Acta Informatica 19, 235-253 (1983)

An Operational Semantics for Specifications of Abstract Data Types with Error Handling

G. Engels[1], U. Pletat[2], and H.-D. Ehrich[3]

[1] Institut für Angewandte Informatik, Universität Osnabrück, Postfach 4469, D-4500 Osnabrück (Fed. Rep.)
[2] Institut für Informatik, Universität Stuttgart, Azenbergstraße 12, D-7000 Stuttgart 1 (Fed. Rep.)
[3] Lehrstuhl B für Informatik, TU Braunschweig, Gauss-Straße 12, D-3300 Braunschweig (Fed. Rep.)

Summary. A new approach to an operational treatment of errors and exceptions in specifications of abstract data types is presented. Considering a specification as a term rewriting system, we define an operational semantics and give conditions that are sufficient for its well-definedness (Church-Rosser property). Also, we give conditions that are sufficient for the termination of reduction strategies, respecting the specified error and exception handling.

1. Introduction

Algebraic specifications of software modules have been discussed widely from several viewpoints. The attractive feature of this approach is that it is practically useful in many cases of interest [14, 2, 8, 3, 11], that its mathematical foundations are well investigated [1, 6, 4, 5, 7], and that there is also an operational treatment in terms of term rewriting systems that has been studied extensively and offers quite an amount of useful results [16, 21, 19, 22, 15]. There are, however, practical needs that are not yet well understood, both in terms of algebraic and operational semantics. One of the most challenging problems is error and exception handling. Although some theoretical aspects of this problem have been addressed [1, 12, 13, 18], a treatment is still missing that is satisfactory for both theoreticians and practitioners in the field.

In this paper, we present a new approach to an operational treatment of errors and exceptions that we think is acceptable from a practical viewpoint since it is simple and allows all perceivable forms of error handling. Especially, we do not force errors to propagate in any case as [12, 2], and [3] do. Instead, we offer the flexibility to eliminate or recover errors under certain circumstances. Also, we avoid to specify error handling totally explicitly as [1] do since this results in very tedious specifications.

In the next section, we present the ideas of our approach informally by means of an example. In Sect. 3, we develop some formal background needed in the rest of the paper. We introduce ok- and error sorts and variables of ok- and error sorts, respectively, and give a modified definition for the syntax of terms. The main part is Sect. 4 giving an operational semantics for our version of specification in terms of term rewriting systems. Our results are based on those of O'Donnell [19] about sufficient syntactical criteria for uniqueness of normal forms and termination of certain reduction strategies. Because of our modified syntax of terms we need some additional criteria to show analogous results for term rewriting systems derived from algebraic specifications with error handling. In Sect. 5, we illustrate our results by revisiting the introductory example. We give the operational model for the example explicitly in order to show what the specification really defines.

The issues discussed in this paper have been treated in more detail in [9].

2. Motivation

An algebraic specification of an abstract data type consists of a finite set of sorts, a finite set of function symbols on different domains and ranges, and a finite set of axioms built from these function symbols and from variables. A simple well-known example is the following specification of the abstract data type *stack*.

Example 2.1. Let {stack, nat} be a set of sorts.

NEW: $\to$stack
0: $\to$nat
PUSH: stack $\times$ nat$\to$stack
POP: stack$\to$stack
TOP: stack$\to$nat

POP(NEW)=error	(1)
POP(PUSH(S, E))=S	(2)
TOP(NEW)=error	(3)
TOP(PUSH(S, E))=E	(4)

The set of axioms is often called semantic part of a specification and the axioms can be used to derive the equality of constant terms built from function symbols without variables. For that purpose, the variables in the axioms are substituted by constant terms to get constant axioms. Then the derivation of equalities is a reduction process, using the constant axioms as rewrite rules from left to right, for instance:

$$\text{TOP(PUSH(POP(PUSH(NEW,0)),0))} \xrightarrow[\text{using axiom (2)}]{}$$
$$\text{TOP(PUSH(NEW,0))} \xrightarrow[\text{using axiom (4)}]{} 0$$

The reduction process terminates if no further axiom can be applied.

In the above example the application of the two function symbols POP and TOP to an empty stack yields an error or exception situation. This is indicated by the two "error" axioms (1) and (3). A term containing an error will be called *error term.*

Unfortunately, undesired contradictions can be derived if functions yielding an error situation in some cases or error axioms occur during the reduction process. A detailed discussion how such contradictions appear can be found in [1] or [18]. The problems arise because it is possible to substitute the variables in the axioms (2) and (4) by constant terms, which can be reduced to error terms. For example, it is allowed to substitute the variable E in axiom (2) by TOP(NEW). This shows that the existing syntactical description means are not powerful enough to distinguish between so-called "error" axioms and axioms describing situations without error terms as it is intended in axiom (2) or (4).

There exist several approaches to modify the given specification in order to guarantee that variables may only be substituted by terms which cannot be reduced to error terms. For example, in [1] so-called OK-predicates OK_s for each sort s are introduced and all axioms are given in an error-conditioned form. This means that an axiom holds only if the OK-predicates of terms substituted for variables are true. To get a correct specification in the sense of [1] a lot of function symbols and axioms have to be added to the specification. In [14] the given specification is extended by a so-called restriction specification indicating whether the value of an operation is well-defined or not. There exist further approaches (e.g. [18]), but all of them are very tedious or restrictive.

In this paper, we shall describe a new simple way of specifying error and exception situations in abstract data types. In contrast to other authors (e.g. [12]) we don't restrict ourselves on error propagation, but allow all possible forms of error handling, including error recovery. The main idea of our approach is the introduction of two kinds of sorts, *ok-sorts* $s(-\mathrm{ok})$ and *error sorts* $s\,\&\,\mathrm{err}$, and the declaration of a partial ordering on the set of sorts being described by the Hasse diagram

$$\begin{tikzcd}
s\,\&\,\mathrm{err} \arrow[d, dash] \\
s(-\mathrm{ok}).
\end{tikzcd}$$

Goguen described a similar idea in [13] in order to handle coercions and error situations. He introduced a strict lower semi-lattice s where for each ground sort s there exist three sorts s, $s-\mathrm{ok}$, and $S\,\&\,\mathrm{err}$ related according to the following Hasse diagram

$$\begin{tikzcd}
 & s \arrow[dl, dash] \arrow[dr, dash] & \\
s-\mathrm{ok} \arrow[dr, dash] & & s\,\&\,\mathrm{err} \arrow[dl, dash] \\
 & \perp &
\end{tikzcd}$$

Furthermore he requires the existence of function symbols $f\colon u' \to v'$ for each function symbol (e.g.) $f\colon u \to v$ where u, $v \in S$ and $u' \leqq u$ and $v \leqq v'$. This implies that we have at least the function symbols POP: stack $\to$ stack, POP: stack $-$ ok $\to$ stack and POP: stack&err $\to$ stack in our example of the specification of the stack. This overloading of function symbols yields specifications where it is difficult to see which function symbol really occurs in a term.

Now we give a second specification of the abstract data type *stack* and explain our treatment of error and exception situations.

Example 2.2. Let {stack, stack & err, nat, nat & err} be a set of sorts.

0: →nat
NEW: →stack
PUSH: stack × nat→stack
POP: stack→stack & err
TOP: stack→nat & err
ERR "UNDERFLOW": →stack & err
ERR "NO-ENTRY": →nat & err

POP(PUSH(S, E)) = S (1)
POP(NEW) = ERR "UNDERFLOW" (2)
POP(ERR "UNDERFLOW") = ERR "UNDERFLOW" (3)
TOP(PUSH(ST, E)) = E (4)
TOP(NEW) = ERR "NO-ENTRY" (5)

where S is a variable of sort stack, ST a variable of sort stack & err and E a variable of sort nat.

The error sorts always have the form s & err, where s is the corresponding ok-sort. In analogy to the distinction of two kinds of sorts we have two kinds of functions symbols regarding their range sorts. All function symbols describing normal situations have ok-range sorts, e.g. NEW, 0, PUSH. All function symbols that may introduce or propagate an error situation have error range sorts, e.g. POP and TOP. Error situations are denoted by so-called error constants that have the form ERR "..." and an error sort as range sort.

In order to allow the construction of terms like POP(ERR "UNDERFLOW") (see axiom (3)), we generalize the usual syntax of terms. This means for example, that the error constant ERR "UNDERFLOW" with an error sort as range sort is allowed to occur as argument of the function symbol POP, which has an ok-domain sort. On the other hand, the argument may have an error sort while the domain sort is an ok-sort (see for example the variable ST in axiom (4)).

Introducing two kinds of sorts implies that we have also two kinds of variables: variables of an ok-sort and variables of an error sort. We shall use this concept to restrict the generation of constant axioms from axioms in the following way: variables of an ok-sort may only be substituted by constant terms which can never be reduced to error terms. We shall show that in our approach it will be possible to decide without further reduction steps that a term cannot be reduced to an error term. In axiom (1) both variables have an ok-sort; this implies that no error terms may be substituted for the variables. Therefore, we shall call this a *normal situation.* In axiom (2) and (5) we have two *error introduction situations* since the righthand sides of the axioms contain error constants, in contrast to the lefthand sides. Axiom (3) gives an example of an *error propagation situation*. The error constant occurs on the lefthand side as well as on the righthand side of the axiom. In axiom (4) the variable ST may be substituted by any term of sort stack or stack & err. This means especially that the variables ST may be substituted by an error term. Since the variable ST doesn't occur on the righthand side, axiom (4) describes a so-called *error recovery situation*.

These are only a few examples of possible error handling situations in specifications of abstract data types. But we hope that these examples show how it is possible to specify different forms of error handling by the choice of variables and sorts as domains and ranges.

Now we shall formalize our ideas and define an operational semantics for specifications including error handling. Furthermore, we shall investigate reduction strategies that respect the specified error handling. In Sect. 5 we shall return to Example 2.2 again.

3. Basic Definitions

In this section we give some basic definitions for signature, terms and specifications of an abstract data type. First we introduce a set of sorts S with a partial order relation: A set S is called a *structured set of sorts* (induced by a set of sorts S_0) iff S contains S_0 and for each s of S_0 there is an s & err in S. We call a sort s of S_0 *ok-sort* and s & err *error sort*. Furthermore there is a *partial order relation* $\leqq$ defined on a structured set of sorts by: for all $s, s' \in S$ we have $s' \leqq s$ iff s' and s are identical or s' is an ok-sort and $s = s'$ & err is an error sort.

Two sorts $s, s' \in S$ are called *comparable* ($s \sim s'$) iff $s \leqq s'$ or $s' \leqq s$.

Example 3.1. Let S_0 be the set of ok-sorts {stack, nat}.

Then the structured set of sorts S induced by S_0 is {stack, nat, stack & err, nat & err}, and we have for example:

stack $\leqq$ stack, nat $\leqq$ nat & err, nat & err $\sim$ nat, but not stack $\sim$ nat.

A *signature* $\Sigma = \langle \Sigma_{w,s} \rangle_{w \in S^*, s \in S}$ is an $S^* \times S$-indexed family of function symbols. Let $V = \langle V_s \rangle_{s \in S}$ be an S-indexed family of sets of *formal variables* with $\Sigma \cap V = \emptyset$.

In order to simplify the notation of subterm replacements we represent terms as trees.

Definition 3.2. A *term domain* D is a finite subset of N^* such that the empty word ε is an element of D and D contains x and $x.(n)$ if it contains $x.(1)$ and $x.(n+1)$, respectively, for $n \geqq 1$.

An element of D is called *node address*; ε is called *root address*. Intuitively speaking, natural numbers from 1 to n are used as selectors for the n subtrees of a node, and a node address is the sequence of selectors from the root to that node.

In the sequel we consider terms as functions $A\colon D \to \Sigma \cup V$ where D is a term domain, sometimes denoted by *dom* A. We call $ns(A, x)$ the *node sort* s (at node address x) iff $x \in dom\,A$ and $Ax \in \Sigma_{w,s}$ ($w \in S^*$) or $Ax \in V_s$; for $x = \varepsilon$ we write only $ns(A)$ and call this *root sort* of A.

In addition to the usual construction of terms over Σ and V we allow the argument sort at a node address to be comparable to the source sort of the function symbol standing at this node. This yields the following definition of terms:

Definition 3.3. A function $A: D \to \Sigma \cup V$ is called a *disordered* (Σ, V)*-term* iff (1) D is a term domain
(2) for all $x \in D$
(a) if $Ax \in \Sigma_{w,s}$ where $s \in S$, $w = s1 \ldots sn \in S^*$ ($n \geq 0$) then $x.(i) \in D$ and $ns(A, x.(i)) \sim si$ holds for all $i \in \{1, \ldots, n\}$ and $x.(n+1) \notin D$
or (b) $Ax \in V_s$ where $s \in S$ and $x.(1) \notin D$.

In the following examples we illustrate the notion of term using the signature of Example 2.2.

Example 3.4. $D := \{\varepsilon, (1), (1,1), (1,2)\}$ is a term domain and a disordered (Σ, V)-term $A: D \to \Sigma \cup V$ can be represented as a tree as follows:

$$A: \quad \begin{tikzcd}
 & \mathrm{POP} \arrow[d, dash, "1"'] & \\
 & \mathrm{PUSH} \arrow[dl, dash, "1"'] \arrow[dr, dash, "2"] & \\
\mathrm{NEW} & & 0
\end{tikzcd}$$

Due to this modified term construction it may happen in a disordered (Σ, V)-term A that at node address $x.(i)$ the node sort $ns(A, x.(i))$ is an error sort and the corresponding sort si of the function symbol $Ax \in \Sigma_{s1\ldots sn,s}$ is an ok-sort. In this case, the node address $x.(i)$ is called a *real place of disorder* in a disordered (Σ, V)-term A. If there exists no real place of disorder in a disordered (Σ, V)-term A, A is called an *ordered* (Σ, V)*-term.*

(Σ, V)-terms without variables are called *Σ-terms.*

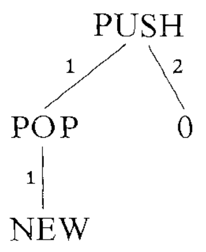
We give some examples for disordered Σ-terms using both string and tree notation to describe terms.

Example 3.5

(1) PUSH(PUSH(PUSH(NEW, 0), 0), 0) is an ordered Σ-term with an ok-root sort.

(2)
$$\begin{tikzcd}
 & \mathrm{POP} \arrow[d, dash, "1"'] & \\
 & \mathrm{PUSH} \arrow[dl, dash, "1"'] \arrow[dr, dash, "2"] & \\
\mathrm{NEW} & & 0
\end{tikzcd}$$
is an ordered Σ-term with an error root sort.

(3)



is a disordered Σ-term that is not ordered. (1) is a real place of disorder.

Now we are able to formalize the two notions ok-term and error-term introduced in Sect. 2. Since all function symbols that have an error sort as range sort may yield an error situation for certain arguments (e.g. the function symbol POP), only ordered Σ-terms with an ok-root sort represent *ok-terms*. On the other hand, all disordered Σ-terms with an error root sort or a real place of disorder in their term domain represent *error terms*. In this sense, the first term in Example 3.5 is an ok-term, while the second and third term are error terms.

Since we define a term as a function, it is easy to denote the subterm of a term A at a node address x of $dom\,A$.

Definition 3.6. The *subterm* A/x of a (Σ, V)-term A at $x \in dom\,A$ is the function A/x: $D_x \to \Sigma \cup V$ with $D_x := \{y \in N^* \mid x \cdot y \in dom\,A\}$ and $(A/x)y := A(x \cdot y)$ for all $y \in D_x$.

In order to compare node addresses in a term domain, we introduce the following relation *anc* on N^*: for $x, y \in N^*$ let x *anc* y iff there is a $w \in N^*$ with $x \cdot w = y$. Then x is called *prefix* of y.

If x *anc* y and $x \neq y$ holds, we write x $anc \neq y$ and call x *proper prefix* of y. Two addresses x and y are said to be *independent* ($x \perp y$) iff x is not a prefix of y and y is not a prefix of x. A subset M of N^* is called *independent* ($\perp M$) iff any two distinct elements from M are independent. Two subsets $M, N \subseteq N^*$ are *independent* ($M \perp N$) iff for all $x \in M$, $y \in N$ we have $x \perp y$.

The replacement of subterms in disordered (Σ, V)-terms is now defined as follows:

Definition 3.7. Let A, B be disordered (Σ, V)-terms, let $x \in dom\,A$ and $ns(A, x) \sim ns(B)$. The disordered (Σ, V)-*term* $A(x \leftarrow B)$, where the subtree A/x is replaced by the tree B, is defined by:

(1) $dom\,A(x \leftarrow B) := x \cdot dom\,B \cup (dom\,A - x \cdot N^*)$

(2) for all $y \in dom\,A(x \leftarrow B)$

$$A(x \leftarrow B)\,y := \begin{cases} Bz & \text{if } y = x \cdot z \\ Ay & \text{otherwise} \end{cases}$$

Let $M = \{x1, \ldots, xn\} \subseteq dom\,A$ and $\perp M$.

The disordered (Σ, V)-term $A(x \leftarrow B_x \mid x \in M)$ is defined by:

$$A(x \leftarrow B_x \mid x \in M) := A(x1 \leftarrow B_{x1}) \ldots (xn \leftarrow B_{xn}).$$

We introduce two particular types of disordered (Σ, V)-terms.

Definition 3.8. A node address x in the term domain of a disordered (Σ, V)-term A is called *place of disorder* iff x is a real place of disorder or we have the following: $x=\varepsilon$ and Ax is a variable of an error sort.

An *internally ordered* (Σ, V)*-term* is a disordered (Σ, V)-term that has all places of disorder only at the leaves. And an internally ordered (Σ, V)-term is called *disordered at variables* iff all places of disorder are at leaves labelled by variable names.

Based on the well-known equational specification approach, we will use the following definition of specification.

A *specification of an abstract data type* is a 4-tuple $\langle S, \Sigma, V, R\rangle$ where S is a structured set of sorts, Σ is a signature, V is an S-indexed family of sets of formal variables and R is a set of axioms. We restrict ourselves to the case where all axioms are equation schemes of the form $A=B$ where A and B are disordered (Σ, V)-terms with comparable root sorts.

From the set R of axioms, a set $E(R)$ of *constant axioms* can be generated by substituting the variables by Σ-terms as follows. (For *a* (Σ, V)-term A and a variable X let $A^{-1}(X)$ be the set of node addresses x where $Ax=X$).

Definition 3.9. Let $A=B$ be an axiom and let $X1, \ldots, Xn$ be the formal variables occuring in A and B.

A constant axiom $\bar{A}=\bar{B}$ is called *generated* from $A=B$

iff (1) $\bar{A}\equiv A(x\leftarrow\overline{C1}\,|\,x\in A^{-1}(X1))\ldots(x\leftarrow\overline{Cn}\,|\,x\in A^{-1}(Xn))$

(2) $\bar{B}\equiv B(x\leftarrow\overline{C1}\,|\,x\in B^{-1}(X1))\ldots(x\leftarrow\overline{Cn}\,|\,x\in B^{-1}(Xn))$

where for all $i\in\{1, \ldots, n\}$, $\overline{Ci}$ is a disordered Σ-term such that $ns(\overline{Ci})\leqq ns(Xi)$, and $\overline{Ci}$ is an ordered Σ-term if $Xi\in V_s$ has an ok-sort s. ($\equiv$means that the two functions are identical.)

The essential point is that only ordered Σ-terms may be substituted for a variable with an ok-sort. We have already mentioned in Sect. 2 that this restriction has been motivated by the intention to specify several forms of error handling.

Example 3.10. In axiom (1), POP(PUSH(S, E))$=S$, of Example 2.2 the variables S and E are variables of ok-sorts. Thus, they can only be substituted by ok-terms. This means that, for example, the variable S may only be substituted by NEW or PUSH(...(PUSH(NEW, 0), ..., 0).

4. An Operational Semantics for Specifications

In this section we define an operational semantics for specifications of abstract data types by transforming a specification into a term replacement system. The well-known normal form semantics of a term replacement system – if it is well-defined – will then be used as an operational semantics for specifications. The following diagram illustrates our approach.

$$\begin{tikzcd}
\text{specification of an abstract data type} \arrow[r, "\text{transformation}"] \arrow[dr, swap, "\text{operational semantics}"] & \text{term replacement system} \arrow[d, "\text{normal form semantics}"] \\
 & \text{normal form algebra}
\end{tikzcd}$$

We describe properties of a term replacement system which guarantee the well-definedness of the normal form semantics and allow to compute the normal form semantics effectively.

Furthermore, we describe properties of a specification which are sufficient for these corresponding properties of a term replacement system and can be checked syntactically.

4.1. Term Replacement Systems

A term replacement system (TRS) is a triple $\langle S, \Sigma, E\rangle$ where S is a structured set of sorts, Σ is a signature and E is a set of rules. *Rules* have the form $\bar{A} \to \bar{B}$, where $\bar{A}$ and $\bar{B}$ are disordered Σ-terms with comparable root sorts.

In a TRS the set of rules has to form a partial function on the set of disordered Σ-terms. These rules are intended to describe which subterm replacements are allowed within a term replacement system. Each set of rules E induces a *reduction relation* $\xrightarrow{E} = \langle \xrightarrow{E}_s \rangle_{s\in S}$ on all disordered Σ-terms by: $\bar{A} \xrightarrow{E}_s \bar{B}$ iff $ns(\bar{A}) \sim s$ holds and there is an $x \in dom\,\bar{A}$ where $\bar{A}/x \to \bar{C} \in E$ and $\bar{B} \equiv \bar{A}(x \leftarrow \bar{C})$. x is called a *redex* in $\bar{A}$ iff $\bar{A}/x$ is the lefthand side of some rule.

Sometimes we write $\bar{A} \xrightarrow[x]{E} \bar{B}$ instead of $\bar{A} \xrightarrow{E}_s \bar{B}$. By $\xrightarrow{E}{}^*$ we denote the reflexive and transitive closure of $\xrightarrow{E}$, and $\bar{A} \xrightarrow{E}{}^* \bar{B}$ means that there exists an $s \in S$ where $\bar{A} \xrightarrow{E}{}^*_s \bar{B}$.

If $M = \{x1, \ldots, xn\}$ is an independent set of redexes in $\bar{A}$, we abbreviate the notation of the sequence $\bar{A} \xrightarrow[x1]{E} \bar{A}1 \xrightarrow[x2]{E} \ldots \xrightarrow[xn]{E} \bar{A}n$ by $\bar{A} \xrightarrow[M]{E} \bar{A}n$ and call it a *reduction step.*

A sequence of reduction steps is called a *reduction sequence.*

If there exists no redex in the term domain of a disordered Σ-term, this Σ-term is called *normal form.* The sets of normal forms $N_{\Sigma,s} := \{\bar{A} \mid \bar{A}$ is a disordered Σ-term where $ns(\bar{A}) \sim s$ and there is no $\bar{B}$ such that $\bar{A} \xrightarrow{E}_s \bar{B}\}$ for each s of S will be the carrier sets of the normal form semantics of a TRS. A disordered Σ-term $\bar{B}$ is called *normal form of a Σ-term $\bar{A}$* (denoted by $NF(\bar{A})$) iff $\bar{A} \xrightarrow{E}{}^*_s \bar{B}$ and $\bar{B}$ is a normal form.

The normal form is uniquely determined if the set E of rules has the *Church-Rosser property wrt* $\xrightarrow{E}$; i.e. that for $\bar{A} \xrightarrow{E}{}^* \bar{B}$ and $\bar{A} \xrightarrow{E}{}^* \bar{C}$ there always is a term $\bar{D}$ with $\bar{B} \xrightarrow{E}{}^* \bar{D}$ and $\bar{C} \xrightarrow{E}{}^* \bar{D}$.

$$\begin{tikzcd}
 & \bar{A} \arrow[dl, swap, "E\,*"] \arrow[dr, "E\,*"] & \\
\bar{B} \arrow[dr, swap, "E\,*"] & & \bar{C} \arrow[dl, "E\,*"] \\
 & \bar{D} &
\end{tikzcd}$$

Now we can give the definition of the normal form semantics for a term replacement system $\langle S, \Sigma, E\rangle$.

Definition 4.1. Let $\xrightarrow{E}$ be the reduction relation induced by E and let E have the Church-Rosser property *wrt* $\xrightarrow{E}$.

The *normal form semantics* of a term replacement system $\langle S, \Sigma, E\rangle$ is a pair $\langle N_\Sigma, F\rangle$ where

(1) $N_\Sigma = \langle N_{\Sigma,s}\rangle_{s\in S}$ is the S-indexed family of carrier sets and

(2) $F = \langle F_{w,s}\rangle_{w\in S^*, s\in S}$ contains for each function symbol $\sigma \in \Sigma_{s1\ldots sn,s}$ a function $\delta_\sigma : N_{\Sigma,s1} \times \ldots \times N_{\Sigma,sn} \to N_{\Sigma,s}$ in $F_{s1\ldots sn,s}$ defined by $\delta_\sigma(t1, \ldots, tn) := NF(\sigma(t1, \ldots, tn))$.

Assuming the Church-Rosser property, all functions are well-defined, but generally these functions are only partial, since the normal form of a term need not exist.

The normal form semantics of a TRS can be extended to arbitrary disordered Σ-terms by assigning the normal form $NF(\bar{A})$ to a disordered Σ-term $\bar{A}$.

The definition of the operational semantics of a specification can now be formulated. Definition 3.9 shows how each set of axioms R induces a set of constant axioms $E(R)$. By omitting the symmetric property and reading everything from left to right, these axioms can be regarded as rules of a term replacement system, i.e. $\bar{A} = \bar{B}$ will be simplified to $\bar{A} \to \bar{B}$.

Definition 4.2. Let $P = \langle S, \Sigma, V, R\rangle$ be a specification of an abstract data type and let $E(R)$ be the set of generated axioms. Assume that $E(R)$, considered as rules, forms a partial function on the set of disordered Σ-terms and has the Church-Rosser property.

The normal form semantics of the corresponding TRS $\langle S, \Sigma, E(R)\rangle$ is called the *operational semantics* of the specification $P = \langle S, \Sigma, V, R\rangle$.

4.2. Properties of Rules

One advantage of the normal form semantics is the possibility of computing it by syntactical means. To do so, it must be guaranteed that the underlying set of rules has the Church-Rosser property and that the applied computation strategy terminates. A computation strategy is called *terminating* iff the normal form of a given Σ-term will eventually be found if it exists. In order to find syntactical criteria which are sufficient for these properties, we use the results of Rosen [21] and O'Donnell [19] about subtree replacements. First it must be guaranteed that the set E of rules has the Church-Rosser property. Rosen has shown in [21] that the Church-Rosser property is satisfied if the set of rules of a TRS is closed *wrt* a residual map. This residual map is a technical aid used in the proofs to describe how the redexes are rearranged during the reduction steps. We give the following definition:

Definition 4.3. Let $P\perp(N^*)$ be the set of all independent subsets of N^* and E a set of rules.

A *residual map for* E is a function $r: E \to (N^* \to P\perp(N^*))$, defined by:

(1) if x is a redex in $\bar{A}$ and $y \in r\langle \bar{A} \to \bar{B} \rangle x$ holds, then y is a redex in $\bar{B}$

(2) if $x \perp y$ holds, then $r\langle \bar{A} \to \bar{B} \rangle x \perp r\langle \bar{A} \to \bar{B} \rangle y$ holds

(3) $r\langle \bar{A} \to \bar{B} \rangle \varepsilon = \emptyset$ for all $\bar{A} \to \bar{B} \in E$.

Definition 4.4. The set of rules E of a TRS $\langle S, \Sigma, E \rangle$ is called *closed wrt a residual map* r iff for $\bar{A} \to \bar{B} \in E$, $x \in dom\,\bar{A}$, $x \neq \varepsilon$ and $\bar{A} \xrightarrow[x]{E} \bar{A}'$ there is a disordered Σ-term $\bar{B}'$

with (1) $\bar{A}' \to \bar{B}' \in E$ and $\bar{B} \xrightarrow[r\langle \bar{A} \to \bar{B} \rangle x]{E} \bar{B}'$

(i.e.

$$\begin{tikzcd}
& \bar{A} \arrow[dl, "E"', "x"] \arrow[dr, "E", "\varepsilon"'] & \\
\bar{A}' \arrow[dr, "E", "\varepsilon"'] & & \bar{B} \arrow[dl, "E"', "r\langle \bar{A} \to \bar{B} \rangle x"] \\
& \bar{B}' &
\end{tikzcd}$$

)

and (2) $x \perp y$ implies $r\langle \bar{A}' \to \bar{B}' \rangle y = r\langle \bar{A} \to \bar{B} \rangle y$.

Theorem 4.5 (cf. [21, 19]). *Let* $\langle S, \Sigma, E \rangle$ *be a TRS and* r *a residual map for* E. *If* E *is closed wrt* r, *then* E *has the Church-Rosser property.*

The proofs of this theorem and of the following results about termination of reduction strategies heavily depend on the notions of a reduction relation induced by a set of rules E and of a redex. Since our definitions of these notions coincide with those given by O'Donnell, his proofs can immediately be carried over to TRSs with a structured set of sorts.

When computing the normal form of a given disordered Σ-term it depends on the choice of the redexes to be reduced in each reduction step whether an existing normal form will be found during the reductions. Computation strategies are also called *reduction strategies*, and terminating reduction strategies are of particular interest.

One simple strategy is to reduce in each reduction step at all redexes in a disordered Σ-term from innermost to outermost. O'Donnell has shown that this strategy, called *full substitution*, is terminating if the underlying TRS is closed wrt a residual map (cf. [19], Corollary 3).

Another reduction strategy, called *parallel outermost*, produces shorter reduction sequences in general. It is defined by reducing at all outermost redexes in each step. A redex x is an *outermost redex* in a disordered Σ-term $\bar{A}$ iff no proper prefix of x is a redex in $\bar{A}$. The parallel outermost strategy is terminating (cf. [19], Theorem 10), if the set of rules of the underlying TRS is closed *wrt* a residual map r and additionally has the outer property which is defined as follows:

Definition 4.6. Let $\langle S, \Sigma, E \rangle$ be a TRS, let $\bar{A}, \bar{B}$ be Σ-terms and let $x, y, z \in dom\,\bar{A}$ such that x *anc* y *anc*$\neq z$. Let y, z be redexes in $\bar{A}$ and $\bar{A} \xrightarrow[z]{E} \bar{B}$. E is called *outer*

iff, whenever x is no redex in $\bar{A}$, then x is no redex in $\bar{B}$.

4.3. Properties of Rule Schemes

In this section we describe simple syntactical criteria for the set of axioms R, which can be automatically tested and imply that the set of generated rules $E(R)$ forms a partial function on the set of disordered Σ-terms and has the Church-Rosser property. This guarantees the well-definedness of the operational semantics. Furthermore, we give syntactical criteria for a set of axioms which imply the outer property of the set of generated rules. In this case, the termination of the parallel outermost strategy can be guaranteed.

Our investigations base on the results of O'Donnell [19]. But, in contrast to the level of term replacement systems, the introduction of a structured set of sorts, a modified syntax of terms and a new definition for the generation of rules imply that the results of O'Donnell cannot carried over to our approach immediately. Therefore, we need some additional criteria for the axioms on the specification level in order to have corresponding results.

In the sequel we consider only special types of axioms in a specification:

Definition 4.7. An axiom $A=B$ is called a *rule scheme*

iff (1) A is internally ordered and
(2) A is not a single variable and
(3) each variable occurs only once in A and
(4) all variables of B occur in A.

It can now easily be shown that a set of generated rules $E(R)$ forms a partial function on the set of disordered Σ-terms, if the set of rule schemes R has the following property:

Definition 4.8. A set R of rule schemes is called *consistent*

iff for all pairs of rule schemes $A=B$ and $C=D$ from R the following holds: If there are Σ-terms $\bar{A}, \bar{B}, \bar{C}, \bar{D}$ such that $\bar{A}\rightarrow\bar{B}$ and $\bar{C}\rightarrow\bar{D}$ are rules generated from these rule schemes, such that $\bar{A}\equiv\bar{C}$, then the rule schemes $A=B$ and $C=D$ are identical up to renaming of the occuring variables and up to comparability of the variable sorts.
(This means formally:

(1) $dom\,B=dom\,D$ and
(2) $\forall x\in dom\,B$: $\exists w\in S^*$, $s\in S$ with $Bx=Dx\in\Sigma_{w,s}\vee$
$(\exists$ ok-sort s with $Bx, Dx\in V_s\cup V_{s\&err}$
$\wedge A^{-1}(Bx)=C^{-1}(Dx))$.

O'Donnell gives a simple criterion to check the consistency of a set of rule schemes.

Lemma 4.9. *If for any two distinct rule schemes $A=B$ and $C=D$ from R we have that there is an $x\in dom\,A\cap dom\,C$ such that Ax and Cx are distinct function symbols, then R is consistent.*

Moreover, the consistency of a set of rule schemes R guarantees the existence of a residual map for a set of generated rules $E(R)$ as follows. For each rule $\bar{A} \to \bar{B}$, generated from the rule scheme $A = B$, the residual map is defined by:

$$r\langle \bar{A} \to \bar{B} \rangle x := \begin{cases} \{y \cdot z \mid y \in B^{-1}(Av)\} & \text{if } x = v \cdot z \text{ and } Av \in V \\ \emptyset & \text{otherwise.} \end{cases}$$

We call this the residual map generated by R.

In Sect. 4.2. we have said that the Church-Rosser property of a set of rules E is satisfied, if E is closed wrt a residual map. Now we describe syntactical criteria for a set of rule schemes R which imply the closure of the set of generated rules $E(R)$.

O'Donnell introduces the following property for a set of rule schemes and gives a simple sufficient criterion for this property.

Definition 4.10 (cf. [19], Def. 48). Let R be a set of rule schemes, $A = B$ and $C = D$ of R, let $X1, \ldots, Xn$ be the formal variables occuring in A, let $\bar{A} \to \bar{B}$ be a rule generated by substituting $X1, \ldots, Xn$ by disordered Σ-terms $\bar{H}1, \ldots, \bar{H}n$, and let $\bar{C} \to \bar{D}$ be generated from $C = D$.

R is called *nonoverlapping* iff, in any such situation and for any $x \neq \varepsilon$, $A/x \equiv \bar{C}$ implies that there is a $y \in dom A$, $az \in N^*$, and an $i \in \{1, \ldots, n\}$ such that $Ay = Xi$ and $x = y \cdot z$.

Lemma 4.11. *If for any two not necessarily distinct rule schemes $A = B$ and $C = D$ of R and for all x of dom C such that $x \neq \varepsilon$ we have $A\varepsilon \neq Cx$, then R is nonoverlapping.*

Example 4.12. All axioms in Example 2.2 are rule schemes and the set of rule schemes is consistent and nonoverlapping.

O'Donnell shows that consistency and the nonoverlapping property of a set R of rule schemes are sufficient for the closure of the set of generated rules $E(R)$ wrt the residual map generated by R (cf. [19], Theorem 17). Since we require that only ordered Σ-terms may be substituted for variables of an ok-sort we need an additional property.

Definition 4.13. A rule scheme $A = B$ *propagates order* iff, whenever A is disordered at variables, we have the following:

(1) B is disordered at variables and

(2) for all places of disorder x in B holds that $A^{-1}(Bx)$ is a real place of disorder in A and

(3) $B\varepsilon \in \Sigma \Rightarrow ns(B) \leq ns(A)$.

A set R of rule schemes *propagates order* iff all rule schemes from R propagate order.

Conditions (1) and (2) imply that in rules $\bar{A} \to \bar{B}$ generated from $A = B$ the righthand side $\bar{B}$ is always ordered if the lefthand side $\bar{A}$ is ordered. Conditions (2) and (3) imply that for all generated rules $\bar{A} \to \bar{B}$ we have $ns(\bar{B}) \leq ns(\bar{A})$.

It is easy to see that an ordered Σ-term with an ok-root sort remains such a term during the reduction process, if the underlying set of rule schemes propagates order. Thus an ok-term can't be reduced to an error term during further reduction steps.

We use this idea to prove the following theorem:

Theorem 4.14. *Let $P = \langle S, \Sigma, V, R\rangle$ be a specification where R is consistent, nonoverlapping and propagates order.*

Then $E(R)$ is closed wrt a residual map r generated by R.

Proof. Let $X1, \ldots, Xn$ be the variables occuring in the lefthand side A of a rule scheme $A = B$, and let $\bar{A} \to \bar{B}$ be generated from $A = B$ by substituting the variables Xi by disordered Σ-terms $\bar{H}i$ for $i = 1, \ldots, n$. Let $x \neq \varepsilon$ be a redex in $\bar{A}$, i.e. there exists $\bar{A}/x \to \bar{D} \in E(R)$ and $\bar{A} \xrightarrow[x]{E(R)} \bar{A}(x \leftarrow \bar{D})$. Let $\bar{C} := \bar{A}/x$.

If R is nonoverlapping, then there is a $y \in dom A$, a $z \in N^*$, and an $i \in \{1, \ldots, n\}$ such that $Ay = Xi$ and $x = y \cdot z$.

We first show that a second rule $\bar{A}' \to \bar{B}'$ can be generated from $A = B$ such that $\bar{A}' \equiv \bar{A}(x \leftarrow \bar{D})$ and $\bar{B}' \equiv B(w \leftarrow \bar{H}i(z \leftarrow \bar{D}) | w \in B^{-1}(Xi))$. This holds if we can show that $\bar{H}i(z \leftarrow \bar{D})$ can be substituted for Xi instead of $\bar{H}i$.

Case 1. $Xi \in V_s$ has an ok-sort s. Then $\bar{H}i$ is ordered and has root sort s. Since $\bar{C}$ is a subtree of $\bar{H}i$ it follows that $\bar{C}$ is ordered. As R propagates order this implies for $\bar{C} \to \bar{D} \in E(R)$ that $\bar{D}$ is ordered and $ns(\bar{D}) \leqq ns(\bar{C})$ holds. So $\bar{H}i(z \leftarrow \bar{D})$ is ordered, has root sort s, and can be substituted for Xi.

Case 2. If Xi is a variable of an error sort, the proposition is trivial. Now the theorem follows directly from the definition of the residual map generated by the set of rule schemes.

In practice, the signature often has the property that the domains of all function symbols consist only of ok-sorts. In this case condition (2) of Definition 4.13 is always satisfied for a rule scheme.

O'Donnell shows that consistency together with the nonoverlapping property is also sufficient for the outer property of a set of rules ([19], Theorem 8). Again we need an additional property for the set of rule schemes to prove the same result for specifications with error handling. In order to motivate this new property the following considerations are helpful. The property of the rule schemes to propagate order can be viewed as implication 'P(lefthand side) $\Rightarrow$ Q(righthand side)' for two predicates P and Q. In contrast to this we need now a property that represents the implication 'P(righthand side) $\Rightarrow$ Q(lefthand side)' for the same predicates P and Q.

At first we need the following additional property for a rule scheme, which is an analogon to our requirement that all variables occuring on the righthand side of a rule scheme must occur on the lefthand side. Here the following weaker property is sufficient.

Definition 4.15. A rule scheme $A = B$ *propagates error variables* iff for all $y \in dom A$ where Ay is a variable of an error sort there exists a $z \in dom B$ where $Bz = Ay$.

Example 4.16. Rule scheme (4) TOP(PUSH(ST, E)) $= E$ in Example 2.2 doesn't propagate error variables, as the variable ST doesn't occur on the righthand side.

We define two further properties for rule schemes.

Definition 4.17. A rule scheme $A=B$ *propagates places of disorder* iff for all $y \in dom\, A$ where y is a place of disorder there exists a $z \in B^{-1}(Ay)$ where z is a real place of disorder.

Example 4.18. Rule scheme (4) TOP(PUSH(ST, E))$=E$ in Example 2.2 doesn't propagate places of disorder, as $(1,1)$ is a real place of disorder in TOP(PUSH(ST, E)) and the set $B^{-1}(A((1,1,1)))$ is empty.

Definition 4.19. A rule scheme $A=B$ *discovers order* iff, whenever B is disordered at variables, we have the following:

(1) A is disordered at variables and
(2) $ns(A) \leq ns(B)$ and
(3) $A=B$ propagates error variables and places of disorder.

A set R of rule schemes *discovers order* iff all rule schemes from R discover order.

Condition (1) and (3) imply that in rules $\bar{A} \to \bar{B}$ generated from $A=B$ the lefthand side $\bar{A}$ is always ordered if the righthand side $\bar{B}$ is ordered.

Now we can show the following theorem:

Theorem 4.20. *Let $P = \langle S, \Sigma, V, R \rangle$ be a specification, where R is consistent, nonoverlapping, and propagates and discovers order.*

Then the set of generated rules $E(R)$ of the corresponding TRS $\langle S, \Sigma, E(R) \rangle$ has the outer property.

Proof. Let $\bar{A}, \bar{A}'$ be disordered Σ-terms, let $x, y, z \in dom\, \bar{A}$ where x *anc* y *anc*$\neq z$, let x be no redex in $\bar{A}$ and $\bar{A} \xrightarrow[z]{E(R)} \bar{A}'$. Then there is a rule scheme $A'_x = F$ in R from which the rule $\bar{A}'/x \to \bar{F}$ can be generated by substituting the variables $X1, \ldots, Xn$ in A'_x by disordered Σ-terms $\bar{H}1, \ldots, \bar{H}n$.

By assumption, we have that P is closed and that y *anc*$\neq z$ holds. This implies that y is a redex in $\bar{A}'$ and that there is a $w \neq \varepsilon$ where $y \cdot w = z$. Since R has the nonoverlapping property, there exist s, t, and j with $y = x \cdot s \cdot t$ and $A'_x(s) = Xj$.

By assumption, z is a redex in $\bar{A}$; so there exists a rule $\bar{A}/z \to \bar{D}$ and we have $\bar{A}'/z \equiv \bar{D}$.

Let $\bar{H}'j := \bar{A}/(x \cdot s)$; then we have $\bar{H}'j/(t \cdot w) \equiv \bar{A}/z$ and $\bar{H}j \equiv \bar{H}'j((t \cdot w) \leftarrow \bar{D})$.

We now show that a second rule $\bar{A}''/x \to \bar{F}'$ can be generated from $A'_x = F$ by substituting Xi by $\bar{H}'j$ instead of $\bar{H}j$.

Case 1. $Xj \in V_s$ has an ok-sort s. Then $\bar{H}j$ is ordered and has root sort s. Since $\bar{H}j/(t \cdot w) \equiv \bar{D}$ holds, $\bar{D}$ is ordered. As the rule $\bar{A}/z \to \bar{D}$ is in $E(R)$ and R discovers order, we can conclude that $\bar{A}/z$ is ordered and $ns(\bar{A}/z) \leq ns(\bar{D})$ holds. So $\bar{H}'j$ is ordered and, since $w \neq \varepsilon$ holds, $\bar{H}'j$ has the same root sort as $\bar{H}j$.

Thus, Xj can be substituted by $\bar{H}'j$.

Case 2. If Xj is a variable of an error sort, the proposition is trivial.

Because of $\bar{A}''/x = \bar{A}/x$, we have a contradiction to the assumption that x is no redex in $\bar{A}$.

If the domains of all function symbols of the signature consist only of ok-sorts, the property of a rule scheme to discover order can be weakened, since in this case a rule scheme that propagates order propagates places of disorder, too.

5. The Example Revisited

In this section we continue to discuss the specification of the abstract data type *stack* in Example 2.2. At first we can state that all axioms in the specification are rule schemes (cf. Definition 4.6). So we can check whether these rule schemes satisfy the properties described in Sect. 4.3. Since any two rule schemes have a node address with different function symbols in the common term domain of their lefthand sides, the set of rule schemes is consistent (cf. Lemma 4.9). For example, look at the axioms (1) and (2), where PUSH and NEW, respectively, are the function symbols standing at node address (1). The set of rule schemes is nonoverlapping, since the prerequisites of Lemma 4.10 are fulfilled. This can easily be seen as the function symbols POP and TOP stand only at the root node.

Furthermore, it can easily be checked that the set of rule schemes propagates order (cf. Definition 4.12).

So Theorem 4.13 yields the closure of the set of generated rules and therefore the Church-Rosser property. This implies that the operational semantics of the specification of the abstract data type *stack* is well-defined, since the normal form semantics is.

The normal form model for this example can be described inductively as follows: (All normal forms that are ok-terms are called *ok-normal forms* and all normal forms being error terms are called *error normal forms*.)

(1) Ok-normal forms of sort stack are NEW and all terms PUSH(...(PUSH(NEW, 0), ..., 0).
(2) Ok-normal form of sort nat is the term 0.
(3) Error normal forms of sort stack are PUSH(x & err, n), PUSH(x, n & err) and PUSH(x & err, n & err) where x and n are ok-normal forms and x & err and n & err are error normal forms of the appropriate sorts.
(4) Error normal forms of sort stack & err are ERR "UNDERFLOW" and all terms POP^i(y & err) where $i \geqq 1$ and y & err is an error normal form of sort stack.
(5) Error normal forms of sort nat & err are the terms ERR "NO_ENTRY", TOP(PUSH(x &, n & err)), TOP(z & err) where x & is an error or ok-normal form, z & err is an error normal form of sort stack & err and n & err is an error normal form.

Example 5.1

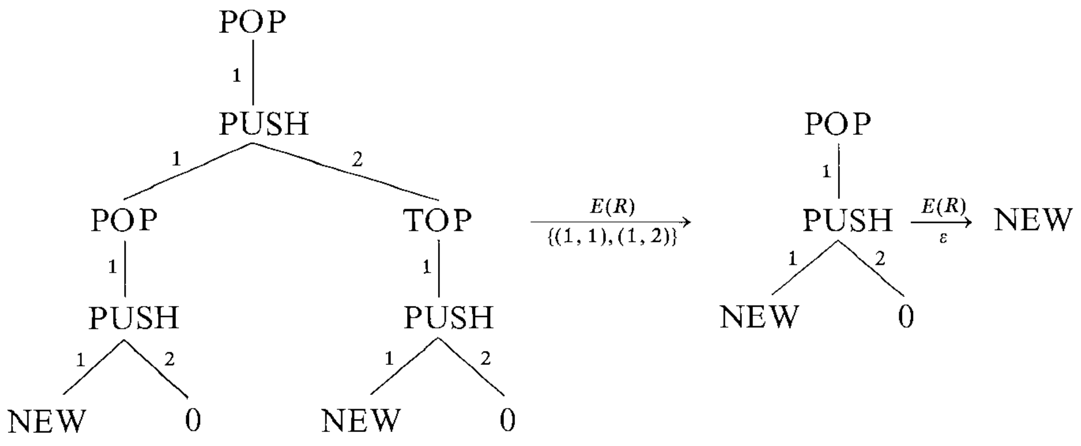
ok-normal forms: 0; NEW; PUSH(NEW, 0); ...

error normal forms: PUSH(ERR "UNDERFLOW", 0)
POP(PUSH(NEW, ERR "NO_ENTRY"))
TOP(POP(PUSH(NEW, ERR "NO_ENTRY")))
TOP(PUSH(NEW, TOP(ERR "UNDERFLOW")))...

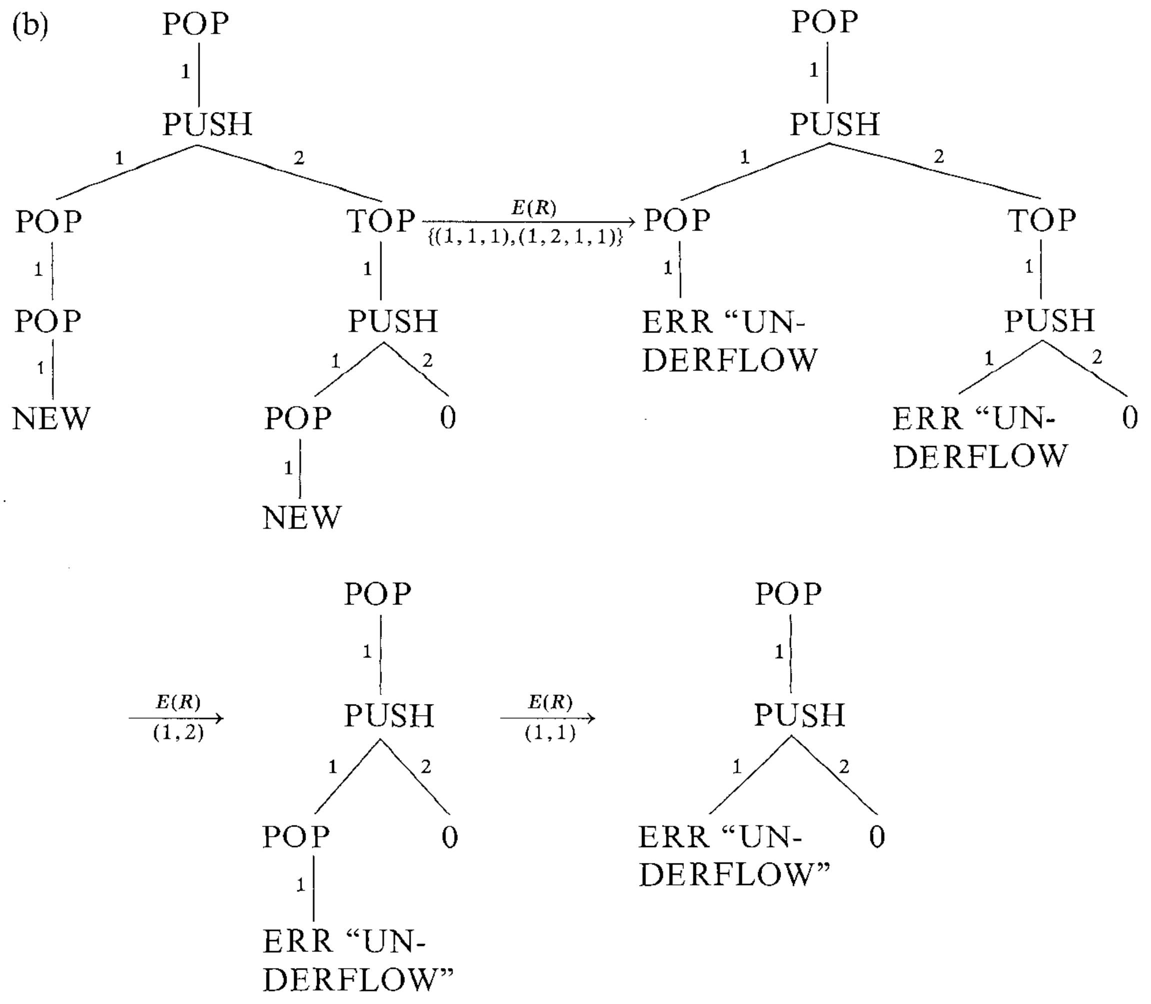
Since we know by Theorem 4.13 that the set of generated rules is closed, the termination of the full substitution reduction strategy is guaranteed for this specification.

We consider some disordered Σ-terms and reduction sequences starting with them:

(a)


POP
PUSH
POP
TOP
PUSH
PUSH
NEW
0
NEW
0
$\xrightarrow[\{(1,1),(1,2)\}]{E(R)}$
POP
PUSH
NEW
0
$\xrightarrow[\varepsilon]{E(R)}$
NEW

(b)



The second example shows how error situations may be recovered or propagated according to the specified error handling in the axioms.

Rule scheme (1) POP(PUSH(S, E)) doesn't discover order, as the right-hand side is a variable of an ok-sort and the lefthand side hasn't an ok-root sort.

So it cannot be guaranteed by our syntactical tests that the set of generated rules has the outer property and that the parallel outermost reduction strategy terminates.

6. Conclusions

In this paper we have defined an operational semantics for specifications of abstract data types with error handling. Our main results are criteria for a specification which are sufficient for this semantics to be well-defined and for certain reduction strategies to terminate, respecting the specified error handling. The reduction strategies and the sufficient criteria have been implemented in a system for interpreting abstract data types (cf. [17, 9]).

We also investigated some other efficient reduction strategies. Especially, we studied a new heuristic strategy that is a mixture of the full substitution and the parallel outermost reduction strategies. We strongly conjecture that this new reduction strategy terminates even if the strong property of a rule scheme to discover order doesn't hold. The main idea of this strategy is to use full substitution during a reduction step only if the outer property may be violated (cf. [10]).

The operational semantics given in this paper is only a partial solution of the problem of error and exception handling in algebraic specifications. The main point missing is a corresponding algebraic semantics, i.e. an algebraic characterization of a semantic algebra associated with a specification, probably a sort of quotient term algebra as in [1], that is isomorphic to the normal form algebra if the latter exists. This is subject to further study. Other open problems to be investigated concern the extension of the concepts of implementation and parametrization of abstract data types such that error and exception handling is included.

Acknowledgements. Thanks are due to the referee for valuable comments and suggestions. We have taken great benefits from discussions with M. Gogolla, U. Lipeck, and K. Drosten. The third author also had helpful discussions with H. Ehrig, H.-J. Kreowski, P. Padawitz, J. Thatcher, and E. Wagner.

References

1. 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. Yeh, R.T. (ed.). Prentice Hall, pp. 80–149, 1978
2. Burstall, R.M., Goguen, J.A.: Putting Theories Together to Make Specifications. Proc. 5th Int. Joint Conf. Artificial Intelligence, MIT, Cambridge, MA, 1977
3. Burstall, R.M., Goguen, J.A.: The Semantics of CLEAR, a Specification Language. Proc. 1979 Copenhagen Winter School Abstr. Software Specifications. Bjorner, D. (ed.). LNCS 86, pp. 292–331. Berlin, Heidelberg, New York: Springer, 1980
4. Ehrich, H.-D.: Extensions and Implementations of Abstract Data Type Specifications. Proc. 7th Symp. MFCS 78. Winkowski, J. (ed.). LNCS 64, pp. 155–164, Berlin, Heidelberg, New York: Springer, 1978
5. Ehrich, H.-D.: On the Theory of Specification, Implementation, and Parametrization of Abstract Data Types. Bericht Nr. 82/79, Abt. Informatik, Universität Dortmund

6. Ehrig, H., Kreowski, H.-J., Padawitz, P.: Stepwise Specification and Implementation of Abstract Data Types. Internal Report, TU Berlin, FB 20, Inst. Software Theor. Inform., 1977
7. Ehrig, H., Kreowski, H.-J., Thatcher, J.W., Wagner, E.G., Wright, J.B.: Parameter Passing in Algebraic Specification Languages. Internal Report, FB 20, TU Berlin, 1980
8. Ehrig, H., Kreowski, H.-J., Weber, H.: Algebraic Specification Schemes for Data Base Systems. Hahn-Meitner-Institut f. Kernforschung Berlin GmbH, HMI-B 266, Februar 1978
9. Engels, G., Pletat, U.: Analyse von Regelschemata für Unterbaumersetzungssysteme. Diplomarbeit, Abt. Informatik, Univ. Dortmund, März 1980
10. Engels, G., Pletat, U., Ehrich, H.-D.: Handling Errors and Exceptions in the Algebraic Specification of Data Types. Osnabrücker Schriften zur Mathematik, Reihe Informatik, OSM I3, 1981
11. Ganzinger, H.: Parameterized Specifications: Parameter Passing and Implementation. Internal Report, EECS-Comp. Sc. Division, UC Berkeley, September 1980
12. Goguen, J.A.: Abstract Errors for Abstract Data Types. Proc. Conf. on Formal Description of Programming Concepts. Neuhold, E.J. (ed.). Amsterdam: North-Holland, 1978
13. Goguen, J.A.: Order Sorted Algebras: Exception and Error Sorts, Coercions, and Overloaded Operators. Semantics and Theory of Computation Reports No. 14, University of California, Los Angeles, December 1978
14. Guttag, J.V., Horowitz, E., Musser, D.R.: Abstract Data Types and Software Validation. Information Sciences Institute, Report RR-76-48, Marina del Rey, CA, August 1976
15. Huet, G.: Confluent Reductions: Abstract Properties and Applications to Term Rewriting Systems. 18th IEEE Symp. Foundations Comput. Sci., pp. 30–45, 1977
16. Knuth, D., Bendix, P.: Simple word problems in universal algebras, Computational Problems in Abstract Data Types. Leech, J. (ed.), Pergamon Press, pp. 263–297, 1970
17. Kraus, T., Wiegand, J.: Entwurf und Implementierung einer Spezifikationssprache für abstrakte Datentypen. Diplomarbeit, Abt. Informatik, Universität Dortmund, 1980
18. Majster, M.E.: Treatment of Partial Operations in the Algebraic Specification Technique. Institut für Informatik, München, In: Proc. Specifications of Reliable Software, IEEE, 1979
19. O'Donnell, M.J.: Computing in Systems Described by Equations. LNCS 58, Berlin, Heidelberg, New York: Springer, 1977
20. Pletat, U., Engels, G., Ehrich, H.-D.: An Operational Approach to Conditional Algebraic Specifications. In: Proc. 7eme Colloque sur les Arbres en Algebre et en Programmation, caap 82, Lille, 1982 (to appear)
21. Rosen, B.K.: Tree-Manipulating Systems and Church-Rosser Theorems. JACM **20**, 160–187 (1973)
22. Wand, M.: Algebraic Theories and Tree Rewriting Systems. Technical Report No. 66, Bloomington, IN: Indiana Univ., 1977

Received July 24, 1981/December 8, 1982