ON PARAMETRIC ALGEBRAIC SPECIFICATIONS WITH CLEAN ERROR HANDLING

martin gogolla
Informatik B, TU Braunschweig
Postfach 3329, D-3300 Braunschweig

ABSTRACT
Usual algebraic specification techniques can be extended to treat partially ordered sorts. This allows the introduction of sub- and supersorts as well as overloaded operators, while pleasant features (e.g. existence of initial algebras and equivalence of algebraic and operational semantics) of the equational specification method are preserved. On this basis error and exception handling is studied. For each sort an ok and an error subsort is introduced and clean algebras (i.e. algebras which are ok/error-consistent and ok/error-complete) are considered. This new approach allows to prove an extension lemma for persistent parametric specifications which permit error handling.

1. INTRODUCTION

During the last years algebraic specifications proved to be a promising method for the specification of abstract data types in programming languages and software engineering. There are many approaches and philosophies for the algebraic semantics of such specifications. Among them are initial [ADJ 76, ADJ 81, EKMP 82, Kl 84], final [Wa 79, WPPDB 83, Ga 83] and observational semantics [GGM 76, ST 85]. Research in the field led to the development of specification languages like OBJ [FGJM 85], ACT ONE [EFH 83], ASL [SW 83] and many others.

Partially ordered sorts first introduced in [Go 78] have been treated in a series of papers [Go 83, Po 84, GM 84, GJM 85, etc.]. They are the basis for our approach to error and exception handling, a topic which is studied extensively in the literature

[ADJ 76, Go 77, Go 78, BGP 82, GDLE 82, Bi 84, Po 84, BBC 86, etc.]. The fundamental new notions introduced here are that of clean algebras and clean specifications, where clean refers to ok/error-consistency and ok/error-completeness. This approach allows the use of pure error variables, which was not possible before. In the literature only [Po 84] considers parametric specifications in connection with error handling, which is quite important because special problems arise here. [Po 84] works with non persistent specifications, whereas we carry over persistency to the exception handling case. By this we can apply the R-extension lemma of [Eh 81] and use it for our clean algebra approach, guaranteeing the well definedness of the application of parametric specifications.

The paper is organized as follows. Chapter 2 introduces the basic ideas by means of some examples. Chapter 3 reviews the fundamental definitions and facts concerning subsorts in algebraic specifications. Chapter 4 treats clean algebras and clean specifications. Chapter 5 discusses parametrization and our extension lemma. Chapter 6 gives some short concluding remarks. Due to space limitations all proofs are omitted.

2. THE BASIC IDEA

Our main new concept for error and exception handling is that of a clean algebra. This means that our algebras have two subsorts for the ok and error part of each sort and the carriers are ok/error-consistent (there is no element which is both ok and error) and ok/error-complete (every element is either ok or error). The approach is explained best by an example. Here is our specification of the natural numbers.

<u>spec</u> NaturalNumbersWithErrorHandling =

  <u>sorts</u> Nat

  <u>opns</u> 0 : -> Nat-Ok

        Succ : Nat-Ok -> Nat-Ok

        Error : -> Nat-Error

```
        Succ, Pred : Nat -> Nat
        Plus, Times : Nat Nat -> Nat
  vars  n:Nat n+,m+:Nat-Ok n-:Nat-Error
  eqns  Succ(n-) = n-
        Pred(0) = Error
        Pred(Succ(n+)) = n+
        Pred(n-) = n-
        Plus(0,n+) = n+
        Plus(Succ(n+),m+) = Succ(Plus(n+,m+))
        Plus(n-,n) = Plus(n,n-) = n-
        Times(0,n+) = 0
        Times(Succ(n+),m+) = Plus(Times(n+,m+),m+)
        Times(n-,n) = Times(n,n-) = n-
end spec
```

The semantics of the specification is an algebra having as carriers for Nat-Ok the natural numbers and for Nat-Error one distinguished error constant. There are some peculiarities in the specification above worth to be mentioned. (1) The sort Nat has implicitly the subsorts Nat-Ok and Nat-Error. (2) The function Succ is declared twice in the signature. The first occurrence assures that Succ yields an ok value when applied to such one. The second occurrence indicates that Succ may also be applied to all Nat values, but makes no statement about the nature of the result. (3) Three different kinds of variables corresponding to the three sorts and subsorts are used. (4) It is important to use an ok variable in the axiom Times(0,n+) = 0, otherwise this axiom would cause an error recovery. (5) The functions can be classified into constructors (line 1-3 of the opns-part) and derived functions (line 4-5 of the opns-part). (6) The error variable in Succ(n-) = n- assures error propagation for the function Succ. The use of pure error variables is essential for parametric specifications, as the next example shows.

```
spec ParametricBinaryTrees =
  parm sorts Entry
       opns  NoEntry : -> Entry-Error
  body sorts Tree
       opns  Leaf : Entry-Ok -> Tree-Ok
```

```
        Node : Tree-Ok Tree-Ok -> Tree-Ok
        NoTree : -> Tree-Error
        Leaf : Entry -> Tree
        Node : Tree Tree -> Tree
        GetEntry : Tree -> Entry
        GetRight, GetLeft : Tree -> Tree
   vars e+:Entry-Ok e-:Entry-Error t:Tree t1+,t2+:Tree-Ok
   eqns Leaf(e-) = NoTree
        Node(NoTree,t) = Node(t,NoTree) = NoTree
        GetEntry(Leaf(e+)) = e+
        GetEntry(Node(t1+,t2+)) = NoEntry
        GetEntry(NoTree) = NoEntry
        GetRight(Leaf(e+)) = GetLeft(Leaf(e+)) = NoTree
        GetRight(Node(t1+,t2+)) = GetLeft(Node(t2+,t1+)) = t2+
        GetRight(NoTree) = GetLeft(NoTree) = NoTree
end spec
```

The specification builds binary trees with given entries at the leaves when it is applied. The given parameter sort Entry persists in the resulting specification, especially because the function GetEntry is well defined. This can only be achieved by the use of the error variable e- in the axiom Leaf(e-) = NoTree. If one would specify only Leaf(NoEntry) = NoTree, then the construction would not be persistent for parameter algebras having more exceptions than the single error NoEntry. Again, lines 1-3 of the opns-part can be considered as the signature specification for the constructors and lines 4-7 for the derived functions. The ideas sketched above are now made precise in the following chapters.

3. REVIEW OF ALGEBRAIC SPECIFICATIONS WITH SUBSORTS

The following remarks review the fundamental definitions and facts and our notation concerning algebraic specifications and subsorts. Readers familar with [Go 78, Go 83, Po 84, GM 84, etc.] will find many common details.

3.1 Definition (Signature, Algebra, Morphism)

A signature $(S,\leq,\Sigma)$ consists of (1) a set S of sorts, (2) a partial order $\leq$ on S and (3) a family $\Sigma=\langle\Sigma_{w,s}\rangle_{w\in S^*,s\in S}$ of sets of function symbols such that (4) $\sigma:w\text{->}s$, $v\leq w$ and $r\geq s$ implies $\sigma:v\text{->}r$. $Name(\Sigma) = \{\sigma^{w,s}|\sigma\in\Sigma_{w,s}\}$ denotes the function names and $Symb(\Sigma) = \{\sigma|\sigma\in\Sigma_{w,s}\}$ the function symbols of Σ.

A Σ-algebra (A,F) consists of (1) a family $A=\langle A_s\rangle_{s\in S}$ of sets such that (2) $s\leq r$ implies $A_s\subseteq A_r$ and (3) a family $F=\langle\sigma_A^{w,s}\rangle_{\sigma^{w,s}\in Name(\Sigma)}$ of functions with $\sigma_A^{w,s}:A_w\text{->}A_s$ such that, (4) if $\sigma:w\text{->}s$, $\sigma:v\text{->}r$ and $a\in A_w\cap A_v$, then $\sigma_A^{w,s}(a)=\sigma_A^{v,r}(a)$.

A Σ-morphism $f:A\text{->}B$ between Σ-algebras A and B is a family $\langle f_s\rangle_{s\in S}$ of mappings such that (1) $f_s(\sigma_A^{w,s}(a))=\sigma_B^{w,s}(f_w(a))$ for $a\in A_w$ and (2) $a\in A_s\cap A_t$ implies $f_s(a)=f_t(a)$.

3.2 Definition (Term algebra)

The Σ-term algebra (T_Σ,F_Σ) has as carriers the least family $\langle T_s\rangle_{s\in S}$ of sets satisfying (1) $\sigma:\text{->}s$ implies $\sigma\in T_s$ and (2) σ : $s1...sn\text{->}s$ and $ti\in T_{si}$ implies $\sigma(t1...tn)\in T_s$ and the functions $\langle\sigma_T^{w,s}\rangle_{\sigma^{w,s}\in Name(\Sigma)}$ are determined by (3) $\sigma_T^{\lambda,s}:=\sigma$ for $\sigma:\text{->}s$ and (4) $\sigma_T^{s1...sn,s}(t1...tn):=\sigma(t1...tn)$ for $\sigma:s1...sn\text{->}s$ and $ti\in T_{si}$.

3.3 Fact (Initiality of the term algebra)

The Σ-term algebra T_Σ is initial in the category ALG_Σ of all Σ-algebras with all Σ-morphisms between them.

3.4 Definition (Congruence, Quotient)

A Σ-congruence $\equiv$ on a Σ-algebra A is a family $\langle\equiv_s\rangle_{s\in S}$ of relations $\equiv_s$ on A_s such that (1) $\equiv_s = [\equiv_{EQ} \cap A_s\times A_s]$ and (2) $ai \equiv_{EQ} bi$ implies $\sigma_A^{s1...sn,s}(a1...an) \equiv_{EQ} \sigma_A^{u1...un,r}(b1...bn)$ for $ai,bi \in A_{si}\cap A_{ui}$, $\sigma:s1...sn\text{->}s$ and $\sigma:u1...un\text{->}r$, where $\equiv_{EQ}$ is the equivalence on $\bigcup_{s\in S}A_s$ generated by $\equiv$.

The quotient $A/\equiv$ of a Σ-algebra A by a Σ-congruence $\equiv$ has (1) the carriers $A/\equiv_s = \{[a]|a\in A_s\}$, where $[a]=\{b\in\bigcup_{s\in S}A_s|a\equiv_{EQ}b\}$, and (2) the functions $\langle\sigma_{A/\equiv}^{w,s}\rangle_{\sigma^{w,s}\in Name(\Sigma)}$ with $\sigma_{A/\equiv}^{s1...sn,s}([a1]...[an]):=[\sigma_A^{s1...sn,s}(b1...bn)]$, where $[ai]\in A/\equiv_{si}$, $[ai]=[bi]$ and $bi\in A_{si}$.

3.5 Definition (Equation, Satisfaction, Specification)

A Σ-equation L=R is a pair of $\Sigma(V)$-terms, where $\Sigma(V)$ is the signature Σ having additionally the variables V as constants. A Σ-algebra A satisfies L=R, if all evaluations of L and R coincide. A specification (Σ,E) consists of a signature Σ and a set E of Σ-equations.

3.6 Fact (Induced Congruence)

A set of Σ-equations E induces uniquely a set of constant equations $E(T_\Sigma)$, which again induces a least congruence $\equiv_E$ on T_Σ containing $E(T_\Sigma)$.

3.7 Fact (Initiality of the quotient term algebra)

The quotient term algebra $T_\Sigma/\equiv_E$ is initial in the category $ALG_{\Sigma,E}$ of all (Σ,E)-algebras satisfying the equations E.

3.8 Example (Bitstrings avoiding error handling)

The following lines define bitstrings of arbitrary length (sort $String^*$) having as subsorts non empty bitstrings (sort $String^+$) and single bits (sort Bit).

```
spec BitStringsAvoidingErrorHandling =
  sorts Bit < String+ < String*
  opns  0,1 : -> Bit
        λ : -> String*
        .|. : String* String* -> String*
        .|. : Bit String* -> String+
        .|. : String* Bit -> String+
        First, Last : String+ -> Bit
  vars  b:Bit s,s1,s2,s3:String*
  eqns  s1|(s2|s3) = (s1|s2)|s3
        s|λ = λ|s = s
        First(b|s) = Last(s|b) = b
end spec
```

Please note that the specification part between the key words sorts and vars has not really to be a signature, but it uniquely determines a signature in the sense of our definition. Furthermore the functions First and Last returning the first respectively last bit are well defined, because all applications syntactically allowed by the signature either yield 0 or 1.

3.9 Remark (Declarations)
One can also use so called declarations in specifications [Go 78, Go 83]. A declaration consists of a term and a sort, assuring that the term will always evaluate to an element of the given sort (e.g. i*i:NonNegative, where i is a variable of sort int).

4. CLEAN SPECIFICATIONS

4.1 Definition (Clean, ok/error-consistent, ok/error-complete)
A signature $(S,\leq,\Sigma)$ is called (ok/error-)clean, if S=S-MAIN$\cup$ S-OK$\cup$S-ERROR, S-OK={s-Ok | s$\in$S-MAIN}, S-ERROR={s-Error | s$\in$S-MAIN} and $\leq$={s$\leq$s | s$\in$S}$\cup${s-Ok$\leq$s, s-Error$\leq$s | s$\in$S-MAIN}. A Σ-algebra A with Σ a clean signature is called (1) ok/error-consistent, if $A_{s\text{-}Ok} \cap A_{s\text{-}Error} = \emptyset$, (2) ok/error-complete, if $A_{s\text{-}Ok} \cup A_{s\text{-}Error} = A_s$, and (3) clean, if A is ok/error-consistent and ok/error-complete. A specification (Σ,E) is called clean, if the initial (Σ,E)-algebra is clean. A set E of equations is called clean, if $e\in E(T_\Sigma)$ implies either $e\in T_{s\text{-}Ok}\times T_{s\text{-}Ok}$ or $e\in T_{s\text{-}Error}\times T_{s\text{-}Error}$ for a suitable sort s. $ALG_{\Sigma,E,CLEAN}$ denotes the category of all clean (Σ,E)-algebras with all morphisms between them.

4.2 Characterisation (Specifications with clean term algebras)
Given a specification (Σ,E) with a clean term algebra T_Σ, then the specification (Σ,E) is clean, if and only if the set E of equations is clean.

4.3 Characterisation (Clean specifications)
A specification (Σ,E) is clean, if and only if
(1) $T_{\Sigma,E}$ is ok/error-consistent and
(2) there is a subspecification $(\Sigma G,EG)\subseteq(\Sigma,E)$ with
(a) ΣG containing all sorts and subsorts and all operations with ok or error result sorts and
(b) EG containing all clean equations of E and

(c) there is a unique surjective morphism $f: T_{\Sigma G,EG} \rightarrow U_{\Sigma \rightarrow \Sigma G}(T_{\Sigma,E})$.

4.4 Remark (Surjective morphism in (c) above)

If the morphism f is also injective, then (Σ,E) is an enrichment of $(\Sigma G,EG)$: $T_{\Sigma G,EG}$ and $U_{\Sigma \rightarrow \Sigma G}(T_{\Sigma,E})$ are isomorphic. If it is not injective, then there are terms t1 and t2 both ok or both error such that $T_{\Sigma G,EG} \models [t1] \neq [t2]$ and $T_{\Sigma,E} \models [t1]=[t2]$. But EG is a maximal set of equations applicable to ok and error terms, so the additional identification in $T_{\Sigma,E}$ is done via a term t3 neither ok nor error : $t3 \in T_s-(T_{s\text{-}Ok} \cup T_{s\text{-}Error})$, t1=t3 and t3=t2. This identification can also be done choosing different equations involving only ok or error terms. It is also much smoother to rule out this case from a methodological point of view and to establish a clear distinction between ok and error constructors and derived functions.

4.5 Concept (Pragmatics for clean specifications)

A clean specification (Σ,E) should have a subspecification $(\Sigma G,EG)$ with $T_{\Sigma G}$ and EG clean such that (Σ,E) is an enrichment of $(\Sigma G,EG)$.

4.6 Example (Bitstrings with error handling)

This clean specification defines bitstrings of arbitrary length. Errors are introduced by the functions Head and Tail when applied to the empty string.

```
spec BitStrings =
  sorts Bit, String
  cons  0,1 : -> Bit-Ok
        NoHead : -> Bit-Error
        λ : -> String-Ok
        .|. : String-Ok Bit-Ok -> String-Ok
        NoTail : -> String-Error
  funcs .|. : String Bit -> String
        Head : String -> Bit
        Tail : String -> String
  vars  s:String s+:String-Ok b:Bit b+,b1+,b2+:Bit-Ok
  eqns  NoTail|b = s|NoHead = NoTail
```

Head(s+|b1+|b2+) = Head(s+|b1+)

Head(λ|b+) = b+

Head(λ) = Head(NoTail) = NoHead

Tail(s+|b1+|b2+) = Tail(s+|b1+)|b2+

Tail(λ|b+) = λ

Tail(λ) = Tail(NoTail) = NoTail

end spec

The parts for the ok and error constructors and for the derived functions are indicated by the keywords cons and funcs. In general there will be an equation part for the constructors as well. For the subsorts the following equations hold : $T_{\Sigma,E,Bit-Ok} \cong \{0,1\}$, $T_{\Sigma,E,Bit-Error} \cong \{NoHead\}$, $T_{\Sigma,E,String-Ok} \cong (\{0,1\})^*$ and $T_{\Sigma,E,String-Error} \cong \{NoTail\}$. On this basis the functions Head and Tail are defined such that the subsorts are respected.

5. CLEAN PARAMETRIC SPECIFICATIONS

5.1 Definition (Signature morphism, specification morphism)

A signature morphism $f:\Sigma 1 \to \Sigma 2$ between signatures $(S1,\leq_{S1},\Sigma 1)$ and $(S2,\leq_{S2},\Sigma 2)$ consists of mappings $f:S1 \to S2$ and $f:Symb(\Sigma 1) \to Symb(\Sigma 2)$ such that $s \leq_{S1} r$ implies $f(s) \leq_{S2} f(r)$ and $\sigma \in \Sigma 1_{w,s}$ implies $f(\sigma) \in \Sigma 2_{f(w),f(s)}$. A signature morphism f is called strict, if $s <_{S1} r$ implies $f(s) <_{S2} f(r)$. A signature morphism f induces a forgetful functor $U_f:ALG_{\Sigma 2} \to ALG_{\Sigma 1}$. A signature morphism f is called specification morphism from (Σ1,E1) to (Σ2,E2), if every equation of E1, when translated by f, belongs to E2 : $f(E1) \subseteq E2$. A specification morphism is called simple, if $S1 \subseteq S2$, $Symb(\Sigma 1) \subseteq Symb(\Sigma 2)$ and $f:S1 \to S2$ and $f:Symb(\Sigma 1) \to Symb(\Sigma 2)$ are inclusions.

5.2 Definition (Parametric specification, persistent)

A parametric specification consists of a parameter specification (ΣP,EP) and a body specification (ΣB,EB) such that $\Sigma P \subseteq \Sigma B$ and $EP \subseteq EB$. The semantics of a parametric specification is the free construction $F:ALG_{\Sigma P,EP} \to ALG_{\Sigma B,EB}$ [ADJ 78, Po 84]. A parametric specification is called persistent, if A and U(F(A)) are "naturally" [WE 85] isomorphic for all (ΣP,EP)-algebras A, where U is the forgetful functor $U: ALG_{\Sigma B} \to ALG_{\Sigma P}$ induced by the signatures ΣP and ΣB.

5.3 <u>Definition</u> (Application of a parametric specification)
The result of <u>applying</u> a parametric specification with parameter (ΣP,EP) and body (ΣB,EB) to an actual specification (ΣA,EA) by means of a specification morphism $h:(\Sigma P,EP) \to (\Sigma A,EA)$ is the specification (ΣR,ER), where $\Sigma R=\Sigma A+hR(\Sigma B)$, $ER=EA+hR(EB)$, $hR(s)$ = IF $s \in SP$ THEN $h(s)$ ELSE s FI and $hR(\sigma)$ = IF $\sigma \in Symb(\Sigma P)$ THEN $h(\sigma)$ ELSE σ FI.

$$\begin{array}{ccc} (\Sigma P,EP) & \xrightarrow{s} & (\Sigma B,EB) \\ h \downarrow & & \downarrow hR \\ (\Sigma A,EA) & \xrightarrow[sR]{} & (\Sigma R,ER) \end{array}$$

The result specification is the pushout of the actual specification (ΣA,EA) and the body specification (ΣB,EB) with respect to the parameter (ΣP,EP) and the specification morphisms h and s, where s is the simple specification morphism induced by the inclusion of the parameter in the body.

5.4 <u>Definition</u> (Clean parametric specification)
A parametric specification with parameter (ΣP,EP) and body (ΣB,EB) is called <u>clean</u>, if the signatures ΣP and ΣB are clean, the free construction F is persistent on $ALG_{\Sigma P,EP,CLEAN}$ and the free construction F preserves cleanness : $A \in ALG_{\Sigma P,EP,CLEAN}$ implies $F(A) \in ALG_{\Sigma B,EB,CLEAN}$.

5.5 <u>Extension Lemma</u> (for clean parametric specifications)
Let there be given a clean parametric specification with parameter (ΣP,EP) and body (ΣB,EB), an actual clean specification (ΣA,EA), a strict specification morphism $h:(\Sigma P,EP) \to (\Sigma A,EA)$ and the result specification (ΣR,ER) as defined above.

(1) The resulting parametric specification with parameter (ΣA,EA) and body (ΣR,ER) is clean: FR is persistent on $ALG_{\Sigma A,EA,CLEAN}$ and it preserves cleanness.

(2) $F \circ U_h = U_{hR} \circ FR$.

$$\begin{array}{ccc} ALG_{\Sigma P,EP,CLEAN} & \xrightarrow{F} & ALG_{\Sigma B,EB,CLEAN} \\ U_h \uparrow & & \uparrow U_{hR} \\ ALG_{\Sigma A,EA,CLEAN} & \xrightarrow[FR]{} & ALG_{\Sigma R,ER,CLEAN} \end{array}$$

5.6 Remark (concerning the extension lemma)

The proof of our extension lemma applies the R-extension lemma of [Eh 81]. The restriction of $ALG_{\Sigma P,EP}$ to clean algebras can be expressed as predicate formula requirements. This restriction to clean algebras is essential for the underlying specification method, because one does not want to care about elements being neither ok nor error. The strictness of the parameter passing morphism h implies that ok or error operations of the formal parameter will also be ok or error operations in the actual parameter.

5.7 Concept (Pragmatics for clean parametric specifications)

Analogously to the case without parameters a clear distinction between ok and error constructors and derived functions should be established. Therefore a clean parametric specification with parameter $(\Sigma P,EP)$ and body $(\Sigma B,EB)$ should have a subspecification $(\Sigma P,EP) \subseteq (\Sigma G,EG) \subseteq (\Sigma B,EB)$ with $T_{\Sigma G}(A)$ and EG clean such that G is persistent on $ALG_{\Sigma P,EP,CLEAN}$ and F(A) is an enrichment of G(A) for all $A \in ALG_{\Sigma P,EP,CLEAN}$, where G is the free construction induced by the parametric specification with parameter $(\Sigma P,EP)$ and body $(\Sigma G,EG)$.

5.8 Example (Parametric strings with error handling)

This clean parametric specification defines strings over an arbitrary parameter sort Char. Again errors are introduced by the functions Head and Tail when applied to the empty string.

```
spec ParametricStrings =
  parm sorts Char
       opns  NoHead : -> Char-Error
```

```
  body sorts String
       cons  λ : -> String-Ok
             .|. : String-Ok Char-Ok -> String-Ok
             NoTail : -> String-Error
       funcs .|. : String Char -> String
             Head : String -> Char
             Tail : String -> String
       vars  s:String s+:String-Ok
             c:Char c+,c1+,c2+:Char-Ok c-:Char-Error
       eqns  NoTail|c = s|c- = NoTail
             Head(s+|c1+|c2+) = Head(s+|c1+)
             Head(λ|c+) = c+
             Head(λ) = Head(NoTail) = NoHead
             Tail(s+|c1+|c2+) = Tail(s+|c1+)|c2+
             Tail(λ|c+) = λ
             Tail(λ) = Tail(NoTail) = NoTail
end spec
```

The parts for the parameter and the body are indicated by the keywords parm and body. In general there will be an equation part for the parameter and the constructors as well. Please note that it is essential for persistency to use the variable c- of sort Char-Error in the equation s|c- = NoTail. If a clean parameter algebra A with sets $A_{Char-Ok}$ and $A_{Char-Error}$ is given, then the resulting algebra F(A) will have the following carriers : $F(A)_{Char-Ok} \cong A_{Char-Ok}$, $F(A)_{Char-Error} \cong A_{Char-Error}$, $F(A)_{String-Ok} \cong (A_{Char-Ok})^*$ and $F(A)_{String-Error} \cong$ {NoTail}. Furthermore the corresponding free construction is not persistent on $ALG_{\Sigma P,EP}$, if no restriction to clean algebras is made.

5.9 Remark (Pointed algebras and specifications)

All considerations presented here can be specialized to pointed algebras [Go 86], where there is only one error element for each sort. In this case error recovery is not supported too well, but especially error propagation can be done automatically.

6. CONCLUSION

The notion of a clean algebra is just a special case of an algebra satisfying certain sort equations which especially make sense in the context of partially ordered sorts and which can be considered as another construct for algebraic specification languages. For example in clean algebras the sort equations

$$s\text{-Ok} \cap s\text{-Error} = \emptyset \text{ and}$$

$$s\text{-Ok} \cup s\text{-Error} = s$$

are valid for all sorts s. A sort equation consists of a pair of sort terms built over the given set of sorts and set operations like union, intersection, difference, complement and empty set. An algebra satisfies a sort equation, if the set theoretic evaluations of the two expressions with respect to the given algebra coincide. This topic is subject to future research.

7. REFERENCES

ADJ 76 J.A.Goguen/J.W.Thatcher/E.G.Wagner : 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.

ADJ 78 J.W.Thatcher/E.G.Wagner/J.B.Wright : Data type specification : Parametrization and the power of specification techniques. Proc. 10th STOC, 1978, San Diego.

ADJ 81 H.Ehrig / H.-J.Kreowski / J.W.Thatcher / E.G.Wagner / J.B.Wright : Parameter passing in algebraic specification languages. LNCS 134, Berlin 1982, pp.322-369.

BBC 86 G.Bernot / M.Bidoit / C.Choppy : Abstract data types with exception handling : An initial approach based on a distinction between exceptions and errors. To appear.

Bi 84 M.Bidoit : Algebraic specification of exception handling and error recovery by means of equations and declarations. Proc. 11th ICALP 1984, LNCS 172, pp.95-109.

BGP 82 F.Boisson / G.Guiho / D.Pavot : Multioperator algebras. L.R.I. Report, Orsay 1982.

Eh 81 H.Ehrig : Algebraic theory of parametrized specifications with requirements. Proc. 6th CAAP 1981, Genova.

EFH 83 H.Ehrig/W.Fey/H.Hansen : ACT ONE : An algebraic specification language with two levels of semantics. Techn. Report No. 83-03, TU Berlin, 1983.

EKMP 82 H.Ehrig/H.-J.Kreowski/B.Mahr/P.Padawitz : Algebraic implementation of abstract data types. TCS, Vol.20 1982.

FGJM 85 K.Futasugi / J.A.Goguen / J.-P.Jouannaud / J.Meseguer : Principles of OBJ2. Proc. POPL 1985, pp.52-66.

Ga 83 H.Ganzinger : Parametrized specification : Parameter passing and implementation. ACM TOPLAS, Vol.5 1983.

GGM 76 V.Giarratana/F.Gimona/U.Montanari : Observability concepts in abstract data type specification. Proc. 5th MFCS 1976, Gdansk, LNCS 45.

GDLE 82 M.Gogolla/K.Drosten/U.Lipeck/H.-D.Ehrich : Algebraic and operational semantics of specifications allowing exceptions and errors. TCS, Vol.34 1984, pp.289-313.

Go 83 M.Gogolla : Partially ordered sorts in algebraic specifications. Proc. 9th CAAP 1984, Bordeaux, B. Courcelle (ed), Cambridge University Press, pp.139-153.

Go 86 M.Gogolla : über partiell geordnete Sortenmengen und deren Anwendung zur Fehlerbehandlung in Abstrakten Datentypen. Dissertation, TU Braunschweig, 1986.

Go 77 J.A.Goguen : Abstract errors for abstract data types. Proc. Conference on Formal Description of Programming Concepts 1978, E.J.Neuhold (ed), North Holland.

Go 78 J.A.Goguen : Order sorted algebras : Exception and error sorts, coercions and overloaded operators. Semantics and Theory of Computation Report No.14, UCLA, 1978.

GM 84 J.A.Goguen/J.Meseguer : Order-sorted algebra I : Partial and overloaded operators, errors and inheritance. Technical Report, SRI International, 1984.

GJM 85 J.A.Goguen/J.-P.Jouannaud/J.Meseguer : Operational semantics for order-sorted algebra. Proc. 12th ICALP 1985.

Kl 84 H.Klaeren : A constructive method for abstract algebraic software specification. TCS, Vol.30, No.2 1984.

Po 84 A.Poigne : Modularization techniques for algebraic specifications with subsorts. Imperial College, London.

ST 85 D.Sannella/A.Tarlecki : On observational equivalence and algebraic specification. Proc. 10th CAAP 1985, Berlin.

SW 83 D.Sannella/M.Wirsing : A kernel language for algebraic specification and implementation. Proc. FCT 1983.

Wa 79 M.Wand : Final algebra semantics and data type extensions. JCSS, Vol.19, No.1 1979, pp.27-44.

WE 85 E.G.Wagner/H.Ehrig : Canonical constraints for parametrized data types. Research Report RC 11248, IBM, 1985.

WPPDB 83 M.Wirsing/P.Pepper/H.Partsch/W.Dosch/M.Broy : On hierarchies of abstract data types. Acta Informatica 1983.