PARTIALLY ORDERED SORTS IN ALGEBRAIC SPECIFICATIONS

Martin Gogolla
Abteilung Informatik, Universität Dortmund
Postfach 500500, D-4600 Dortmund 50

Abstract

Conventional algebraic specification techniques cannot express relationships between sorts formally. The approach presented here puts more structure into the specification by allowing a partial ordering on the set of sorts to describe that one sort is a subsort of another sort. This concept implies that one function can occur more than one time in the signature with different domains and codomains. The initial algebra semantics of our equational specifications with a partial ordering on the set of sorts are studied.

Key words

Abstract data type, algebraic specification, initial algebra semantics, equation, subsort, partial ordering, overloading, error and exception handling.

1. Introduction

Algebraic specification of data types is regarded today as an instrument of great promise for program developement. The first papers in this field [LZ 74 , Gu 75] date from the middle seventies and a large amount of work has been done since then. The mathematical foundations of this specification method [ADJ 78 , Eh 79 , WPPDB 80 , Li 82] as well as precise concepts of implementation and parametrization [EKMP 80 , EKTWW 81 , SW 82] have been studied and led to the developement of specification languages.

But by conventional algebraic specifications, relationships between sorts cannot be expressed formally. For example it is impossible to point out clearly in a specification that the natural numbers are included in the integers, which again are included in the rationals. In our approach this can be done straightforward. The set of sorts is classified by a partial ordering, which expresses that one sort is a subsort of another sort. This concept implies that one function can occur more than one time in the signature with different domains and codomains or - for short - that functions are overloaded. The main results are theorems concerning initial models in the spirit of [ADJ 78] and a method to treat errors in abstract data types.

Martin Gogolla: Partially Ordered Sorts in Algebraic Specifications. Proc. 9th Colloquium on Trees in Algebra and Programming (CAAP), Bordeaux, Bruno Courcelle (Ed.), Cambridge University Press, Cambridge (1984), pp. 139-153.

The idea of structuring the set of sorts by a partial ordering can already be found in [Go 78], but there are some difficulties with that approach. For example the usual relationship between the concepts of morphism and congruence - in the sense that a morphism induces a congruence and vice versa - is not valid there. Apart from this, the basic notions of signature, algebra and congruence and the philosophy of error and exception handling are different here. In [Go 78] disjoint ok and error subsorts are wanted, whereas we only introduce an ok subsort representing the ok values [GDLE 82]. Similar problems of structuring the set of sorts are also discussed in [Gu 83 , Bi 83] and the concept of a declaration to structure the sorts is taken up in [Wa 82, GE 83] .

2. The basic idea

The conventional algebraic specification technique presents some circumstantialities if the data type to be specified includes substructures, so called subtypes or subsorts.

Example 2.1

Consider a conventional algebraic specification for the integers, which have the natural numbers as a subtype.

```
spec nat included in int - conventional version
srts nat , int
opns zero : ---> nat ;
     succ : nat ---> nat ;
     0 : ---> int ;
     _+1 : int ---> int ;
     _-1 : int ---> int ;
     mk : nat ---> int
vars n : nat ; i : int
eqns (i+1)-1 = (i-1)+1 = i ;
     mk(zero) = 0 ;
     mk(succ(n)) = mk(n)+1
ceps
```

The fact that the natural numbers are included in the integers is only implicit by the conversion function mk. There is no formal relation between the sorts nat and int. Another lack of the specification is the fact that a term denoting a natural number does not denote an integer, but every natural number is an integer number and therefore an adequate notation should be found. ***

The approach presented here allows to classify the set of sorts by a partial ordering, which expresses that one sort is a subsort of another sort. This concept has a deep influence on our notion of sig= nature : it is possible and sometimes necessary that one function has many domains and codomains or - for short - functions are overloaded :

- If a function is defined on certain sorts, the same function must be defined on all subsorts of the given ones.
- If a function has a certain value sort, then the same function must also be seen as a function having a supersort of the given one.

For example if nat < int is in the partial ordering and we have _-_ : int x int ---> int, then _-_ : nat x nat ---> int has to be in the signature too. And if 0 : ---> nat is a function, the the same will hold for 0 : ---> int. The exact definitions will be given in the following chapter.

Example 2.2

Let us now look at the example again and present our specification.

```
spec nat included in int - subsort version
srts nat , int with nat < int
opns 0 : ---> nat ;
     _+1 : nat ---> nat ;
     _+1 : int ---> int ;
     _-1 : int ---> int
vars i : int
eqns (i+1)-1 = (i-1)+1 = i
ceps
```

The ordering nat < int indicates that every term denoting a natural number also denotes an integer number. Therefore 0 is also a constant of sort int and _+1 : nat ---> nat can also be seen as a function from nat to int. But it is neccessary to declare that _+1 is a function from int to int, because not every function from nat to nat is also a function from int to int, for example the factorial function. The main point here is the fact that terms of the form $0(+1)^n$ can be recognised as natural numbers looking only at the signature.

It is also possible to define functions from subsorts into arbitrary sorts or from arbitrary sorts into subsorts. For example we can now specify the factorial function and the square

```
_! : nat ---> nat
_^2 : int ---> nat
```

by giving the equations

$0! = 0+1$

$(n+1)! = (n+1)*(n!)$

$i^2 = i*i$

using a variable n of sort nat and the multiplication _*_ : int x int ---> int . Please note that the equations are well typed. For example the left hand side of the last equation is of sort nat and therefore also of sort int and this matches the sort of the right side. ***

3. Signatures, algebras and morphisms

Our notions of signature, algebra and morphism must take into account that the same function name can be defined for different domains or codomains and that the same elements can occur in different carriers. But giving the same name to different functions implies that the functions behave in a similar way. Furthermore different occurences of identical elements in different carriers are not distinguished.

Definition 3.1

A signature is a triple $(S,\leq,\Sigma)$, where

(1) S is a set (of sorts) ,

(2) $\leq$ is a partial ordering of S and

(3) $\Sigma=\langle\Sigma_{w,s}\rangle_{w\in S^*,s\in S}$ is an S^*xS-indexed family of sets of operation symbols such that

(4) $w'\leq w$ and $s\leq s'$ implies $\Sigma_{w,s}\subseteq\Sigma_{w',s'}$.

- By a partial ordering we mean a reflexsive, transitive and antisymmetric relation. If $s\leq s'$, then we say that s is a subsort of s' and s' is a supersort of s. $s<s'$ means $s\leq s'$ and $s\neq s'$. If $w=s_1...s_n$ and $w'=s'_1...s'_n$, $w\leq w'$ means $s_i\leq s'_i$ for $i=1,...,n$.
- The sets $\Sigma_{w,s}$ are arbitrary, so the same operation symbol may occur in different sets even with a different number of arguments.
- Condition (4) means the following : If an operation symbol σ has arity $s_1...s_n$ and sort s, then σ will also be defined on all subsorts $s'_1...s'_n$ of the arity $s_1...s_n$ and σ will also be a function symbol having a supersort s' of sort s. For example if we have nat<int and _-_ : int x int ---> int, then _-_ : nat x nat ---> int will also be a part of the signature. On the other hand if 0 : ---> nat is in the signature, then the same will hold for 0 : ---> int.
- We will abbreviate $(S,\leq,\Sigma)$ by Σ and define $\tilde{\Sigma}=\{\sigma^{w,s}\mid w\in S^*,s\in S,\sigma\in\Sigma_{w,s}\}$.

Example 3.2

We give a signature describing some operations on integer numbers. Let S be the set {nat,int,bool} and nat<int be the partial ordering on S. The operation symbols are denoted in the usual way :

```
0 : ---> nat
0 : ---> int
_+1 : nat ---> nat
_+1 : nat ---> int
_+1 : int ---> int
_-1 : nat ---> int
_-1 : int ---> int
false , true : ---> bool
_≡_ : nat x nat ---> bool
_≡_ : nat x int ---> bool
_≡_ : int x nat ---> bool
_≡_ : int x int ---> bool
```

The given family of operations symbols satisfies condition (4) of our signature definition. ***

It is quite troublesome to repeat all operation symbols of the sig= nature with appropriate domains and codomains. A shorter notation for a signature would be useful and therefore we define what the com= pletion of an arbitrary family of operation symbols means.

Definition 3.3

Let $(S,\leq,\Sigma)$ be a triple consisting of a set S, a partial ordering $\leq$ on S and an arbitrary S^*xS-indexed family of sets Σ .
$comp(\Sigma) = \langle comp(\Sigma)_{w,s} \rangle_{w\in S^*,s\in S}$ with $comp(\Sigma)_{w,s} = \bigcup_{w\leq w',s'\leq s} \Sigma_{w',s'}$ is called the completion of Σ.

- For any arbitrary family Σ the completion $(S,\leq,comp(\Sigma))$ will be a signature. It will be convenient not to repeat the same function symbol in all sets. When we give an arbitrary family Σ and we speak of the signature Σ, we will allways mean the completion of Σ.

Example 3.4

Let S with $\leq$ be the same as in example 3.2 and let Σ be following family of operation symbols.

```
0 : ---> nat
_+1 : nat ---> nat
_+1 : int ---> int
_-1 : int ---> int
```

false , true : ---> bool

_≡_ : int x int ---> bool

The completion of the above family of operation symbols is identical to the signature given in example 3.2. It is also the least family such that its completion yields that signature. Note that _+1 : nat ---> nat reflects some important information about the successor operation +1 and that we do not have _-1 : nat ---> nat. ***

Definition 3.5

Let signature Σ be given. A Σ-algebra is a tuple (A,F), where

(1) $A=\langle A_s\rangle_{s\in S}$ is an S-indexed family of sets such that

(2) $s\leq s'$ implies $A_s \subseteq A_{s'}$ and

(3) $F=\langle \sigma_A^{w,s}\rangle_{\sigma w,s\in\tilde{\Sigma}}$ is a $\tilde{\Sigma}$-indexed family of functions, for every $\sigma^{w,s}\in\tilde{\Sigma}$ there is a function $\sigma_A^{w,s} : A_w \dashrightarrow A_s$ such that

(4) $\sigma\in\Sigma_{w,s}\cap\Sigma_{w',s'}$ and $a\in A_w\cap A_{w'}$ implies $\sigma_A^{w,s}(a)=\sigma_A^{w',s'}(a)$.

- If $w = s_1\ldots s_n$ is given , then A_w means the product $A_{s_1} \times \ldots \times A_{s_n}$.
- Clause (2) requires that if s is a subsort of s', then the corresponding carrier A_s will be included in $A_{s'}$. Thus for example the natural numbers will be included in the integer numbers.
- Condition (4) especially expresses : if $w=w'=\lambda$, then $\sigma_A^{w,s}=\sigma_A^{w',s'}$. It will also guarantee that we can speak of the function σ_A . Especially if $\sigma\in\Sigma_{w,s}$, $w'\leq w$ and $s\leq s'$, which of course implies $\sigma\in\Sigma_{w',s'}$, $A_{w'} \subseteq A_w$ and $A_s \subseteq A_{s'}$, then $\sigma_A^{w,s}|_{A_{w'}} = \sigma_A^{w',s'}$. This assures for example that the addition on integers is compatible with the addition on rational numbers.

  But even if the sorts are unrelated condition (4) must be valid. For example if the same constant symbol c occurs in $\Sigma_{\lambda,s}$ and $\Sigma_{\lambda,s'}$ and neither $s\leq s'$ nor $s'\leq s$ holds, then $c_A^{\lambda,s}=c_A^{\lambda,s'}$ must be true. We do not think it is natural in such a case that $c_A^{\lambda,s}$ and $c_A^{\lambda,s'}$ evaluate to different elements. If different evaluations are wanted, then there should be different names in the signature. Of course the same applies to functions. If f is in $\Sigma_{w,s}\cap\Sigma_{w',s'}$ and a is in $A_w\cap A_{w'}$, then $f_A^{w,s}(a)$ and $f_A^{w',s'}(a)$ have to be equal, because giving the same name to a function from A_w to A_s and to a function from $A_{w'}$ to $A_{s'}$, expresses that there are similarities between the functions. These similarities are the requirement that the functions behave in the same way when applied to the same arguments.
- We abbreviate (A,F) by A and define $\tilde{A} = \bigcup_{s\in S} A_s$.

Example 3.6

Recall the signature given in example 3.2. Let A_{nat} , A_{int} and A_{bool} be the set of natural numbers, integer numbers and truth values, respectively. Let 0, +1 and -1 be the obvious functions and let ≡ be the equality predicate on integers. Then the described entities form an algebra in the sense of definition 3.5. ***

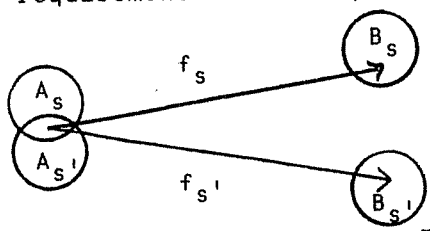
Definition 3.7

Let signature Σ and Σ-algebras A and B be given. A Σ-algebra morphism $f : A \dashrightarrow B$ is an S-indexed family of mappings $f=\langle f_s \rangle_{s \in S}$, each $f_s : A_s \dashrightarrow B_s$ such that

(1) if $\sigma^{w,s} \in \tilde{\Sigma}$ with $w = s_1 \ldots s_n$ and $a_i \in A_{s_i}$ for $i = 1,\ldots,n$, then
$f_s(\sigma_A^{w,s}(a_1,\ldots,a_n)) = \sigma_B^{w,s}(f_{s_1}(a_1),\ldots,f_{s_n}(a_n))$ and

(2) if $a \in A_s \cap A_{s'}$, then $f_s(a) = f_{s'}(a)$.

- Condition (1) is the usual requirement that morphisms respect the operational structure.
- Clause (2) reflects that we do not want to distinguish between equal elements in different carrier sets. It especially holds for the case $s \leq s'$, because then $A_s \subseteq A_{s'}$.
  If condition (2) would be dropped, then a situation like the one depicted in the figure below is possible. But a mapping like this does not reflect the structure of the algebra A in the algebra B, which is the usual requirement for a morphism from A to B.



- A morphism $f : A \dashrightarrow B$ induces a mapping $\tilde{f} : \tilde{A} \dashrightarrow \tilde{B}$ that satisfies $\tilde{f}(A_s) \subseteq B_s$ for $s \in S$.
- The category of all Σ-algebras with all morphisms between them is denoted by ALG_Σ .

Definition 3.8

Let signature Σ be given. The Σ-term algebra (T_Σ, F_Σ) is defined in the usual way.

$T_\Sigma = \langle T_s \rangle_{s \in S}$ is the least family of sets satisfying :

(1) $\Sigma_{\lambda,s} \subseteq T_s$ and

(2) $\sigma \in \Sigma_{w,s}$ with $w = s_1 \ldots s_n$ and $t_i \in T_{s_i}$ for $i = 1,\ldots,n$ implies
$\sigma(t_1,\ldots,t_n) \in T_s$.

$F_\Sigma = \langle \sigma_T^{w,s} \rangle_{\sigma^{w,s} \in \tilde{\Sigma}}$.

(3) $\sigma \in \Sigma_{\lambda,s}$ implies $\sigma_T^{\lambda,s} = \sigma$.

(4) $\sigma \in \Sigma_{w,s}$ with $w = s_1 \ldots s_n$ and $t_i \in T_{s_i}$ for $i = 1,\ldots,n$ implies $\sigma_T^{w,s}(t_1,\ldots,t_n) = \sigma(t_1,\ldots,t_n)$.

- T_Σ is a Σ-algebra satisfying the conditions of our definition.

  (1) Let $s \leq s'$ be given. We have to show $T_s \subseteq T_{s'}$. Assume $t \in T_s$. If depth(t)=1, then $t=\sigma \in \Sigma_{\lambda,s}$. This implies $\sigma \in \Sigma_{\lambda,s'}$ and so $\sigma = t \in T_{s'}$. If depth(t)=n+1, then $t=\sigma(t_1,\ldots,t_n)$, $\sigma \in \Sigma_{w,s}$ with $w=s_1 \ldots s_n$ and $t_i \in T_{s_i}$. This implies $\sigma \in \Sigma_{w,s'}$ and therefore $\sigma(t_1,\ldots,t_n) = t \in T_{s'}$.

  (2) Let $\sigma \in \Sigma_{w,s} \cap \Sigma_{w',s'}$ with $w=s_1 \ldots s_n$ and $w'=s_1' \ldots s_n'$ and $t_i \in T_{s_i} \cap T_{s_i'}$ be given. We have to show : $\sigma_T^{w,s}(t_1,\ldots,t_n) = \sigma_T^{w',s'}(t_1,\ldots,t_n)$. But $\sigma_T^{w,s}(t_1,\ldots,t_n) = \sigma(t_1,\ldots,t_n) = \sigma_T^{w',s'}(t_1,\ldots,t_n)$ and if $w=w'=\lambda$, then $\sigma_T^{\lambda,s} = \sigma = \sigma_T^{\lambda,s'}$.

- We defined the term algebra in the standard way, but used in our examples a mix-fix notation for operation symbols. This does not present any difficulties as long as terms are put in parenthesis unequivocally.

Example 3.9

The term algebra of the signature of example 3.2 can be described by the following context-free productions :

```
<nat>  ::= 0 | (<nat>)+1
<int>  ::= 0 | (<int>)+1 | (<int>)-1
<bool> ::= false | true | (<int>)≡(<int>)
```

Of course we have $L(\langle nat \rangle) \subseteq L(\langle int \rangle)$. Therefore terms of the form $0(+1)^n$ are of sort nat and of sort int and cannot be parsed uniquely.

***

Theorem 3.10

Let signature Σ and Σ-algebra A be given. Then T_Σ is initial in ALG_Σ , i.e. there is a unique morphism $f : T_\Sigma \dashrightarrow A$.

Proof

We define $f = \langle f_s \rangle_{s \in S}$ in the usual way according to the structure of the terms.

$$f_s(t) = \begin{cases} \sigma_A^{\lambda,s} & \text{if } t=\sigma \text{ and } \sigma \in \Sigma_{\lambda,s} \\ \sigma_A^{w,s}(f_{s_1}(t_1),\ldots,f_{s_n}(t_n)) & \text{if } t=\sigma(t_1,\ldots,t_n) \text{ , } \sigma \in \Sigma_{w,s} \text{ with } w=s_1 \ldots s_n \text{ and } t_i \in T_{s_i} \end{cases}$$

First we have to show that the definition is well-defined, because the choice of w is not unique. The following situation can occur :

$t = \sigma(t_1,\ldots,t_n)$, $\sigma\in\Sigma_{w,s}\cap\Sigma_{w',s}$ with $w = s_1\ldots s_n$, $w' = s'_1\ldots s'_n$ and $t_i \in T_{s_i}\cap T_{s'_i}$. We have to prove that $\sigma_A^{w',s}(f_{s'_1}(t_1),\ldots,f_{s'_n}(t_n)) = \sigma_A^{w,s}(f_{s_1}(t_1),\ldots,f_{s_n}(t_n))$ is valid. But this is part of the proof for condition (2) of our morphism definition :

Let $t\in T_s\cap T_{s'}$ be given. If depth(t)=1 , then $t=\sigma$ and $\sigma \in \Sigma_{\lambda,s}\cap\Sigma_{\lambda,s'}$ and $f_s(t)=f_s(\sigma)=\sigma_A^{\lambda,s}=\sigma_A^{\lambda,s'}=f_{s'}(\sigma)=f_{s'}(t)$ is valid according to our definition of algebra.

If depth(t)=n+1 , then $t=\sigma(t_1,\ldots,t_n)$, $\sigma\in\Sigma_{w,s}\cap\Sigma_{w',s'}$ with $w=s_1\ldots s_n$ and $w'=s'_1\ldots s'_n$ and $t_i\in T_{s_i}\cap T_{s'_i}$. According to the induction assumption $f_{s_i}(t_i)$ and $f_{s'_i}(t_i)$ are equal and therefore $f_{s_i}(t_i)=f_{s'_i}(t_i)\in A_{s_i}\cap A_{s'_i}$. Then the following equalities hold :

$$f_s(t) = f_s(\sigma(t_1,\ldots,t_n)) = \sigma_A^{w,s}(f_{s_1}(t_1),\ldots,f_{s_n}(t_n))$$

$$\overset{(*)}{=} \sigma_A^{w',s'}(f_{s'_1}(t_1),\ldots,f_{s'_n}(t_n)) = f_{s'}(\sigma(t_1,\ldots,t_n)) = f_{s'}(t) .$$

(*) holds due to our definition of algebra. The proof that f respects the operations and its uniqueness property is analogous to the conventional case [ADJ 78]. q.e.d.

Example 3.11

The unique morphism $f : T_\Sigma \dashrightarrow A$ from the term algebra of example 3.9 into the algebra A of example 3.6 evaluates terms to the corresponding natural numbers, integer numbers and truth values, respectively. ***

4. Equations

Definition 4.1

Let signature Σ and Σ-algebra A be given. An S-indexed, pairwise disjoint family of sets $V = \langle V_s \rangle_{s\in S}$ with each V_s disjoint from all $\Sigma_{w,s'}$ with $w\in S^*$ and $s'\in S$ denotes variables for Σ .

An assignment to (or interpretation of) the variables is an S-indexed family of mappings $I = \langle I_s \rangle_{s\in S}$, $I_s : V_s \dashrightarrow A_s$.

The extended signature $(S,\leq,\Sigma(V))$ is the completion of $\langle\Sigma'_{w,s}\rangle_{w\in S^*,s\in S}$ with $\Sigma'_{w,s}$ = if $w=\lambda$ then $\Sigma_{\lambda,s} \cup V_s$ else $\Sigma_{w,s}$ fi .

The result of applying the forgetful functor $U : ALG_{\Sigma(V)} \dashrightarrow ALG_\Sigma$ to $T_{\Sigma(V)}$ is denoted by $T_\Sigma(V)$.

Lemma 4.2

Let signature Σ, variables V, Σ-algebra A and assignment $I : V \dashrightarrow A$ be given. Then there is a unique Σ-algebra morphism $I^\# : T_\Sigma(V) \dashrightarrow A$ that extends I in the sense that $I(v)=I^\#(v)$ if $v\in V$.

Proof

The Σ-algebra A can be made into a $\Sigma(V)$-algebra A_V by defining $v_A=I(v)$.

$T_{\Sigma(V)}$ is initial in $ALG_{\Sigma(V)}$ and therefore there is a unique morphism $f_V : T_{\Sigma(V)} \dashrightarrow A_V$. Then $U(f_V) : T_\Sigma(V) \dashrightarrow A$ is the unique morphism with $U : ALG_{\Sigma(V)} \dashrightarrow ALG_\Sigma$ the corresponding forgetful functor. q.e.d.

Definition 4.3

Let signature Σ, Σ-algebra A and an S-indexed family of relations $\equiv = \langle \equiv_s \rangle_{s\in S}$, $\equiv_s \subseteq A_s \times A_s$ be given. Let $\tilde{\equiv}$ denote the equivalence on $\tilde{A}$ generated by $\bigcup_{s\in S} \equiv_s$.

The family of relations $\equiv = \langle \equiv_s \rangle_{s\in S}$ is called an equivalence on A, iff $\tilde{\equiv}|_{A_s \times A_s} = \equiv_s$.

An equivalence $\equiv = \langle \equiv_s \rangle_{s\in S}$ is called a congruence on A, iff for all $\sigma\in\Sigma_{w,s}\cap\Sigma_{w',s'}$ with $w=s_1\ldots s_n$, $w'=s'_1\ldots s'_n$ and $a_i\in A_{s_i}$, $a'_i\in A_{s'_i}$ with $a_i\tilde{\equiv}a'_i$ for $i=1,\ldots,n$ $\sigma_A^{w,s}(a_1,\ldots,a_n)\tilde{\equiv}\sigma_A^{w',s'}(a'_1,\ldots,a'_n)$ is valid.

The factorisation $A/\equiv$ of an algebra by a congruence is defined in the following way :

$A/\equiv \;= \langle A/\equiv_s \rangle_{s\in S}$.

$A/\equiv_s \;= \{[a]\,|\,a\in A_s\}$ with $[a]=\{a'\in\tilde{A}\,|\,a\tilde{\equiv}a'\}$.

$F/\equiv \;= \langle \sigma_{A/\equiv}^{w,s} \rangle_{\sigma^{w,s}\in\tilde{\Sigma}}$.

$\sigma_{A/\equiv}^{w,s}([a_1],\ldots,[a_n]) = [\sigma_A^{w,s}(a'_1,\ldots,a'_n)]$ with
$w=s_1\ldots s_n$, $[a_i]\in A/\equiv_{s_i}$, $a'_i\in A_{s_i}$ and $[a_i]=[a'_i]$.

- It is not sufficent to require each $\equiv_s$ to be an equivalence relation on $A_s \times A_s$. Consider the following example with sorts s and s' and operations

  a : ---> s a : ---> s'

  b : ---> s b : ---> s'

  Then it would be possible that a and b are equivalent under $\equiv_s$, but not under $\equiv_{s'}$: $a \equiv_s b$ and $a \not\equiv_{s'} b$. But a case like this should be excluded.
- Also a weaker definition of congruence in the sense that "$a_i \equiv_{s_i} a'_i$ implies $\sigma_A^{w,s}(a_1,\ldots,a_n) \equiv_s \sigma_A^{w,s}(a'_1,\ldots,a'_n)$ with $w=s_1\ldots s_n$" does not work as well. Consider the following example with sorts s, s', s_1 and s_2 and operations

  a , b : ---> s

  a : ---> s_1

  b : ---> s_2

  c , d : ---> s'

  f : s_1 ---> s'

  f : s_2 ---> s'

and the algebra which is depicted below and where f(a)=c and f(b)=d is valid.

$A_s = \{ a , b \}$

$A_{s_1} = \{ a \}$ $A_{s_2} = \{ b \}$

$A_{s'} = \{ f(a)=c , d=f(b) \}$

Under the above weaker definition a congruence could then consist only of $a \equiv_s b$, but this would not imply $f(a)=c \equiv_{s'} d=f(b)$, which should be the case for a congruence.

- The factorisation $A/\equiv$ of an algebra by a congruence is a Σ-algebra satisfying our definition.

(1) Let $s \leq s'$ be given. We have to show $A/\equiv_s \subseteq A/\equiv_{s'}$. If $[a] \in A/\equiv_s$, then there is an $a' \in A_s$ with $[a]=[a']$. But $a' \in A_s$ implies $a' \in A_{s'}$, because $s \leq s'$. Therefore $[a]=[a'] \in A/\equiv_{s'}$.

(2) Let $\sigma \in \Sigma_{w,s} \cap \Sigma_{w',s'}$ be given. If $w=w'=\lambda$, then $\sigma^{\lambda,s}_{A/\equiv} = [\sigma^{\lambda,s}_A] = [\sigma^{\lambda,s'}_A] = \sigma^{\lambda,s'}_{A/\equiv}$.

If $w=s_1 \ldots s_n$, $w'=s'_1 \ldots s'_n$ and $[b_i] \in A/\equiv_{s_i} \cap A/\equiv_{s'_i}$ for $i=1,\ldots,n$, then there are $a_i \in A_{s_i}$ and $a'_i \in A_{s'_i}$ with $[a_i] = [a'_i] = [b_i]$ for $i=1,\ldots,n$. Then the following equalities hold :

$$\sigma^{w,s}_{A/\equiv}([b_1],\ldots,[b_n]) = \sigma^{w,s}_{A/\equiv}([a_1],\ldots,[a_n]) =$$

$$[\sigma^{w,s}_A(a_1,\ldots,a_n)] \overset{(*)}{=} [\sigma^{w',s'}_A(a'_1,\ldots,a'_n)] =$$

$$\sigma^{w',s'}_{A/\equiv}([a'_1],\ldots,[a'_n]) = \sigma^{w',s'}_{A/\equiv}([b_1],\ldots,[b_n])$$

(*) is valid due to our definition of congruence.

- If an arbitrary family of relations on an algebra is given, then there is allways a least congruence containing the family. It is called the congruence generated by this family of relations. This is the same as in the conventional case [ADJ 78].

Lemma 4.4

Let signature Σ and Σ-algebras A and B be given.

(1) A morphism $f : A \dashrightarrow B$ induces a congruence $\equiv$ on A.

(2) A congruence $\equiv$ on A induces an epimorphism $f : A \dashrightarrow A/\equiv$.

Proof

(1) As usual we define $\equiv = \langle \equiv_s \rangle_{s \in S}$ by $a \equiv_s a'$ iff $f_s(a) =_s f(a')$. Assume $\equiv$ is not an equivalence on A. Then there is a sort s with $\equiv_s$ a proper subset of $\tilde{\equiv}|_{A_s \times A_s}$ and there are $a,a' \in A_s$ with $a \tilde{\equiv} a'$ and not $a \equiv_s a'$. But $\tilde{\equiv}$ is the equivalence generated by $\equiv$ and this implies $f_s(a)=f_s(a')$. Therefore $a \equiv_s a'$.

The congruence property is valid as well. Let $\sigma \in \Sigma_{w,s} \cap \Sigma_{w',s'}$ with $w = s_1 \ldots s_n$ and $w' = s'_1 \ldots s'_n$ and $a_i \in A_{s_i}$, $a'_i \in A_{s'_i}$ with $f_{s_i}(a_i) = f_{s'_i}(a'_i)$ for $i = 1, \ldots, n$ be given. Then the following equalities are valid :

$$f_s(\sigma_A^{w,s}(a_1, \ldots, a_n)) = \sigma_B^{w,s}(f_{s_1}(a_1), \ldots, f_{s_n}(a_n)) =$$

$$\sigma_B^{w,s}(f_{s'_1}(a'_1), \ldots, f_{s'_n}(a'_n)) = \sigma_B^{w',s'}(f_{s'_1}(a'_1), \ldots, f_{s'_n}(a'_n)) =$$

$$f_{s'}(\sigma_A^{w',s'}(a'_1, \ldots, a'_n)) .$$

This implies $\sigma_A^{w,s}(a_1, \ldots, a_n) \tilde{\equiv} \sigma_A^{w',s'}(a'_1, \ldots, a'_n)$.

(2) We define $f = \langle f_s \rangle_{s \in S}$ by $f_s(a) = [a]$ as usual.

f respects the operational structure :

$$f_s(\sigma_A^{w,s}(a_1, \ldots, a_n)) = [\sigma_A^{w,s}(a_1, \ldots, a_n)] = \sigma_{A/\equiv}^{w,s}([a_1], \ldots, [a_n]) =$$

$$\sigma_{A/\equiv}^{w,s}(f_{s_1}(a_1), \ldots, f_{s_n}(a_n))$$

Condition (2) of our morphism definition is valid as well : $a \in A_s \cap A_{s'}$ implies $[a] \in A/\equiv_s \cap A/\equiv_{s'}$. Therefore $f_s(a) = [a] = f_{s'}(a)$ is true.

Furthermore f is surjective : Let $[a] \in A/\equiv_s$ be given. Then there is an $a' \in A_s$ with $[a] = [a']$ and so $f_s(a') = [a'] = [a]$. q.e.d.

Definition 4.5

Let signature Σ, variables V, $T_\Sigma(V)$ and Σ-algebra A be given. An equation is a tuple $l = r$ with $l, r \in T_\Sigma(V)_s$. An equation $l = r$ is true in A, iff for all assignments $I : V \dashrightarrow A$ $I^\#(l) = I^\#(r)$.

Let E be a set of equations.

$E(T_\Sigma) = \{ \langle I^\#(l), I^\#(r) \rangle \mid l = r \in E \text{ and } I : V \dashrightarrow T_\Sigma \text{ assignment} \}$

$\equiv_E$ denotes the congruence generated by $E(T_\Sigma)$.

Example 4.6

We give equations for the signature of example 3.2 using variables i and j of sort int and n of sort nat.

$(i+1)-1 = i$

$(i-1)+1 = i$

$0 \equiv 0 = \text{true}$

$0 \equiv n+1 = \text{false}$

$n+1 \equiv 0 = \text{false}$

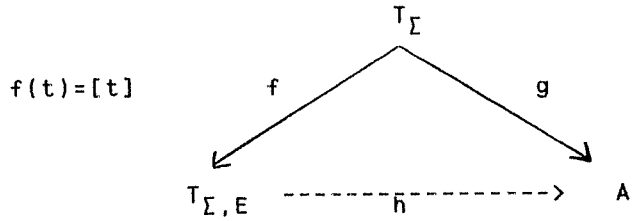
$i+1 \equiv j+1 = i \equiv j$ ***

Theorem 4.7

Let signature Σ, variables V and equations E be given and let $T_{\Sigma,E}$ denote the factorisation of T_Σ by $\equiv_E$. Then $T_{\Sigma,E}$ is initial in the category of all Σ-algebras satisfying E.

Proof

We know T_Σ is initial in the category of all Σ-algebras and therefore we have the following situation with unique morphisms f and g for a given Σ-algebra satisfying E.



We define h([t]) = g(t). h is independant of representatives, it respects the operations and is unique. The proof is analogous to the conventional case [ADJ 78]. So we only have to prove condition (2) of our morphism definition is valid.

Let $[t] \in T/\equiv_s \cap T/\equiv_{s'}$ be given. Then $h_s([t]) = g_s(t) = g_{s'}(t) = h_{s'}([t])$ is true. q.e.d.

Example 4.8

The quotient of the term algebra described in example 3.9 by the congruence $\equiv_E$ induced by the equations of example 4.6 is isomorphic to the algebra A of example 3.6 . Please note that the equality predicate on integers has been specified without hidden functions. This cannot be done by the conventional algebraic specification technique. ***

5. Error and exception handling

A quite useful application of our concepts is error and exception handling in abstract data types. One can introduce for each sort s a subsort s_{ok} representing only the ok values of the data type and classify the functions into those preserving ok values and those introducing errors when applied to ok elements.

Example 5.1

spec nat with error handling
srts nat, nat_{ok} with nat_{ok} < nat
opns 0 : ---> nat_{ok} ;
_+1 : nat_{ok} ---> nat_{ok} ;
_+1 : nat ---> nat ;
_-1 : nat ---> nat ;
error : ---> nat
vars n : nat_{ok}

eqns (n+1)-1 = n ;
0-1 = error ;
error+1 = error-1 = error

ceps

The specified data type consists of the natural numbers and exactly one error element. The main point here is that the variable n has the sort nat_{ok} and is actualized only by terms of the form $0(+1)^n$, which represent the ok values. 0 and +1 are functions preserving ok elements, -1 is an error introduction function and error of course an error constant. ***

The details of this approach can be found in [GDLE 82] , where a slightly different and easier notation is used. It is also shown there that this concept allows all forms of error handling : error introduction, error propagation and error recovery.

Acknowledgements

Thanks to Hans-Dieter Ehrich, Udo Lipeck, Axel Poigne and Klaus Drosten for fruitful discussions and constructive criticism.

References

ADJ 78 Goguen,J.A./Thatcher,J.W./Wagner,E.G. : An Initial Algebra Approach to the Specification, Correctness and Implementation of Abstract Data Types. Current Trends in Programming Methodology, Vol. IV (R.T.Yeh, ed.). Prentice Hall, Englewood Cliffs, 1978, pp. 80-149.

Bi 83 Bidoit,M.: Algebraic Specification of Exception Handling and Error Recovery in Abstract Data Types. 2nd Workshop on Theory and Applications of Abstract Data Types. Passau 1983.

Eh 79 Ehrich,H.-D.: On the Theory of Specification, Implementation and Parametrization of Abstract Data Types. Journal ACM, Vol.29, 1982, pp. 206 - 227.

EKMP 80 Ehrig,H./Kreowski,H.-J./Mahr,B./Padawitz,P.: Algebraic Implementation of Abstract Data Types. Theoretical Computer Science, Vol. 20, 1982, pp. 209-263.

EKTWW 81 Ehrig,H./Kreowski,H.-J./Thatcher,J.W./Wagner,E.G./Wright,J.B.: Parameter Passing in Algebraic Specification Languages. Proc. Workshop on Program Specification, Aarhus 1981, LNCS 134, Berlin 1982, pp. 322-369.

GDLE 82 Gogolla,M./Drosten,K./Lipeck,U./Ehrich,H.D.: Algebraic and Operational Semantics of Exceptions and Errors. Proceedings 6th GI-Conference Theoretical Computer Science, Dortmund 1983, LNCS 145, pp. 141-151. Also : Forschungsbericht Nr. 140, 1982, Abteilung Informatik, Universitaet Dortmund.

GE 83 Gogolla,M./Ehrich,H.D.: Algebraic Specification with Subsorts Using Declarations. Bulletin of the EATCS, Vol. 21, October 1983.

Go 78 Goguen,J.A.: Order Sorted Algebras : Exception and Error sorts, Coercions and Overloaded Operators. Semantics and Theory of Computation Report No. 14, University of California, Los Angeles, Dec. 1978.

Gu 83 Guiho,G.: Multioperator Algebras. 2nd Workshop on Theory and Applications of Abstract Data Types. Passau 1983.

Gu 75 Guttag,J.V.: The Specification and Application to Programming of Abstract Data Types. Techn. Report CSRG-59, Univ. of Toronto, 1975.

Li 82 Lipeck,U.: Ein algebraischer Kalkuel fuer einen strukturierten Entwurf von Datenabstraktionen (Dissertation). Forschungsbericht Nr. 148, 1982, Abteilung Informatik, Universitaet Dortmund.

LZ 74 Liskov,B./Zilles,S.: Programming with Abstract Data Types. SIGPLAN Notices Vol. 9, No. 4, April 1974, pp. 50-59.

SW 82 Sanella,D./Wirsing,M.: Implementation of Parametrized Specifications. Proc. 9th ICALP, LNCS 140, 1982, pp. 473-488.

Wa 82 Wadge,W.W.: Classified Algebras. University of Warwick, Theory of Computation, Report No. 46, October 1982.

WPPDB 80 Wirsing,M. / Pepper,P. / Partsch,H. / Dosch,W./ Broy,M. : On Hierachies of Abstract Data Types. Bericht TUM-I8007,Institut fuer Informatik, Technische Univ. Muenchen, Mai 1980.