Partiality as nondeterminism

Yngve Lamo   Michał Walicki

Abstract

The paper presents a novel approach, based on multialgebras, to develop specifications with partial operations. It is shown that multialgebras allow to combine various features of several earlier approaches. A wide range of possibilities for error handling is presented. Moreover the possibility of reusing partial algebra specifications is discussed and formally proved based on the concept of (maps of) institutions.

1   Introduction

The problem of partial operations is of fundamental importance in specifying and deriving programs. At the abstract level, one would like to avoid detailed treatment of error situations. However, at the level close to actual implementation, it will often be mandatory to address explicitly possible error situations and to describe the program's behaviour in such situations.

In the tradition of algebraic specifications, there are two main approaches to describing partiality: the partial algebras on the one hand [4, 17, 6], and total algebras with explicit definition domains and error elements, on the other. This later approach comprises manifold variations, including error-algebras [7], labeled algebras [2], order-sorted algebras [8], and techniques using predicates [15] or functions [10] to specify definition domains of operations.

In partial algebras, an undefined term has no interpretation in the carrier. During the refinement process, one makes terms gradually more defined, by adding values for undefined terms. Terms which remain undefined til the very end of the specification process are then understood as errors whose handling is left for the implementation. Partial algebras offer an abstract model due to a strictness assumption which releases the specifier from the need to explicitly treat error situations. This, however, turns out to be a drawback when the specification approaches the implementation level. Strictness of all operations makes explicit error handling difficult, if at all possible. Extensions of partial algebras to handle this level of specification are indicated in [6, 5]. Typically, they involve a translation of partial algebra specifications into some total framework.

The approaches based on total algebras explore the possibilities of explicit error handling, by using specific values as error elements. They are well suited to describe detailed error handling. Their main drawback is that one is from the very beginning forced to treat explicitly error elements.

We thus have, on the one hand, an abstract framework of partial algebras and, on the other hand, the total algebra approaches forcing one to specify error situations explicitly from the very start. We are proposing a single framework capable of addressing both these aspects. Following the idea originally expressed and formalised in [20], we use nondeterminism in specification process as a means of *abstraction* – here, from the unknown results returned by partial operations. We view undefinedness as nondeterminism – a term whithout a well-defined value may result in any value. At the early stages of development, this expresses our ignorance or disinterest in what exactly will happen in a

given – error – situation. This preserves the intuition that any operation in any actual run of a program actually produces some – even if uninteded and unexpected – result. At the later development stages, such nondeterminism may be constrained leading, eventually, to explicit error values and their treatment.

As a model of nondeterminism, we use multialgebras [20, 21]. These are summarized in section 2 which also gives the basic definitions of mappings of institutions [9]. Section 3 shows by a simple example how the problems of partiality can be addressed in $\mathcal{MA}$ – the institution of multialgebras. Since multialgebras offer a non-strict (or rather, not necessarily strict) framework, we also show how to deal with strictness, if it is desirable for some reasons. Section 4 shows two particular mappings of the institution of partial algebras into $\mathcal{MA}$ formalizing also the intuition of reusing partial algebra specifications for further development in a non-strict context. Similar possibilities of embedding are indicated for other frameworks thus suggesting that $\mathcal{MA}$ may be a useful setting for both *comparing* and *combining* advantages of different formalisms for partiality. More detailed discussion and proofs can be found in [11].

2 Preliminaries

Categories are written with the bold font **Cat**, functors with Sans Serif Func, and institutions with script $\mathcal{I}$. $|A|$ denotes the carrier of an algebra A. A signature is usually written Σ and consists of a pair of sets $(\mathbf{S}, \Omega)$ with sort- and operation-symbols. Sequences $s_1, \ldots, s_k$ are usually written as $\overline{s}$. Application of functions are then understood *not* to distribute over the elements, i.e., $f(\overline{s})$ denotes the term $f(s_1, \ldots, s_k)$.

Multialgebras

For an overview of multialgebras see [21, 18]). Multialgebraic signatures (and signature morphisms) are the same as in the classical case and form the category **Sign**. Also terms, $T(\Sigma, X)$, are defined in the usual way. Signature morphisms are extended to terms in the canonical way. A multialgebra is an algebra where operations may be set-valued. $\mathcal{P}(y)$ denotes the powerset of a set y.

Definition 2.1 *Given a signature $\Sigma = (\mathbf{S}, \Omega)$, a Σ-multialgebra A is given by:*
- *a set s^A, the carrier set, for each sort symbol $s \in \mathbf{S}$*
- *a subset $c^A \in \mathcal{P}(s^A)$, for each constant, $c :\to s$*
- *an operation $\omega^A : s_1^A \times \cdots \times s_k^A \to \mathcal{P}(s^A)$ for each symbol $\omega : s_1 \times \cdots \times s_k \to s \in \Omega$*

Operations are defined on sets by pointwise extension.

Notice that multialgebras are strict on the empty set since operations are defined by pointwise extension. Note also that for a constant (or any ground term) c, c^A denotes a subset of the carrier s^A. This allows us to use constants as predicates and to refine such predicates to individual values (denoted by terms) in a way not requiring any additional transition between the two.

As homomorphisms we use weak homomorphisms (see [18] for alternative notions).

Definition 2.2 *Given two Σ-multialgebras A and B, a function $h : |A| \to |B|$ is a (weak) homomorphism if:*
1. *$h(c^A) \subseteq c^B$, for each constant $c :\to s$*
2. *$h(\omega^A(a_1, \ldots, a_n)) \subseteq \omega^B(h(a_1), \ldots, h(a_n))$, for each operation $\omega : s_1 \times \cdots \times s_n \to s \in \Omega$ and for all $a_i \in s_i^A$.*

Multialgebraic specifications are written using the following formulae:

Definition 2.3 *Formulae of multialgebraic specifications are of the following forms* $(t, t' \in T(\Sigma, X))$:

1. *Atomic formulae:*
   - $t \doteq t'$ *(equality), t and t' denote the same one-element set.*
   - $t \prec t'$ *(inclusion), the set interpreting t is included in the set interpreting t'.*

2. $a_1 \vee \ldots \vee a_n \Rightarrow b_1 \wedge \ldots \wedge b_m$, *where either $n > 0$ or $m > 0$ and each a_i and b_j is atomic.*

Given a set of variables X, an assignment is a function $\alpha : X \to |A|$ assigning *individual* elements of the carrier to variables. It induces a unique interpretation $\overline{\alpha} : T(\Sigma, X) \to A$ of terms (with variables from X) in A in the standard way $(\overline{\alpha}(x) = \alpha(x); \overline{\alpha}(\omega(t_1, \ldots, t_n)) = \omega^A(\overline{\alpha}(t_1), \ldots, \overline{\alpha}(t_n)))$.

Definition 2.4 *[Satisfaction] Given an assignment $\alpha : X \to |A|$:*

1. $A \models_\alpha t \doteq t'$ *iff* $\alpha(t) = \alpha(t') = \{e\}$, *for some* individual value $e \in |A|$
2. $A \models_\alpha t \prec t'$ *iff* $\alpha(t) \subseteq \alpha(t')$
3. $A \models_\alpha a_1 \vee \ldots \vee a_n \Rightarrow b_1 \wedge \ldots \wedge b_m$ *iff* $\exists i : 1 \leq i \leq n : A \not\models_\alpha a_i$ *or* $\exists j : 1 \leq j \leq m : A \models_\alpha b_j$
4. $A \models \varphi$ *iff* $A \models_\alpha \varphi$ *for all* α

Notice that $\doteq$ denotes *deterministic* equality – the two sets must contain exactly one (the same) element. Multialgebras with weak homomorphisms form an institution $\mathcal{MA}$ [11]. There exist sound and complete calculii for multialgebraic specifications: for multialgebras without operations returning the empty set [19, 20], and for the ones admitting empty result sets (like in def. 2.1) [3].

3 Developing specifications in $\mathcal{MA}$

We illustrate the possibilities of development in $\mathcal{MA}$ from abstract specifications analogous to partial-algebra specifications, through a series of refinement steps, to specifications with explicit error handling. At the initial, most abstract level, 3.1, error situations are not addressed at all. Operations known to be total are specified as deterministic, while others remain underspecified which allows, in particular, for their nondeterministic interpretation. At the next stage, 3.2, error situations are identified and we indicate several possible ways to do that. Then one can begin explict error handling, first, 3.3, by specifying the behaviour of other operations in error situations and, eventually, 3.4, by introducing explicit error values. A great flexibility of error treatment is offered enabling one to introduce error values, exceptions and various ways of reacting to them.

We emphasize that the whole process is merely a gradual refinement of the initial specification by extending its signature and the set of axioms. (**det** abbreviates axioms of the form $t \doteq t$, i.e. writing ω is in the **det** part means the same as includcing the axiom $\omega() \doteq \omega()$ under **axioms**.)

3.1 Initial specification

We start with a standard specification not addressing any error situations explicitly (we assume given some standard specification **Nat** of natural numbers).

$$\begin{aligned}
&\textbf{spec Stack}^{\mathcal{MA}} = \\
&\quad\textbf{include}\qquad \textbf{Nat} \\
&\qquad\quad \textbf{S}: \qquad Stack \\
&\qquad\quad \Omega: \qquad empty :\to Stack \\
&\qquad\qquad\qquad\quad top : Stack \to Nat \\
&\qquad\qquad\qquad\quad pop : Stack \to Stack \\
&\qquad\qquad\qquad\quad push : Nat \times Stack \to Stack \\
&\qquad\;\; \textbf{det}: \qquad empty \\
&\quad \textbf{axioms}: \quad 1.\quad push(x,s) \doteq push(x,s) \Rightarrow top(push(x,s)) \doteq x \\
&\qquad\qquad\qquad 2.\quad push(x,s) \doteq push(x,s) \Rightarrow pop(push(x,s)) \doteq s
\end{aligned}$$

At this level, only *empty* is explicitly specified to be deterministic. Axioms 1. and 2. are the "usual" stack axioms, equipped, however with aditional guards. They illustrate the general strategy of guarding which requires well-defined (deterministic) results only on arguments which are well-defined (deterministic). This is the way we will do it in general – the outermost operation in a composition of functions will be guarded by a deterministic assertion. The guarding is necessary when we (later) come to particular error recovery. As we will see, this specification can be refined to a specification of bounded stack. It is possible to give an alternative specification where the guards in both axioms are dropped, but it would force *top* and *pop* to be defined, irrespectively of wheter $push(s,x)$ is defined or not, which preclude error recovery. Since variables only range over individual elements, we need no additional guards of the form $s \doteq s$. Also, the models for the specification may display flexible behaviour on nondeterministic values resulting from *push*.

Notice that this is essentially the same as a partial algebra specification – just replace the sign $\doteq$ by $\stackrel{e}{=}$. Reuse of partial algebra specifications in our framework is discussed in section 4.

3.2 Error situations

Error situations may be treated in various ways. At first error situations may be identified and marked by appropriate error constants:

1. **spec ErrStack1**$^{\mathcal{MA}}$ =
   enrich Stack by:
   $$\begin{aligned}
   &\quad \Omega: \qquad errStack :\to Stack \\
   &\qquad\qquad\quad errNat :\to Nat \\
   &\textbf{axioms}: \quad 1.\quad pop(empty) \prec errStack \\
   &\qquad\qquad\quad 2.\quad top(empty) \prec errNat
   \end{aligned}$$

Such error constants do not have any direct influence on the semantics; they function mainly as labels visualizing the special situations: $pop(empty)$ may still be deterministic or not – it has only been marked as a special term "of type" $errStack$.

Instead of introducing explicit error constants, one may indicate error situations by forcing respective terms to be nondeterministic:

2. **spec ErrStack2**$^{\mathcal{MA}}$ =
   enrich Stack by:
   $$\begin{aligned}
   &\textbf{axioms}: \quad 1.\quad pop(empty) \doteq pop(empty) \Rightarrow \\
   &\qquad\qquad\quad 2.\quad top(empty) \doteq top(empty) \Rightarrow
   \end{aligned}$$

The inequalities ensure that the error terms can't be interpreted as individuals of the carrier; they must be sets – possibly empty. (The associated logic then prevents one from substituting such terms for variables.) This precludes later treatment of particular error

situations by means of deterministic error constants. Still, it offers significant flexibility to be illustrated in 3.4.

Further refinement of these two possibilities will take full advantage of non-strict semantics. Notice that, so far, no specific error treatment has been described – $pop(empty)$ is merely labeled in the first case and made nondeterministic in the second.

Finally, one can follow the order-sorted approach with constants for appropriate subsorts:

3. **spec OsStack$^{\mathcal{MA}}$ =**
   enrich Stack by:
   $$\Omega: \qquad nonempty :\to Stack$$
   axioms :   1. $s \prec nonempty \Rightarrow top(s) \doteq top(s)$
                  2. $s \prec nonempty \Rightarrow pop(s) \doteq pop(s)$
                  3. $empty \prec nonempty \Rightarrow$
                  4. $push(x,s) \doteq push(x,s) \Rightarrow push(x,s) \prec nonempty$

The new constant $nonempty$ is used as a subsort of non-empty stacks (by axiom 3. $empty$ does not belong to this subsort) for which pop and top are defined.

3.3   Behaviour on errors

Error constants, like those introduced in **ErrStack1**, may be used for a uniform specification of behaviour of other operations on all elements (error values) included in the constants:

1. **spec ErrStack1a$^{\mathcal{MA}}$ =**
   enrich ErrStack1 by:
   axioms :   1. $pop(push(x, errStack)) \doteq empty$
                  2. $top(push(x, errStack)) \doteq x$
                  3. $pop(push(errNat, s)) \prec pop(s), s \prec errStack$
                  4. $top(push(errNat, s)) \prec top(s), s \prec errStack$

In the first two axioms, the prescribed results are always well-defined. If the first argument happens to be $errNat$, axiom 1. will give $empty$ and axiom 2. will result in $errNat$ (whether it happens to be deterministic or not). The last two axioms use $\prec$ and not $\doteq$. This is so because the terms on the right-hand-side may, possibly, be error terms (when s is $empty$). The alternatives give precedence to $errStack$ over $errNat$. If both arguments are err, axiom 1., respectively 2., will be applied – instead of axiom 3., axiom 1. will yield $empty$ as the result of pop. Similarly in axiom 4.

Another possibility is to treat each error situation separately. The following specification refines **ErrStack2** but it might as well be a direct refinement of **ErrStack1**:

2. **spec ErrStack2a$^{\mathcal{MA}}$ =**
   enrich ErrStack2 by:
   axioms :   1. $pop(push(x, pop(empty))) \doteq empty$
                  2. $top(push(x, pop(empty))) \doteq x$
                  3. $pop(push(top(empty), s)) \prec pop(s), s \prec pop(empty)$
                  4. $top(push(top(empty), s)) \prec top(s), s \prec pop(empty)$

At this stage we have still not determined any specific error values but only the behaviour of other operations applied in situations identified so far as errors – $errStack$ may later be specified to be a particular value or else remain nondeterministic.

The most dramatic possibility is to delegate all responsibility for error treatment to the implementation. Unlike in approaches with implicit assumptions about (non)strictness, here this is a decision to be made explicitly by the specifier. Explicit specification which

excludes further description of error makes the result of respective error terms empty (this can be read as a requirement of raising a run-time exception):

3. **spec ErrStack3**$^{\mathcal{MA}}$ =
   enrich ErrStack2 by:
   axioms :   1. $s \prec pop(empty) \Rightarrow$
           2. $x \prec top(empty) \Rightarrow$
           3. $s \prec push(errNat, s') \Rightarrow$

This implies strictness of other operations applied to these arguments, since empty argument will always lead to empty result. (For instance, $push(x, pop(empty))$ will now return the empty set, too.) Note to ensure strictness we have to specifiy it explicitly with multialgebras, where partial algebras is always strict. Making analogous extension of **ErrStack1** is possible, though it does not seem quite purposeful – labeling error situations will, typically, involve later their explicit treatment.

3.4 Error values

We now arrive at the lowest level and introduce explicit error values.

1. **spec ErrValStack1**$^{\mathcal{MA}}$ =
   enrich ErrStack1a by:
   axioms :   1. $pop(empty) \doteq empty$
           2. $top(empty) \doteq errNat$
           3. $push(errNat, s) \doteq s$

This is apparently consistent with the intention of the behaviour from **ErrStack1a** (according to first two axioms from **ErrStack1a**, $errStack$ behaves as $empty$, and according to the last two, pushing $errNat$ on s behaves then as s). However, there are serious problems with this refinement.

A counter-intuitive consequence of this enrichment is that $empty \prec errStack$ by axiom 1. from **ErrStack1**. Now axiom 1. says that, no matter what was earlier said about the error situation $pop(empty)$, it can be disregarded and that we do, instead, immediate error recovery. Since $pop(empty)$ has been earlier identified as an error $errStack$, a more plausible refinement would be to identify this error value which can take care of a possible indication of the error situation.

The really serious problem is caused by the last axiom. It makes $push(errNat, s)$ deterministic. Thus, since $errNat$ is deterministic (axiom 2), we can substitute it into axioms from **Stack** and may conclude that $top(push(errNat, s)) \doteq errNat$. However, according to **ErrStack1a**, if $s \not\prec errStack$ we have that $top(push(errNat, s)) \prec top(s)$. Thus, this may lead to collapsing the sort of elements (here Nat).

Forcing some error terms to be deterministic requires revisiting earlier specifications and checking for such unintended coincidences. The uniform way of introducing errors, which is safe, is to force errors to be sets. The following specification makes $pop(empty)$ a new deterministic constants (this is ok in this example), but illustrates this generally recommended way by axiom 2.

2. **spec ErrValStack2**$^{\mathcal{MA}}$ =
   enrich ErrStack1a by:
       Ω : $popEmpty :\rightarrow Stack$
   axioms :   1. $pop(empty) \doteq popEmpty$
           2. $push(errNat, s) \doteq push(errNat, s) \Rightarrow$

We are forcing $push(errNat, s)$ to be a set in order to avoid interference with the axioms of **Stack**. The axioms from **ErrStack1a** prescribe recovery from this situation. This strategy does *not* require revisiting earlier specifications – it ensures that newly identified

error sets will not interfere with earlier assuumptions about results prodused on defined (deterministic) values.

The last remaining question concerns the actual value to be returned by $push(errNat, s)$. We specify it is a set including some error value, (axiom 2.), and the value to be used for the recovery purpose (axiom 1.):

3. **spec ErrValStack3**$^{\mathcal{M}\mathcal{A}} =$
   enrich ErrValStack2 by:

   $\Omega :$ $pushErr :\to Stack$

   det : $pushErr$

   ax : 1. $s \prec push(errNat, s)$
           2. $pushErr \prec push(errNat, s)$
           3. $s \prec pop(pushErr) \Rightarrow$
           4. $x \prec top(pushErr) \Rightarrow$
           5. $s \prec push(x, pushErr) \Rightarrow$

These axioms ensure the desired behaviour (from **ErrStack1a**). Axioms 3.-5. make the results of other operations applied to $pushErr$ empty, so that they will only consider the recovery value.[1] $pushErr$ can be naturally interpreted as a side effect of $push(errNat, s)$, and implemented as, say, sending an error message to the user. On the other hand, the presence of such an error value in the result set can be interpreted as an exception. The axioms 3.-5. represent immediate catching this exception. Replacing, e.g., the axiom 3. by $pushErr \prec pop(pushErr)$ will then correspond to throwing this exception also from an application of pop until one arrives at a situation where other operations ignore this error value (specified with axioms corresponding to 3.-5.)

At any previous development step we could have extended the stack specification to a specification of bounded stack. This can be also done now as a refinement of the last specification:

4. **spec BoundedStack**$^{\mathcal{M}\mathcal{A}} =$
   enrich ErrValStack3 by:

   $\Omega :$ $max :\to Nat$
             $hgh : Stack \to Nat$

   det : max, hgh

   axioms : 1. $hgh(empty) \doteq 0$
               2. $push(x, s) \doteq push(x, s) \Rightarrow hgh(push(s, x)) \doteq succ(hgh(s))$
               3. $hgh(s) \doteq max \Rightarrow hgh(push(x, s)) \doteq succ(max)$
               4. $hgh(s) < max \Rightarrow push(x, s) \doteq push(x, s)$
               5. $hgh(s) \geq max \Rightarrow push(s, x) \prec errStack$

Notice that the last axiom makes the error resulting from exceeding the bound behave as *empty* according to **ErrStack1a**. If this is not desirable, one would have to identify it as a new error type with appropriate axioms.

4   Partial algebras and multialgebras

For a survey of partial algebras, the reader is referred to [4, 6, 16, 17]. A partial algebra A is an algebra where each operation $\omega^A : s_1^A \times ... \times s_n^A \to s^A$ may be partial with a subset of the argument as the domain of defintion, $\mathbf{dom}(\omega^A) \subseteq s_1^A \times ... \times s_n^A$. (Weak) homomorphisms are homomorphisms preserving the defintion domains, i.e., $\phi(\mathbf{dom}(\omega^A)) \subseteq \mathbf{dom}(\omega^B)$. The atomic formulae are *existential equations*, written $t \stackrel{e}{=} t'$, where $A \models_\alpha t \stackrel{e}{=} t'$ iff both

---

[1]The intention here is to interpret the result set, at least $\{s, pushErr\}$, not as a nondeterminisitc choice but as an actual union of the elements. The only difference between further refinement and eventual implementation is that the former may further restric the range of the respective sets, while the latter may interpret elements of the set as "simultaneously present", e.g., as a recovery value and an error label/message.

$\alpha(t)^A$ and $\alpha(t')^A$ are defined and deliver the same value. In particular, $A \models t \overset{e}{=} t$ means that t is total (in A). The institution of partial algebras, $\mathcal{PA}$, has as sentences conditional existential equations, written $a_1 \wedge \ldots \wedge a_n \rightarrow b$, where $n \geq 0$ and $A \models_\alpha a_1 \wedge \ldots \wedge a_n \rightarrow b$ if $A \models_\alpha b$ or $\exists i : 1 \leq i \leq n : A \not\models a_i$.

$\mathcal{PA}$ specification of stacks will look exactly as the one given in subsection 3.1 with all $\doteq$ replaced by $\overset{e}{=}$. Conversely, replacing all $\overset{e}{=}$ by $\doteq$ in an arbitrary $\mathcal{PA}$ specification, yields an $\mathcal{MA}$ specification with intuitively the same meaning but with a larger model class, where the development down to explicit error handling as illustrated in section 3 can take place. This section makes this intuitive correspondance precise by using (maps of) institutions.

4.1 Institutions

Definition 4.1 *An* institution *[9] is a quadruple* $\mathcal{I} = (\mathbf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$, *where:*

- **Sign** *is a category of signatures.*
- $\mathsf{Sen} : \mathbf{Sign} \rightarrow \mathbf{Set}$ *is a functor which associates a set of sentences to each signature.*
- $\mathsf{Mod} : \mathbf{Sign}^{op} \rightarrow \mathbf{Cat}$ *is a functor which associates a category of models, whose morphisms are called Σ-morphisms, to each signature Σ.*
- $\models$ *is a satisfaction relation – for each signature Σ, a relation $\models_\Sigma \subseteq |\mathsf{Mod}(\Sigma)| \times \mathsf{Sen}(\Sigma)$, such that the satisfaction condition holds: for any $M' \in \mathsf{Mod}(\Sigma'), \mu : \Sigma \rightarrow \Sigma', \phi \in \mathsf{Sen}(\Sigma)$*
$$M' \models_{\Sigma'} \mathsf{Sen}(\mu)(\phi) \ \textit{iff} \ (\mathsf{Mod}(\mu))(M') \models_\Sigma \phi$$

We write: $\Gamma \models_\Sigma \varphi$ iff $\forall M \in \mathsf{Mod}(\Sigma) : M \models_\Sigma \Gamma \Rightarrow M \models_\Sigma \varphi$. $\Gamma^\bullet$ denotes the semantical consequences of Γ i.e. $\Gamma^\bullet = \{\varphi : \Gamma \models \varphi\}$

A *theory* (specification) in an institution is a pair $Th = (\Sigma, \Gamma)$ where $\Sigma \in \mathbf{Sign}$ and $\Gamma \subseteq \mathsf{Sen}(\Sigma)$. For a given institution $\mathcal{I}$, we have the corresponding category of theories $\mathbf{Th}_\mathcal{I}$ with theories as objects and theory morphisms $\mu : (\Sigma, \Gamma) \rightarrow (\Sigma', \Gamma')$, where $\mu : \Sigma \rightarrow \Sigma'$, is a signature morphism such that: $\Gamma' \models_{\Sigma'} \mathsf{Sen}(\mu)(\Gamma)$. There are a canonical projection functor $\mathsf{sign} : \mathbf{Th} \rightarrow \mathbf{Sign}$ and an embedding functor $\mathsf{th} : \mathbf{Sign} \rightarrow \mathbf{Th}$ defined by $\mathsf{th}(\Sigma) = (\Sigma, \emptyset)$. The category of models for the theory $Th = (\Sigma, \Gamma)$ is the full subcategory $\mathsf{Mod}(\Sigma, \Gamma)$ of $\mathsf{Mod}(\Sigma)$ where $M \in \mathsf{Mod}(\Sigma, \Gamma)$ iff $M \models_\Sigma \varphi, \forall \varphi \in \Gamma$. A theory morphism $\mu : (\Sigma, \Gamma) \rightarrow (\Sigma', \Gamma')$ is *axiom conserving* if $\mathsf{Sen}(\mu)(\Gamma) \subseteq \Gamma'$. This defines the subcategory $\mathbf{Th_0}$ with theories as objects and axiom conserving theory morphisms as morphisms.

Definition 4.2 *Let* $\mathcal{I} = (\mathbf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$ *and* $\mathcal{I}' = (\mathbf{Sign}', \mathsf{Sen}', \mathsf{Mod}', \models')$ *be two institutions.*

1. *Given a functor* $\Phi : \mathbf{Sign} \rightarrow \mathbf{Th_0'}$ *and a natural transformation* $\alpha : \mathsf{Sen} \Rightarrow \mathsf{Sen}' \circ \Phi$, *an α-extension to theories of Φ is a functor* $\Phi : \mathbf{Th_0} \rightarrow \mathbf{Th_0'}$ *mapping a theory* $Th = (\Sigma, \Gamma)$ *to the theory $\Phi(Th)$ with signature $\Phi(\Sigma)$ and with axioms $\Phi(\Sigma) \cup \alpha_\Sigma(\Gamma)$*

2. *Given a functor* $\Phi : \mathbf{Th_0} \rightarrow \mathbf{Th_0'}$ *and a natural transformation* $\alpha : \mathsf{Sen} \Rightarrow \mathsf{Sen}' \circ \Phi$, Φ *is α-sensible iff:*

   - *There is a functor* $\Phi^\diamond : \mathsf{Sign} \rightarrow \mathsf{Sign}'$ *such that* $\mathsf{sign}' \circ \Phi = \Phi^\diamond \circ \mathsf{sign}$
   - $(\Gamma')^\bullet = (\emptyset'_\Sigma \cup \alpha_\Sigma(\Gamma))^\bullet$

   where the set of axioms induced by $\Phi(\Sigma)$ is denoted by $\emptyset'_\Sigma$.

To relate different institutions we will use various maps of institutions.

Definition 4.3 *Let* $\mathcal{I} = (\mathbf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$ *and* $\mathcal{I}' = (\mathbf{Sign}', \mathsf{Sen}', \mathsf{Mod}', \models')$ *be two institutions.*

1. *A* map of institutions *[14] is a triple* $(\Phi, \alpha, \beta) : \mathcal{I} \rightarrow \mathcal{I}'$ *where:*

   - $\alpha : \mathsf{Sen} \Longrightarrow \mathsf{Sen}' \circ \Phi$ *is a natural transformation.*

- $\Phi : \mathbf{Th_0} \to \mathbf{Th'_0}$ *is an* α*-sensible functor*
- $\beta : \mathsf{Mod'} \circ \Phi^{op} \Longrightarrow \mathsf{Mod}$ *is a natural transformation*

such that for each $\phi \in \mathsf{Sen}(\Sigma)$ *and* $M' \in \mathsf{Mod'}(\Phi(\Sigma, \emptyset))$ *the following condition holds:*

$$M' \models_{sign'(\Phi(\Sigma,\emptyset))} \alpha_\Sigma(\phi) \ \text{iff} \ \beta_{(\Sigma,\emptyset)}(M') \models_\Sigma \phi$$

2. *An* embedding of institutions *[15] is a map of institutions, written* $(\Phi, \alpha, \beta) : \mathcal{I} \hookrightarrow \mathcal{I}'$, *where the functor* $\beta_T : \mathsf{Mod'}(\Phi(T)) \to \mathsf{Mod}(T)$ *is an equivalence of categories for each* $T \in \mathbf{Th}_\mathcal{I}$.

3. *A map of institutions* (Φ, α, β) *is:*
   - (α) simple *iff* Φ *is an* α*-extension to theories of a functor* $\mathsf{F} : \mathbf{Sign} \to \mathbf{Th'_0}$, *i.e.* Φ *maps axioms to axioms.*
   - (α) plain *iff* Φ *is an* α*-extension to theories of a functor* $\mathsf{F} : \mathbf{Sign} \to \mathbf{Th'_0}$ *that maps* Σ *to* $(\Sigma', \emptyset)$, *i.e.* Φ *maps signatures to signatures.*

4. *A* subinstitution *[14] is a map of institutions* $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ *that is plain, with* Φ *faithful and injective on objects,* α *injective and with* β *a natural isomorphism.*

5. *An* institution transformation *[12, 13] is a triple* $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$ *where:*
   - $\alpha : \mathsf{Sen} \Longrightarrow \mathsf{Sen'} \circ \Phi$ *is a natural transformation.*
   - $\Phi : \mathbf{Th_0} \to \mathbf{Th'_0}$ *is an* α *plain,* α*-sensible functor*
   - *a natural transformation* $\beta : \mathsf{Mod} \Longrightarrow \mathsf{Mod'} \circ \Phi^{op}$

such that for each $\phi \in \mathsf{Sen}(\Sigma)$ *and* $M \in \mathsf{Mod}(\Sigma, \emptyset)$ *the following condition holds:*

$$\beta_{(\Sigma,\emptyset)}(M) \models_{sign'(\Phi(\Sigma,\emptyset))} \alpha_\Sigma(\phi) \ \text{iff} \ M \models_\Sigma \phi$$

4.2 Formal corespondence of $\mathcal{PA}$ and $\mathcal{MA}$

First we consider a simple embedding of $\mathcal{PA}$ into $\mathcal{MA}$.

Any Σ-partial algebra can be trivially converted into a Σ-multialgebra by making all undefined operations return the empty set. Since operations in multialgebra are strict on the empty set, the implicit strictness assumption from partial algebras, will be enforced automatically.

The embedding of $\mathcal{PA}$ into $\mathcal{MA}$ is now obtained by augmenting the partial algebra specification with additional axioms forcing all operations to return either a unique element or the empty set. This is the underlying model in partial algebras which in multialgebraic context need explicit axioms. The required axioms are given in (the proof sketch of) the following proposition.

Proposition 4.4 *There is a (simple map) embedding* $(\Psi, \alpha, \beta) : \mathcal{PA} \hookrightarrow \mathcal{MA}$.

Proof sketch. We merely indicate the construction.
- The functor $\Psi : \mathbf{Sign}_{\mathcal{PA}} \to \mathbf{Th}_{0\mathcal{MA}}$ is given by: $\Psi(S, \Omega) = ((S, \Omega), \emptyset_\Sigma)$, where $\emptyset_\Sigma$ contains an axiom $y \prec f(\overline{x}) \to f(\overline{x}) \doteq f(\overline{x})$ for each $f \in \Omega$. For morphisms $\Psi(\mu_S, \mu_\Omega)$ is the identity.
- The natural transformation $\alpha : \mathsf{Sen}_{\mathcal{PA}} \to \mathsf{Sen}_{\mathcal{MA}} \circ \Psi$ is given by:
  1. $\alpha(t \overset{e}{=} t') \equiv t \doteq t'$ for each axiom $t \overset{e}{=} t'$
  2. $\alpha(a_1 \wedge \cdots \wedge a_n \Rightarrow a) \equiv \alpha(a_1) \wedge \cdots \wedge \alpha(a_n) \Rightarrow \alpha(a)$ for each clause $a_1 \wedge \cdots \wedge a_n \Rightarrow a$

  Ψ is extended to a functor $\Psi : \mathbf{Th}_{0\mathcal{PA}} \to \mathbf{Th}_{0\mathcal{MA}}$ by letting $\Psi(\Sigma, \Gamma) = (\Sigma, \emptyset_\Sigma \cup \alpha_\Sigma(\Gamma))$.
- The components of the natural transformation $\beta : Mod_{\mathcal{MA}} \circ \Psi^{op} \to Mod_{\mathcal{PA}}$ are given by:
  - $|\beta_\Sigma(M')| = |M'|$

$$- f(x_1, \ldots, x_n)^{\beta_\Sigma(M')} = \begin{cases} undefined \text{ if } f(x_1, \ldots, x_n)^{M'} = \emptyset \\ x \text{ such that } f(x_1, \ldots, x_n)^{M'} = \{x\} \text{ otherwise} \end{cases}$$

For a homomorphism: $h \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \Gamma))$, we define $\beta_\Sigma(h) = h$. $\qquad\square$

We also have an immediate consequence of the above proof:

Fact 4.5 *For a $\mathcal{PA}$ theory (Σ, Γ), the functor $\beta_{(\Sigma, \Gamma)}$ is an equivalence (in fact, an isomorphism) of categories $\mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \Gamma))$ and $\mathsf{Mod}_{\mathcal{PA}}(\Sigma, \Gamma)$.*

As shown in [16], $\mathcal{PA}$ allows to specify exactly the finitely locally presentable categories [1], i.e. we have identified the subinstitution of $\mathcal{MA}$ allowing to specify these classes of models.

Now, the intention of passing from a partial to a multialgebra specification is, on the one hand, to reuse specifications written in the former framework and, on the other, to allow for their further development with explicit error handling. The embedding from proposition 4.4 does not address this latter issue since it yields essentially the same model class.

The desired generalization is made by importing $\mathcal{PA}$ specifications *without* augmenting them with additional axioms. This results in a larger model class where, in addition to essentially the same partial models, we also have the models with non-strict operations. Further development can now take place in the multialgebraic framework, allowing one to refine the specification to the level of explicit error treatment. The following proposition, stating the existence of *institution transformation*, def. 4.3.5, formalizes the possibility of such a scenario.

Proposition 4.6 *There is an institution transformation $(\Psi^*, \alpha, \beta^-) : \mathcal{PA} \to \mathcal{MA}$.*

Proof sketch. The definition is as in proposition 4.4 except that $\Psi^* : \mathbf{Sign}_{\mathcal{PA}} \to \mathbf{Th}_{0\mathcal{MA}}$ does not add any new axioms, while the components of the natural transformation $\beta_{\overline{\Sigma}}^- : \mathsf{Mod}_{\mathcal{PA}} \to \mathsf{Mod}_{\mathcal{MA}} \circ (\Psi^*)^{op}$ is the functor $\beta_{\overline{\Sigma}}^- : \mathsf{Mod}_{\mathcal{PA}}(\Sigma) \to \mathsf{Mod}_{\mathcal{MA}}(\Sigma)$ that maps a partial algebra A to a multialgebra given by:

- $|\beta_{\overline{\Sigma}}^-(A)| = |A|$

- for all $\overline{x} \in |\beta_{\overline{\Sigma}}^-(A)|$ and $f \in \Omega$: $f(\overline{x})^{\beta_{\overline{\Sigma}}^-(A)} = \begin{cases} \{f(\overline{x})^A\} & -\text{if it is defined} \\ \emptyset & -\text{otherwise} \end{cases}$

For a Σ homomorphism: $h \in \mathsf{Mod}(\Psi(\Sigma, \Gamma))$, we define $\beta_{\overline{\Sigma}}^-(h) = h$.

For a Σ multialgebra M where all nondeterministic operations return empty set, $\beta_{\mathbf{\Sigma}}(M)$ will denote the corresponding partial algebra, i.e., $\beta_{\overline{\Sigma}}^-(\beta_\Sigma(M)) = M$. $\qquad\square$

Note that the above proposition is a new (practical) way of using institution transformations, as an essential extension of the model class, where in [12, 13] institution transformations are mainly used for conceptual reasons.

If desired we can axiomatize the image of $\beta_{\overline{\Sigma}}^-$ by using Ψ from proposition 4.4 and we get the same relation as the (simple map) embedding in 4.4.

Fact 4.7 *For a $\mathcal{PA}$ theory (Σ, Γ), the functor $\beta_{(\Sigma, \Gamma)}$ is an equivalence (in fact, an isomorphism) of categories $\mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \Gamma))$ and $\mathsf{Mod}_{\mathcal{PA}}(\Sigma, \Gamma)$.*

5 Conclusions and future work

We have presented a multialgebra based approach to developing specifications with partial operations. The novelty of the proposed framework lies in thinking about and modeling undefined operations by nondeterministic ones – an operation applied to an argument outside its definition domain may result in an unexpected and initially unknown value. This

view leads actually to the combination of various features of several earlier approaches. It allows one to start with high level specifications, where error situations can be dealt with at the same level of abstraction as in partial algebras. Narrowing the range of nondeterminism modeling undefinedness, one can refine such specifications to a low level error handling. We have illustrated by examples a wide range of possibilities for error handling admitted by the proposed framework. In particular, utilizing sets to model error situations allows a function to return both a marking that such a situation occurred and relevant recovery values. Thus can be used for specifications of exceptions.

From the methodological perspective, we have shown the possiblity of reusing partial algebra specifications without the necessity to perform any translation (except for the trivial replacement of $\stackrel{e}{=}$ by $\doteq$.) Thus, we believe the proposed framework may be more useful, and in any case easier to apply, when extending partial algebra specifications to explicit error handling, than the frameworks based on translation of such specifications into deterministic specifications with predicates.

We have shown that multialgebras can treat the partiality problem at an abstract level, like partial algebras. One problem with partial algebras is that they only have defined (ordinary) values or undefined (empty) terms. Introduction of a new element (term) intended as an "error element" makes it to behave as all other defined elements (terms). Thus for instance the specification of stack from 3.1 cannot be refined within $\mathcal{PA}$ to a specification of bounded stack. Introducing an error element "err" for $pushing$ a stack out of bound so that if s is a maximal stack is $push(x, s) \stackrel{e}{=} err$. But then we obtain that $push(x, s) \stackrel{e}{=} err \stackrel{e}{=} push(x, s)$, so axiom 1 from **Stack** applies, i.e. $x \stackrel{e}{=} err$, colapsing the sort **Nat**.

Multialgebras allowe one to distinguish error values from ordinary values already at an abstract specification level. Later, error values can be specified so that they don't come into conflict with earlier axioms inteded only for defined arguments.

We have also indicated (proofs are to be found in [11]) that other frameworks and institutions can be embedded into the institution of multialgebras $\mathcal{MA}$ (e.g., membership algebras, order-sorted algebras). We thus suggest that $\mathcal{MA}$ may provide an adequate framework for both *comparing* and *combining* the advantages of earlier approaches within a unified framework. For this purpose, it would be desirable to restrict the generality of $\mathcal{MA}$. Looking for a suitable subinstitution may start, for instance, by trying to identify the conditions for $\mathcal{MA}$ specifications ensuring the existence of initial models.

Another, though probably, less problematic issue concerns the reasoning system. The earlier systems need to be adjusted to the present context: the complete logic from [19, 20] did not allow for empty result sets in a model, while the one from [3] used only inclusions $\prec$, but not the element equality $\doteq$.

References

[1] J. Adámek and J. Rosický. *Locally Presentable and Accessible Categories*. Cambridge University Press, 1994.

[2] G. Bernot and P. L. Gall. Label algebras: a systematic use of terms. In M. Bidot and C. Choppy, editors, *Recent Trends in Data Type Spesification*, volume 655 of *Lecture Notes in Computer Science*, pages 144–163. Springer, 1993.

[3] M. Białasik and B. Konikowska. Reasoning with first-order nondeterministic specifications. Technical Report 830, Depeartment of Theoretical Informatics, Polish Academy of Sciences, 1997.

[4] P. Burmeister. Partial algebra - an introductory survey. *Algebra Universalis*, 15:306–358, 1982.

[5] M. Cerioli. A lazy approach to partial algebras. In E. Astesiano, G. Reggio, and A. Tarlecki, editors, *Recent Trends in Data Type Specification: 10th Workshop on Specification of Abstract Data Types-Selected Papers*, volume 906 of *Lecture Notes in Computer Science*. Springer, 1995.

[6] M. Cerioli, T. Mossakowski, and H. Reichel. From total equational to partial first order. In E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors, *on Algebraic Foundations of Systems Specification*, chapter 3. Springer, 1999.

[7] J. A. Goguen. Abstract errors for abstract data types. In *IFIP Working Conference on the Formal Description of Programming Concepts*, volume 116, pages 89–103. North-Holland, 1978.

[8] J. A. Goguen. Order-sorted algebra i. Semantics and Theory of Computation Series 14, UCLA Computer Science Department, 1978.

[9] J. A. Goguen and R. M. Burstal. Institutions: Abstract model theory for specification and programming. *Journal of the ACM*, 39:95–146, 1992.

[10] M. Haveraaen and E. G. Wagner. Guarded algebras and data type specification. Technical Report 108, Depeartment of Informatics, University of Bergen, 1995.

[11] Y. Lamo and M. Walicki. Modeling partiality by nondeterminism - from abstract specifications to flexible error handling. Technical Report 178, Depeartment of Informatics, University of Bergen, 1999.

[12] A. Martini and U. Wolter. A systematic study of mappings between institutions. In F. P. Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*, pages 300–315. Springer, 1998.

[13] A. Martini and U. Wolter. A Single Perspective on Arrows between Institutions. In A. M. Haeberer, editor, *Algebraic Methodology and Software Technology*, volume 1548 of *Lecture Notes in Computer Science*, pages 486–501. 7th International Conference, AMAST'98, Brazil, January 1999, Procedings, Springer Verlag, 1999.

[14] J. Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Logic colloquium'87*, pages 275–329. Elsevier Science Publisher B.V.(North-Holland), 1989.

[15] J. Meseguer. Membership algebra as a logical framework for equational specification. In F. P. Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*, pages 18–61. Springer, 1998.

[16] T. Mossakowski. Equivalences among various logical frameworks of partial algebras. In H. K. Büning, editor, *Computer Science Logic*, volume 1092 of *Lecture Notes in Computer Science*, pages 403–433. Springer, 1996.

[17] H. Reichel. *Initial Computability Algebraic Specifications and Partial Algebras*. Oxford Science Publications, 1987.

[18] M. Walicki and M. Białasik. Categories of relational structures. In F. P. Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*. Springer, 1998.

[19] M. Walicki and S. Meldal. A complete calculus for the multialgebraic and functional semantics of nondeterminism. *ACM TOPLAS*, 17(2), 1995.

[20] M. Walicki and S. Meldal. Multialgebras, power algebras and complete calculi of identities and inclusions. volume 906 of *Lecture Notes in Computer Science*. Springer, 1995.

[21] M. Walicki and S. Meldal. Algebraic approaches to nondeterminism-an overview. *ACM Computing Surveys*, 29, 1997.