with D as parameter and $\underline{K}$ as data type constructor)!

Hence a parameterized data type in the sense of Scott is a unique fixedpoint of an endofunctor $\underline{K}: \underline{\underline{Cpo}} \to \underline{\underline{Cpo}}$.

Oneargument-parameterization and fixedpoint-construction for

recursive domain equations are in this sense equivalent.

But what about parameterization in the ADJ-approach? Scott's method suggests looking for endofunctors on suitable categories. Nevertheless the (strongly) persistent functors which extend the parameter algebras to build parameterized data types can't be supposed to be endofunctors in all relevant cases. The main reason for this argument is that in the relevant cases of constructing parameterized data types new sorts, operations and equations will be added to the parameter. On the syntactical level we have a parameterized specification p: X → D between different source- and target-specifications. The corresponding equivalent on the semantical level is a data type constructor P: $\underline{\underline{Alg}}_X \to \underline{\underline{Alg}}_D$ with different source- and target-categories.
Hence the data type constructor P above will not suffice to allow specifications of data types as fixedpoints of certain recursive 'algebraic' domain equations.
On the other hand we are able to specify the following endofunctors.
As already indicated

$\bar{F} \circ P \quad \underline{\underline{Alg}}_X \to \underline{\underline{Alg}}_X$

is an endofunctor where $\bar{F}: \underline{\underline{Alg}}_D \to \underline{\underline{Alg}}_X$ is the forgetful functor corresponding to a $\underline{\underline{Spec}}$-morphism e: X → D.
If P is a (strongly) persistent data type constructor the the equation

$\forall A \in |\underline{\underline{Alg}}_X| .\ A \cong |\bar{F} \circ P|(A)$

holds. The algebra A is obviously a fixedpoint of the endofunctor $\bar{F} \circ P$.
But what will be gained by considering fixedpoints for $\bar{F} \circ P$?
My answer is: nothing interesting will be gained, because we loose all relevant information about the parameterized data type |P|(A) if we applicate $\bar{F}$ afterwards. $\bar{F}$ is a forgetful functor and will therefore reduce most of the relevant structure of |P|(A) to the $\underline{\underline{Alg}}_X$-components Therefore fixedpoints of $\bar{F} \circ P$ will not concern us in our further discussion. The other endofunctor we may specify is

$Po\bar{F}$: $\underline{\underline{Alg}}_D \to \underline{\underline{Alg}}_D$.

Here we can be sure that no relevant structure of a parameterized data type will be lost by applicating $Po\bar{F}$; we shall always have the full $\underline{\underline{Alg}}_D$-structure when looking at fixedpoints of $Po\bar{F}$.

But nevertheless $Po\bar{F}$ will not give exactly what we want. A slight modification of fixedpoints of $Po\bar{F}$ will be used in the projected 'algebraic domain equations' (ade's). Please consider the following example taken from [E/L 81] as a motivation: The solution of Scott's recursive domain equation $X \cong X+\underline{1}$ is the domain of natural numbers.

Now let's stick to an algebraic interpretation of this equation. Let X be a variable ranging over pointed sets (M,i). Each (M,i) is a one-sorted algebra with carrier (M,i) for sort s_0 without operations. Let P be a functor taking each pointed set (M,i) to the two-sorted algebra with carrier (M,i) for s_0 and carrier (M+{0},0) for sort s_1 and two operations σ:(M,i) $\to$ (M+{0},0) and π:(M+{0},0) $\to$ (M,i). Here (M+{0},0) is the pointed set (M,i) together with a new element denoted by '0'.

The operations are defined in the following way

σ: (M,i) $\to$ (M+{0},0)

$$y \to \begin{cases} y & \text{if } y \neq i \\ 0 & \text{if } y = i \end{cases}$$

π: (M+{0},0) $\to$ (M,i)

$$y \to \begin{cases} y & \text{if } y \neq 0 \\ i & \text{if } y = 0 \end{cases}$$

σ denotes in a sense a successor operation with the new point 0 as the successor of i.

π is a predecessor operation: if the new point 0 is generated by i then i is the predecessor of 0.

Let $\bar{F}$ be a functor which takes the two sorted algebra ((M,i),(M+{0},0),σ,π) to the one sorted-algebra (M+{0},0) where (M+{0},0) is the carrier of sort s_0. (Consider the underlying signature-morphism e for $\bar{F}$ to be such that e maps s_0 to s_1!)

Now in Scott's inverse limit construction we got the solution of

$X \cong X+1$ by iterated application of the functor $\underline{K}$ with $|\underline{K}|(D) := D+\underline{1}$. The solution consists of a sequence of domains

$D_0 := \underline{1}$

$D_1 := |\underline{K}|(D_0) := D_0 + \underline{1}$

.

.

.

$D_{k+1} := |\underline{K}|(|\underline{K}|^k(D_0)) := D_k + \underline{1}$

.

.

.

and a sequence of retractions (i_k, j_k) such that

$i_0: D_0 \to |\underline{K}|(D_0)$

$j_0: |\underline{K}|(D_0) \to D_0$

$i_1 := /\underline{K}/(i_0): |\underline{K}|(D_0) \to |\underline{K}|^2(D_0)$

$j_1 := /\underline{K}/(j_0): |\underline{K}|^2(D_0) \to \underline{K}(D_0)$

.

.

.

$i_{k+1} := /\underline{K}/(i_k): |\underline{K}|^k(D_0) \to |\underline{K}|^{k+1}(D_0)$

$j_{k+1} := /\underline{K}/(j_k): |\underline{K}|^{k+1}(D_0) \to |\underline{K}|^k(D_0)$

.

.

.

Let A be the following two sorted algebra

$A_{s0} := (\{0\},0)$, $A_{s1} := (\{0,1\},1)$

$\sigma: (\{0\},0) \to (\{0,1\},1)$ with $\sigma(0) := 1$

$\pi: (\{0,1\},1) \to (\{0\},0)$ with $\pi(0) := 0$, $\pi(1) := 0$

Then iterating A by the composed functor $Po\bar{E}$ gives the following sequence of algebras:

$|Po\bar{E}|^0(A) := A$

$|Po\bar{E}|^1(A) := A'$ with

$A'_{s0} := (\{0,1\},1)$, $A'_{s1} := (\{0,1\}+\{2\},2)$

σ': $A'_{s0} \to A'_{s1}$

$y \to \begin{cases} 2 & \text{if } y = 1 \\ \sigma(y) & \text{otherwise} \end{cases}$

π': $A'_{s1} \to A'_{s0}$

$z \to \begin{cases} 1 & \text{if } y = 2 \\ \pi(z) & \text{otherwise} \end{cases}$

$|Po\bar{E}|^2(A) := A''$ with

$A''_{s0} := (\{0,1,2\},2)$, $A''_{s1} := (\{0,1,2\}+\{3\},3)$

σ'': $A''_{s0} \to A''_{s1}$

$y \to \begin{cases} 3 & \text{if } y = 2 \\ \sigma'(y) & \text{otherwise} \end{cases}$

π'': $A''_{s1} \to A''_{s0}$

$z \to \begin{cases} 2 & \text{if } z = 3 \\ \pi'(z) & \text{otherwise} \end{cases}$

$|Po\bar{E}|^{k+1}(A) := A'^{(k+1)}$

$A'^{(k+1)}_{s0} := (A'^{(k)}_{s1},k+2)$,

$A'^{(k+1)}_{s1} := (A'^{(k)}_{s1}+\{k+2\},k+2)$

$\sigma'^{(k+1)}$: $A'^{(k+1)}_{s0} \to A'^{(k+1)}_{s1}$

$y \to \begin{cases} k+2 & \text{if } y = k+1 \\ \sigma'^{(k)}(y) & \text{otherwise} \end{cases}$

$\pi'^{(k+1)}$: $A'^{(k+1)}_{s1} \to A'^{(k+1)}_{s0}$

$z \to \begin{cases} k+1 & \text{if } z = k+2 \\ \pi'^{(k)}(z) & \text{otherwise} \end{cases}$

The iteration involves not only algebras but also algebra homomorphisms.

Define

$h: A \to |Po\bar{E}|(A)$

with

$h: \langle h_{s0}, h_{s1} \rangle$

such that

$h_{s0}: A_{s0} \to A'_{s0}$

$0 \rightarrow 0$

$h_{s1}: A_{s1} \rightarrow A'_{s1}$

$0 \rightarrow 0$

$1 \rightarrow 1$

(h_{s0} and h_{s1} are ´embeddings´)

Then you can easily verify, that h is a homomorphism,

e.g. $h_{s1}(\sigma(0)) = h_{s1}(1) = 1$

$$\begin{array}{ccc} h_{s1}(\sigma(0)) & = h_{s1}(1) = & 1 \\ \| & & \| \\ \sigma'(h_{s0}(0)) & = \sigma'(0) = & 1. \end{array}$$

Then you can iterate h itself by $/Po\bar{E}/$ and get the sequence of homomorphisms:

$h: A \rightarrow |Po\bar{E}|(A)$

$h' := /Po\bar{E}/(h): |Po\bar{E}|(A) \rightarrow |Po\bar{E}|^2(A)$

$h'^{(k+1)} := /Po\bar{E}/(h'^{(k)}): |Po\bar{E}|^k(A) \rightarrow |Po\bar{E}|^{k+1}(A)$

The whole process corresponds to the following diagram.

$$A \xrightarrow{h} |Po\bar{E}|(A) \xrightarrow{/Po\bar{E}/(h)} |Po\bar{E}|^2(A) \xrightarrow{/Po\bar{E}/^2(h)} |Po\bar{E}|^3(A) \ \ldots$$

where in a sense (which will be explained later on) each algebra $|Po\bar{E}|^{k+1}(A)$ contains ´more information´ than $|Po\bar{E}|^k(A)$.

´More information´ here means

$|Po\bar{E}|^k(A) \subseteq |Po\bar{E}|^{k+1}(A)$

and

$\sigma'^{(k+1)} = \sigma'^{(k)} \cup \{(k+1,k+2)\}$

$\pi'^{(k+1)} = \pi'^{(k)} \cup \{(k+2,k+1)\}.$

If we follow this iteration process, we shall get at least an algebra $\hat{A}$ with

$\hat{A}_{s0} := \mathbf{N}$

$\hat{A}_{s1} := \mathbf{N}'+\{0\}$ where $\mathbf{N}'$ is the set of successors of natural numbers

together with

$\hat{\sigma}$: $\mathbf{N} \to \mathbf{N}'+\{0\}$

$h \mapsto h+1$

$\hat{\mathbf{f}}$: $\mathbf{N}'+\{0\} \to \mathbf{N}$

$h+1 \to h$

$0 \to 0$

One can now verify that $\hat{A}$ is a fixedpoint of $P \circ \bar{E}$. But we have still 'redundant' information due to the fact that $\hat{A}_{s0}$, $\hat{A}_{s1}$ are isomorphic as sets. Therefore we have to apply a sort of identification process on $\hat{A}$ (coequalizer-construction in categories!). The resulting 'solution' will then be a one sorted algebra consisting of the carrier $\mathbf{N}$ and operations σ': $\mathbf{N} \to \mathbf{N}$ and $\mathbf{f}'$: $\mathbf{N} \to \mathbf{N}$.

As we have seen we need two functors P: $\underline{\underline{Alg}}_X \to \underline{\underline{Alg}}_D$ and $\bar{E}$: $\underline{\underline{Alg}}_D \to \underline{\underline{Alg}}_X$ to define the endofunctor $P \circ \bar{E}$: $\underline{\underline{Alg}}_D \to \underline{\underline{Alg}}_D$ and then look for fixedpoints. In the previous example the reason for this was that if we want to iterate a D-algebra A by $P \circ \bar{E}$ we have to reduce it first to an X-algebra by $\bar{E}$ and then to apply the data type constructor P.
As certain fixedpoints of $P \circ \bar{E}$ should give meaning to 'recursive domain equations' in an algebraic framework it should be obvious that these 'equations' should be defined by a pair of $\underline{\underline{Spec}}$-morphisms p,e: X → D Here p is a parameterized specification and e is used to define the forgetful functor $\bar{E}$: $\underline{\underline{Alg}}_D \to \underline{\underline{Alg}}_X$. The following sequence of definitions and theorems formalizes the ideas we have given in the introduction.

II.1. Definition

Let X and D be specifications.
An algebraic domain equation (ade), denoted by

$$X \ (\underline{\underline{p,e}}) \ D$$

consists of a pair of $\underline{\underline{Spec}}$-morphisms

$p,e: X \to D$

where p is a (strongly) persistent parameterized specification and e describes a forgetful functor.
Remarks: (i) The double-arrow $(\underline{\underline{\ ,\ }})$ in $X \ (\underline{\underline{p,e}}) \ D$ is used to indicate

that p,e are directed from X to D.

(ii) By I.3.2. we know that each Spec-morphism defines a forgetful functor. As we have indicated in convention I.3.11. each parameterized specification should use unique names (for sort, constants, equality). Then for e it suffices to map the formal parameter components to the (new) components in the resulting specification which have unique names. So if the formal parameter is denoted by X and the target specification by P then

$e(X) := P$

$e(\bar{x}) := \bar{p}$

$e(\equiv_X) := \equiv_P$

As I have already mentioned in the introduction to this chapter we have two endofunctors described by p and e, namely

$\bar{E} \circ P: \underline{Alg}_X \to \underline{Alg}_X$

and

$P \circ \bar{E}: \underline{Alg}_D \to \underline{Alg}_D$

We shall use this later on but the two functors are related in the following way:

II.2. Lemma

Let $A \in |\underline{Alg}_X|$. Then the following assumption holds:
If A is fixedpoint of $\bar{E} \circ P$ then $|P|(A)$ is a fixedpoint of $P \circ \bar{E}$ and
if B is a fixedpoint of $P \circ \bar{E}$ then $|\bar{E}|(B)$ is a fixedpoint of $\bar{E} \circ P$.

Proof:
By persistency of P: $\underline{Alg}_X \to \underline{Alg}_D$ we have by

$\forall A \in |\underline{Alg}_X|.\ A \cong |\bar{E} \circ P|(A)$

that A is a fixedpoint of $\bar{E} \circ P$.
But since persistent functors respect isomorphy we have

$|P|(A) \cong |P| \circ |\bar{E} \circ P|(A) \iff$

$|P|(A) \cong |P \circ \bar{E}| \circ |P|(A)$

Thus $|P|(A)$ is a fixedpoint of $P \circ \bar{E}$.
The proof of the second part of the assumption works due to the fact that forgetful functors respect isomorphy.

II.3. Lemma

B is a fixed point of $P \circ \bar{E}$ iff $|\bar{E}|(B) \cong |\bar{P}|(B)$ and there exists an X-algebra A such that $|P|(A) \cong B$.

Proof:

B is a fixed point of $P \circ \bar{E}$ <=>

$B \cong |P \circ \bar{E}|(B)$

P is (strongly) persistent <=>

$\forall A \in |\underline{\underline{Alg}}_X|.\ A \cong |\bar{P} \circ P|(A)$

Since $|\bar{E}|(B)$ is an object of $\underline{\underline{Alg}}_X$ we have

$|\bar{E}|(B) \cong |\bar{P} \circ P|(|\bar{E}|(B))$ <=>

$|\bar{E}|(B) \cong |\bar{P}| \circ |P \circ \bar{E}|(P)$ <=>

$|\bar{E}|(B) \cong |\bar{P}|(B)$

They define $A := |\bar{E}|(B)$ and the proof is completed.

The next theorem is very important. It states (informally spoken) that each fixedpoint of $P \circ \bar{E}$ has a certain (syntactically determined) form and that we can restrict the search for fixedpoints to a subcategory of $\underline{\underline{Alg}}_D$. The argumentation is the following:

Let $X \overset{(p,e)}{\rightrightarrows} D$ be an ade. We know by I.1.3. that each pair of morphisms in $\underline{\underline{Spec}}$ has a coequalizer. This coequalizer leads to a renaming of sorts, operations and equations in the specification D as shown in the construction of coequalizers is $\underline{\underline{Spec}}$ (the section following theorem I.1.5.). So we know that (p e) has a coequalizer in $\underline{\underline{Spec}}$. This coequalizer consists of a unique specification Q and a unique $\underline{\underline{Spec}}$-morphism $q: D \to Q$ (see theorem I.1.3.).

Let coeq(p,e)=(q Q) denote this relation formally. Now it turns out that if $B \in |\underline{\underline{Alg}}_D|$ is a fixedpoint of $P \circ \bar{E}$ then there exists an Q-algebra C such that $|\bar{Q}|(C) \cong B$. That means that we may restrict our search for fixedpoints of $P \circ \bar{E}$ to coequalizer-

algebras ($\underline{Alg}_Q$). And here in coequalizer algebras an identification (renaming) between p- and e-components of D has been made. Now we turn to the theorem.

II.4. Theorem

Let X $\overset{(p,e)}{\Rightarrow}$ D be an ade and let (q Q) = coeq(p,e).

If B is a fixedpoint of $P \circ \bar{F}$ then there exists a unique Q-algebra C such that $B \cong |\bar{Q}|(C)$.

Proof:

Our argumentation here is the following:

(i) (q Q) = coeq(p,e). Now the functor **alg**: $\underline{Spec} \to \underline{Cat}^{op}$ sends each $\underline{Spec}$-morphism r: S → T to the forgetful functor $\bar{R} := \text{alg-r}: \underline{Alg}_T \to \underline{Alg}_S$. **alg** transforms coequalizers in $\underline{Spec}$ to equalizers in $\underline{Cat}$. If (q,Q) = coeq(p,e) in $\underline{Spec}$ then $(\bar{Q}, \underline{Alg}_Q) = eq(\bar{P},\bar{F})$ in $\underline{Cat}$ (where eq() denotes the equalizer).

Thus for each Q-algebra C $|\bar{P} \circ \bar{Q}|(C) = |\bar{F} \circ \bar{Q}|(C)$ holds. (equalizer-property!).

(ii) Since B is a fixed-point of $P \circ \bar{F}$ we know that $|\bar{P}|(B) \cong |\bar{F}|(B)$. But we can even construct a D-algebra $B' \cong B$ such that $|\bar{P}|(B') = |\bar{F}|(B')$. This looks similar to the equalizer property $|\bar{P} \circ \bar{Q}|(C) = |\bar{F} \circ \bar{Q}|(C)$.

(iii) What is still missing is an algebra $C \varepsilon |\underline{Alg}|_Q$ suchthat $B' \cong |\bar{Q}|(C)$. We show how we can construct a suitable Q-algebra C out of B' such that this property holds.

Then we shall have $B \cong B' \cong |\bar{Q}|(C)$ and we are ready.

(ii) Construction of B' using (i):

Let $S_X := \{s_X^1,\ldots,s_X^k\}$ and $S_D := \{s_D^1,\ldots,s_D^m\}$ denote the respective sort-sets of X and D in X $\overset{(p,e)}{\Rightarrow}$ D. Then since B is a D-algebra we have the carriers ${}^B s_D^1,\ldots,{}^B s_D^m$. Now construct B' as follows:

(a) $\forall\, s' \varepsilon S_D\ \forall s \varepsilon X.\ \ s' \neq p(s)\ \&\ s' \neq e(s) \Rightarrow B'_{s'} := B_{s'}$

(b) $\forall\, s' \varepsilon S_D\ \forall s \varepsilon S_X.\ \ p(s) = s' = e(s) \Rightarrow B'_{s'} := B_{s'}$

(c) $\forall s \in S_X \; \exists s', s'' \in S_D.\; p(s)=s' \neq s''=e(s) \Rightarrow B_{s'} := B_{s''} := B_{s''}$

Have we kept in mind $B \cong B'$?

Indeed we have!

In the cases (a) and (b) we have nothing to do here $B' \cong B$ is obviously satisfied.

But even case (c) works well because from $|\bar{P}|(B) \cong |\bar{E}|(B)$ we know that for $p(s) = s' \neq s'' = e(s)$ $B_{s'} \cong B_{s''}$. And so we have indeed $B' \cong B$. And furthermore by identification $B_{s'} := B_{s''} =: B_{s''}$ we have $|\bar{P}|(B') = |\bar{E}|(B')$.

(iii) Construction of C such that $B' \cong |\bar{Q}|(C)$.

We use the coequalizer property of $(q,Q) = coeq(p,e)$ for the construction of C. The most important fact is the construction of the carriers. We use the case distinction (a)-(c) from the construction of B':

For any sort $s' \in S_D$ which satisfies (a) define $C_{s'} := B_{s1}$ (q sends s' to $[s'] = \{s'\}$). For any sort $s' \in S_D$ satisfying (b) define $C_{s'} := B_{s'}$ (q sends s' to $[s'] = [p(s),e(s)]=\{s'\}$)

For any sorts s',s'' which satisfy (c) use a unique new sort name $\hat{s}$ for $[s',s''] = [p(s),e(s)]$ and define $C_{\hat{s}} := B_{s''}$.

Then the reduction $|\bar{Q}|(C)$ is isomorphic to B'.

So we have

$$B \cong B' \cong |\bar{Q}|(C) \Rightarrow B \cong |\bar{Q}|(C).$$

The uniqueness of C is due to the uniqueness of coequalizers!

II.5 Corollary

B is a fixedpoint of $P \circ \bar{E}$ iff the following conditions hold:

(a) $B \cong |P|(A)$ for a X-algebra A

(b) $B \cong |\bar{Q}|(C)$ for a Q-algebra C

Proof:

'=>': B is a fixedpoint of $P \circ \bar{E}$. Then by theorem II.4. there exists a unique Q-algebra C such that

$B \cong |\bar{Q}|(C)$. Furthermore $B \cong |P \circ \bar{E}|(B) \iff B \cong |P|(|\bar{E}|(B))$.

'$\Leftarrow$': Since $(\bar{Q}, \underline{Alg}_Q)$ is the equalizer of $\bar{P}$, $\bar{E}$ in $\underline{Cat}^{op}$ then we have for the unique $C \in \underline{Alg}_Q$ in (2): $|\bar{P}|(|\bar{Q}|(C)) = |\bar{E}|(|\bar{Q}|(C))$. Since $B \cong |\bar{Q}|(C)$ we have $|\bar{P}|(B) \cong |\bar{E}|(B)$. By Lemma II.3. this means that B is a fixedpoint of $P \circ \bar{E}$.

Each fixedpoint B of $P \circ \bar{E}$ has the property $|\bar{P}|(B) \cong |\bar{E}|(B)$. This means that all the carriers $B_{p(s)}$, $B_{e(s)}$ $(s \in S_X)$ are isomorphic and may therefore be identified. This identification is just what the coequalizer of p and e leads to. Moreover the identification process via coequalizers in $\underline{Spec}$ yields a unique category $\underline{Alg}_Q$ which consists in a sense of all D-algebras in which isomorphic $\bar{P}$ and $\bar{E}$-parts are identified. Motivated by the example in the introduction and by Theorem II.4. we consider solutions of ade's $X \overset{(p,e)}{\Rightarrow} D$ to be Q-algebras ($(q\ Q) = coeq(p,e)$) which have the fixedpoint property for $P \circ \bar{E}$, when they are reduced by $\bar{Q}$.

II.6. Definition

Let $X \overset{(p,e)}{\Rightarrow} D$ be an ade and $(q,Q) = coeq(p,e)$. Then a solution of $X \overset{(p,e)}{\Rightarrow} D$ is a Q-algebra C such that

$$|\bar{Q}|(C) \cong |P \circ \bar{E}|(|\bar{Q}|(C))$$

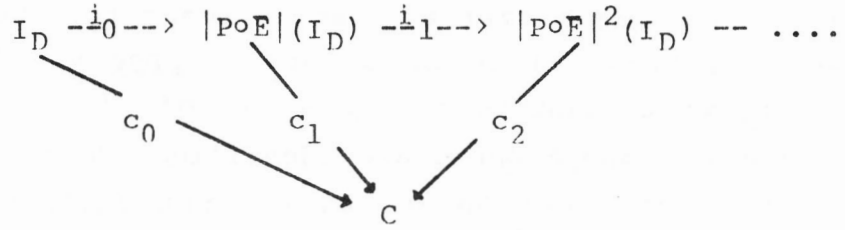
($|\bar{Q}|C$ is a fixedpoint of $P \circ \bar{E}$!)

Now according to this definition there may be a variety of Q-algebras C which are solutions of $X \overset{(p,e)}{\Rightarrow} D$. But when we started we had in mind to use ade's for implicit specifications of parameterized data types. Due to the ADJ-philosophy there ade's should define an (up to isomorphism) unique data type. So the question is: Is there a uniquely determined Q-algebra C which is a solution for $X \overset{(p,e)}{\Rightarrow} D$? Fortunately Ehrich and Lipek have a positive answer to this question:
The initial Q-algebra denoted by I_Q is uniquely determined and is a solution of $X \overset{(p,e)}{\Rightarrow} D$!

The following theorems and definitions show the development of this result. We proceed by using an analogon to Scott's inverse limit construction.

II.7. Therorem

Let $X \overset{(p,e)}{\rightleftarrows} D$ be an ade and let I_D denote the initial D-algebra. Furthermore let $i_0: I_D \to |P\circ\bar{E}|(I_D)$ be the unique initial homomorphism and let $i_{k+1} := /P\circ\bar{E}/(i_k)$. Then the colimit D-algebra of the diagram

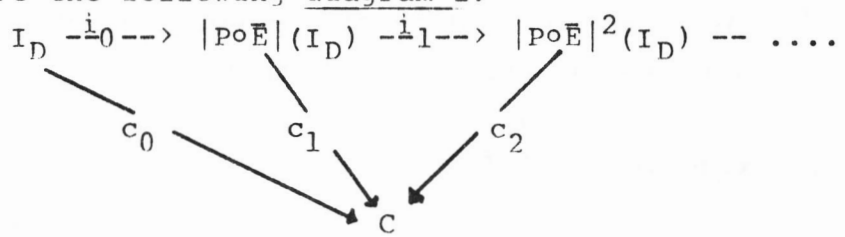


is a fixedpoint of $\bar{E}\circ P$.

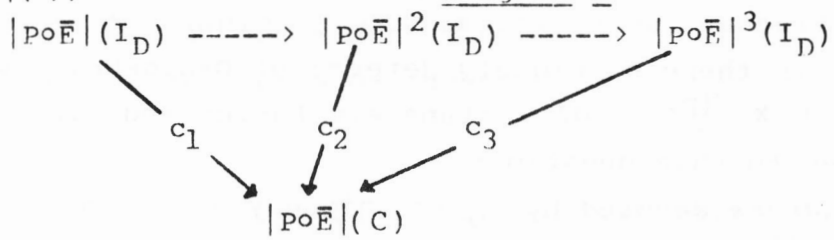
Proof

We know that for any $D \in |\underline{Spec}|$ the category $\underline{Alg}_D$ of D-models is cocomplete. Thus the colimit of the above diagram exists. Let it be denoted by $((c_n: |P\circ\bar{E}|^n \to C \mid n \in \mathbf{N}), C)$.

Thus we have the following diagram 1:



Now forgetful functors (here: $\bar{E}$) and their left-adjoints (here: P) respect colimits of chains. Therefore $((/P\circ\bar{E}/(c_n) \mid n \in \mathbf{N}_0), |P\circ\bar{E}|(C))$ is the colimit of diagram 2



But since I_D is initial in $\underline{\underline{Alg}}_D$ we are able to add $i_0: I_D \to |P\circ\bar{F}|(I_D)$ and $c_0: I_D \to |P\circ\bar{F}|(C)$. So $((/P\circ\bar{F}/(c_n)|n\in N_0), (P\circ\bar{F}|(C))$ is another colimit of diagram 1. Since colimits are unique up to isomorphism, we have

$$C \cong |P\circ\bar{F}|(C).$$

Thus C is a fixedpoint of $P\circ\bar{F}$.

Remarks: (1) As an analogy to Scott's inverse limit for $D \cong \underline{K}(D)$ we know that

$$D_\infty \overset{(\psi,\Phi)}{\underline{\cong}} \underline{K}(D_\infty).$$

(2) By Theroem II.4. there exists a unique (up to isomorphism) O-algebra $A\in|\underline{\underline{Alg}}_O|$ such that

$$C \cong |\bar{O}|(A).$$

Now we want to prove that the initial O-algebra I_O is a solution of $X \overset{(P,e)}{\underline{=>}} D$. Before giving the proof I shall outline the argumentation.

(i) From the fixedpoint property we know: if $B\in\underline{\underline{Alg}}_D$ is a fixedpoint of $P\circ\bar{F}$ then $|\bar{F}|(B) \cong |\bar{P}|(B)$. According to this fact we get a category $\underline{\underline{Iso}}$ with:

objects: The class of all pairs (B,β) where B is a fixedpoint of $P\circ\bar{F}$ and $\beta: |\bar{F}|(B) \to |\bar{P}|(B)$ is the isomorphism for $|\bar{F}|B \cong |\bar{P}|(B)$.

morphisms: $f: (B_1,\beta_1) \to (B_2,\beta_2)$ where $f: B_1 \to B_2$ is an $\underline{\underline{Alg}}_D$-morphism such that $|\bar{P}|(f)\circ\beta_1 = \beta_2\circ|\bar{F}|(f)$.

Note that $\underline{\underline{Iso}}$ is a full subcategory of $\underline{\underline{Alg}}_D$. We can consider $\underline{\underline{Iso}}$ to be the category of all fixedpoints of $P\circ\bar{F}$.

(ii) Theorem II.4 shows that for each fixedpoint B of $P\circ\bar{F}$ we can construct a $B' \in \underline{\underline{Alg}}_D$ such that $B \cong B'$ and $|\bar{F}|(B') = |\bar{P}|(B')$.

This fact leads to the category $\underline{\underline{Equ}}$ with:

objects: all pairs $(B', id_{|\bar{F}|(B')})$ from $\underline{\underline{Iso}}$.

morphisms: all $f: (B_1, id_{|\bar{F}|(B1)}) \to (B_2, id_{|\bar{F}|(B2)})$

such that

$/\bar{E}/(f) = /\bar{P}/(f)$. (because $/\bar{P}/(f).id = \bar{P}/(f)$
$= /\bar{E}/(f)$
$= id_{O}/\bar{E}/(F)$

On the other hand since $\bar{O}$: $\underline{\underline{Alg}}_O \to \underline{\underline{Alg}}_D$ is an equalizer of $\bar{E}$ and $\bar{P}$ the characterizing property of the morphisms and objects in $\underline{\underline{Equ}}$ (where $|\bar{E}|(B) = |\bar{P}|(B)$ and $/\bar{E}/(F) = /\bar{P}/(F)$) is similar to $\bar{O}$'s characteristic property namely
$\bar{E} \circ \bar{O} = \bar{P} \circ \bar{O}$.
These observations culminate in the assumption that $\underline{\underline{Equ}}$ and $\underline{\underline{Alg}}_O$ are isomorphic as categories (provided the range of $\bar{O}$ is restricted.)

(iii) We had $((c_n|n\varepsilon\mathbf{N}_0),C)$ as the colimit of diagram 1 in Theorem II.7 The object C was isomorphic to $|P\circ\bar{E}|(C)$.
Let γ: $|P\circ\bar{E}|(C) \to C$ be this isomorphism. Then the pair $(C\ /\bar{P}/(\gamma))$ is initial in $\underline{\underline{Iso}}$. Since $\underline{\underline{Iso}}$ and $\underline{\underline{Equ}}$ are equivalent as categories there exists a D-algebra C' isomorphic to C such that $(C', id_{|\bar{E}|(C')})$ is an object of $\underline{\underline{Equ}}$. Equivalence of $\underline{\underline{Iso}}$ and $\underline{\underline{Equ}}$ imply that initiality of $(C,/\bar{P}/(\gamma))$ is respected. Isomorphy of $\underline{\underline{Equ}}$ and $\underline{\underline{ALG}}_O$ then implies that I_O is a solution of $X \overset{(p,e)}{\underline{\underline{\Rightarrow}}} D$ since isomorphy respects initiality. Thus the argumentation is closed.

Now we can state the main theorem.

II.8. Theorem

Let $X \overset{(p,e)}{\underline{\underline{\Rightarrow}}} D$ be an ade where $(q,O) = coeq(p,e)$. Then the initial O-algebra I_O is a solution.

Proof:
Suppose $((c_n|n\varepsilon\mathbf{N}_0),C)$ is the colimit of diagram 1 in Theorem II.7. Then $((/P\circ\bar{E}/(c_n)|n\varepsilon\mathbf{N}_0),|P\circ\bar{E}|(C))$ is another colimit of this diagram and thus $C \cong |P\circ\bar{E}|(C)$.

Let γ: $|P\circ\bar{E}|(C) \to C$ be the isomorphism as stated above.
Then $/\bar{P}/(\gamma)$: $|\bar{E}|(C) \to |\bar{P}|(C)$ is an isomorphism.
$(\bar{P}\circ(P\circ\bar{E}) = (\bar{P}\circ P)\circ\bar{E} = id\circ\bar{E} = \bar{E})$.
$(C,/\bar{P}/(\gamma))$ is an $\underline{Iso}$-object and furthermore $(C,/\bar{P}/(\gamma))$ is initial in $\underline{Iso}$.
Let (B,β) be an $\underline{Iso}$-object. Then what remains to be done is the construction of a unique $\underline{Iso}$-morphism

$$f: (C,/\bar{P}/(\gamma)) \to (B,\beta).$$

Since $C \cong |P\circ\bar{E}|(C)$ and γ: $|P\circ\bar{E}|(C) \to C$ is an isomorphism we have as a characteristic property for the c_n: $|P\circ\bar{E}|^n(I_D) \to C$:

(i) $c_{n+1} = \gamma\circ/P\circ\bar{E}/(c_n)$

Furthermore there exists an initial morphism b_0: $I_D \to B$ and a unique morphism f: $C \to B$ since C is a colimit.
So we can construct b_n: $|P\circ\bar{E}|^n(I_D) \to B$ by using b_0 and

(ii) $/\bar{P}/(b_{k+1}) = \beta\circ/\bar{E}/(b_n)$

Furthermore we have

(iii) $b_n = b_{n+1}\circ i_n$.

And with the unique morphism
f: $C \to B$ we get

(iv) $b_n = f\circ c_n$

But since we want to use f to construct the required initial morphism in $\underline{Iso}$ then f must have the following property:

(v) $/\bar{P}/(f)\circ/\bar{P}/(\gamma) = \beta\circ/\bar{E}/(f)$.

It remains to show the equivalence of (iv) and (v).

(iv) => (v)

We have $b_{n+1} = f\circ c_{n+1}$ $(n\in\mathbf{N}_0)$. From (ii) we get
$/\bar{P}/(b_{n+1}) = P\circ/\bar{E}/(b_n)$ and from (i) we get $c_{n+1} = \gamma\circ/P\circ\bar{E}/(c_n)$.
Then

$$\begin{aligned} /\bar{P}/(b_{n+1}) &= /\bar{P}/(f)\circ/\bar{P}/(c_{n+1}) \\ &= (/\bar{P}/(f)\circ/\bar{P}/(\gamma))\circ\bar{E}(c_n) \\ &\overset{(ii)}{=} \beta\circ/\bar{E}/(b_n) \\ &= \beta\circ/\bar{E}/(f)\circ/\bar{E}/(c_n). \end{aligned}$$

So $/\bar{P}/(f)\circ/\bar{P}/(\gamma) = \beta\circ/\bar{E}/(f)$ for all n.

(v) => (iv)

We use induction on n:

For $n=0$ we have $b_0 = f \circ c_0$ by initiality. Suppose $b_n = f \circ c_n$ for all $k \leq n$. Then by (ii)

$$
\begin{aligned}
/\bar{P}/(b_{n+1}) &= \beta \circ /\bar{E}/(b_n) \\
&= \beta \circ (/\bar{E}/(f) \circ /\bar{E}/(c_n)) \quad \text{(induction hypothesis)} \\
&\overset{(v)}{=} (/\bar{P}/(f) \circ /\bar{P}/(\gamma)) \circ /\bar{E}/(c_n) \\
&\overset{(i)}{=} /\bar{P}/(f) \circ /\bar{P}/(c_{n+1}) \\
&= /\bar{P}/(f \circ c_{n+1})
\end{aligned}
$$

Thus $b_{n+1} = f \circ c_{n+1}$ and the proof is complete.

III. Continuous Data Types and Parameterization

I'll return now to our basic question, namely: Is it possible to combine the data type approaches of Scott and the ADJ-group to get a concept of parameterized continuous data types?
Is it possible to specify parameterized continuous data types by a sort of algebraic domain equations as presented in the preceeding chapter?
We'll try to clarify some aspects of this problem in this chapter.

III.1. Continuous Data Types

First some standards from the theory of domains.

III.1.1 Definition

Let $(D,\leq)$ be a partial order with a minimal element $\perp_D$. $X \subseteq D$ is directed if every finite subset $Z \subseteq X$ has an upper bound in X.

III.1.2. Definition

Let $(D,\leq)$ again be a partial order with $\perp_D$. Then D is a cpo (complete partial order) if every directed subset $X \subseteq D$ has a least upper bound $lub(X)$ in D.

III 1.3. Definition

Let $(D,\leq)$ and $(D',\leq')$ be partially ordered sets as in III.1.1.

(i) A mapping $f: D \to D'$ is monotonic if
$\forall\, d,d' \in D.\ d \leq d' \Rightarrow f(d) \leq' f(d')$

(ii) f is continuous if
$\forall\, X \subseteq D.\ X$ is directed $\Rightarrow f(lub(X)) = lub'\{f(x) \mid x \in X\}$

(iii) f is strict if
$f(\perp_D) = \perp_{D'}$

Now a continuous data type is merely an algebra whose carriers are partially ordered and whose operations respect monotonicity and existing limits of directed sets.

III.1.4. Definition

Let $\underline{\Sigma} = \langle S, \Sigma \rangle$ be a signature.
A $\underline{\Sigma}$-algebra A is continuous iff
(i) A_s is a cpo (for each $s \in S$).
(ii) $\sigma_A: A_{s1} \times \ldots \times A_{sn} \to A_s$ is a continuous function for each $\sigma \in \Sigma_{s1,\ldots,sn,s}$.
Let $\underline{\underline{CAlg}}_\Sigma$ denote the category of continuous $\underline{\Sigma}$-algebras together with strict continuous homomorphisms. Then $\underline{\underline{CAlg}}_\Sigma$ has an initial object: the continuous term algebra $CT\Sigma$. $CT\Sigma$ is in a sense $T\Sigma$ equipped with an order-theoretical structure.

III.1.5. Definition

Let $\underline{\Sigma} = \langle S, \Sigma \rangle$ be a signature. Then the continuous term-algebra $CT\Sigma$ is defined by
(i) For each $s \in S$ $\quad \bot_s$ is the least element of $CT_{\Sigma,S}$
$\forall\, t \in CT\Sigma_s.\ \bot_s \leqslant t$.
(ii) Let $\sigma \in \Sigma_{s1,\ldots,sn,s}$ and for $i = 1,\ldots,n$ let $t_i \leqslant t_i'$ $(t_i, t_i' \in T\Sigma_{si})$
Then $\sigma(t_1,\ldots,t_n) \leqslant \sigma(t_1',\ldots,t_n')$.
(iii) Let $\sigma \in \Sigma_{s1,\ldots,sn,s}$. The operation corresponding to σ in $CT\Sigma$ is denoted by σ_c and defined by
(a) $\sigma_c := \sigma$ if $\sigma \in \Sigma_{\lambda,s}$
(b) $\sigma_c(t_1,\ldots,t_n) = \sigma_{t1,\ldots,tn}$.

The following theorem is due to [ADJ 77].

III.1.6. Theorem

$CT\Sigma$ is initial in $\underline{\underline{CAlg}}_\Sigma$.

So even in the category of continuous Σ-algebras we have a uniquely determined model for a signature. Again, choosing this model as the continuous meaning of a signature is standard.

III.1.7. Definition

Let Σ be a signature.
The abstract continuous data specified by Σ is the isomorphy class of $CT\Sigma$.
But what about initial models for specifications Spec = ⟨S,Σ,E⟩? The search for initial models in $\underline{\underline{CAlg}}_{\Sigma,E}$ is successful for a class of specifications which give rise to so-called continuous congruences. If there exists a strict continuous function nf: $CT\Sigma \to CT\Sigma$ which takes each member $t \in CT_{\Sigma,S}$ to a normalform nf(t) and which behaves 'well' with respect to the continuous congruence generated by D then we have an initial algebra $CT\Sigma/\equiv_E$ in $\underline{\underline{CAlg}}_{\Sigma,E}$. The following definitions and theorems concerning continuous congruences, normalizers and initiality of $CT\Sigma/\equiv_E$ are taken from [L/M 82].
The set of equations E in Spec = ⟨S Σ,E⟩ generates in the normal framework exposed by the ADJ-group the congruence $\equiv_E$ on $T\Sigma$. So the initial quotient $T\Sigma/\equiv_E$ is generated. Moreover $\equiv_E$ is unique in the sense that it is the least congruence containing E. One critical point about continuous data types is that limit points for chains and directed sets exist. But normally the congruence relation $\equiv_E$ generated is not defined on those limits because only finite terms are considered to be elements of $T\Sigma$. But $CT\Sigma$ contains finite and infinite terms. Then a continuous congruence has to take into account limits of directed sets.

III.1.8. Definition

Let Σ be a signature. Let A be a continuous Σ-algebra and $R \subseteq A \times A$ a relation on A.
Then the continuous Σ-congruence on A generated by R (denoted by $\equiv_R$) is the least congruence on A containing R and whenever there

are two chains $\langle a_i | i\varepsilon I\rangle$, $\langle b_i | i\varepsilon I\rangle$ is A_s with $(a_i,b_i)\varepsilon\equiv_R$ then
$(\mathrm{lub}(\langle a_i | i\varepsilon I\rangle), \mathrm{lub}(\langle b_i | i\varepsilon I\rangle)) \varepsilon \equiv_R$.

A congruence class of $\equiv_R$ (denoted by $[\ ]_{\equiv R}$) contains equivalent elements. It is useful to choose one representative for each congruence-class and to use this representative in proofs concerning the whole congruence-class. If certain properties hold we can speak of a normalform. A normalizer then is roughly spoken a mapping which takes terms to their normalforms. In the framework of continuous algebras order-theoretical requirements have to be considered.

III.1.9. Definition

Let $\equiv_R$ denote a continuous congruence on $CT\Sigma$. A continuous mapping $nf: CT\Sigma \to CT\Sigma$ is called a normalizer with respect to $\equiv_R$ iff the following properties hold:
(i) $\forall\ t,t'\varepsilon CT\Sigma_s.\ ([t] = [t']) \Rightarrow nf(t) = nf(t')$
(ii) $\forall\ t\varepsilon CT\Sigma_s.\ [nf(t)] = [t]$

Let $\equiv_R$ denote in the sequel the continuous congruence on a continuous Σ-algebra as in III.1.8. To make $CT\Sigma/\equiv_R$ a continuous algebra we have to define a partial ordering between congruence classes. We do this by using the normalizer nf.

III.1.10. Definition

Let $CT\Sigma/\equiv_R$ be the quotient of $CT\Sigma$ with respect to the congruence $\equiv_R$ and let nf be a normalizer.
We define a partial ordering $'\leq_R'$ on $CT\Sigma/\equiv_R$ in the following way:
$\forall\ t,t'\varepsilon CT\Sigma_s.\ [t] \leq_R [t']: \Leftrightarrow nf(t) \leq nf(t')$

It is easy to verify that $'\leq_R'$ is indeed a well-defined partial ordering on $CT\Sigma/\equiv_R$ due to the fact that $'\leq'$ is well defined on $CT\Sigma$ and that $nf: CT\Sigma \to CT\Sigma$ is a normalizer.
Furthermore $CT\Sigma/\equiv_R$ is chain complete.

III.1.11. Lemma

Let $\langle [t_i] \mid i \in I \rangle$ be a chain in $CT\Sigma/\equiv_R$.
Define $lub(\langle [t_i] \mid i \in I \rangle) := [nf(lub(\{t_i \mid i \in I\}))]$.
Then $CT\Sigma/\equiv_R$ is chain-complete.

Let Spec = $\langle S\ \Sigma, E \rangle$ be a specification. Then E generates a continuous congruence on $CT\Sigma$ (denoted by $\equiv_E$). This congruence always exists and is uniquely determined, since it is the least congruence on $CT\Sigma$ which contains E and for which the continuity property holds. Now the question arises if $CT\Sigma/\equiv_E$ is initial in $\underline{\underline{CAlg}}_{\Sigma,E}$? Indeed, the main result in the work of [L/M 82] has shown that if a normalizer nf: $CT\Sigma \to CT\Sigma$ with respect to $\equiv_E$ exists, then $CT\Sigma/\equiv_E$ is initial in $\underline{\underline{CAlg}}_{\Sigma,E}$.

III.1.12. Theorem

Let Spec = $\langle S, \Sigma, E \rangle$ be a specification and $\equiv_E$ be the continuous congruence on $CT\Sigma$ generated by E.
If there exists a normalizer nf $CT\Sigma \to CT\Sigma$ with respect to $\equiv_E$ then $CT\Sigma/\equiv_E$ is initial in $\underline{\underline{CAlg}}_{\Sigma,E}$.

III.2. Parameterized Continuous Data Types

The preceding result is very important when using continuous algebras as data types. If Spec = $\langle S, \Sigma, E \rangle$ is a specification we know that there always exists a least continuous congruence on $CT\Sigma/\equiv_E$ containing E. Furthermore if there exists a normalizer with respect to $\equiv_E$ then the quotient $CT\Sigma/\equiv_E$ is initial in $\underline{\underline{CAlg}}_{\Sigma,E}$. As we have seen in Theorems II.7. and II.8. the existence of initial algebras is essential in constructing a unique solution for algebraic domain equations. As we wanted to use this method for specifying parameterized continuous algebras it seems reasonable to restrict our attention to those specifications which possess an initial continuous model. In the context we have shown in the preceeding sections this means we

consider only those specifications for which a normalizer exists. This category of specifications will be denoted by NSpec (Specifcations with normalizers).

III.2.1. Definition

The category NSpec is defined by
objects: all Spec-objects with finite sort-, operation-, equation-sets for which a normalizer does exist.
Morphisms: the morphisms between two NSpec-objects are those in Spec which exist between these two objects.

III.2.2. Corollary

NSpec is a full subcategory of Spec.
Proof: obvious from the definition of NSpec.

As we have seen in chapter II the cocompleteness-property is essential for using algebraic domain equations.

III.2.3. Corollary

NSpec is cocomplete.
Proof:
Since Spec is cocomplete and NSpec is a full subcategory of Spec it follows immediately that cocompleteness holds for NSpec.

We use the same parameterization concept for specifications from NSpec as defined for Spec in chapter I, namely that a parameterized specification is an injective NSpec-morphism and that the result of parameter passing is the pushout of a certain diagram in NSpec (see definitions I 3.1. and I.3.8.). But what about our forgetful functors and persistent data type constructors? Do they fit into the framework of continuous algebras?
Principally the answer is 'yes' but we have to take into account some merely slight modifications.

In definition I.3.2. we used the functor **alg**: $\underline{Spec} \to \underline{Cat}^{OP}$ sending each specification $Spec = \langle \Sigma, E \rangle$ to the category $\underline{Alg}_{\Sigma,E}$ and sending each $\underline{Spec}$-morphism $e: \langle \Sigma, E \rangle \to \langle \Sigma', E' \rangle$ to a forgetful functor $\bar{F}: \underline{Alg}_{\Sigma',E'} \to \underline{Alg}_{\Sigma,E}$. In the framework of continuous data types we modify **alg** explicitly in the following way:

III.2.4. Definition

calg: $\underline{NSpec} \to \underline{Cat}^{OP}$ is the following functor

(i) Each specification $\underline{S} = \langle \Sigma, E \rangle \in |\underline{NSpec}|$ is send to $\underline{CAlg}_{\Sigma,E}$.

(ii) Let $e: \langle \Sigma, E \rangle \to \langle \Sigma', E' \rangle$ be a $\underline{NSpec}$-morphism.

Then **calg** sends e to the forgetful functor

$\bar{F}: \underline{CAlg}_{\Sigma',E'} \to \underline{CAlg}_{\Sigma,e}$ defined by

(a) Each $\underline{CAlg}_{\Sigma',E'}$-algebra A' is sent to it's Σ-reduct A by:

$\forall\, s \in S.\quad A_s := A'_{e(s)}$ (notice that $\perp_s := \perp_{e(s)}$!)

(b) If A', B' are $\underline{CAlg}_{\Sigma',E'}$-algebras and $h': A' \to B'$ is a strict continuous algebra homomorphism then the resulting $\underline{CAlg}_{\Sigma,E}$ morphism $h := /\bar{F}/(h'): |\bar{F}|(A') \to |\bar{F}|(B')$ is given by

$\forall\, s \in S.\quad h_s := h'_{e(s)}$.

(notice that $h_s(\perp_{As}) = \perp_{Bs} = \perp_{B'e(s)} = h'_{e(s)}(\perp_{A'e(s)})$!)

It is important to see that each forgetful functor $\bar{F}: \underline{CAlg}_{\Sigma',E'} \to \underline{CAlg}_{\Sigma,E}$ which sends each continuous (Σ',E')-algebra to its continuous (Σ,E)-reduct respects continuity by definition (since reduction here is a renaming operation which doesn't affect the order-theoretical properties of algebras and homomorphisms).

In the case of persistent functors serving as data-type constructors the continuity of the target and resulting types requires the strong persistency property. If $p: X \to D$ is a parameterized specification in $\underline{NSpec}$ then weak persistency of the corresponding data type constructor is equivalent to

$\forall\, A \in \underline{CAlg}_X.\quad |\bar{P} \circ P_{free}|(A) \cong A$

But Scott's theory of domains shows that pure isomorphy of

domains is not strong enough to ensure that they possess the same order theoretical structure (e.g. that are homoeomorphic).
Thus weakly persistent data type constructions don't necessarily lead to equivalent order-theoretical structures of $|\bar{P}oP_{free}|(A)$ and the argument-algebra A.
The following theorem shows why.

III.2.5. Theorem

Let p: X → D be a specification in NSpec.
Let P_{free}: $\underline{CAlg}_X \to \underline{CAlg}_D$ be a weakly persistent functor with respect to p which is not strongly persistent.
There is an algebra $A \in |\underline{CAlg}_X|$ such that for a directed set $M \subseteq A_s$ (for a sort s∈sorts(X)) and the restricted persistent functor
$\tilde{P}$: $\underline{CAlg}_X \to \underline{CAlg}_{p(X)}$ the following inequality holds:
$|\tilde{P}|(lub_{As}(M)) \neq lub_{|\tilde{P}|(A)p(S)}\{|\tilde{P}|(m) \mid m \in M\}$
(This means that weak persistency doesn't necessarily respect continuity!)
Proof:
Let X and D be the following specifications:
$X := \langle \{s\}, \sigma: s \to s \rangle$
$D := \langle \{r\}, \psi: r \to r \rangle$
with $f_{sort}(s) := r$
$f_{op}(\sigma) := \psi$ } renaming of operations
Let A be the following $\underline{CAlg}_X$-algebra·
$A_s := \{\perp_s, 0, 1\}$ with $\perp_s \leq 0$ and $\perp \leq 1$
$\sigma_A : A_s \to A_s$
$\perp_s \to \perp_s$
$0 \to \perp_s$
$1 \to 1$
Let $A' \in |\underline{CAlg}|_{p(X)}$ be the following algebra:
$A'_r := \{\perp_r, b, c\}$ with $\perp_r \leq b$, $b \leq c$ ($\Rightarrow \perp_r \leq c$)
$\psi: A'_r \to A'_r$
$\perp_r \to \perp_r$
$b \to c$
$c \to c$

Let the functor $\tilde{P}$ be defined by

$|\tilde{P}|(A) := A'$

$|\tilde{P}|(\bot_S) := c$

$|\tilde{P}|(0) := b$

$|\tilde{P}|(1) := \bot_r$

$|\tilde{P}|(\sigma_A) := \psi_{A'}$

Then we have $A \cong |\tilde{P}|(A)$ which is equivalent to the persistency-property (we have defined $\tilde{P}$ to be the restricted functor defined by $p: X \to D$ and thus may omit the forgetful functor $\bar{P}: \underline{CAlg}_D \to \underline{CAlg}_X$)).

Let M be the directed set:

$M := \{\bot_S, 1\}$

Then we have:

$|\tilde{P}|(lub_S M) = |\tilde{P}|(1) = \bot_r$

and

$lub_r\{ |\tilde{P}|(m) | m \in M\} = lub_r\{\bot_r, c\} = c.$

Thus $|\tilde{P}|(lub_S(M)) \neq lub_r(\{|\tilde{P}|(m) | m \in M\})$ and $\tilde{P}$ doesn't respect continuity.

III.2.6. Definition

A parameterized continuous data type consists of a parameterized specification

$p: X \to D$

in $\underline{NSpec}$ and a strongly persistent data type constructor $P_{free}: \underline{CAlg}_X \to \underline{CAlg}_D$ that takes each $\underline{CAlg}_X$-algebra A to it's free continuous extension over A with respect to p such that

$|\bar{P} \circ P_{free}|(A) = A.$

We have defined algebraic domain equations to be a pair $X \overset{(p,e)}{\Rightarrow} D$ where $p: X \to D$ is a parameterized specification. Therefore p is an injective $\underline{Spec}$-morphism. The $\underline{Spec}$-morphism e is only supposed to define a forgetful functor $\bar{E}: \underline{Alg}_D \to \underline{Alg}_X$. But in all interesting cases of applications of algebraic domain equations it turns out that even e is an injective $\underline{Spec}$-morphism. Thus it seems reasonable to modify the concept of algebraic domain

equations in this sense:

III 2.7. Definition

Let X, D be two $\underline{\underline{NSpec}}$-objects.
Then by a continuous algebraic domain equation we mean a pair of injective $\underline{\underline{NSpec}}$-morphisms p,e: X → D.
According to solutions of continuous algebraic domain equations we proceed in a similar way as shown in chapter II. The proofs of the various theorems are not yet worked out in detail but will appear in an extended version of this paper.
We have the following facts:

Fact 1
$\underline{\underline{NSpec}}$ is cocomplete. Thus colimits, coproducts and coequalizers exist in $\underline{\underline{NSpec}}$.

Fact 2
$\forall D \in |\underline{\underline{NSpec}}|$. $\underline{\underline{CAlg}}_D$ has an initial object.

This initial object is the quotient $CT\Sigma/\equiv_E$ ($\Sigma := sig(D)$) which is guaranteed to exist in $\underline{\underline{CAlg}}_D$ since a normalizer
nf: $CT\Sigma \to CT\Sigma$ exists for D and the continuous congruence generated by the equations in D.

Fact 3
According to the definition of the forgetful functor $\bar{P}$: $\underline{\underline{CAlg}}_D \to \underline{\underline{CAlg}}_X$ (p: X → D is a $\underline{\underline{NSpec}}$-morphism) there exists a left-adjoint P_{free}: $\underline{\underline{CAlg}}_X \to \underline{\underline{CAlg}}_D$ which respects continuity and takes each $\underline{\underline{CAlg}}_X$-algebra to its free continuous extension.

Fact 4
The functor **calg**: $\underline{\underline{NSpec}} \to \underline{\underline{Cat}}^{op}$ respects colimits in $\underline{\underline{NSpec}}$.
Thus we are able to define what a solution of a continuous algebraic domain equation is.

III.2.6. Definition

Let $X \overset{(p,e)}{\Rightarrow} D$ be a continuous ade.
Let $(q,Q) := coeq(p.e)$ be the coequalizer of p and e guaranteed to exist by cocompleteness of $\underline{\underline{NSpec}}$. Then a continuous solution of $X \overset{(p,e)}{\Rightarrow} D$ is a $\underline{\underline{CAlg}}_Q$-algebra C´ such that

$$|\bar{Q}|(C) \cong |p\circ\bar{E}|(|\bar{Q}|(C))$$

Since the initial $\underline{\underline{CAlg}}_Q$-algebra I_Q exists we have to prove the following theorems in the framework of continuous algebras to hold. With these theorems we get the result that I_Q is a solution of $X \overset{(p,e)}{\Rightarrow} D$.

III.2.8. Assumption

Let $D \in |\underline{\underline{NSpec}}|$ be a specification.
Then $\underline{\underline{CAlg}}_D$ has colimits of chains of the form

$$I_D \xrightarrow{i_0} |p\circ\bar{E}|(I_D) \xrightarrow{i_1} |p\circ\bar{E}|^2(I_D) \dashrightarrow \dots$$

(where I_D is the initial algebra in $\underline{\underline{CAlg}}_D$)

III.2.9. Assumption

Let p be a $|\underline{\underline{NSpec}}|$ morphism.
Then **calg**-p respects colimits of chains.

III.2.10. Assumption

Let $(p,e) \overset{}{\Rightarrow} D$ be a continuous ade.
Let B be a $\underline{\underline{CAlg}}_X$-algebra Moreover B is a fixedpoint of $p\circ\bar{E}$.
Then there exists a D-algebra B´ such that $|\bar{E}|(B´) = |\bar{P}|(B´)$.

Under these conditions I_Q is a continuous solution of $X \overset{(p,e)}{\Rightarrow} D$ and due to the initiality property it is the unique continuous solution of $X \overset{(p,e)}{\Rightarrow} D$.

Bibliography

[ADJ 76]: Goguen J.A., Thatcher J.W., Wagner E.G.
An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types
IBM Research Report RC 6487, 1976.

[ADJ 77]: Goguen J.A., Thatcher J.W., Wagner E.G., Wright J.B.
Initial Algebra Semantics and Continuous Algebras
Journal of the ACM Vol. 24, No. 1., 1977, pp. 68-95.

[ADJ 82]: Thatcher J.W., Wagner E.G., Wright J.B.
Data Type Specification: Parameterization and the Power of Specification Techniques
ACM Toplas Vol. 4, No. 4, 1982, pp. 711-732.

[B/G 80]: Burstall R.M., Goguen J.A.
The Semantics of Clear. A Specification Language
University of Edinburgh, Internal Report CSR-65-80, 1980.

[E/L 81]: Ehrich H.D., Lipeck U.
Algebraic Domain Equations
Universität Dortmund, Forschungsbericht Nr. 125, 1981.

[L/M 82]: Levy M.R., Maibaum T.S.E.
Continuous Data Types
SIAM Journal of Computing, Vol. 11, No. 2, 1982.

[Rau 72]: Raulefs, P.
Semantik von Programmiersprachen
Vorlesungsnotizen, Universität Bonn, 1979.

[Sco 72]: Scott D.S.
Continuous Lattices
Springer Lecture Notes in Mathematics. Vol. 274, 1972,

pp 97-136.

[Sco 76]: Scott D.S.
Data Types as Lattices
SIAM Journal of Computing, Vol. 5, 1976, pp. 522-587.

11/22/1983

SEKI Memos

The following memos are available free of charge from

Mrs. Dorothea Kilgore
Universität Kaiserslautern
Fachbereich Informatik
Postfach 3049
D-6750 Kaiserslautern
West Germany

MEMO SEKI-81-01 U. Bartels, W. Olthoff and P. Raulefs: APE: An Expert System for Automatic Programming from Abstract Specifications of Data Types and Algorithms.

MEMO SEKI-81-03 Peter Raulefs: Expert Systems: State of the Art and Future Prospects.

MEMO SEKI-81-04 Christoph Beierle: Programmsynthese aus Beispielsfolgen.

MEMO SEKI-81-05 Erich Rome: Implementierungen Abstrakter Datentypen in terminaler Algebrasemantik.

MEMO SEKI-81-06 Christoph Beierle: Synthesizing Minimal Programs from Traces of Observable Behaviour.

MEMO SEKI-81-07 Dieter Wybranietz: Ein verteiltes Betriebssystem für CSSA.

MEMO SEKI-81-08 Ulrich Bartels and Walter Olthoff: APE - Benutzerbeschreibung.

MEMO SEKI-82-01 Hans Voß: Programming in a Distributed Environment: A Collection of CSSA Examples.

MEMO SEKI-82-02 Hartmut Grieneisen: Eine algebraische Spezifikation des Software-Produkts INTAKT.

MEMO SEKI-82-03 Christian Beilken, Friedemann Mattern and Michael Spenke: Entwurf und Implementierung von CSSA - Beschreibung der Sprache, des Compilers und des Mehrrechnersimulationssystems.
Printed in 6 volumes, which can be ordered individually:
Vol-A:Konzepte
Vol-B: CSSA-Sprachbeschreibung
Vol-C: CSSA-Systembenutzung
Vol-D: CSSA-Programmbeispiele

Vol-E1: Programmdokumentation Teil I
Vol-E2: Programmdokumentation Teil II

MEMO SEKI-83-01 Wilfried Schrupp and Johann Tamme: Spezifikation und abstrakte Implementierung des Aufbereitungsteils von INTAKT.

MEMO SEKI-83-02 Frank Puppe and Bernd Puppe: Overview on MED1: A Heuristic Diagnostic System with an Efficient Control-Structure.

MEMO SEKI-83-03 Elisabeth Hülsmann: LISP-SP : A portable INTERLISP Subset Interpreter for Mini-Computers.

MEMO SEKI-83-04 FrankPuppe: MED1 - Ein heuristisches Diagnosesystem mit effizienter Kontrollstruktur.

MEMO SEKI-83-05 Horst Peter Borrmann: MODIS - Ein Expertensystem zur Erstellung vonReparaturdiagnosen für den Ottomotor und seineAggregate.

MEMO SEKI-83-06 Harold Boley: From Pattern-Directed to Adapter-Driven Computation via Function-Applying Matching.

MEMO SEKI-83-07 Christoph Beierle and Angi Voß: Canonical Term Functors and Parameterization-by-use for for the Specification of Abstract Data Types.

MEMO SEKI-83-08 Christoph Beierle and Angi Voß: Parameterization-by-use for hierarchically structured objects.

MEMO SEKI-83-09 Christoph Beierle, Michael Gerlach and Angi Voß: Parameterization without parameters in : The History of a Hierarchy of Specifications.

MEMO SEKI-83-10 Ulrike Petersen: Elimination von Rekursionen.

MEMO SEKI-83-11 Gerd Krützer: An Approach to Parameterized Continuous Data Types.

MEMO SEKI-83-12 Richard Göbel: A Completion Procedure for Globally Finite Term Rewriting Systems.

MEMO SEKI-83-13 Michael Gerlach: A Second-Order Matching Procedure for the Practical Use in a Program Transformation System.