Theoretical Computer Science 34 (1984) 289-313
North-Holland

ALGEBRAIC AND OPERATIONAL SEMANTICS OF SPECIFICATIONS ALLOWING EXCEPTIONS AND ERRORS

M. GOGOLLA, K. DROSTEN, U. LIPECK and H.-D. EHRICH
Fachbereich Informatik B, Technical University of Braunschweig, D-3300 Braunschweig, Fed. Rep. Germany

Communicated by R. Milner
Received September 1982
Revised March 1984

Abstract. The specification of abstract data types requires the possibility to treat exceptions and errors. We present an approach allowing all forms of error handling: error introduction, error propagation and error recovery. The algebraic semantics of our method and a new correctness criterion are given. We also introduce an operational semantics of a subclass of our specifications which coincides with the algebraic semantics.

Key words. Specification of abstract data types, error and exception handling, algebraic semantics, correctness of specifications, operational semantics.

1. Introduction

Abstract data types offer promising tools for the specification and implementation of programs. Research in this field has been initiated by Guttag [9] and Liskov and Zilles [11]. Pleasant features of the method are that it is well-founded algebraically [1, 3, 4, 16] and operationally [14, 10, 13, 15] and that it is a sound basis for specification languages.

The problem of handling exceptions and errors in abstract data types has been studied in [6, 1, 7, 8, 12, 2] and an operational treatment has been given in [5].

We here modify the approach of Engels et al. [5] and study the algebraic and operational semantics of specifications allowing error and exception handling. We distinguish syntactically between error introducing and normal functions and allow two different types of variables for the same sort. Thus all forms of error and exception handling, i.e., error introduction, error propagation and error recovery, may be treated. We avoid the strict propagation of errors as in [1, 7, 12], the transformation of axioms via new operations as in [1] and the introduction of a semi-lattice structure on the set of sorts as in [8].

Section 2 gives an informal introduction to our method. In Sections 3 and 4 we present the algebraic semantics of our specifications. Section 5 gives a new correctness criterion and Section 6 introduces the operational semantics of a subclass of our specifications and shows that it coincides with the algebraic semantics. Appendices A and B offer more examples.

2. The basic idea

The *natural numbers* are an example of a simple data type, which needs error and exception handling. We are going to use the natural numbers in different versions throughout the paper.

Example 2.1. To specify the natural numbers we need the following functions:

$$\begin{aligned}&0: \to \text{nat}\\&\text{pred}, \text{succ}: \text{nat} \to \text{nat}\\&\text{plus}, \text{times}: \text{nat} \times \text{nat} \to \text{nat}.\end{aligned}$$

The axioms are given by

$$\text{pred}(\text{succ}(n)) = n \tag{A1}$$
$$\text{pred}(0) = \text{error} \tag{A2}$$
$$\text{plus}(0, n) = n \tag{A3}$$
$$\text{plus}(\text{succ}(n), m) = \text{succ}(\text{plus}(n, m)) \tag{A4}$$
$$\text{times}(0, n) = 0 \tag{A5}$$
$$\text{times}(\text{succ}(n), m) = \text{plus}(m, \text{times}(n, m)). \tag{A6}$$

But what is the *semantics* of an axiom like pred(0) = error? If we treat 'error' as an extra constant in nat, we will find times(0, pred(0)) = times(0, error) = 0 and so the introduced error has been forgotten by axiom (A5). This might suggest the idea that *errors should propagate* and so we could add the following axioms:

$$\text{succ}(\text{error}) = \text{error} \tag{E1}$$
$$\text{pred}(\text{error}) = \text{error} \tag{E2}$$
$$\text{plus}(\text{error}, n) = \text{error} \tag{E3}$$
$$\text{plus}(n, \text{error}) = \text{error} \tag{E4}$$
$$\text{times}(\text{error}, n) = \text{error} \tag{E5}$$
$$\text{times}(n, \text{error}) = \text{error}. \tag{E6}$$

But unfortunately we did not really specify what we had intended, because unwanted *contradictions* occur. 0 = times(0, error) = error holds due to equations (A5) and (E6) and so $\text{succ}^n(0) = \text{error}$ for every $n \in N_0$ due to (E1). (This critical problem was first pointed out in [1].) There are other reasons which do not support the idea of strict error propagation. For example, consider a straightforward specification of the factorial function on the natural numbers:

$$\text{fac}(n) = \text{if}(\text{eq}(n, 0), \text{succ}(0), \text{times}(n, \text{fac}(\text{pred}(n)))).$$

With the error propagation idea in mind we will find

$$\begin{aligned}\text{fac}(0) &= \text{if}(\text{eq}(0,0), \text{succ}(0), \text{times}(0, \text{fac}(\text{pred}(0))))\\ &= \text{if}(\text{true}, \text{succ}(0), \text{times}(0, \text{fac}(\text{error})))\\ &= \text{if}(\text{true}, \text{succ}(0), \text{times}(0, \text{error}))\\ &= \text{if}(\text{true}, \text{succ}(0), \text{error}) = \text{error}.\end{aligned}$$

We present an approach which allows all forms of error handling, i.e. error introduction, error propagation and error recovery, to be treated in an easy way and which avoids difficulties like the above. Our *main instruments* are:
- the *partition of the carrier sets* into a normal and an error part,
- the *syntactical classification of functions* into those which introduce errors in normal situations and those which preserve ok states, and
- the introduction of *two types of variables*. The first type will serve for non error situations only, the other for ok and exceptional states as well.

Example 2.2. We give a specification for the intended natural number algebra including an error constant.

$0: \to \text{nat}$
$\text{succ}: \text{nat} \to \text{nat}$
$\text{pred}: \text{nat} \to \text{nat}$: *unsafe*
$\text{plus}, \text{times}: \text{nat} \times \text{nat} \to \text{nat}$
$\text{error}: \to \text{nat}$: *unsafe.*

Functions, which might introduce errors and error constants, *unsafe functions* as we call them, are indicated in the signature by "*:unsafe*". The other functions are called *ok functions.*

The signature gives a *classification for all terms t.* If an unsafe function occurs in *t*, it is not known whether an error might be introduced or not and so *t* is viewed as a possible error and called *unsafe term.* If only ok functions occur in *t*, we know *t* corresponds to a natural number. In this case, *t* is called an *ok term.*

We mark pred as an unsafe function, because it introduces an error when applied to the ok value 0. For the functions succ, plus and times we know that they will return ok values when they are applied to such ones.

The *ok part* of the intended carrier set are terms of the form $\text{succ}^n(0)$ with $n \in N_0$ corresponding to the natural numbers. The *error part* of the carrier set are terms in which the function symbol 'error' occurs. These terms can be seen as error messages informing about illegal application of pred to 0.

We now use *ok variables n and m* which means they serve for non error situations only, i.e., only for ok terms. The axioms are exactly (A1)–(A6) of Example 2.1, but there are semantical differences. It is not allowed to substitute for example the term 'error' for variable *n* in axiom (A5) and therefore the difficulties described in Example 2.1 do not occur.

Now the following identities hold:

- pred(succ(succ(pred(succ(0))))) = pred(succ(succ(0))) = succ(0).
  Please note it is not allowed to substitute succ(pred(succ(0))) into the ok variable *n* in axiom (A1), but we may substitute the semantically equivalent term succ(0). This will be clarified later.
- times(0, pred(0)) = times(0, error).
  No further simplifications can be made on the last term, because axiom (A5) cannot be applied. The term can be seen as an error message within its environment.

3. Algebras with ok predicates

In this and the following two sections we show how the results of Goguen et al. [1] carry over to our notion of algebra. The syntax of our many-sorted algebras with ok predicates is defined via a signature, which gives names corresponding to the sorts of the carrier sets and to the operations on these sets. Our carrier sets are not homogeneous, because we want to distinguish between normal situations and exceptions. For this reason the signature for an algebra with ok predicates identifies those function symbols which correspond to operations introducing errors in normal situations. Therefore, there are ok predicates on function symbols and ok predicates on each sort.

Definition 3.1. A *signature* (with ok predicates) is a quintuple (S, Σ, arity, sort, ok_Σ), where

(a) S is a set (of sorts), Σ is a set (of function symbols) and

(b) arity, sort and ok_Σ are the following mappings:

$$\begin{aligned} \text{arity}&: \Sigma \to S^*,\\ \text{sort}&: \Sigma \to S,\\ \mathrm{ok}_\Sigma&: \Sigma \to \text{BOOL}. \end{aligned}$$

- A signature (S, Σ, arity, sort, ok_Σ) will often be denoted by Σ only.
- Given a function symbol σ, arity$(\sigma) = s1 \dots sn$ denotes the sorts of the arguments, sort$(\sigma) = s$ gives the result sort. This is written as $\sigma: s1 \times \cdots \times sn \to s$.
- $\mathrm{ok}_\Sigma(\sigma) = \text{TRUE}$ means σ is an *ok function symbol*, while $\mathrm{ok}_\Sigma(\sigma) = \text{FALSE}$ indicates an *unsafe function symbol*. The notion of ok and unsafe functions will be made clear by the following definition.

Definition 3.2. Let signature Σ be given. A Σ-*algebra* (with ok predicates) is a triple (A, F, ok_A), where

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

(b) $F = \langle \sigma_A \rangle_{\sigma \in \Sigma}$ is a Σ-indexed family of functions, for every function symbol $\sigma: s1 \times \cdots \times sn \to s$ we have a function $\sigma_A: A_{s1} \times \cdots \times A_{sn} \to A_s$, and

(c) $\mathrm{ok}_A = \langle \mathrm{ok}_s \rangle_{s\in S}$ is an S-indexed family of predicates, $\mathrm{ok}_s : A_s \to \text{Bool}$, such that

(d) for every function symbol $\sigma : s1 \times \cdots \times sn \to s$ with $\mathrm{ok}_\Sigma(\sigma) = \text{True}$ and $(a1, \ldots, an) \in A_{s1} \times \cdots \times A_{sn}$ with $\mathrm{ok}_{si}(ai) = \text{True}$ for $i = 1, \ldots, n$, we have $\mathrm{ok}_s(\sigma_A(a1, \ldots, an)) = \text{True}$.

- Parts (a) and (b) correspond to carrier sets and functions of the conventional notion of algebras without ok predicates.
- A Σ-algebra (A, F, ok_A) will often be denoted by A only. Whenever no ambiguity arises, we will omit indices of the predicates: for a function symbol σ, $\mathrm{ok}(\sigma)$ means $\mathrm{ok}_\Sigma(\sigma)$ and, for $a \in A_s$, $\mathrm{ok}(a)$ means $\mathrm{ok}_s(a)$. If $\mathrm{ok}(\sigma) = \text{True}$ holds, we call σ_A an *ok function*, otherwise an *unsafe function*.
- For $a \in A_s$, $\mathrm{ok}_s(a) = \text{True}$ will mean a is an *ok element* of A_s, otherwise it is called *error element*. The ok predicates split the carrier sets into an ok part A_{ok} and an error part A_{err}:

$$A_{\mathrm{ok}} = \langle A_{\mathrm{ok},s} \rangle_{s\in S}, \qquad A_{\mathrm{ok},s} = \{a \in A_s \mid \mathrm{ok}_s(a) = \text{True}\},$$

$$A_{\mathrm{err}} = \langle A_{\mathrm{err},s} \rangle_{s\in S}, \qquad A_{\mathrm{err},s} = \{a \in A_s \mid \mathrm{ok}_s(a) = \text{False}\}.$$

  For the application we have in mind the ok elements correspond to normal situations, while the error elements indicate exceptional states.
- Part (d) of the definition requires that *ok functions yield ok values for ok arguments*, or in other words, only unsafe functions may introduce errors when applied to normal situations. Because we have to treat these unsafe functions carefully, we distinguish them syntactically from ok functions. This guarantees that whenever an expression consisting only of ok functions is applied to ok arguments, it will result in an ok element. For expressions including unsafe functions this is not known.
- Every algebra with ok predicates can also be interpreted as a conventional *algebra without ok predicates* by just omitting the predicates. On the other hand, every conventional algebra without ok predicates can be made into an algebra with ok predicates by demanding all functions to be ok functions and all elements to be ok elements. The same holds for signatures.
- The notion of *partial algebra* may be embedded into the notion of algebra with ok predicates by making all partially defined operations into unsafe functions, completing them using bottom elements $\bot_s$ for every $s \in S$ and then interpreting these elements as the only error elements in the algebra.

Example 3.3. We describe the *natural numbers* together with an extra error element, which is introduced because we want to apply the predecessor function to 0. Let $S = \{\text{bool}, \text{nat}\}$ be the set of sorts. (The sort 'bool' used here is of course different from the meta-sort 'Bool' used in the previous definitions.) The set of function symbols together with their arity and sort are denoted conventionally in a list. Unsafe function symbols will be indicated by "*:unsafe*".

$$\begin{aligned}
&\text{false}, \text{true}: \to \text{bool}\\
&0: \to \text{nat}\\
&\text{succ}: \text{nat} \to \text{nat}\\
&\text{pred}: \text{nat} \to \text{nat}: \textit{unsafe}\\
&\text{negative}: \to \text{nat}: \textit{unsafe}\\
&\text{if}: \text{bool} \times \text{nat} \times \text{nat} \to \text{nat}.
\end{aligned}$$

The carrier sets and the ok predicates on them are given by

$$A_{\text{bool}} = \{f, t\} \qquad \text{ok}_{\text{bool}}(b) = \text{TRUE}$$

$$A_{\text{nat}} = N_0 \cup \{e_{\text{nat}}\} \qquad \text{ok}_{\text{nat}}(n) = (n \in N_0).$$

The functions corresponding to the function symbols are defined by

$$\text{false}_A: \mapsto f \qquad 0_A \quad : \mapsto 0$$

$$\text{true}_A \ : \mapsto t \qquad \text{negative}_A: \mapsto e_{\text{nat}}$$

$$\text{succ}_A: n \mapsto \begin{cases} n+1 & \text{if } n \in N_0,\\ e_{\text{nat}} & \text{if } n = e_{\text{nat}}, \end{cases}$$

$$\text{pred}_A: n \mapsto \begin{cases} n-1 & \text{if } n \in N,\\ e_{\text{nat}} & \text{if } n = 0 \text{ or } n = e_{\text{nat}}, \end{cases}$$

$$\text{if}_A: (b, n1, n2) \mapsto \begin{cases} n1 & \text{if } b = t,\\ n2 & \text{if } b = f. \end{cases}$$

pred_A is an unsafe function, because only for some ok arguments it yields ok results, for the ok value 0 it returns the error element e_{nat}. In this sense we call pred_A an *error introduction function.* Of course, negative_A is unsafe as well, but all others functions are ok, because for ok inputs they yield ok results. We call if_A an *error recovery function*, because for some error arguments (that means at least one of the arguments is an error element) it returns an ok result, e.g., $\text{if}_A(t, 0, e_{\text{nat}}) = 0$. succ_A is a *strict function*, in the sense that ok arguments return ok values and error arguments return error values. negative_A, returning always error elements, is an *error function.*

For every signature Σ we define the term algebra with ok predicates in the following way.

Definition 3.4. Let signature Σ be given. The *Σ-term algebra* $(T_\Sigma, F_\Sigma, \text{ok}_T)$ is defined by:

(a) (T_Σ, F_Σ) is the conventional term algebra without ok predicates. $T_\Sigma = \langle T_s \rangle_{s \in S}$. $F = \langle \sigma_T \rangle_{\sigma \in \Sigma}$.

(b) $\text{ok}_T = \langle \text{ok}_s \rangle_{s \in S}$.

$$\text{ok}_s(t) = \begin{cases} \text{FALSE} & \text{if an unsafe function symbol occurs in } t,\\ \text{TRUE} & \text{otherwise.} \end{cases}$$

- A term is an ok term if and only if all function symbols occurring in the term are ok function symbols.
- Our term algebras are well defined, that means T_Σ is a Σ-algebra with ok predicates satisfying part (d) of Definition 3.2: Given an ok function symbol $\sigma : s1 \times \cdots \times sn \rightarrow s$ and ok terms ti of sort si for $i = 1, \ldots, n$ (i.e., only ok function symbols occur in ti), then $\sigma_T(t1, \ldots, tn) = \sigma(t1, \ldots, tn)$ is of course an ok term.

Example 3.5. For the signature of Example 3.3 the term algebra is described by the following context-free productions, where L is the generated language.

$$\begin{array}{ll}
\langle\text{bool}\rangle & ::= \text{false}\,|\,\text{true} \\
\langle\text{nat}\rangle & ::= 0\,|\,\text{negative}\,|\,\text{succ}(\langle\text{nat}\rangle)\,|\,\text{pred}(\langle\text{nat}\rangle)\,| \\
 & \quad\ \text{if}(\langle\text{bool}\rangle, \langle\text{nat}\rangle, \langle\text{nat}\rangle) \\
T_{\text{bool}} & = L(\langle\text{bool}\rangle) \\
\text{ok}_{\text{bool}}(b) & = \text{TRUE} \\
T_{\text{nat}} & = L(\langle\text{nat}\rangle) \\
\text{ok}_{\text{nat}}(t) & = \begin{cases} \text{FALSE} & \text{if negative or pred occur in } r, \\ \text{TRUE} & \text{otherwise.} \end{cases}
\end{array}$$

For example, if(true, 0, succ(0)) is an ok term and pred(succ(0)), pred(0) or if(true, 0, negative) are error elements in the term algebra.

Σ-algebras with ok predicates may be compared by structure preserving mappings called Σ-algebra morphisms.

Definition 3.6. Let Σ-algebras $(A1, F1, \text{ok}_{A1})$, $(A2, F2, \text{ok}_{A2})$ be given. An S-indexed family of functions $h = \langle h_s \rangle_{s \in S}$, $h_s : A1_s \rightarrow A2_s$ is called *Σ-algebra morphism* if:

(a) h is a morphism between $A1$ and $A2$, viewed as algebras without ok predicates, and

(b) for $s \in S$ and $a \in A_s$, $\text{ok}_{A1}(a)$ implies $\text{ok}_{A2}(h_s(a))$.

A morphism is called *injective* (respectively *surjective*) if every h_s is injective (respectively surjective).

A morphism is called *strict* if, for $s \in S$ and $a \in A1_s$, $\text{ok}_{A1}(a) = \text{ok}_{A2}(h_s(a))$.

- Because of the additional predicate structure on signatures and algebras we require in part (b) that no ok element may be mapped onto an error element or in other words that the *ok property for elements is preserved.*
- The *isomorphisms* are the injective, surjective and strict morphisms. The strictness property is necessary, because we do not only want the operational structure but also the predicate structure to be respected by isomorphisms.
- h_{ok} and h_{err} denote the restriction of h to ok and error elements.

$$\begin{array}{ll}
h_{\text{ok}} = \langle h_{\text{ok},s} \rangle_{s \in S}, & h_{\text{ok},s} : A1_{\text{ok},s} \rightarrow A2_{\text{ok},s}, \\
h_{\text{err}} = \langle h_{\text{err},s} \rangle_{s \in S}, & h_{\text{err},s} : A1_{\text{err},s} \rightarrow A2_s.
\end{array}$$

For strict morphisms we can denote $h_{\mathrm{err},s}$ of course by

$$h_{\mathrm{err},s}: A1_{\mathrm{err},s} \to A2_{\mathrm{err},s}.$$

Example 3.7. The following is an example of a morphism between algebras with ok predicates and a motivation for the freedom of *allowing error elements to be mapped to ok elements.* We give a morphism from the term algebra of Example 3.5 into the algebra of Example 3.3. It is the uniquely determined morphism between these algebras taken without ok predicates.

$$\begin{array}{lll}
h_{\mathrm{bool}}: & T_{\mathrm{bool}} & \to A_{\mathrm{bool}} \\
& \mathrm{false} & \mapsto f \\
& \mathrm{true} & \mapsto t \\
h_{\mathrm{nat}}: & T_{\mathrm{nat}} & \to A_{\mathrm{nat}} \\
& 0 & \mapsto 0 \\
& \mathrm{negative} & \mapsto e_{\mathrm{nat}} \\
& \mathrm{succ}(t) & \mapsto \mathrm{succ}_A(h_{\mathrm{nat}}(t)) \\
& \mathrm{pred}(t) & \mapsto \mathrm{pred}_A(h_{\mathrm{nat}}(t)) \\
& \mathrm{if}(b, t1, t2) & \mapsto \mathrm{if}_A(h_{\mathrm{bool}}(b), h_{\mathrm{nat}}(t1), h_{\mathrm{nat}}(t2)).
\end{array}$$

h respects the operations and h preserves ok elements, which can be seen directly from its definition. What h does is simply that it sends a term to the corresponding value in A when the term is evaluated. So succ(succ(0)) will naturally be mapped to 2 and, of course, the error (or unsafe) terms pred(succ(succ(0))), if(true, 0, negative) and pred(0) will result in 1, 0 and e_{nat}, respectively.

The next theorem will show that this morphism is the only morphism between such algebras, i.e., our term algebras are initial.

Theorem 3.8. *Let signature Σ, term algebra T_Σ and Σ-algebra A be given. Then there exists a unique morphism $h: T_\Sigma \to A$, or, in other words, T_Σ is initial in the class of all Σ-algebras.*

Proof. We already know from Goguen et al. [1] that there exists a unique morphism $h: T_\Sigma \to A$ viewed as a mapping between algebras without ok predicates.

$$h_s: t \mapsto \begin{cases} \sigma_A & \text{if } t=\sigma,\ \sigma: \to s, \\ \sigma_A(h_{s1}(t1), \ldots, h_{sn}(tn)) & \text{if } t=\sigma(t1, \ldots, tn), \\ & \quad \sigma: s1 \times \cdots \times sn \to s. \end{cases}$$

We have to prove that this mapping is a morphism between algebras with ok predicates as well, i.e., part (b) of our morphism definition holds. By induction on the depth of terms, we show that, for each $t \in T_\Sigma$, ok(t) implies ok$(h(t))$.

Let $t=\sigma$, $\sigma: \to s$. If $\mathrm{ok}_{T,s}(\sigma)=\mathrm{TRUE}$, then $\mathrm{ok}_\Sigma(\sigma)=\mathrm{TRUE}$ and therefore $\mathrm{TRUE}=\mathrm{ok}_{A,s}(\sigma_A)=\mathrm{ok}_{A,s}(h_s(\sigma))$ according to the definition of algebra part (d) and the definition of h.

Let $t = \sigma(t1, \ldots, tn)$, $\sigma : s1 \times \cdots \times sn \to s$. If $\mathrm{ok}_{T,s}(t) = \text{TRUE}$ holds, then $\mathrm{ok}_\Sigma(\sigma) = \text{TRUE}$ and $\mathrm{ok}_{T,si}(ti) = \text{TRUE}$ for $i = 1, \ldots, n$. This implies

$$\begin{aligned}\mathrm{ok}_{A,s}(h_s(t)) &= \mathrm{ok}_{A,s}(h_s(\sigma(t1, \ldots, tn)) \\ &= \mathrm{ok}_{A,s}(\sigma_A(h_{s1}(t1), \ldots, h_{sn}(tn))) = \text{TRUE}\end{aligned}$$

according to the definition of h, the induction assumption and the definition of algebra part (d). □

4. Specifications

An important difference between our specification technique and the usual algebraic specification without error handling is that we introduce two different types of variables for the same sort. Variables of the first type will serve for the ok part of the corresponding carrier set only, variables of the second type for the whole carrier set.

Definition 4.1. Let signature Σ be given. A tuple (V, ok_V) is called a *set of variables* (with ok predicates) for Σ if:

(a) $V = \langle V_s \rangle_{s \in S}$ is an S-indexed, pairwise disjoint family of sets (of variables), each V_s disjoint from Σ, and

(b) $\mathrm{ok}_V = \langle \mathrm{ok}_{V,s} \rangle_{s \in S}$, $\mathrm{ok}_{V,s} : V_s \to \text{BOOL}$ is an S-indexed family of predicates.

- A set of variables (V, ok_V) is often denoted by V only.
- When no ambiguity arises, $\mathrm{ok}(v)$ means $\mathrm{ok}_{V,s}(v)$ for $v \in V_s$. In analogy to ok and unsafe functions we use the notions of *ok and unsafe variables.*

Definition 4.2. Let signature Σ, Σ-algebra A and variables V be given. An *assignment* to (or interpretation of) the variables is an S-indexed family of functions $I = \langle I_s \rangle_{s \in S}$, $I_s : V_s \to A_s$ such that $\mathrm{ok}_{V,s}(v)$ implies $\mathrm{ok}_s(I_s(v))$ for $s \in S$ and $v \in V_s$.

- If $\mathrm{ok}(v) = \text{TRUE}$ holds, it is not allowed to assign an error element to v, $\mathrm{ok}(v) = \text{FALSE}$ indicates that v may hold ok or error values.

Example 4.3. We give variables for the signature of Example 3.3.

$$\begin{array}{ll} V_{\text{bool}} = \emptyset & V_{\text{nat}} = \{n, n+\} \\ \mathrm{ok}(n) = \text{TRUE} & \mathrm{ok}(n+) = \text{FALSE}. \end{array}$$

Assignments into the algebra of Example 3.3 are

$$\begin{array}{ll} I_{\text{nat}} : n \mapsto 154 & n+ \mapsto 154 \\ I_{\text{nat}} : n \mapsto 154 & n+ \mapsto e_{\text{nat}}. \end{array}$$

But the following mapping is not an assignment:

$$I_{\text{nat}}: n \mapsto e_{\text{nat}} \qquad n+ \mapsto 154.$$

Definition 4.4. Let signature Σ and variables V be given. The *extended signature* $(S, \Sigma(V), \text{arity}_{\Sigma(V)}, \text{sort}_{\Sigma(V)}, \text{ok}_{\Sigma(V)})$ is defined as follows:

(a) $\Sigma(V) = \Sigma \cup \bigcup_{s\in S} V_s$,

(b) $\text{arity}_{\Sigma(V)}(\sigma) = \textbf{if } \sigma \in V_s \textbf{ then } \varepsilon \textbf{ else } \text{arity}(\sigma) \textbf{ fi}$,

(c) $\text{sort}_{\Sigma(V)}(\sigma) = \textbf{if } \sigma \in V_s \textbf{ then } s \textbf{ else } \text{sort}(\sigma) \textbf{ fi}$,

(d) $\text{ok}_{\Sigma(V)}(\sigma) = \textbf{if } \sigma \in V_s \textbf{ then } \text{ok}_{V,s}(\sigma) \textbf{ else } \text{ok}_{\Sigma}(\sigma) \textbf{ fi}$.

- Variables are treated as constants of the according type.
- $(T_{\Sigma(V)}, F_{\Sigma(V)}, \text{ok}_{T,\Sigma(V)})$ is a $\Sigma(V)$-algebra, but we want to treat it as a Σ-algebra. So we omit the variables from the signature, leaving the carrier sets and predicates unchanged, and we forget the operations corresponding to the variables as well. the resulting algebra is denoted by $T_\Sigma(V)$.

  We next show that for our notion of algebra and morphism there always exist free algebras.

Lemma 4.5. *Let signature Σ, variables V, Σ-algebra A and assignment $I: V \to A$ be given. Then there is a unique Σ-algebra morphism $\underline{I}: T_\Sigma(V) \to A$, that extends I in the sense that $I_s(v) = \underline{I}_s(v)$ for $s \in S$ and $v \in V_s$.*

Proof. There is a unique $\underline{I}$ viewed as a morphism between algebras without ok predicates. This $\underline{I}$ is a morphism due to our definition as well. We have to show that $\text{ok}(\underline{I}(t)) = \text{TRUE}$ is valid for every ok term t.

If $t \in T_\Sigma$, then $\text{ok}(\underline{I}(t)) = \text{TRUE}$ holds, because T_Σ is initial.

If $t \in T_\Sigma(V) - T_\Sigma$, then there are $s \in S$ and $v \in V_s$ occurring in t and, for all variables v in t, $\text{ok}(v) = \text{TRUE}$ holds. This implies $\text{ok}(I(v)) = \text{TRUE}$ and because t is an ok term, all function symbols σ occurring in t have $\text{ok}(\sigma) = \text{TRUE}$. So there are only ok functions applied to ok elements and A is of course a Σ-algebra. So $\text{ok}(\underline{I}(t)) = \text{TRUE}$ holds. □

The notions of *Σ-equation* and of equations *satisfied* by a Σ-algebra are defined as usual. A *congruence* on an algebra with ok predicates is just an algebra congruence. But please note that our definition of assignment implies that there is a *restriction to the substitution of variables.* An equation may be valid although it does not hold for error elements substituted for ok variables. Such an equation had been given in Example 2.2 with axiom (A5).

Example 4.6. Let n, $n1+$ and $n2+$ be variables of sort nat with $\text{ok}(n) = \text{TRUE}$ and $\text{ok}(n1+) = \text{ok}(n2+) = \text{FALSE}$. Then the algebra of Example 3.3 satisfies the following equations (among others):

$$\begin{aligned}
&\text{pred}(\text{succ}(n)) = n\\
&\text{pred}(0) = \text{negative}\\
&\text{if}(\text{false}, n1+, n2+) = n2+\\
&\text{if}(\text{true}, n1+, n2+) = n1+\\
&\text{succ}(\text{negative}) = \text{negative}\\
&\text{pred}(\text{negative}) = \text{negative}.
\end{aligned}$$

But, for example, the equation

$$\text{succ}(\text{pred}(n)) = n$$

does not hold, because $\text{succ}_A(\text{pred}_A(0)) = \text{succ}_A(e_{\text{nat}}) = e_{\text{nat}}$.

Given a Σ-algebra A and a congruence relation $\equiv$ on it, the *quotient* $A/\equiv$ of A by $\equiv$ can be made into a Σ-algebra with ok predicates by defining the carrier sets and operations in the usual way and by letting a class be ok if and only if there is an ok element of the algebra in it. In this sense, TRUE dominates FALSE with respect to the ok predicate of a class.

Definition 4.7. Let signature Σ, Σ-algebra A and congruence $\equiv\; = \langle \equiv_s \rangle_{s\in S}$ be given.

(a) $(A/\equiv, F_{A/\equiv})$ denotes the usual quotient of an algebra by a congruence relation on it.

(b) $\text{ok}_{A/\equiv} = \langle \text{ok}_{\equiv,s} \rangle_{s\in S}$ is an S-indexed family of predicates.

$$\text{ok}_{\equiv,s}([a]) = \begin{cases} \text{TRUE} & \text{if there is a } b \in [a] \text{ with } \text{ok}_{A,s}(b) = \text{TRUE},\\ \text{FALSE} & \text{otherwise.} \end{cases}$$

We now have to show that our quotient really is a Σ-algebra with ok predicates.

Lemma 4.8. $(A/\equiv, F_{A/\equiv}, \text{ok}_{A/\equiv})$ *is a Σ-algebra with ok predicates.*

Proof. We already know $A/\equiv$ is a Σ-algebra without ok predicates. So we only prove:

For every $\sigma : s1 \times \cdots \times sn \to s$ with $\text{ok}_\Sigma(\sigma) = \text{TRUE}$ and every $ai \in A_{si}$ with $\text{ok}_{\equiv,si}([ai]) = \text{TRUE}$ for $i = 1, \ldots, n$ we have

$$\text{ok}_{\equiv,s}(\sigma_{A/\equiv}([a1], \ldots, [an])) = \text{TRUE}.$$

$\text{ok}_{\equiv,si}([ai]) = \text{TRUE}$ implies there are $bi \in [ai]$ and $\text{ok}_{A,si}(bi) = \text{TRUE}$. It follows that $\text{ok}_{A,s}(\sigma_A(b1, \ldots, bn)) = \text{TRUE}$ holds. So we have

$$\begin{aligned}
&\text{ok}_{\equiv,s}(\sigma_{A/\equiv}([a1], \ldots, [an])) = \text{ok}_{\equiv,s}(\sigma_{A/\equiv}([b1], \ldots, [bn]))\\
&\quad = \text{ok}_{\equiv,s}([\sigma_A(b1, \ldots, bn)]) = \text{ok}_{A,s}(\sigma_A(b1, \ldots, bn)) = \text{TRUE}. \qquad \Box
\end{aligned}$$

For a given set of equations E with variables the induced set of *constant equations* $E(T_\Sigma)$ and the *generated least congruence relation* denoted by $\equiv_E = \langle \equiv_{E,s} \rangle_{s \in S}$ are defined in the usual way. There always exists such a $\equiv_E$ since we know that there always is a least congruence generated by a given relation if we deal only with algebras without ok predicates and our congruence definition did not involve the predicates. For brevity we often denote $\equiv_E$ by $\equiv$ and $a \equiv_{E,s} b$ by $a \equiv b$ if no ambiguities arise.

Example 4.9. If we look at the equations of Example 4.6, we find that the following pairs are in $E(T_\Sigma)_{\text{nat}}$ due to the first equation:

$$\langle \text{pred}(\text{succ}(0)), 0 \rangle$$
$$\langle \text{pred}(\text{succ}(\text{succ}(0))), \text{succ}(0) \rangle.$$

But the following pairs are not in $E(T_\Sigma)_{\text{nat}}$:

$$\langle \text{pred}(\text{succ}(\text{negative})), \text{negative} \rangle$$
$$\langle \text{pred}(\text{succ}(\text{pred}(\text{succ}(0)))), \text{pred}(\text{succ}(0)) \rangle.$$

On the other hand, the last pair is *in the congruence relation* generated by $E(T_\Sigma)$:

$$\begin{aligned} &\langle \text{pred}(\text{succ}(0)), 0 \rangle \in E(T_\Sigma)_{\text{nat}} \Rightarrow \text{pred}(\text{succ}(0)) \equiv 0 \Rightarrow \\ &\quad \text{succ}(\text{pred}(\text{succ}(0))) \equiv \text{succ}(0) \Rightarrow \\ &\quad \text{pred}(\text{succ}(\text{pred}(\text{succ}(0)))) \equiv \text{pred}(\text{succ}(0)) \equiv 0. \end{aligned}$$

The pleasant thing about out approach to error and exception handling is that the fundamental initiality result of Goguen et al. [1] is still valid.

Theorem 4.10. *$T_\Sigma/\equiv_E$ is initial in the class of all Σ-algebras satisfying E, i.e., given a Σ-algebra A, which satisfies E, we have a unique morphism $g: T_\Sigma/\equiv_E \to A$.*

Remark. $T_\Sigma/\equiv_E$ is denoted by $T_{\Sigma,E}$ and called the *quotient term algebra.*

Proof of Theorem 4.10. We know T_Σ is initial in the class of all Σ-algebras and therefore we have unique morphisms h_E and h_A:

$$\begin{tikzcd} & T_\Sigma \arrow[dl, "h_E"'] \arrow[dr, "h_A"] & \\ T_{\Sigma,E} \arrow[rr, dashed, "g"'] & & A \end{tikzcd}$$

We define $g([t]) := h_A(t)$. g is independent of representatives, it respects the operations and is unique. The proof is analogous to the initiality proof for algebras without ok predicates. So we only have to show condition (b) of our morphism definition is valid for this g.

If $\mathrm{ok}_{\Sigma,E}([t]) = \mathrm{TRUE}$ holds, we know there is a $t' \in [t]$ with $\mathrm{ok}_T(t') = \mathrm{TRUE}$. So we conclude

$$\mathrm{ok}_A(g([t])) = \mathrm{ok}_A(g([t'])) = \mathrm{ok}_A(h_A(t')) = \mathrm{TRUE},$$

because of the independency of representatives and the definition of g and the requirement that a morphism has to respect the ok property. □

Example 4.11. The quotient of T_Σ in Example 3.5 by the equations in Example 4.6 is isomorphic to the algebra of natural numbers in Example 3.3.

Example 4.12. We want to explain exactly how the quotient term algebra of Example 2.2 looks. The operation declarations had been given by (we use 'negative' instead of 'error'):

$$\begin{aligned}
&0 : \to \mathrm{nat}\\
&\mathrm{succ} : \mathrm{nat} \to \mathrm{nat}\\
&\mathrm{negative} : \to \mathrm{nat} : \textit{unsafe}\\
&\mathrm{pred} : \mathrm{nat} \to \mathrm{nat} : \textit{unsafe}\\
&\mathrm{plus}, \mathrm{times} : \mathrm{nat} \times \mathrm{nat} \to \mathrm{nat}.
\end{aligned}$$

The equations E had used ok variables n and m of sort nat:

$$\mathrm{pred}(\mathrm{succ}(n)) = n \tag{A1}$$
$$\mathrm{pred}(0) = \mathrm{negative} \tag{A2}$$
$$\mathrm{plus}(0, n) = n \tag{A3}$$
$$\mathrm{plus}(\mathrm{succ}(n), m) = \mathrm{succ}(\mathrm{plus}(n, m)) \tag{A4}$$
$$\mathrm{times}(0, n) = 0 \tag{A5}$$
$$\mathrm{times}(\mathrm{succ}(n), m) = \mathrm{plus}(m, \mathrm{times}(n, m)). \tag{A6}$$

We give a description of $T_{\Sigma,E}$ by means of a canonical term algebra. Every congruence class will be represented by a word of the context-free languages defined by the following productions:

$$\begin{aligned}
\langle\text{ok-nat}\rangle \quad &::= 0 \mid \mathrm{succ}(\langle\text{ok-nat}\rangle)\\
\langle\text{err-nat}\rangle \quad &::= \mathrm{negative} \mid \mathrm{succ}(\langle\text{err-nat}\rangle) \mid \mathrm{pred}(\langle\text{err-nat}\rangle) \mid\\
&\qquad \mathrm{plus}(\langle\text{err-nat}\rangle, \langle\text{nat}\rangle) \mid\\
&\qquad \mathrm{plus}(\langle\text{ok-nat}\rangle, \langle\text{err-nat}\rangle) \mid\\
&\qquad \mathrm{times}(\langle\text{err-nat}\rangle, \langle\text{nat}\rangle) \mid\\
&\qquad \mathrm{times}(\langle\text{ok-nat}\rangle, \langle\text{err-nat}\rangle)\\
\langle\text{nat}\rangle \quad &::= \langle\text{ok-nat}\rangle \mid \langle\text{err-nat}\rangle\\
T_{\Sigma,E,\mathrm{nat},\mathrm{ok}} \quad &= L(\langle\text{ok-nat}\rangle)\\
T_{\Sigma,E,\mathrm{nat},\mathrm{err}} \quad &= L(\langle\text{err-nat}\rangle).
\end{aligned}$$

The ok part of the carrier set represents the natural numbers, the error elements can be seen as error messages informing about illegal applications of the predecessor function to 0. Examples of operation applications are

$$\begin{aligned}
&\mathrm{pred}_{\Sigma,E}(\mathrm{succ}(0)) = 0\\
&\mathrm{pred}_{\Sigma,E}(0) = \mathrm{negative}\\
&\mathrm{pred}_{\Sigma,E}(\mathrm{negative}) = \mathrm{pred}(\mathrm{negative})\\
&\mathrm{plus}_{\Sigma,E}(\mathrm{succ}(0), \mathrm{succ}(0)) = \mathrm{succ}(\mathrm{succ}(0))\\
&\mathrm{times}_{\Sigma,E}(0, \mathrm{succ}(0)) = 0\\
&\mathrm{times}_{\Sigma,E}(0, \mathrm{negative}) = \mathrm{times}(0, \mathrm{negative}).
\end{aligned}$$

Please note equation (A5) is valid and $\mathrm{times}_{\Sigma,E}(0, \mathrm{negative})$ evaluates not to 0.

Example 4.13. Appendix A gives the obligatory stack example this time including exception handling.

We now know that for given signature Σ, variables V and equations E there always exists an initial Σ-algebra which can be chosen as a *standard semantics*. So we put together signatures, variables and equations as usual, getting a specification.

Definition 4.14. A *specification* is a triple (Σ, V, E), where Σ is a signature with ok predicates, V is a set of variables with ok predicates and E is a set of Σ-equations.

5. Correctness of specifications

The usual notion of correctness of specifications—the isomorphism between the specified algebra and the given model—is somewhat too strong for our purpose. Our main interest lies in the ok part of the carrier sets. If we look at Example 4.12, we would not like the whole bunch of error elements to appear in our model. The main thing about terms like succ(negative) and pred(negative) is that they are error elements and it is not important here that they are different. So we allow different error elements of the specified algebra to be identified in our model.

Definition 5.1. Let specification (Σ, V, E) and Σ-algebra A be given. (Σ, V, E) is called *correct* with respect to A if:

(a) there is a strict morphism $h: T_{\Sigma,E} \to A$, such that
(b) $h_{\mathrm{ok}}: T_{\Sigma,E,\mathrm{ok}} \to A_{\mathrm{ok}}$ is bijective, and
(c) $h_{\mathrm{err}}: T_{\Sigma,E,\mathrm{err}} \to A_{\mathrm{err}}$ is surjective.

(Σ, V, E) is *strongly correct* with respect to A if it is correct with respect to A and the morphism h is an isomorphism.

- We permit that the morphism h identifies certain error elements, as long as the operational structure on the ok and error part is preserved. Because there may be more error elements in the specified algebra than in the model, the specification may define a somewhat richer algebra than the model.

- The *conventional notion of correctness of specifications* for algebras without ok predicates may be embedded into this correctness criterion, because then we only deal with ok functions and ok elements and so $T_{\Sigma,E,\mathrm{err}} = A_{\mathrm{err}} = \emptyset$.
- Of course, sometimes it is necessary for the specification of a given model to introduce new sorts and operations that are not part of the model. In this case not $T_{\Sigma,E}$ but the reduct of $T_{\Sigma,E}$ with respect to the signature of the model will be considered in the correctness criterion. We prefer our notion here for simplicity of formalism.

Example 5.2. We give a *correct specification* for the algebra A defined in Example 3.3. The signature has been given by

$$\begin{aligned}
&\text{false, true}: \to \text{bool}\\
&0: \to \text{nat}\\
&\text{succ}: \text{nat} \to \text{nat}\\
&\text{pred}: \text{nat} \to \text{nat}: \textit{unsafe}\\
&\text{negative}: \to \text{nat}: \textit{unsafe}\\
&\text{if}: \text{bool} \times \text{nat} \times \text{nat} \to \text{nat}.
\end{aligned}$$

The axioms E using ok variable n and unsafe variables $n1+$ and $n2+$ of sort nat are given by

$$\begin{aligned}
&\text{pred}(\text{succ}(n)) = n\\
&\text{pred}(0) = \text{negative}\\
&\text{if}(\text{true}, n1+, n2+) = n1+\\
&\text{if}(\text{false}, n1+, n2+) = n2+.
\end{aligned}$$

Again we give a description of $T_{\Sigma,E}$ by means of a canonical term algebra using the context-free languages defined by the following productions:

$$\begin{aligned}
&\langle\text{bool}\rangle && ::= \text{false} \,|\, \text{true}\\
&\langle\text{nat}\rangle && ::= \langle\text{nat-ok}\rangle \,|\, \langle\text{nat-err}\rangle\\
&\langle\text{nat-ok}\rangle && ::= 0 \,|\, \text{succ}(\langle\text{nat-ok}\rangle)\\
&\langle\text{nat-err}\rangle && ::= \text{negative} \,|\, \text{succ}(\langle\text{nat-err}\rangle) \,|\, \text{pred}(\langle\text{nat-err}\rangle).
\end{aligned}$$

$$\begin{aligned}
&T_{\Sigma,E,\mathrm{bool}} = L(\langle\text{bool}\rangle) && \mathrm{ok}_{\mathrm{bool}}(b) = \textsc{True}\\
&T_{\Sigma,E,\mathrm{nat}} = L(\langle\text{nat}\rangle) && \mathrm{ok}_{\mathrm{nat}}(n) = (n \in L(\langle\text{nat-ok}\rangle)).
\end{aligned}$$

The operations in $T_{\Sigma,E}$ are defined in the usual way, e.g.,

$$\text{succ}_{\Sigma,E} : t \mapsto \text{succ}(t)$$

$$\text{pred}_{\Sigma,E} : t \mapsto \begin{cases} \text{succ}^{n-1}(0) & \text{if } t = \text{succ}^n(0) \text{ and } n > 0,\\ \text{negative} & \text{if } t = 0,\\ \text{pred}(t) & \text{if } t \in L(\langle\text{nat-err}\rangle). \end{cases}$$

We now define a mapping $h: T_{\Sigma,E} \to A$:

$$\begin{array}{lll} h_{\text{bool}} & : T_{\Sigma,E,\text{bool}} \to A_{\text{bool}} & \\ & \quad \text{true} \quad \mapsto t & \\ & \quad \text{false} \quad \mapsto f & \\ h_{\text{nat}} & : T_{\Sigma,E,\text{nat}} \to A_{\text{nat}} & \\ & \quad t \quad \mapsto \begin{cases} n & \text{if } t \in L(\langle\text{nat-ok}\rangle),\ t = \text{succ}^n(0), \\ e_{\text{nat}} & \text{if } t \in L(\langle\text{nat-err}\rangle). \end{cases} & \end{array}$$

To prove that h is a strict morphism, we have to show:

(1) $h(\sigma_{\Sigma,E}(a1, \ldots, an)) = \sigma_A(h(a1), \ldots, h(an))$ holds for every $\sigma \in \Sigma$.

(2) $\text{ok}_{\Sigma,E}(a) = \text{ok}_A(h(a))$ for $s \in S$ and $a \in A_s$.

We prove (1) for pred, the other functions symbols may be treated in an analogous way: $h(\text{pred}_{\Sigma,E}(t)) = \text{pred}_A(h(t))$.

We distinguish three cases for t:

Case 1. $t \in L(\langle\text{nat-ok}\rangle)$ and $t = \text{succ}^n(0)$ with $n > 0$, $h(\text{pred}_{\Sigma,E}(\text{succ}^n(0))) = h(\text{succ}^{n-1}(0)) = n - 1 = \text{pred}_A(n) = \text{pred}_A(h(\text{succ}^n(0)))$.

Case 2. $t = 0$, $h(\text{pred}_{\Sigma,E}(0)) = h(\text{negative}) = e_{\text{nat}} = \text{pred}_A(0) = \text{pred}_A(h(0))$.

Case 3. $t \in L(\langle\text{nat-err}\rangle)$, $h(\text{pred}_{\Sigma,E}(t)) = h(\text{pred}(t)) = e_{\text{nat}} = \text{pred}_A(e_{\text{nat}}) = \text{pred}_A(h(t))$.

The strictness of h, its bijectivity on the ok part and its surjectivity on the error part can be seen directly from its definition. The specified algebra is correct with respect to the algebra A of Example 3.3, although all error elements are mapped to the one error element in A. If we want to get a *strongly correct* specification, we have to add the equations succ(negative) = negative and pred(negative) = negative.

Example 5.3. In [1], a specification for the data type bool including error handling and its correctness proof is given. We want to specify this data type with our method.

$$\begin{array}{l} \text{false, true} : \to \text{boole} \\ \text{error} : \to \text{boole} : \textit{unsafe} \\ \text{not, ok} : \text{boole} \to \text{boole} \\ \text{and} : \text{boole} \times \text{boole} \to \text{boole} \\ \text{ife} : \text{boole} \times \text{boole} \times \text{boole} \to \text{boole}. \end{array}$$

The carrier set, the ok predicate and the operations of our model are given by

$$\begin{array}{ll} A_{\text{boole}} = \{f, t, e\} & \text{ok}(b) = (b \neq e) \\ \text{false}_A : \mapsto f & \text{error}_A : \mapsto e \\ \text{true}_A : \mapsto t & \end{array}$$

$$\text{not}_A : b \mapsto \begin{cases} f & \text{if } b = t, \\ t & \text{if } b = f, \\ e & \text{if } b = e, \end{cases}$$

$$\text{ok}_A : b \mapsto \begin{cases} t & \text{if } b = f \text{ or } b = t, \\ f & \text{if } b = e, \end{cases}$$

$$\text{and}_A : (b, b') \mapsto \begin{cases} t & \text{if } b = t \text{ and } b' = t, \\ f & \text{if } b, b' \neq e \text{ and } (b = f \text{ or } b' = f), \\ e & \text{if } b = e \text{ or } b' = e, \end{cases}$$

$$\text{ife}_A : (b, b', b'') \mapsto \begin{cases} b' & \text{if } b = t, \\ b'' & \text{if } b = f, \\ e & \text{if } b = e. \end{cases}$$

The equations using ok variables b, $b1$ and $b2$ and unsafe variables $b+$, $b1+$ and $b2+$ look like

$$\begin{array}{ll} \text{not(true)} = \text{false} & /*\ \text{axioms for not}\ */ \\ \text{not}(\text{not}(b)) = b & \\ \text{and}(\text{true}, b) = b & /*\ \text{axioms for and}\ */ \\ \text{and}(\text{false}, b) = \text{false} & \\ \text{and}(b1, b2) = \text{and}(b2, b1) & \\ \text{ife}(\text{true}, b1+, b2+) = b1+ & /*\ \text{axioms for ife}\ */ \\ \text{ife}(\text{false}, b1+, b2+) = b2+ & \\ \text{not(error)} = \text{error} & /*\ \text{error propagating axioms}\ */ \\ \text{and}(\text{error}, b+) = \text{error} & \\ \text{and}(b+, \text{error}) = \text{error} & \\ \text{ife}(\text{error}, b1+, b2+) = \text{error} & \\ \text{ok(false)} = \text{true} & /*\ \text{axioms for ok}\ */ \\ \text{ok(true)} = \text{true} & \\ \text{ok(error)} = \text{false}. & \end{array}$$

The specified algebra is *strongly correct* with respect to the given model. The error propagating axioms are only needed because the function ok has to be defined completely. If we would delete the function ok from the signature, we could drop the error propagating axioms and those for the function ok. This would result in a shorter specification and it would cause the error part of our specified algebra to blow up. But we could map all these error elements by the correctness morphism onto the only error element in A_{boole}.

Thus a *correct* specification for the model algebra without the ok function will consist only of axioms for normal situations.

Example 5.4. Appendix B gives a specification and a correctness proof for an error recovery stack.

6. Operational semantics of specifications

A set of equations can be viewed as a set of rewrite rules interpreting equations from left to right. By substituting constant terms for the variables we get a set of constant rewrite rules. These rules determine a reduction process on terms which stops if none of the axioms can be applied further. In this way we give an operational semantics for specifications which is well defined if the set of constant rewrite rules has the finite Church–Rosser property.

In this section we show the following: For specifications allowing error and exception handling and operationally well defined in the above sense the algebraic and operational semantics coincide.

Definition 6.1. Let specification (Σ, V, E) and the set of constant equations $E(T_\Sigma)$ be given. $\rightarrow_E = \langle \rightarrow_s \rangle_{s \in S}$ is the family of relations on T_Σ defined as follows:

(a) If $\langle t, t' \rangle \in E(T_\Sigma)_s$, then $t \rightarrow_s t'$.

(b) If $\sigma : s1 \times \cdots \times sn \rightarrow s$, $ti \in T_{si}$ for $i = 1, \ldots, n$ and $j \in \{1, \ldots, n\}$ with $tj \rightarrow_{sj} tj'$ are given, then

$$\sigma(t1, \ldots, tj, \ldots, tn) \rightarrow_s \sigma(t1, \ldots, tj', \ldots, tn)$$

$\rightarrow_E^* = \langle \rightarrow_s^* \rangle_{s \in S}$ is the reflexive and transitive closure of $\rightarrow_E$ and called the family of *subterm replacements* induced by E.

A term t of sort s has the *normal form* t' if $t \rightarrow_s^* t'$ and there is no $t' \rightarrow_s \bar{t}$. This is denoted by $\mathrm{nf}(t) = t'$.

Example 6.2. If we look at the specification of Example 5.2 we have, for example, the following derivations from terms to their normal forms (we here abbreviate pred, succ, true and negative by p, s, t and n, respectively).

- $p(s(s(p(s(0))))) \rightarrow_{\mathrm{nat}} p(s(s(0))) \rightarrow_{\mathrm{nat}} s(0)$
- $p(s(p(0))) \rightarrow_{\mathrm{nat}} p(s(n))$
-

$$\begin{tikzcd}
 & \mathrm{if}(t, p(s(0)), p(0)) \arrow[ddl] \arrow[d] \arrow[dr] & \\
 & \mathrm{if}(t, p(s(0)), n) \arrow[dl] \arrow[d] & \mathrm{if}(t, 0, p(0)) \arrow[dl] \arrow[ddl] \\
p(s(0)) \arrow[dr] & \mathrm{if}(t, 0, n) \arrow[d] & \\
 & 0 &
\end{tikzcd}$$

The normal forms of Examples 5.2 and 4.12 are identical to the elements of the carrier sets of the given canonical terms algebras.

Definition 6.3. $\rightarrow_E^*$ is called *finite Church–Rosser* if every term t or sort s has a normal form, and if $t \rightarrow_s^* \bar{t}$ and $t \rightarrow_s^* \bar{t}'$, then there is a t' with $\bar{t} \rightarrow_s^* t'$ and $\bar{t}' \rightarrow_s^* t'$.

- If $\rightarrow_E^*$ is finite Church–Rosser, each term has a *unique normal form.*

Example 6.4. For the sets of equations in Examples 4.6, 4.12 and 4.13 and for the specifications in Example 5.2 and Appendices A and B the families of subterm replacements $\rightarrow_E^*$ are finite Church-Rosser.

Definition 6.5. Let specification (Σ, V, E) with finite Church-Rosser $\rightarrow_E^*$ be given. The *normal form algebra* $(\mathrm{NF}, F_{\mathrm{NF}}, \mathrm{ok}_{\mathrm{NF}})$ is defined as follows:

(a) $\mathrm{NF} = \langle \mathrm{NF}_s \rangle_{s \in S}$. NF_s are the normal forms of sort s.

(b) $F_{\mathrm{NF}} = \langle \sigma_{\mathrm{NF}} \rangle_{\sigma \in \Sigma}$. For $\sigma : s1 \times \cdots \times sn \rightarrow s$ and normal forms ti of sort si for $i = 1, \ldots, n$, the function σ_{NF} is given by

$$\sigma_{\mathrm{NF}}(t1, \ldots, tn) = \mathrm{nf}(\sigma(t1, \ldots, tn)).$$

(c) $\mathrm{ok}_{\mathrm{NF}} = \langle \mathrm{ok}_{\mathrm{NF},s} \rangle_{s \in S}$. For a normal form t of sort s, $\mathrm{ok}_{\mathrm{NF},s}$ is defined by

$$\mathrm{ok}_{\mathrm{NF},s}(t) = \begin{cases} \textsc{True} & \text{if there is an ok term } t' \text{ with } \mathrm{nf}(t') = t, \\ \textsc{False} & \text{otherwise.} \end{cases}$$

- The finite Church-Rosser property guarantees the well-definedness of the functions σ_{NF}.
- $(\mathrm{NF}, F_{\mathrm{NF}}, \mathrm{ok}_{\mathrm{NF}})$ is a Σ-algebra with ok predicates satisfying part (d) of our definition for algebra: Let ok function symbol $\sigma : s1 \times \cdots \times sn \rightarrow s$ and normal forms ti of ok terms ti' with sort si for $i = 1, \ldots, n$ be given. Then we have

  $$\mathrm{ok}_{T,s}(\sigma(t1', \ldots, tn')) = \textsc{True} \quad \text{and}$$
  $$\sigma(t1', \ldots, tn') \rightarrow_s^* \sigma(t1, \ldots, tn) \rightarrow_s^* \mathrm{nf}(\sigma(t1, \ldots, tn)).$$

  This implies

  $$\mathrm{ok}_{\mathrm{NF},s}(\sigma_{\mathrm{NF}}(t1, \ldots, tn)) = \mathrm{ok}_{\mathrm{NF},s}(\mathrm{nf}(\sigma(t1, \ldots, tn))) = \textsc{True}.$$

- A normal form t is ok in the normal form algebra if and only if there is an ok term t' which has t as its normal form. In this sense the ok terms dominate the error terms or, in other words, if an error term is equivalent to an ok term, this 'heals' the error term. If the rules are *ok term preserving*, which means there is no $\langle t, t' \rangle \in E(T_\Sigma)$ with $\mathrm{ok}(t) = \textsc{True}$ and $\mathrm{ok}(t') = \textsc{False}$, the ok predicates in the normal form algebra are determined by the normal forms themselves.
- Every normal form algebra is a canonical term algebra.

Example 6.6. As mentioned above, the normal forms of the specification in Example 5.2 are identical with elements of the carrier sets of the canonical term algebra given in that example. The same holds for the functions σ_{NF} and $\sigma_{\Sigma,E}$ for every $\sigma \in \Sigma$ and the predicates $\mathrm{ok}_{\mathrm{NF},s}$ and $\mathrm{ok}_{\Sigma,E,s}$ for every $s \in S$. So the normal form algebra and the quotient term algebra are isomorphic.

The next theorem shows that this is not incidentally so.

Theorem 6.7. *Let specification (Σ, V, E) with finite Church-Rosser $\rightarrow_E^*$ be given. Then the quotient term algebra $T_{\Sigma,E}$ and the normal form algebra* NF *are isomorphic.*

Proof. Parts of the proof follow some ideas of Wand [15], who deals with algebras without ok predicates. Let us first characterise $\equiv$ by the following proposition:

$$t \equiv_s t' \Leftrightarrow \mathrm{nf}(t) = \mathrm{nf}(t') \quad \text{for } t, t' \in T_s.$$

($\Leftarrow$): If we have $\mathrm{nf}(t) = \mathrm{nf}(t')$, then

$$t = t1 \rightarrow_s t2 \ldots tn-1 \rightarrow_s \mathrm{nf}(t) = \mathrm{nf}(t')\, {}_s\!\leftarrow tm-1' \ldots t2'\, {}_s\!\leftarrow t1' = t'$$

and so $t \equiv_s t'$.

($\Rightarrow$): If we have $t \equiv_s t'$, then $t = t1 \leftrightarrow_s t2 \leftrightarrow_s t3 \ldots tn-1 \leftrightarrow_s tn = t'$ with each $\leftrightarrow_s = \rightarrow_s$ or $\leftrightarrow_s = {}_s\!\leftarrow$. We prove our proposition by induction on the length n of this derivation.

If $n=0$, then $t=t'$ and so $\mathrm{nf}(t) = \mathrm{nf}(t')$ is valid.

In the induction step we assume $\mathrm{nf}(t1) = \mathrm{nf}(tn-1)$.

If we have $tn-1\, {}_s\!\leftarrow tn$, *then* $tn \rightarrow_s tn-1 \rightarrow_s^* \mathrm{nf}(tn-1) = \mathrm{nf}(t1)$ and so $\mathrm{nf}(t1) = \mathrm{nf}(tn) \Leftrightarrow \mathrm{nf}(t) = \mathrm{nf}(t')$ holds.

If we have $tn-1 \rightarrow_s tn$, then $tn \rightarrow_s^* \mathrm{nf}(tn-1) = \mathrm{nf}(t1)$ holds due to the finite Church-Rosser property and so $\mathrm{nf}(t1) = \mathrm{nf}(tn) \Leftrightarrow \mathrm{nf}(t) = \mathrm{nf}(t')$ is valid.

We now define a family of mappings from the normal form algebra into the quotient term algebra:

$$h : \mathrm{NF} \rightarrow T_{\Sigma,E}, \quad h = \langle h_s \rangle_{s \in S}.$$
$$h_s : t \mapsto [t']$$

We now show: h is a strict, injective and surjective morphism.

- *h respects the operations*:

  Let $\sigma : s1 \times \cdots \times sn \rightarrow s$ and normal forms ti of sort si be given.

$$h_s(\sigma_{\mathrm{NF}}(t1, \ldots, tn)) = h_s(\mathrm{nf}(\sigma(t1, \ldots, tn))) = [\mathrm{nf}(\sigma(t1, \ldots, tn))]$$
$$= [\sigma(t1, \ldots, tn)] = \sigma_{\Sigma,E}([t1], \ldots, [tn]) = \sigma_{\Sigma,E}(h_{s1}(t1), \ldots, h_{sn}(tn)).$$

- *h is strict*:

  Let normal form t of sort s be given. If $\mathrm{ok}_{\mathrm{NF},s}(t) = \textsc{True}$, then there is a t' with $\mathrm{ok}_{T,s}(t') = \textsc{True}$ and $t' \rightarrow_s^* t$. So we have

$$\mathrm{ok}_{\Sigma,E,s}(h_s(t)) = \mathrm{ok}_{\Sigma,E,s}([t]) = \mathrm{ok}_{\Sigma,E,s}([t']) = \textsc{True}.$$

  If $\mathrm{ok}_{\mathrm{NF},s}(t) = \textsc{False}$, then for all terms t' which have t as their normal form, $\mathrm{ok}_{T,s}(t') = \textsc{False}$. This implies $\mathrm{ok}_{\Sigma,E,s}(h_s(t)) = \mathrm{ok}_{\Sigma,E,s}([t]) = \textsc{False}$ due to our characterisation of $\equiv$.

- *h is injective*:

  Let normal forms t, t' of sort s be given. Then we have

$$h_s(t) = h_s(t') \Rightarrow [t] = [t'] \Rightarrow t \equiv_s t' \Rightarrow t = \mathrm{nf}(t) = \mathrm{nf}(t') = t'$$

  due to our definitions of h and $\equiv_E$ and our characterisation of $\equiv_E$.

- *h is surjective*:
  Let $[t] \in T_{\Sigma,E,s}$ be given. Then $h_s(\mathrm{nf}(t)) = [\mathrm{nf}(t)] = [t]$ is valid. □

Appendix A

This example gives a stack of nat description including exception handling:

$0: \to \mathrm{nat}$
$\mathrm{succ}: \mathrm{nat} \to \mathrm{nat}$
$\mathrm{new}: \to \mathrm{stack}$
$\mathrm{push}: \mathrm{stack} \times \mathrm{nat} \to \mathrm{stack}$
$\mathrm{pop}: \mathrm{stack} \to \mathrm{stack}$: *unsafe*
$\mathrm{top}: \mathrm{stack} \to \mathrm{nat}$: *unsafe*
$\mathrm{underflow}: \to \mathrm{stack}$: *unsafe*
$\mathrm{topless}: \to \mathrm{nat}$: *unsafe.*

The carrier sets and the ok predicates look like

$$A_{\mathrm{nat}} = N_0 \cup \{e_{\mathrm{nat}}\} \qquad \mathrm{ok}_{\mathrm{nat}}(n) = (n \in N_0)$$
$$A_{\mathrm{stack}} = N_0^* \cup \{e_{\mathrm{stack}}\} \qquad \mathrm{ok}_{\mathrm{stack}}(s) = (s \in N_0^*).$$

The functions are defined by

$$0_A: \mapsto 0$$
$$\mathrm{succ}_A: n \mapsto \begin{cases} n+1 & \text{if } n \in N_0, \\ e_{\mathrm{nat}} & \text{if } n = e_{\mathrm{nat}}, \end{cases}$$
$$\mathrm{new}_A: \mapsto \varepsilon$$
$$\mathrm{push}_A: (s, n) \mapsto \begin{cases} s.n & \text{if } s \in N_0^* \text{ and } n \in N_0, \\ e_{\mathrm{stack}} & \text{otherwise}, \end{cases}$$
$$\mathrm{pop}_A: s \mapsto \begin{cases} s' & \text{if } s \in N_0^+,\ s = s'.n, \\ e_{\mathrm{stack}} & \text{otherwise}, \end{cases}$$
$$\mathrm{top}_A: s \mapsto \begin{cases} n & \text{if } s \in N_0^+,\ s = s'.n, \\ e_{\mathrm{nat}} & \text{otherwise}, \end{cases}$$
$$\mathrm{underflow}_A: \mapsto e_{\mathrm{stack}}$$
$$\mathrm{topless}_A: \mapsto e_{\mathrm{nat}}.$$

The axioms E using ok variables n and s and unsafe variables $n+$ and $s+$ of sort nat and stack, respectively are given by

$\mathrm{pop}(\mathrm{push}(s, n)) = s$ /* classical stack axioms */
$\mathrm{pop}(\mathrm{new}) = \mathrm{underflow}$
$\mathrm{top}(\mathrm{push}(s, n)) = n$
$\mathrm{top}(\mathrm{new}) = \mathrm{topless}$
$\mathrm{succ}(\mathrm{topless}) = \mathrm{topless}$ /* error propagating axioms */

push(underflow, n+) = underflow
push(s+, topless) = underflow
pop(underflow) = underflow
top(underflow) = topless.

The quotient term algebra $T_{\Sigma,E}$ is isomorphic to the given model. It is now easy to specify an error recovery function

$$\text{recover}:\text{stack}\to\text{stack} \qquad \text{recover}_A: s\mapsto\begin{cases} s & \text{if } s\in N_0^*,\\ \varepsilon & \text{if } s=e_{\text{stack}},\end{cases}$$

by giving the equations recover(s) = s and recover(underflow) = new using ok variable s.

Appendix B

We want to specify an error recovery stack which means pop and top have to yield ok results when applied to certain error stacks, e.g.,

pop(push(new, top(new))) = pop(push(new, topless)) = new
top(push(pop(new), 0)) = top(push(underflow, 0)) = 0.

The signature is the same as in Appendix A, but the carrier sets, where we have especially to decode the error recoverable stacks, and the functions are different.

$$A_{\text{nat}} = N_0 \cup \{e_{\text{nat}}\} \qquad \text{ok}(n) = (n\in N_0)$$
$$A_{\text{stack}} = N_0^* \cup \underline{N}_0 \cup \bar{N}_0^* \cup \{e_{\text{stack}}\} \qquad \text{ok}(s) = (s\in N_0^*)$$
$$0_A: \mapsto 0$$
$$\text{succ}_A: n\mapsto\begin{cases} n+1 & \text{if } n\in N_0,\\ e_{\text{nat}} & \text{if } n=e_{\text{nat}},\end{cases}$$
$$\text{new}_A: \mapsto \varepsilon$$
$$\text{push}_A: (s,n)\mapsto\begin{cases} s.n & \text{if ok}(s)\text{ and ok}(n),\\ \bar{s} & \text{if ok}(s)\text{ and not ok}(n),\\ \underline{n} & \text{if not ok}(s)\text{ and ok}(n),\\ e_{\text{stack}} & \text{if not ok}(s)\text{ and not ok}(n),\end{cases}$$
$$\text{pop}_A: s\mapsto\begin{cases} s' & \text{if ok}(s)\text{ and } s=s'.n,\\ s' & \text{if not ok}(s)\text{ and } s=\bar{s}',\\ e_{\text{stack}} & \text{otherwise},\end{cases}$$
$$\text{top}_A: s\mapsto\begin{cases} n & \text{if ok}(s)\text{ and } s=s'.n,\\ n & \text{if not ok}(s)\text{ and } s=\underline{n},\\ e_{\text{nat}} & \text{otherwise},\end{cases}$$
$$\text{underflow}_A: \mapsto e_{\text{stack}}$$
$$\text{topless}_A: \mapsto e_{\text{nat}}.$$

We now give the axioms E using the same variables as in Appendix A:

$$\begin{array}{l}
\text{pop}(\text{push}(s, n+)) = s \\
\text{pop}(\text{new}) = \text{underflow} \\
\text{top}(\text{push}(s+, n)) = n \\
\text{top}(\text{new}) = \text{topless}.
\end{array}$$

Again, we first describe the carrier sets of $T_{\Sigma,E}$ by the following productions:

$$\begin{array}{lrl}
\langle\text{ok-nat}\rangle & ::= & 0 \mid \text{succ}(\langle\text{ok-nat}\rangle) \\
\langle\text{ok-stack}\rangle & ::= & \text{new} \mid \text{push}(\langle\text{ok-stack}\rangle, \langle\text{ok-nat}\rangle) \\
\langle\text{err-stack}\rangle & ::= & \text{underflow} \mid \\
 & & \text{push}(\langle\text{err-stack}\rangle, \langle\text{err-nat}\rangle) \mid \\
 & \text{(i)} & \text{push}(\langle\text{err-stack}\rangle, \langle\text{ok-nat}\rangle) \mid \\
 & \text{(ii)} & \text{push}(\langle\text{ok-stack}\rangle, \langle\text{err-nat}\rangle) \mid \\
 & & \langle\text{err-pop}\rangle \\
\langle\text{err-pop}\rangle & ::= & \text{pop}(\text{underflow}) \mid \\
 & & \text{pop}(\text{push}(\langle\text{err-stack}\rangle, \langle\text{ok-nat}\rangle)) \mid \\
 & & \text{pop}(\text{push}(\langle\text{err-stack}\rangle, \langle\text{err-nat}\rangle)) \mid \\
 & & \text{pop}(\langle\text{err-pop}\rangle) \\
\langle\text{err-nat}\rangle & ::= & \text{topless} \mid \\
 & & \text{top}(\text{underflow}) \mid \\
 & & \text{top}(\text{push}(\langle\text{ok-stack}\rangle, \langle\text{err-nat}\rangle)) \mid \\
 & & \text{top}(\text{push}(\langle\text{err-stack}\rangle, \langle\text{err-nat}\rangle)) \mid \\
 & & \text{top}(\langle\text{err-pop}\rangle).
\end{array}$$

Please note, terms which need the productions (i) or (ii) as the first step in their derivation correspond to the $\underline{N}_0$ and $\bar{N}_0^*$ elements of A_{stack}. The correctness morphism $h: T_{\Sigma,E} \to A$ is defined in the following way:

- $h_{\text{nat}}: T_{\Sigma,E,\text{nat}} \to A_{\text{nat}}$

$$t \mapsto \begin{cases} n & \text{if } t \in L(\langle\text{ok-nat}\rangle) \text{ with } t = \text{succ}^n(0), \\ e_{\text{nat}} & \text{if } t \in L(\langle\text{err-nat}\rangle), \end{cases}$$

- $h_{\text{stack,ok}}: T_{\Sigma,E,\text{stack,ok}} \to A_{\text{stack,ok}}$

$$t \mapsto \begin{cases} \varepsilon & \text{if } t = \text{new}, \\ h_{\text{stack,ok}}(t1).h_{\text{nat}}(t2) & \text{if } t = \text{push}(t1, t2), \end{cases}$$

- $h_{\text{stack,err}}: T_{\Sigma,E,\text{stack,err}} \to A_{\text{stack,err}}$

$$t \mapsto \begin{cases} \overline{h_{\text{stack,ok}}(t1)} & \text{if } t = \text{push}(t1, t2) \text{ with } t1 \in L(\langle\text{ok-stack}\rangle), \\ \underline{h_{\text{nat}}(t2)} & \text{if } t = \text{push}(t1, t2) \text{ with } t2 \in L(\langle\text{ok-nat}\rangle), \\ e_{\text{stack}} & \text{otherwise.} \end{cases}$$

Using this morphism we can prove that the given specification is *correct* with respect to the model algebra A.

If we want to get a *strongly correct* specification, we have to add the following axioms using additionally ok variables *n1* and *n2* of sort nat:

$$\begin{aligned}
&\text{pop(underflow)} = \text{underflow}\\
&\text{pop(push(underflow}, n+)) = \text{underflow}\\
&\text{top(underflow)} = \text{topless}\\
&\text{top(push(}s+, \text{topless)} = \text{topless}\\
&\text{push(underflow, topless)} = \text{underflow}\\
&\text{push(push(}s, \text{topless)}, n) = \text{push(underflow}, n)\\
&\text{push(push(}s, \text{topless), topless)} = \text{underflow}\\
&\text{push(push(underflow}, n1), n2) = \text{push(underflow}, n2)\\
&\text{push(push(underflow}, n), \text{topless)} = \text{underflow}.
\end{aligned}$$

Then the quotient term algebra $T_{\Sigma,E}$ is described by

$$\begin{aligned}
&\langle\text{ok-nat}\rangle && ::= 0 \mid \text{succ}(\langle\text{ok-nat}\rangle)\\
&\langle\text{ok-stack}\rangle && ::= \text{new} \mid \text{push}(\langle\text{ok-stack}\rangle, \langle\text{ok-nat}\rangle)\\
&\langle\text{err-nat}\rangle && ::= \text{topless}\\
&\langle\text{err-stack}\rangle && ::= \text{underflow} \mid\\
& && \quad \text{push}(\langle\text{ok-stack}\rangle, \text{topless}) \mid\\
& && \quad \text{push(underflow}, \langle\text{ok-nat}\rangle)
\end{aligned}$$

Acknowledgment

We thank G. Engels and U. Pletat for their earlier work in the field and for many fruitful discussions.

References

[1] J.A. Goguen, J. W. Thatcher and E.G. Wagner, An initial algebra approach to the specification, correctness and implementation of abstract data types, in: R.T. Yeh, ed., *Current Trends in Programming Methodology, Vol. IV* (Prentice-Hall, Englewood Cliffs, NJ, 1978) pp. 80–149.

[2] A.P. Black, Exception handling and data abstraction, IBM Res. Rept. RC 8059, 1980.

[3] H.-D. Ehrich, On the theory of specification, implementation and parametrisation of abstract data types, *J. ACM* **29** (1982) 206–227.

[4] H. Ehrig, H.-J. Kreowski, J.W. Thatcher, E.G. Wagner and J.B. Wright, Parameter passing in algebraic specification languages, *Proc. Workshop on Algebraic Specification*, Aarhus, 1981; *Theoret. Comput. Sci.*, **28**(1,2) (1984) 45–81.

[5] G. Engles, U. Pletat and H.-D. Ehrich, Handling errors and exceptions in the algebraic specification of data types, *Osnabrücker Schriften zur Mathematik, Reihe Informatik, Heft 3* (Univ. Osnabrück, 1981).

[6] J.V. Guttag, E. Horowitz and D.R. Musser, Some extensions to algebraic specifications, *SIGPLAN Notices* **12**(3) (1977) 63–67.

[7] J.A. Goguen, Abstract errors for abstract data types, in: E.J. Neuhold, ed., *Proc. Conf. on Formal Description of Programming Concepts* (North-Holland, Amsterdam, 1978).

[8] J.A. Goguen, Order sorted algebras: Exception and error sorts, coercions and overloaded operators, Semantics and Theory of Computation Report No. 14, University of California, Los Angeles, 1978.

[9] J.V. Guttag, The specification and application to programming of abstract data types, Tech. Rept. CSRG-59, Univ. of Toronto, 1975.

[10] G. Huet, Confluent reductions: Abstract properties and applications to term rewriting systems, *Proc. 18th IEEE Symp. on Foundations of Computer Science* (1977) pp. 30-45.

[11] B. Liskov and S. Zilles, Programming with abstract data types, *SIGPLAN Notices* **9**(4) (1974) 50-59.

[12] M.E. Majster, Treatment of partial operations in the algebraic specification technique, *Proc. Specifications of Reliable Software* (IEEE, 1979) pp. 190-197.

[13] M.J. O'Donnell, *Computing in Systems Described by Equations*, Lecture Notes in Computer Science **58** (Springer, New York, 1977).

[14] B.K. Rosen, Tree-manipulating systems and Church-Rosser Theorems, *J. ACM* **20** (1973) 160-187.

[15] M. Wand, Algebraic theories and tree rewriting systems, Tech. Rept. No. 66, Indiana Univ., Bloomington, IN, 1977.

[16] M. Wirsing, P. Pepper, H. Partsch, W. Dosch and M. Broy, On hiérarchies of abstract data types, Bericht TUM-I8007, Institut für Informatik, Technische Univ. München, 1980.