Modeling partiality by nondeterminism – from abstract specifications to flexible error handling

Yngve Lamo   Michał Walicki

Department of Informatics
University of Bergen
5020 Bergen, Norway
{yngvel,michal}@ii.uib.no

Abstract

The paper presents a new way to model partial operations by use of nondeterminism: a partial operation is modeled as a nondeterministic operation returning, possibly, any value of the carrier. We introduce an institution of multialgebras $\mathcal{MA}$ (modelling nondeterministic operations by set-valued functions) and illustrate the flexibility of our approach by examples showing uniform treatment of strictness, non-strictness and various error handling. We present a methodology for specification development from an abstract specification to low level error handling. We also study the conditions for $\mathcal{MA}$ specifications to ensure the existence of initial models. Finally we relate $\mathcal{MA}$ to other institutions, in particular, we show embedding of partial algebras and membership algebras into $\mathcal{MA}$. Applying institution transformation instead of embedding leads to the possibility of resuing partial algebra specifications in the proposed framework – a partial algebra specification can be conservatively (preserving the models) imported to $\mathcal{MA}$ while the extension of the model class allows for furhter development towrds explicit error treatement.

1 Introduction

The problem of partial operations is of fundamental importance in specifying and deriving programs. At the abstract level, one would like to be able to ignore many details related to the fact that some operations may happen to be undefined for some particular arguments. 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 partially defined operations: the partial algebra approach on the one hand [5, 21, 7], and total algebras with explicit definition domains or even error elements, on the other. This later approach comprises manifold variations, including error-algebras [9], labeled algebras [2], order-sorted algebras [10], techniques using predicates [18] or functions [13] to specify definition domains of operations. For a closer discussion of the traditional solutions see [20].

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 undefiend 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 and user-friendly model for partiality due to 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 and explicit error handling becomes desirable. Strictness of all operations makes explicit error handling very difficult, if at all possible. Possible extensions of partial algebras to handle this level of specification are indicated in [7, 6]. Often, they suggest some form of translation of a partial algebra specification into some total framework [7].

The approaches based on total algebras explore the possibilities of explicit error handling, by allowing for values which can be used as error values.

The error elements solution is to add distinct elements to the carrier which will be returned by partial operations. The problem with this solution is that operations applied to error elements need to be explicitly specified and thus specifications tend to include – even at an early, abstract stage – a lot of axioms for such situations. In order-sorted algebras, the sorts are arranged in a hierarchy of subsorts and domains of operations are restricted to appropriate subsorts. To some extent, static type checking can then take care of excluding partial terms. On the other hand, static type checking may give unitended consequences [7], e.g. intuitively legal terms may become ill-typed. To handle this one introduces the dynamical concepts like retracs [12]. A retract is much like a predicate telling if some term is defined/well-typed or not. Finally, instead of subsorts one may use the predicates to specify the definition domains of operations [18]. A partial term may remain underspecified by guarding the axioms to hold only for terms satisfying appropriate predicates.

We thus have, on the one hand, an abstract framework of partial algebras and, on the other hand, the total algebra approaches which allow one to specify errors explicitly at a level closer to implementation but for the price of the abstract character of partial algebras. We are proposing a single framework capable of addressing both these aspects. We view undefinedness as nondeterminism – a term whithout a well-defined value may result in any value. At the early stages of development, this nondeterminism expresses our ignorance or disinterest in what exactly will happen in a given – error – situation. At the later stages, such nondeterminism may be narrowed leading, eventually, to explicit error values.

As a model of nondeterminism, we use multialgebras [24, 25]. A multialgebra is an algebra where operations are set valued. A term interpreted by a set with more than one element is called nondeterministic: the set represents the set of possible results returned by various applications of the term. The fact that an operation is well-defined is represented by this operation having a unique value. Such a unique value may be an intended "proper" value, or else it may represent an error value after one started to introduce such error values explicitly into specification.

The opposite, i.e., undefined situation may be represented in two ways. A term may denote the empty set. This turns out to be the exact analogue of undefinedness in partial algebras. Other operations applied to the empty set return empty set, and thus operations' behaviour is strict. The empty set result can also be understood as delegating further error treatement to the level of implementation.

On the other hand, an undefined term may denote a non-empty set of values. Initially, such a set may be thought of as the possibilities of later error recovery, or else it may model abnormal behaviour of the system. In the refinement process, the set may be narrowed to include only the relevant values to be used in error situation. However, one may also *force* a term to return a set. Such "error sets" will then include, on the one hand, some "proper" values to be used by other operations for the recovery purposes. On the other hand, the sets may contain error values functioning as labels and indicating particular error situation. These error values may be then propagated by other operations or else caught and removed – thus we obtain a model of throwing and catching exceptions.

We will present various scenarios for the development process. The one which we favour most, begins with a partial algebra specification which, however, in our context has a richer class of models than the standard partial algebra models – it contains also models where operations are non-strict. Through a simple process of refinement, one can arrive at the level of explicit error handling, either by introducing unique error values, or else error sets which are amenable to further specification of errors and exceptions. On the other hand, one may use the language of multialgebras to introduce nondeterministic constants which are essentially the same as predicates in many total algebra approaches. Such predicates can be used for the specification of domain definitions and specifications can be refined in the way standard total algebra specifications are. At the technical level, we show that institutions of partial algebras and membership alegbras can be embedded into the proposed institution of multialgebras, thus justifying our claims about both flexibility of our approach and the possiblity of reusing the existing specifications.

In section 2 we introduce some notation and give the basic definitions concering multialgebras, following [25]. Section 3 shows how the basic partiality problems can be addressed in $\mathcal{MA}$ – the institution of multialgebras. Since multialgebras offer a nonstrict (or rather, not necessarily strict) framework, we also show how to deal with strictness, if it is desirable for some reasons. In section 4 we present a methodology for developing specifications in $\mathcal{MA}$, illustrated by examples.

The rest of the paper presents the technical results. Section 5 presents the background definitions of institutions [11] and their morphisms, and shows that multialgebras form an institution $\mathcal{MA}$. Section 6 studies the conditions for the existence of initial models in $\mathcal{MA}$. In section 7 we compare $\mathcal{MA}$ with partial algebras $\mathcal{PA}$, showing, firstly, an embedding of institution of partial algebras $\mathcal{PA}$ into $\mathcal{MA}$. Then we show that a direct translation of $\mathcal{PA}$ specifications yields another form of institution morphism which also gives us the possibility to reuse such specifications in a later development toward explicit error handling. Section 8 shows that the membership algebras [18] form a sublogic of multialgebras by constructing apropriate embedding of institutions.

2 Preliminaries

2.1 Notation

We use the notation $|\mathbf{C}|$ to denote the objects of a category $\mathbf{C}$. The same notation is used to denote the carrier $|A|$ of an algebra A. (This shouldn't cause any confusion.) Institutions are written with the script font $\mathcal{I}$, categories with bold **Cat**, and functors with Sans Serif Func. Sequences $s_1, \ldots, s_k$ will be often denoted by $\overline{s}$. Application of functions are then understood to not distribut over the elements, i.e., $f(\overline{s})$ denotes the term $f(s_1, \ldots, s_k)$. Occasionally, a sequence $s_1 \ldots s_k$ may be denoted by s^* – applications of functions are then understood to distribute over the elements, i.e., $f(s^*)$ denotes the sequence $(f(s_1), \ldots, f(s_k))$. f^A interprets the operation f in the (multi)algebra A

2.2 Multialgebras

We will now summarize the relevant notions about multialgebras (for an overview, see [25, 22]). Signatures for multialgebras are the same as classical signatures.

Definition 2.1 *The category of signatures* **Sign** *has:*

- *signatures as objects: a signature Σ is a pair of sets $(\mathbf{S}, \Omega)$ of names of sorts and operations. Each operation $\omega \in \Omega$ is a (k+2)-tuple: $\omega : s_1 \times \cdots \times s_k \to s$, where $s_1, \ldots, s_k, s \in S$ and $k \geq 0$. ω is the name of the operation and $s_1 \times \ldots \times s_k \to s$ its arity. If $k = 0$ then an operation $c :\to s$ is called a constant of sort s.*
- *signature morphisms as arrows: a signature morphism $\mu : \Sigma \to \Sigma'$ is a pair $\mu = (\mu_S, \mu_\Omega)$ of (total) functions: $\mu_S : S \to S'$, $\mu_\Omega : \Omega \to \Omega'$, such that $\mu_\Omega(\omega : s_1 \times \cdots \times s_n \to s) = \omega' : \mu_S(s_1) \times \cdots \times \mu_S(s_n) \to \mu_S(s)$*
- *Identities are the identity signature morphisms; morphism are composed componentwise.*

In the standard way, we extend the signature morphism $\mu : \Sigma \to \Sigma'$ to terms.

Terms are defined in the usual way; ground terms are denoted by $T(\Sigma)$ and terms over a set with variables X is denoted $T(\Sigma, X)$

Definition 2.2 *Extension of a signature morphism μ to terms $\tilde{\mu} : T(\Sigma, X) \to T(\Sigma, X)$ is defined by:*

- $\tilde{\mu}(x_s) = x_{\mu(s)}$
- $\tilde{\mu}(c) = \mu(c)$
- $\tilde{\mu}(\omega(t_1, \ldots, t_n)) = \mu(\omega)(\tilde{\mu}(t_1), \ldots, \tilde{\mu}(t_n))$

We will write $\mu(t)$ instead of $\tilde{\mu}(t)$.

A multialgebra for a signature Σ is an algebra where operations may be set-valued. $\mathcal{P}(y)$ denotes the powerset of set y.

Definition 2.3 *(Multialgebra) A multialgebra A for Σ 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.

One sometimes demands that constants and operations are total [24, 23], i.e. never return empty set and take values only in $\mathcal{P}^+(s^A)$, the nonempty subsets of s^A. We will not make this assumption.

Note that for a constant $c \in \Omega$, c^A denotes a (sub)set of the carrier s^A. This will allow us to use constants as predicates.

As homomorphisms of multialgebras, we will use weak homomorphisms (see [22] for alternative notions).

Definition 2.4 *Given two 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* $a_i \in s_i^A$.

Saying "homomorphism" we will always mean weak homomorphism, unless something else is stated.

Definition 2.5 *The category of Σ-multialgebras, $\mathbf{MAlg}_\Sigma$, has Σ-multialgebras as objects and homomorphisms as arrows. The identity arrows are the identity homomorphisms and composition of arrows is obvious composition of homomorphisms.*

Multialgebraic specifications are written using the following formulae:

Definition 2.6 *Formulae of multialgebraic specifications are of the following forms:*

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. $a_1, \ldots, a_n \Rightarrow b_1, \ldots, b_m$, *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 of A to variables. It induces a unique interpretation $\overline{\alpha} : T(\Sigma(X)) \to A$ of every term t (with variables from X) in A.

Definition 2.7 *Given a Σ-multialgebra A, an assignment to variables X is a function $\alpha : X \to |A|$. An assignment induces a unique interpretation $\overline{\alpha}(t)$ in A of any term t as follows:*

- $\overline{\alpha}(x) = \{\alpha(x)\}$
- $\overline{\alpha}(c) = c^A$
- $\overline{\alpha}(\omega(t_1, \ldots, t_n)) = \bigcup_{a_i \in \overline{\alpha}(t_i)} \omega^A(a_1, \ldots, a_n)$

Keep in mind that variables are assigned not sets but individual elements of the carrier. We will write $\alpha(t)$ instead of $\overline{\alpha}(t)$.

Satisfaction of formulae in a multialgebra is defined as follows:

Definition 2.8 *Given an assignment $\alpha : X \to |A|$:*

1. $A \models_\alpha t \doteq t'$ iff $\overline{\alpha}(t) = \overline{\alpha}(t') = \{e\}$, fore some $e \in |A|$
2. $A \models_\alpha t \prec t'$ iff $\overline{\alpha}(t) \subseteq \overline{\alpha}(t')$
3. $A \models_\alpha a_1, \ldots, a_n \Rightarrow b_1, \ldots, b_m$ 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

Putting these definitions together, the multialgebras form an institution $\mathcal{MA}$. Its formal definition and the proof of the fact that it is an institution are postponed to section 5. We also recall that there exist sound and complete calculii for multialgebraic specifications: for multialgebras without operations returning empty set [23, 24], and for the ones admitting empty result sets (like in 2.3 above) [3].

3 Partiality handling with multialgebras

The straightforward way to model partiality in a multialgebra A is to let an operation undefined on some arguments, $\omega^A(a_1, \ldots, a_n)$, be represented by a multi-function returning the empty set when applied to these arguments, $\omega^A(a_1, \ldots, a_n) = \emptyset$. This was done, for instance, in [22] where one showed that modeled this way, partial algebras form a full subcategory of multialgebras. Classical partial algebras (or the above way of representing them in multialgebras) offer a high level abstraction mechanism. In particular, the underlying strictness assumption releases one from detailed treatment of various error situations. On the other hand, however, there are non-strict operations (like if_then_else) which then require special treatment [6, 7]. More generally, the abstract character of partial algebra formalism does not handle the problems encountered when the specification approaches the implementation level and where explicit error handling may be desirable. At this stage, one will typically recommend a "totalisation" of a partial algebra specification [7]. Most, if not all, total algebra approaches to partiality introduce some form of error elements or predicates, allowing explicit error handling.

It is our contention that multialgebras can provide a framework unifying these two extremes in a single formalism. As indicated in the introduction, besides the above solution (where partial operation returns empty set), we have the possibility to model a partial operation by proper nondeterminism. Thus an undefined $\omega^A(a_1, \ldots, a_n)$ may return various results, potentially, any available element of the carrier. At the very beginning of specification process, this nondeterminism may represent our complete ignorance and disinterest in what particularly will happen when error situation occur. Later it may be used to represent all potential error elements without discriminating against any one. Finally, it may also capture possible "totalisation" when, at a later stage, one refines the earlier specification by choosing a particular (error or not) value to be returned in an error situation.

3.1 Definedness and Undefinedness

3.1.1 Definedness

As noted above, definedness is the same as determinism in our setting. To specify definedness we use element equalities. The axiom $t \doteq t$ holds only when t is defined i.e. when t is deterministic. Likewise the axiom $t \doteq t'$ specifies both terms t and t' to be equal and deterministic, i.e., defined. Specially, the axiom $f(\overline{x}) \doteq f(\overline{x})$ specifies the operation f to be defined (deterministic) on all arguments. Thus, since operations may be nondeterministic, to make an ordinary total specification in $\mathcal{MA}$ one has to add this type of axioms for every constant and operation.

Example 3.1 *Here is a simple example of a specification of the natural numbers, with successor and predecessor, using multialgebras:*

$$
\begin{array}{rll}
\textbf{spec Nat} = & & \\
\textbf{S}: & & Nat \\
\Omega: & & zero :\to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
\textbf{axioms}: & 1. & zero \doteq zero \\
& 2. & succ(x) \doteq succ(x) \\
& 3. & pred(succ(x)) \doteq x
\end{array}
$$

This specification will have all the expected classical (total, as well as partial algebras) among its models. The first two axioms make zero and succ deterministic operations. The last one ensures that pred is deterministic and returns x when applied to $succ(x)$.

However, $pred(zero)$ remains unspecified. In the traditional setting, this amounts to underspecification. So does it here, only that here it opens the possibility for this term to be nondeterministic. $pred(zero)$ may return anything. Consequently, although $succ(x)$ will always return a unique value, $succ(pred(0))$ will be, in general, nondeterministic since it denotes all the values returned by $succ(y)$ where y is returned by $pred(0)$.

Note that in this specification the succ is always defined as a consequence of axiom 2.

In the sequel we will sometimes use **det** as an abbreviation to specify that operations is deterministic; then axioms 1. and 2. above will be abbreviated as **det** $zero, succ$.

3.1.2 Undefinedness

Since element equality, $\doteq$, holds only if both sides of the equality sign denote same element, we have two possible situations for this equality to fail – (at least) one side may be a set with more than one element or else it may be the empty set. We interpret these two possibilities as two types of undefinedness in multialgebras: nondeterminism and partiality.

Partiality corresponding to empty result set leads to strictness: if a term t denotes an empty set, any operation applied to it will also yield empty set. In fact, such partiality can be forced by the specification by adding a negated axiom – '$x \prec t \Rightarrow$' requires that no x be included in t. In terms of program development, we may interpret such axioms as requirements to handle the respective situation, caused by t, at the implementation level. It excludes any further possibility of a more refined treatement of this situation at the specification level.

Undefinedness represented by nondeterminism, e.g., by an axiom '$t \doteq t \Rightarrow$', on the other hand, allows further refinement and will be the main issue in the following. Delegation of the responsibility for error treatement to implementation may occur at the very last steps of specification.

Using element equality $\doteq$ we can only force a term to contain exactly one value. Thus, the intuition behind our element equality $\doteq$ corresponds exactly to the existential equality $\stackrel{e}{=}$ from partial algebras: forcing a term to have a uniquely defined value. The difference concerns the negation: if existential equation $t \stackrel{e}{=} t$ fails, the term t is undefined and other operations are strict; if the element equality $t \doteq t$ fails, the term t may be undefined in the sense of being empty set (leading to strictness of other operations), or else it may be a non-empty set. The choice between these two possibilities is left for the specifier.

Unless stated otherwise, we will in the sequel mean by "undefinedness" proper nondeterminism and by "partiality" empty result set.

3.2 Predicates, order-sortedness and strictness

In the most abstract way one may say that an operation f is strict in an argument if, whenever this argument is undefined, then so is the result of the operation. This is how it is viewed in partial algebras. However, when undefinedness is described more explicitly, typically by an explicit indication of the definition domain, then this needs a reformulation: whenever the argument falls outside the definition domain, then so does the result of the operation. Here "definition domain" is understood simply as a set of designated elements of the carrier considered as well-defined. Even more specific notion is obtained when each operation obtains a specific definition domain. Then strictness means: when the argument of the operation comes from outside of the operation's definition domain, then the result is undefined.

As noted in the introduction multialgebras offer a non-strict framework. Although we believe this to be its advantage, we will now make some remarks on how also strict operations can be specified in this framework.

3.2.1 Non-strictness and non-injectivity

We now start with an example of a non-strict function and then proceed to a more detailed treatment of explicit definition domains.

Example 3.2 *Let us extend the specification from example 3.1 as follows:*

$$
\begin{array}{rll}
\textbf{spec Nat1} = & & \\
\textbf{S}: & & Nat \\
\Omega: & & zero :\to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
& & f : Nat \to Nat \\
\textbf{det}: & & zero, succ \\
\textbf{axioms}: & 1. & pred(succ(x)) \doteq x \\
& 2. & f(x) \doteq succ(zero)
\end{array}
$$

The last axiom makes f non-strict - and non-injective. No matter what argument or set of arguments it receives, its result will always be $succ(zero)$.

All operations are strict on the partial terms – applied to empty set, they return empty set. Unless some axioms of the form 2. are given for some operation, the operation will be allowed to be strict also with respect to nondeterministic undefinedness: applied to an error (nondeterministic term) it will, in some models, result in error (nondeterminate result).

In other models, however, it may be non-strict in that it returns unique result on undefined (nondeterministic) arguments. We think that this generosity is a virtue rather than a sign of indefinite abstractness. One should not be forced to make this kinds of decisions at an early stage of development – forcing strictness damages the possibility for later error recovery, as error recovery is non strict by nature.

3.2.2 Definedness predicates

Since nondeterministic constants denote a set, we can use them as predicates. For instance, we can add definedness predicates (constants) to each sort, e.g. $d_{Nat} :\to Nat$. This gives a possibility to write axioms like $succ(x) \prec d_{Nat}$ with the intended meaning as $succ(x) \doteq succ(x)$. This meaning, however, remains merely "intended" since the constant d_{Nat} functions merely as a label unless the range of its elements is explicitly specified.

The use of definedness predicates implies closer classification of the elements of the carrier, so it is less abstract than the mere use of elements equalities.

Example 3.3 *This following intends to use definedness predicates to make "the same" specification of the natural numbers, with successor and predecessor, as example 3.1.*

$$
\begin{array}{rll}
\textbf{spec Nat2} = & & \\
\textbf{S}: & & Nat \\
\Omega: & & zero :\to Nat \\
 & & succ : Nat \to Nat \\
 & & pred : Nat \to Nat \\
 & & d_{Nat} \to Nat \\
\textbf{det}: & & zero, succ \\
\textbf{axioms}: & 1. & zero \prec d_{Nat} \\
 & 2. & x \prec d_{Nat} \Rightarrow succ(x) \prec d_{Nat} \\
 & 3. & x \prec d_{Nat} \Rightarrow pred(succ(x)) \doteq x \\
 & 4. & succ(x) \prec d_{Nat} \Rightarrow x \prec d_{Nat} \\
 & 5. & pred(succ(x)) \doteq x \Rightarrow x \prec d_{Nat}
\end{array}
$$

The constant d_{Nat} is intended to comprise all "defined" values of sort Nat. The first two axioms specify its minimal range. Axiom 3. needs then a guard, since we do not know what may happen when operations are applied to elements outside d_{Nat}.

Axiom 4. makes succ strict in the more specific sense than mere preservation of nondeterminism: when its argument falls outside d_{Nat}, so does its result. Similar effect is achieved for pred by axiom 5.

3.2.3 Order-sorting and predicates

The previous example illustrated the possibility to identify a subset of the carrier as a set of "defined" elements. A more detailed treatment of undefinedness is achieved by an explicit introduction of predicates identifying definition domains for various operations.

Example 3.4 *We extend the specification from example 3.1 with a new constants representing subsort pos of Nat.*

$$
\begin{array}{rll}
\textbf{spec Nat3} = & & \\
\mathbf{S}: & & Nat \\
\Omega: & & zero :\to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
& & pos :\to Nat \\
\textbf{det}: & & zero, succ \\
\textbf{axioms}: & 1. & succ(zero) \prec pos \\
& 2. & x \prec pos \Rightarrow succ(x) \prec pos \\
& 3. & zero \prec pos \Rightarrow \\
& 4. & x \prec pos \Rightarrow pred(x) \doteq pred(x) \\
& 5. & pred(succ(x)) \doteq x
\end{array}
$$

We have here a more detailed description of definition domain for pred: the first two axioms include there all positive numbers and the third one excludes from it zero. Then, according to axiom 4., if the argument of pred comes from its definition domain pos, then the result is defined in the sense that it is deterministic.

A more detailed notion of strictness than in example 3.3 follows now from this more detailed specification of the definition domain pos. The above specification is consistent with the axiom

$$6. \quad \Rightarrow pred(x) \doteq errorNat, x \prec pos$$

i.e., the requirement that the result of pred(x) $\doteq errorNat$ *for some new (error) constant, when applied to an* $x \not\prec pos$ *(i.e., an* $x \notin pos$*).*

Alternative axiom

$$6'. \quad pred(x) \doteq pred(x) \Rightarrow x \prec pos$$

would make pred strict with respect to pos: if the argument does not belong to the domain pos, then the result is undefined.

If this example reminds you of order-sorted algebras [8], then it was also our intention. Recasting order-sorted formalism within the present framework should be a simple exercise. The additional power can be used, for example, to specify exactly the relations between subsorts (similarly to axiom 3.) Notice also, that we don't run into the known problem with typing intuitively correct terms: $pred(pred(succ(succ(zero))))$ is a correct term. The role played by retracts in order-sorted approach is here overtaken by the proof obligation: to decide that it is defined, one has to show that the argument $pred(succ(succ(zero)))$ actually yields something belonging to pos.

Definedness predicates from example 3.3 in 3.2.2 and subsorts from the above example 3.4 reflect only methodological aspects - formally they are just constants in the specfication. A particular consequence of this is that they can be introduced in arbitrary order during development, and their relations can be specified quite tightly. For instance, one will typically introduce first definedness predicates, like d_{Nat} and later subsorts, like pos, for a tighter specification of definition domains. Extending axioms 1.-3. from example 3.3 with axioms 1.-2. from example 3.4 and adding the axiom

$$x \prec d_{Nat} \Rightarrow x \doteq zero, x \prec pos$$

will enforce the expected relation between the three constants in every model A, namely: $d^A_{Nat} = \{zero^A\} \cup pos^A$ and $zero^A \notin pos^A$.

3.2.4 Strictness

The axioms 4. in example 3.3 and 6. in example 3.4 exemplify the general way of enforcing strictness of required operations.

Schema 3.5 *Enforcing strictness:*

1. *partial function* $f : s_1 \times \cdots \times s_n \to s$
2. *definedness constant for each argument* $1 \le i \le n$ *(relative to the function):* $d_{f_{s_i}} :\to s_i$

3. *value sort function definedness constant:* $d_{f_s} :\to s$
4. *strictness axioms:* $f(x_1,\ldots,x_n) \prec d_{f_s} \Rightarrow x_i \prec d_{fs_i}$*, for each argument* x_i *in which* f *is to be strict.*

If no definition domains are specified but one merely has the definedness constants for the subsets of the well-defined elements of each sort (like d_{Nat}), these later constants will be used instead. In particular, the value sort definedness constant 3. will, typically, be of this kind.

This solution works in many cases but is not fully general. A function with two (or more arguments), $f : s_1 \times s_2 \to s$ may have a definition domain which cannot be described as a product of some subsets of s_1 and s_2. (E.g. $f(1,2)$ and $f(2,3)$ may be defined while $f(1,3)$ is undefined.) In such cases, one needs a more general technique, similar to that used in guarded algebras [13]. It introduces, for each partial function f, an explicit definition domain function dom_f as follows:

Schema 3.6 *General way of enforcing strictness:*

1. *partial function* $f : s_1 \times \cdots \times s_n \to s$
2. *definition domain function:* $dom_f : s_1,\ldots,s_n \to s$
3. *value sort definedness constant:* $d_s :\to s$
4. *definedness axiom:* $dom_f(x_1,\ldots,x_n) \prec d_s \Rightarrow f(x_1,\ldots,x_n) \prec d_s$.

It seems like we have added a lot of constants and functions to treat strictness. Indeed, we have. To defend this wastage of ink we can say two things. Firstly, the above schemata can be added purely mechanically having a mere indication of a function being strict in some arguments. More importantly, one of the schemata is used only when a function is to be strict. We believe that, in practice, most functions are not or, in any case, should not be. Eventually, one wants to describe explicitly error handling. For this purpose, we find it more natural to allow the operations to be, by default, non-strict. And in such a case, our formalism does not force one to do any additional specification work.

3.2.5 Specifying evaluation strategies

In our setting, strictness is specified using explicit constants for definition domains. We finish this section by giving a few simple examples of specifying various evaluation strategies for boolean operations.

Example 3.7 *Basic specification of booleans.*

spec Bool$^{\mathcal{MA}}$ =
include: Nat

S :	$Bool$
Ω :	$d_{Bool} \to Bool$
	$true :\to Bool$
	$false :\to Bool$
	$not : Bool \to Bool$
	$and : Bool \times Bool \to Bool$
	$if\ \_\ then\ \_\ else\ \_ : Bool \times Nat \times Nat \to Nat$
axioms :	**det** $: not, true, false$
	$true \prec d_{Bool}$
	$false \prec d_{Bool}$
	$not(true) \doteq false$
	$not(false) \doteq true$
	$(if\ true\ then\ x\ else\ y) \doteq x$
	$(if\ false\ then\ x\ else\ y) \doteq y$

Adding axiom '$x \prec d_{Bool} \Rightarrow x \doteq true, x \doteq false$*' would make* $true$ *and* $false$ *the only defined boolean values in all models. In the above specification, the same will be the case in the initial model. The operation* $if\ \_\ then\ \_\ else\ \_$ *is non-strict in the second and third argument.*

Example 3.8 *Strict evaluation of and operation.*

$$
\begin{array}{ll}
\textbf{spec StrictBool}^{\mathcal{MA}} = & \\
\textbf{enrich: Bool by:} & \\
\textbf{axioms}: & a.\quad x \prec d_{Bool} \Rightarrow x \; and \; true \doteq x \\
& b.\quad x \prec d_{Bool} \Rightarrow x \; and \; false \doteq false \\
& c.\quad y \prec d_{Bool} \Rightarrow true \; and \; y \doteq y \\
& d.\quad y \prec d_{Bool} \Rightarrow false \; and \; y \doteq false
\end{array}
$$

Note that and is underspecified outside d_{Bool}. The axioms b. and d. force and to have the expected result when the arguments come from d_{Bool} but do not exclude the possibility that this happens also when they do not. Strictness – in the sense that the result of and is included in d_{Bool} only if the arguments are – is obtained by adding two furhter axioms:

$$
\begin{array}{ll}
e. & x \; and \; y \prec d_{Bool} \Rightarrow x \prec d_{Bool} \\
f. & x \; and \; y \prec d_{Bool} \Rightarrow y \prec d_{Bool}
\end{array}
$$

Example 3.9 *Left to right evaluation of and:*

$$
\begin{array}{ll}
\textbf{spec LtoRBool}^{\mathcal{MA}} = & \\
\textbf{enrich: Bool by:} & \\
\textbf{axioms}: & a.\quad true \; and \; y \doteq y \\
& b.\quad false \; and \; y \doteq false
\end{array}
$$

This evaluation may be non-strict in the first argument (it is here underspecified for first argument from outside of d_{Bool}). Left to right evaluation, strict in the the first argument, is obtained by adding the axiom:

$$
c.\quad x \; and \; y \prec d_{Bool} \Rightarrow x \prec d_{Bool}
$$

Example 3.10 *Possibly non-strict parallel evaluation.*

$$
\begin{array}{ll}
\textbf{spec NSPBool}^{\mathcal{MA}} = & \\
\textbf{extend: Bool by:} & \\
\textbf{axioms}: & a.\quad true \; and \; y \doteq y \\
& b.\quad false \; and \; y \doteq false \\
& c.\quad x \; and \; true \doteq x \\
& d.\quad x \; and \; false \doteq false
\end{array}
$$

We have merely indicated the possibilities of obtaining non-stric evaluations. Possible errors and error recovery are left for a more detailed specification of the possible values outside d_{Bool}.

4 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 intial, most abstract level, 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, error situations are identified and we indicate several possible ways to do that. Then one can begin explict error handling, first by specifying the behaviour of other operations in error situations and, eventually, by introducing explicit error values. A great flexibility of error treatement is offered enabling one to introduce error values, exceptions and various ways of reacting to them. It is to be remarked that the whole process involves merely a gradual refinement of initial specification by extending its signature and the set of axioms.

Subsection 4.1 gives a simple example and discusses various alternatives and subsection 4.2 summarizes the methodological observations.

4.1 An example

As an example of the possibilities offered by multialgebras, we discuss various ways of developing a specification of stacks of natural numbers. The process starts from an abstract specification quite similar to partial algebra specification and ends with a specification with explicit error treatement.

4.1.1 Initial specification

We start with a standard development of specification not addressing any error situations explicitly.

$$
\begin{array}{rll}
\multicolumn{3}{l}{\textbf{spec Stack}^{\mathcal{MA}} =} \\
\textbf{include} & & \textbf{Nat} \\
\textbf{S}: & & Stack \\
\Omega: & & empty :\rightarrow Stack \\
 & & top : Stack \rightarrow Nat \\
 & & pop : Stack \rightarrow Stack \\
 & & push : Nat \times Stack \rightarrow Stack \\
\textbf{det}: & & empty \\
\textbf{axioms}: & 1. & push(x,s) \doteq push(x,s) \Rightarrow top(push(x,s)) \doteq x \\
 & 2. & push(x,s) \doteq push(x,s) \Rightarrow pop(push(x,s)) \doteq s
\end{array}
$$

At this abstract level, only *empty* is explicitly specified to be deterministic. Axioms 1. and 2. are the "usual" stack axioms, guarded to yield 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 way axiom 2. should be understood is: if a stack s' is constructed from $push(x, s)$ and s' is defined, then $pop(s')$ is defined and equal to s. Note that 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 specification is essentially the same as a partial algebra specification – just replace the sign $\doteq$ by $\stackrel{e}{=}$. The possibility to reuse partial algebra specifications in our framework is discussed in section 7.

4.1.2 Error situations

Error situations may be identified and marked by appropriate error constants:

1. $$
\begin{array}{rll}
\multicolumn{3}{l}{\textbf{spec ErrStack1}^{\mathcal{MA}} =} \\
\multicolumn{3}{l}{\textbf{enrich Stack by:}} \\
\Omega: & & errStack :\rightarrow Stack \\
 & & errNat :\rightarrow Nat \\
\textbf{axioms}: & 1. & pop(empty) \prec errStack \\
 & 2. & top(empty) \prec errNat
\end{array}
$$

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. $$
\begin{array}{rll}
\multicolumn{3}{l}{\textbf{spec ErrStack2}^{\mathcal{MA}} =} \\
\multicolumn{3}{l}{\textbf{enrich Stack by:}} \\
\textbf{axioms}: & 1. & pop(empty) \doteq pop(empty) \Rightarrow \\
 & 2. & top(empty) \doteq top(empty) \Rightarrow
\end{array}
$$

The inequalities ensure that the error terms cannot be interpreted as individual elements of the carrier. They must be sets – possibly empty. The associated logic prevents then one from substituting such terms for variables. This solution precludes later treatement of particular error situations by means of deterministic error constants. Nevertheless, we will illustrate the flexibility offered by modelling errors by sets.[1]

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

3. **spec OsStack**$^{\mathcal{MA}}$ =
   enrich Stack by:
   Ω : $nonempty :\to Stack$
   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 here as a subsort of non-empty stacks (by axiom 3. *empty* does not belong to this subsort) for which *pop* and *top* are defined. Note that adding "strictness' axioms like: $pop(s) \doteq pop(s) \Rightarrow s \prec nonempty$, would preclude the possibility of deterministic error recovery at a later stage – the axiom requires the result of $pop(empty)$ to be nondeterministic (cf. **ErrStack2** above).

The order-sorted approach may be naturally combined with the explicit error constants, e.g., this specification may extend **ErrStack1**.

4.1.3 Behaviour on errors

Error constants, like those introduced in **ErrStack1**, may be used for a uniform treatement of all errors which they include:

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 determnistic 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 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. $top(push(top(empty), s)) \prec top(s), s \prec pop(empty)$

These two possibilities take full advantage of non-strict semantics allowing one to describe actions to be performed in error situations. Notice that they do not specify explicitly the value of error terms ($pop(empty)$ is merely labeled in the first case and made nondeterministic in the second) – they only specify the behaviour of other operations applied to such error terms. $errStack$ may later be specified to be a particular value or else remain nondeterministic.

[1]Notice that here we are interpreting the result *set* not as a nondeterminisitc result but as an actual union of the elements of the set. Every refinement of this specification – eventually, also the implementation – must conform to this. The only difference between further specification 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. This point is illustrated further in 4.1.4.

The most dramatic possibility is to delegate all responsibility for error treatement to the implementation. 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. $s \prec push(errNat, s') \Rightarrow$

Notice that 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 return the empty set.) Making analogous extension of **ErrStack1** is possible, though it does not seem quite purposeful – labeling error situtations will, typically, involve later their explicit treatement.

4.1.4 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 specification of the behaviour in **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 enirchment is that $empty \prec errStack$ by axiom 1. from **ErrStack1**. Axiom 1. here says that, no matter what we have previously said about the error situation $pop(empty)$, it can be disregarded and that we, instead, do 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.

Another problem might occur from making $errNat$ deterministic. If it is a constant introduced in this specification (as it happened in our example), this might be ok. However, if this error constant comes from another specification **Nat**, we shouldn't force additional restriction since these may interfere with its specification elsewhere.

The really serious problem is caused by the last axiom. It makes $push(errNat, s)$ deterministic. Thus, since $errNat$ is deterministic, 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).

In general, forcing some error terms to be deterministic, requires revisiting earlier developed 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 3.

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

We are forcing $push(errNat, s)$ to be nondeterministic in order to avoid interference with the axioms of **Stack**. The axioms from **ErrStack1a** prescribe recovery from this situation.

The last remaining question concerns now the actual value to be returned by $push(errNat, s)$. We are doing it by specifying the values to be included in the set to be denoted by this term.

On the one hand, we include there some error value – it does not have any influence on the other operations. On the other hand, we include the value which is to be used for the recovery purpose:

spec ErrValStack3$^{\mathcal{MA}}$ =
enrich ErrStack2 by:

Ω : $pushErr :\rightarrow Stack$

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. $s \prec push(x, pushErr) \Rightarrow$

Together, these axioms ensure the desired behaviour (as specified in **ErrStack1a**). The first axiom includes the recovery value s and the second the error value $pushErr$. Thus, the results of other operations applied to $push(errNat, s)$ will be obtained by summing up the results applied to s and to $pushErr$.

The last three axioms make this later value insignificant for other operations – it merely marks the error situation. It can be naturally interpreted as a side effect of $push(errNat, s)$, and can be implemented as, for instance, 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. correspond here to immediate catching this exception. Replacing, e.g., the axiom 3. with $pushErr \prec pop(pushErr)$ 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 as above by the axioms 3.-5. which correspond to catching the exception and performing the further action only on the remaining values in the set – here on s).

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:

3. **spec BoundedStack**$^{\mathcal{MA}}$ =
enrich ErrValStack3 by:

Ω : $max :\rightarrow Nat$
$hgh : Stack \rightarrow Nat$

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 desirbale, one would have to identify it as a new error type with appropriate axioms.

Also, when $hgh(s) \doteq max$, the result of $push(x, s)$ will yield a possibly nondeterministic $errStack$. Nevertheless, its height is defined uniquely and uniformly for all such possible results by axiom 3.

4.2 The methodology for developing partial specifications

The above examples show the wide range of possible error treatements offered by multialgebras and suggest the following fixed methodology for the development of partial specifications in $\mathcal{MA}$.

4.1.1 The first step involves the usual abstract specification which need not address any partiality issues. The only requirement on the axioms – needed for later error handling – is that the arguments of the outermost function applications should be guarded by a definedness assertion, i.e., a determinacy caluse (like $push(x, s) \doteq push(s, x)$ in axioms 2., 3. of **Stack** in 4.1.1). There is no need to guard the variables, since variables are always deterministic. Only operations that can never possibly lead to error situations should be explicitly specified deterministic.

At this level of abstraction the form of the specification is like a partial algebra specification, with $\doteq$ instead of $\stackrel{e}{=}$. In fact, any partial-algebra specification can be reused in this way (see section 7).

4.1.2 The second step identifies error situations and, possibly, introduces order-sorting constants. Among the possibilities here, forcing the error terms to be a set is the solution offering most flexibility in later development (**ErrStack2**).

Altrernatively, one may introduce labels in form of error constants (**ErrStack1**). We do not specify that operations are strict on error constants at this level. The error constants work merely like syntactic marking until we specify the errors closer at a lower level. They are not specified here to be determinisitic (later they may be) and serve merely as negative predicates telling that some terms may yield ill-defined results.

4.1.3 The third step should specify the behaviour of other significant operations when applied to the error constants or terms – this is the stage where we begin the error handling. Error constants may be used for a uniform error handling; alternatively, each error term may obtain separate treatement.

Notice that here we do not introduce explicit error elements but only specify behaviour in error situations. Error recovery means here that we specify an error term to behave as an ordinary term after use of some operations. Strictness means that errors remain (or become other) errors after application of operations. The most extreme case of strictness is to delegate error handling to the implementation by explicitly specifying the result set to be empty (as in **ErrStack3**).

4.1.4 The fourth and last step is to identify the errors with some particular error sets/elements. In some cases, immediate error recovery can be specified (as was the case with $pop(empty)$, which could be made equal $empty$ in **ErrValStack1** or $popEmpty$ in **ErrValStack2**). The general error recovery, however, is that exemplified by $push(errNat, s)$ in **ErrValStack3**: en error situation results in a set comprising the recovery value and an explicit error value which does not affect other operations unless it is passed as an exception.

The first two steps are flexible with respect to error handling. After step 2 we have not taken any decision regarding (non-)strictness. So working at this abstract level it should be possible to have compositional specification building operations, treating the nondeterministic error constants as predicates. From step three one actually specifies consequences of errors so strictness and error handling may interfere with compositionality.

5 Institution of multialgebras

We start this section by recalling the basic notions about institutions.

5.1 Institutions

Definition 5.1 *An institution is a quadruple* $\mathcal{I} = (\mathbf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$*, where:*

- $\mathbf{Sign}$ *a category of signatures.*
- $\mathsf{Sen} : \mathbf{Sign} \to \mathbf{Set}$ *is a functor which associates a set of sentences to each signature.*
- $\mathsf{Mod} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ *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 following satisfaction condition holds: for any* $M' \in \mathsf{Mod}(\Sigma'), \mu : \Sigma \to \Sigma', \phi \in \mathsf{Sen}(\Sigma)$

$$M' \models_{\Sigma'} \mathsf{Sen}(\mu)(\phi) \textit{ iff } (\mathsf{Mod}(\mu))(M') \models_\Sigma \phi$$

The definition can be represented as the following diagram:

$$\begin{tikzcd}
\Sigma \arrow[d, "\mu"'] & \mathsf{Mod}(\Sigma) & \models_\Sigma & \mathsf{Sen}(\Sigma) \arrow[d, "\mathsf{Sen}(\mu)"] \\
\Sigma' & \mathsf{Mod}(\Sigma') \arrow[u, "\mathsf{Mod}(\mu)"'] & \models_{\Sigma'} & \mathsf{Sen}(\Sigma')
\end{tikzcd}$$

Based on the above satisfaction relation we write: $\Gamma \models_\Sigma \varphi$ iff $\forall M \in \mathsf{Mod}(\Sigma) : M \models_\Sigma \Gamma \Rightarrow M \models_\Sigma \varphi$. With this in mind we write $\Gamma^\bullet$ for the semantical consequences of Γ i.e. $\Gamma^\bullet = \{\varphi : \Gamma \models \varphi\}$

A *theory* (specification) in an institution is any pair $\mathbf{Th} = (\Sigma, \Gamma)$ 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 $\mu : (\Sigma, \Gamma) \to (\Sigma', \Gamma')$, where $\mu : \Sigma \to \Sigma'$, is a signature morphism such that: $\Gamma' \models_{\Sigma'} \mathsf{Sen}(\mu)(\Gamma)$. The models for the theory $\mathbf{Th} = (\Sigma, \Gamma)$ is the full sub category $\mathsf{Mod}_\models(\Sigma, \Gamma)$ of $\mathsf{Mod}(\Sigma)$ where $M \in \mathsf{Mod}_\models(\Sigma, \Gamma)$ iff $M \models_\Sigma \varphi, \forall \varphi \in \Gamma$, we will write $\mathsf{Mod}(\Sigma, \Gamma)$ instead of $\mathsf{Mod}_\models(\Sigma, \Gamma)$. The satisfaction condition gives that $\mathsf{Mod}(\mu)(\mathsf{Mod}(\Sigma', \Gamma')) \subseteq (\mathsf{Mod}(\Sigma, \Gamma)$, for each morphism $\mu : (\Sigma, \Gamma) \to (\Sigma', \Gamma') \in \mathbf{Th}$. This means that the functor Mod can be extended to a functor $\mathsf{Mod}_\models : \mathbf{Th}^{op} \to \mathbf{Cat}$. There is a canonic projection functor $\mathsf{sign} : \mathbf{Th} \to \mathbf{Sign}$ and there is an embedding functor $\mathsf{th} : \mathbf{Sign} \to \mathbf{Th}$ defined by $\mathsf{th}(\Sigma) = (\Sigma, \emptyset)$ A theory morphism $\mu : (\Sigma, \Gamma) \to (\Sigma', \Gamma')$ is called axiom preserving if $\mu(\Gamma) \subseteq \Gamma'$. This defines the sub category $\mathbf{Th_0}$ with theories as objects and axiom preserving theory morphisms as morphisms.

Definition 5.2 *Given a functor $\Phi : \mathbf{Th_0} \to \mathbf{Th}'_0$ and a natural transformation $\alpha : \mathsf{Sen} \Rightarrow \mathsf{Sen}' \circ \Phi$, Φ is α-sensible iff:*

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

Where we denote the set of axioms induced by $\Phi(\Sigma)$ by $\emptyset'_\Sigma$.

To relate different institutions we will use *map of institutions* [17].

Definition 5.3 *Given institutions $\mathcal{I} = (\mathbf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$ and $\mathcal{I}' = (\mathbf{Sign}', \mathsf{Sen}', \mathsf{Mod}', \models')$, a map of institutions 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-sensible functor*
- *a natural transformation $\beta : \mathsf{Mod}' \circ \Phi^{op} \Longrightarrow \mathsf{Mod}$*

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

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

The condition from the above definition corresponds to the following commuting diagrams:

$$\begin{tikzcd}
(\Sigma,\Gamma)^{op} \arrow[r, "\Phi^{op}"] \arrow[d, "\mathsf{Mod}"'] & (\Sigma',\Gamma')^{op} \arrow[d, "\mathsf{Mod}'"] \\
\mathbf{Cat} & \mathbf{Cat} \arrow[l, Rightarrow, "\beta"]
\end{tikzcd}
\qquad
\begin{tikzcd}
(\Sigma,\Gamma) \arrow[r, "\Phi"] \arrow[d, "\mathsf{Sen}"'] & (\Sigma',\Gamma') \arrow[d, "\mathsf{Sen}'"] \\
\mathbf{Set} \arrow[r, Rightarrow, "\alpha"] & \mathbf{Set}
\end{tikzcd}$$

Definition 5.4 *An embedding of institutions [18] is a map of institutions $(\Phi, \alpha, \beta) : \mathcal{I} \to \mathcal{I}'$, where the functor $\beta_T : \mathsf{Mod}'(\Phi(T)) \to \mathsf{Mod}(T)$ is an equivalence of categories for each $T \in \mathbf{Th}_\mathcal{I}$. We will use the notation $(\Phi, \alpha, \beta) : \mathcal{I} \hookrightarrow \mathcal{I}'$ to denote an embedding of institutions.*

An α-extension to theories of a functor $\Phi : \mathbf{Sign} \to \mathbf{Th}'_0$ is a functor $\Phi : \mathbf{Th_0} \to \mathbf{Th}'_0$ mapping the theory $\mathbf{Th} = (\Sigma, \Gamma)$ to the theory $\Phi(\mathbf{Th})$ with signature $\Phi(\Sigma)$ and with axioms $\Phi(\Sigma) \cup \alpha_\Sigma(\Gamma)$, for a given natural transformation $\alpha : \mathsf{Sen} \to \mathsf{Sen}' \circ \Phi$.

Definition 5.5 *A map of institutions (Φ, α, β) is:*

- (α) simple *iff Φ is a 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 a 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.*

Definition 5.6 *A subinstitution [17] 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.*

We will use *institutioin transformation*[16] as a formalisiation of model class extension.

Definition 5.7 *Given institutions $\mathcal{I} = (\mathbf{Sign}, \mathsf{Sen}, \mathsf{Mod}, \models)$ and $\mathcal{I}' = (\mathbf{Sign}', \mathsf{Sen}', \mathsf{Mod}', \models')$, a institution transformation 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 satisfaction condition:

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

5.2 Institution of multialgebras, $\mathcal{MA}$

In this section we formally define the institution of multialgebras $\mathcal{MA}$ and prove that it actually is an institution. First we apply the standard concept of reduct to multialgebras.

Definition 5.8 *Let $\mu : \Sigma \to \Sigma'$ be a signature morphism.*
Reduct of an algebra: The μ-reduct of a Σ'-multialgebra A', is the Σ-multialgebra $A'|_\mu$ given by:

$$\begin{aligned} s^{A'|_\mu} &= \mu(s)^{A'} &&, \text{ for all } s \in S, \\ \omega^{A'|_\mu} &= \mu(\omega)^{A'} &&, \text{ for all } \omega \in \Omega, \end{aligned}$$

Reduct of assignment: Given a set of variables X, a Σ'-multialgebra A' and an assignment $\alpha' : \mu(X) \to A'$: the μ-reduct of α' is an assignment $\alpha'|_\mu$ defined by:

$$\begin{aligned} &\alpha'|_\mu : X \to A'|_\mu \text{ and} \\ &(\alpha'|_\mu)_s(x) = \alpha'_{\mu(s)}(\mu(x)) \end{aligned}$$

Reduct of a homomorphism: The μ reduct of a weak Σ' homomorphism $h' : A' \to B'$, is the weak Σ homomorphism $h'|_\mu$ defined by:

$$\begin{aligned} &h'|_\mu : A'|_\mu \to B'|_\mu \text{ and} \\ &(h'|_\mu)_s = h'_{\mu(s)} \end{aligned}$$

Fact 5.9 *The reduct of a multialgebra is a multialgebra, the reduct of an assignment is an assignment to the reduct algebra, the reduct of a weak homomorphism is a weak homomorphism between the reduct of two algebras.*

The proof of this fact is analogous to the classical case.

We are now ready to define the model functor $\mathsf{MAlg} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ which maps each $\Sigma \in |\mathbf{Sign}|$ to the category of all Σ-multialgebras $\mathbf{MAlg}_\Sigma$.

Definition 5.10 *The functor $\mathsf{MAlg} : \mathbf{Sign}^{op} \to \mathbf{Cat}$ is defined by:*

- *objects:* $\mathsf{MAlg}(\Sigma) = \mathbf{MAlg}_\Sigma$
- *arrows:* $\mathsf{MAlg}(\mu : \Sigma \to \Sigma') = \mathsf{MAlg}_\mu : \mathbf{MAlg}_{\Sigma'} \to \mathbf{MAlg}_\Sigma$, *where the functor MAlg_μ is given by:*
  1. $\mathsf{MAlg}_\mu(A') = A'|_\mu$
  2. $\mathsf{MAlg}_\mu(h') = h'|_\mu$

Lemma 5.11 *(Reduct theorem) If $\mu : \Sigma \to \Sigma'$ is a signature morphism, X a set of variables, t a Σ' term, A' a Σ' algebra and $\alpha' : \mu(X) \to A'$ an assignment for A' then we have that:*

$$\alpha'|_\mu(t)^{A'|_\mu} = \alpha'(\mu(t))^{A'}$$

Proof. The proof goes by induction on the complexity of the term t.

1. $t = x \in X_s$.

$$\begin{aligned} &\alpha'|_\mu(x)^{A'|_\mu} \\ =\ &\{\text{ assignment}\} \\ &(\alpha'|_\mu)_s(x) \\ =\ &\{\text{ def. } \alpha'|_\mu\} \\ &(\alpha'_{\mu(s)})(\mu(x)) \\ =\ &\{\text{ assignment}\} \\ &\alpha'(\mu(x))^{A'} \end{aligned}$$

2. $t = c, (c :\to s)$

$$\begin{aligned}
& \alpha'|_\mu(c)^{A'|_\mu} \\
= \quad & \{ \text{ no assignment for constants} \} \\
& c^{A'|_\mu} \\
= \quad & \{ \text{ def. } A'|_\mu \} \\
& \mu(c)^{A'} \\
= \quad & \{ \text{ no assignment, for constants} \} \\
& \alpha'(\mu(c))^{A'}
\end{aligned}$$

3. $t = \omega(t_1, \ldots, t_n), (\omega : s_1 \times \cdots \times s_n \to s)$

$$\begin{aligned}
& \alpha'|_\mu(\omega(t_1, \ldots, t_n)^{A'|_\mu} \\
= \quad & \{ \text{ assignment on function} \} \\
& \textstyle\bigcup_{a_i \in \alpha'|_\mu(t_i)} \omega^{A'|_\mu}(a_1, \ldots, a_n) \\
= \quad & \{ \text{ ind. hyp. and def reduct} \} \\
& \textstyle\bigcup_{a_i \in \alpha'(\mu(t_i))} \mu(\omega)^{A'}(a_1, \ldots, a_n) \\
= \quad & \{ \text{ assignment on function} \} \\
& \alpha'(\mu(\omega)(\mu(t_1), \ldots, \mu(t_n))^{A'} \;=\; \alpha'(\mu(\omega(t_1, \ldots, t_n)))^{A'}
\end{aligned}$$

□

The following lemma leads immediately to the satisfaction condition.

Lemma 5.12 *For any signature morphism $\mu : \Sigma \to \Sigma'$ and Σ' algebra A', given assignment*

- $\alpha : X \to A'|_\mu$, *define* $\alpha' : \mu(X) \to A'$ *by:* $\alpha'_{\mu(s)}(\mu(x)) = \alpha_s(x)$, *and*
- $\alpha' : \mu(X) \to A'$, *define* $\alpha : X \to A'|_\mu$ *by:* $\alpha_s(x) = \alpha'_{\mu(s)}(\mu(x))$, *i.e.* $\alpha = \alpha'|_\mu$

Then for any Σ-formula φ and for any Σ' multialgebra A' we have that:

$$(A'|_\mu) \models_\alpha \varphi \Longleftrightarrow A' \models_{\alpha'} \mu(\varphi)$$

Proof. By induction on the formulae:

1. For atomic formulae $t_1 \doteq t_2$ and $t_1 \prec t_2$, the reduct theorem gives $\alpha'|_\mu(t_i)^{A'|_\mu} = \alpha'(\mu(t_i))^{A'}$, which yields the result.
2. $a_1, \ldots, a_n \Rightarrow b_1, \ldots b_m$
   There are two cases:
   a) for some $a_i : (A'|_\mu) \not\models_\alpha a_i \overset{IH}{\Longleftrightarrow} A' \not\models_{\alpha'} \mu(a_i)$, or
   b) for some $b_k : (A'|_\mu) \models_\alpha b_k \overset{IH}{\Longleftrightarrow} A' \models_{\alpha'} \mu(b_k)$

□

Fact 5.13 *(Satisfaction condition) The satisfaction condition is fullfiled for multialgebras, i.e. for any signature morphism $\mu : \Sigma \to \Sigma'$, for any Σ-formula φ and for any Σ' multialgebra A' we have that:*

$$(A'|_\mu) \models_\Sigma \varphi \Longleftrightarrow A' \models_{\Sigma'} \mu(\varphi)$$

Proof. Let φ be an arbitrary formula.
"$\Leftarrow$": let $\alpha : X \to A'|_\mu$ be arbitrary and let α' be as in lemma 5.12. Then, by assumption, $A' \models_{\Sigma'} \mu(\varphi)$ and, in particular, $A' \models_{\alpha'} \mu(\varphi)$. By lemma 5.12, $(A'|_\mu) \models_\alpha \varphi$. Since α was arbitrary, we obtain $(A'|_\mu) \models_\Sigma \varphi$.
"$\Rightarrow$": let $\alpha' : \mu(X) \to A'$ be arbitrary, and let α be as in lemma 5.12. By assumption, $(A'|_\mu) \models_\Sigma \varphi$, in particular, $(A'|_\mu) \models_\alpha \varphi$. By lemma 5.12, $A' \models_{\alpha'} \mu(\varphi)$. Since α' was arbitrary, we obtain $A' \models_{\Sigma'} \mu(\varphi)$. □

Finally, we define the functor assigning to each signature the set of formulae.

Definition 5.14 *The formula functor* $\mathsf{MSen} : \mathbf{Sign} \to \mathbf{Set}$ *is given by:*

- *objects:* $\mathsf{MSen}(\Sigma)$ = *the set of all* Σ *formulae*

- $\mathsf{MSen}(\mu : \Sigma \to \Sigma') = \mathsf{MSen}_\mu : \mathsf{MSen}(\Sigma) \to \mathsf{MSen}(\Sigma')$ *defined by:*
  1. $\mathsf{MSen}_\mu(t \doteq t') = \tilde{\mu}(t) \doteq \tilde{\mu}(t')$
  2. $\mathsf{MSen}_\mu(t \prec t') = \tilde{\mu}(t) \prec \tilde{\mu}(t')$
  3. $\mathsf{MSen}_\mu(a_1, \ldots, a_n \to b_1, \ldots, b_n)$
     $= \mathsf{MSen}_\mu(a_1), \ldots, \mathsf{MSen}_\mu(a_n) \to \mathsf{MSen}_\mu(b_1), \ldots, \mathsf{MSen}_\mu(b_n)$

With these definitions, lemma 5.13 shows the following:

Proposition 5.15 *The multialgebras form the institution $\mathcal{MA}$ with:*

- *the category* **Sign** *as signatures,*
- *the model functor* MAlg*,*
- *the sentence functor* MSen*,*
- $\models$ *from definition 2.8 as the satisfaction relation.*

5.3 Subinstitutions of $\mathcal{MA}$

As illustrated in sections 3 and 4, multialgebras offer a highly abstract specification formalism. The price of this abstraction is the loss of structure, some closure properties, in particular, such as the existence of initial models. In the following, we will restrict the institution $\mathcal{MA}$ in various ways to relate it to other specification frameworks and, on the other hand, to ensure the existence of initial models.

Fact 5.16 *The multialgebras with Horn clauses is a subinstitution $\mathcal{MAH}$ of $\mathcal{MA}$. All components are as in $\mathcal{MA}$ except for the sentence functor which is* MSen *restricted to Horn clauses over atomic formulae.*

The proof of the satisfaction condition is a special case of the proof for $\mathcal{MA}$, the only difference is that now we only have to consider formulae of the form $a_1, \ldots, a_n \Rightarrow b$. The subinstitution mapping is $(\Phi, \alpha, \beta) : \mathcal{MAH} \to \mathcal{MA}$, where Φ is the identity functor, each component of α is identity and each component of β is the identity functor.

Fact 5.17 *The multialgebras with equational Horn clauses is a subinstitution $\mathcal{MAEH}$ of $\mathcal{MAH}$. All components are as in $\mathcal{MAH}$ except for the sentence functor which is* MSen *restricted to Horn clauses' over element equalities.*

The proof of the satisfaction condition is a special case of the proof for $\mathcal{MAH}$, the only difference is that now we only have to consider formulae build over element equations. The subinstitution mapping is $(\Phi, \alpha, \beta) : \mathcal{MAEH} \to \mathcal{MAH}$, where Φ is the identity functor, each component of α is identity and each component of β is the identity functor.

Fact 5.18 *The multialgebras with atomic sentences is a subinstitution $\mathcal{MAA}$ of $\mathcal{MAH}$. All components are as in $\mathcal{MAH}$ except for the sentence functor which is* MSen *restricted to atomic formulae.*

The proof of the satisfaction condition is a special case of the proof for $\mathcal{MAH}$, the only difference is that now we only have to consider atomic formulas. The subinstitution mapping is $(\Phi, \alpha, \beta) : \mathcal{MAA} \to \mathcal{MAH}$, where Φ is the identity functor, each component of α is identity and each component of β is the identity functor.

6 Initial models for $\mathcal{MA}$ specifications

In general, $\mathcal{MA}$ specifications do not possess initial models [14, 25]. The most powerful logic that admits initial models is Horn logic, [15], so the first restriction would be to consider the subinstitution $\mathcal{MAH}$. However, as showed in [14], even atomic specifications don't always have initial models.

Using essentially the same construction as in [14], we will state the general semantic condition for the existence of initial models – also for specifications using disjunctive axioms (subsection 6.1). Then we will review Hussmann's construction in our framework obtaining conditions ensuring the existence of initial models for atomic specifications (subsection 6.2).

We should emphasize that the results of this section are of relatively limited value and the problem of identifying a satisfactory general subinstitution of $\mathcal{MA}$ admitting initial models is still open.

6.1 The general construction of initial structure

In [14] Hussmann gives sufficient conditions to ensure the existence of initial models when multialgebras are restricted to non empty interpretation of operations and with only atomic formula as axioms. The carrier of the initial model contains all and only deterministic terms. We first review this construction (some lemmata and proofs are different) which leads to a general result showing that it yields an initial model *whenever* it actually belongs to the model class (proposition 6.12).

Definition 6.1 *[14] For a specification $SP = (\Sigma, \Gamma)$, the (deterministic based) term structure $T(SP)$ is defined by:*

- $s^{T(SP)} = \{t : t \in T(\Sigma) \wedge SP \models t \doteq t\}$, *for* $s \in S$.
- *for* $f : \overline{s} \to s$, *the operation* $f^{T(SP)} : \overline{s}^{T(SP)} \to \mathcal{P}(s^{T(SP)})$ *is defined by* $f^{T(SP)}(\overline{t}) = \{t' : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec f(\overline{t})\}$,

The first thing is to ensure that this construction yields a multialgebra. Two conditions from [14] to this goal are: DET-completeness and DET-additivity. The former requires that for every ground term t, there exists at least one deterministic term included in t. Since we are allowing empty set interpretation of terms, this condition is not needed in our case. The second condition is as follows:

Definition 6.2 *[14] A specification $SP = (\Sigma, \Gamma)$ is* DET-additive iff:

$$\forall f : \overline{s} \to s \in \Omega \ : \forall \overline{t}, t \in T(\Sigma) : SP \models t \prec f(\overline{t}) \wedge SP \models t \doteq t \ \Rightarrow$$

$$\exists \overline{t'} \in T(\Sigma) : SP \models t \prec f(\overline{t'}) \wedge SP \models \overline{t'} \prec \overline{t} \wedge SP \models \overline{t'} \doteq \overline{t'}$$

The condition says: if a deterministic value t is produced by an application of an operation f to some terms $\overline{t}$, then this value t must be produced by an application of f to some determnistic elements $\overline{t'}$ of $\overline{t}$. This condition is necessary to be able to define the operations on term structure. It allows to construct the result set of an application $f(\overline{t})$ as the union of the determinisitc terms returned by the applications of f to all deterministic terms (elements) $\overline{t'}$ included in $\overline{t}$.

Fact 6.3 *[14] If SP is a DET-additive specification then $T(SP)$ is a multialgebra.*

We now begin to turn $T(SP)$ into a model for SP.

Lemma 6.4 *There is a unique homomorphism $h : T(SP) \to A$ for every $A \in \mathsf{Mod}(SP)$*

Proof. Define h by $h(t) = t^A$, for all $t \in |T(SP)|$ (remember that $SP \models t \doteq t$).

- h is a homomorphism:

$$\begin{array}{ll} & h(f^{T(SP)}(\overline{x})) \\ = & \{\ \overline{x} = \overline{t} \text{ for some } \overline{t} \in T(\Sigma), \text{ and } SP \models \overline{t} \doteq \overline{t}\} \\ & h(f^{T(SP)}(\overline{t})) \\ = & \{\text{ def. } T(SP)\} \\ & h(\{t' : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec f(\overline{t})\}) \\ = & \{\text{ def. } h\} \\ & \{t'^A : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec f(\overline{t})\} \\ \subseteq & \{\ t'^A \in f^A(\overline{t^A})\} \\ & f^A(\overline{t^A}) \\ = & \{\text{ def. } h \text{ and } SP \models \overline{t} \doteq \overline{t}\} \\ & f^A(\overline{h(t)}) \end{array}$$

- h is unique:
  Suppose $h' : T(SP) \to A$ is another homomorphisms with $h' \neq h$. Then:

$$h'(t) \subseteq t^A$$

  We show this by induction on the term structure of t:

1. $t = c$, c constant:

$$
\begin{array}{ll}
& h'(t) \\
= & \{\text{ assumption}\} \\
& h'(c) \\
\subseteq & \{\ h' \text{ homomorphism}\} \\
& c^A
\end{array}
$$

2. $t = f(\overline{t'})$:

$$
\begin{array}{ll}
& h'(t) \\
= & \{\text{ assumption}\} \\
& h'(f^{T(SP)}(\overline{t'})) \\
\subseteq & \{\ h' \text{ homomorphism}\} \\
& f^A(h'(t'^*)) \\
\subseteq & \{\text{ ind. hyp.}\} \\
& f^A(\overline{t'^A}) \\
= & \{\text{ def. interpretation}\} \\
& (f(\overline{t'}))^A \\
= & \{\text{ assumption}\} \\
& t^A
\end{array}
$$

For the elements $t \in |T(SP)|$ we have that $SP \models t \doteq t$, i.e. $|t^A| = 1$ and from the above we get:

$$h'(t) = t^A = h(t), \forall t \in |T(SP)|$$

□

Definition 6.5 *[22] The* kernel of a homomorphism $h : A \to B$ is the equivalence relation $\sim_h \subset |A| \times |A|$ such that $a \sim_h b$ iff $h(a) = h(b)$

Definition 6.6 *[22] The* quotient of a multialgebra A wrt. an equivalence relation $\sim$ is the multialgebra $A/_\sim$ where:

- $|A/_\sim| = \{[a]_\sim : a \in |A|\}$
- $f^{A/\sim}(\overline{[x]}) = \{[a] : a \in f^A(\overline{x'}), \overline{x'} \in \overline{[x]}\}$

Fact 6.7 *[22] Given an equivalence relation $\sim$ on a multialgebra A then the mapping $h_\sim$: $A \to A/_\sim$, defined by $h_\sim(a) = [a]_\sim$ is a homomorphism (in fact, an epimorphism).*

Fact 6.8 *[22] Given a homomorphism $h : A \to B$, between to Σ multialgebras A, B, there exist homomorphisms $h_1 : A \to A/_{\sim_h}$ and $h_2 : A/_{\sim_h} \to B$, defined by $h_1(a) = [a]_{\sim_h}$ and $h_2([a]_{\sim_h}) = h(a)$ such that h_1 is epi, h_2 is mono and $h = h_1; h_2$.*

Fact 6.9 *[22] If $\sim_1 \subseteq \sim_2$ for two equivalences $\sim_1, \sim_2$ on A then there is a (epi) homomorphism $h : A/_{\sim_1} \to A/_{\sim_2}$.*

Definition 6.10 *Let $\cong$ be the equivalence relation defined on $T(SP)$ by $\cong\ =\ \bigcap_A \sim_{h_A}$: where h_A is the unique homomorphism $h_A : T(SP) \to A$ for each $A \in \mathsf{Mod}(SP)$.*

Lemma 6.11 *There is a unique homomorphism $\phi_A : T(SP)/_\cong \to A$, for each $A \in \mathsf{Mod}(SP)$.*

Proof. We have the following picture:

$$
\begin{tikzcd}
& T(SP) \arrow[dl, "!h_A"'] \arrow[d, "\eta_A"] \arrow[dr, "\eta"] & \\
A & T(SP)/_{\sim_{h_A}} \arrow[l, "h'_A"] & T(SP)/_{\cong} \arrow[l, "h_\sim"]
\end{tikzcd}
$$

There is a unique homomorphism $h_A : T(SP) \to A$, for each $A \in \mathsf{Mod}(SP)$, by lemma 6.4. By fact 6.8 there is then a monomorphism $h'_A : T(SP)/_{\sim_{h_A}} \to A$. By fact 6.9 there is, for each

$A \in \mathsf{Mod}(SP)$, an epimorphism $h_\sim : T(SP)/_{\cong} \to T(SP)/_{\sim_{h_A}}$, since $\sim_{h_A} \supseteq \cong$. Composition of homomorphisms is a homomorphism, so we get a homomorphism $\phi_A : T(SP)/_{\cong} \to A$, defined by $\phi_A = h_\sim; h'_A$. By fact 6.7 there is an epimorphism $\eta : T(SP) \to T(SP)/_{\cong}$. This gives that ϕ_A is unique since h_A is unique and η is epi. □

Lemma 6.11 gives us immediately following proposition:

Proposition 6.12 *If $T(SP)/_{\cong} \in \mathsf{Mod}(SP)$ then $T(SP)/_{\cong}$ is an initial model for SP.*

Of course, checking whether $T(SP)/_{\cong} \in \mathsf{Mod}(SP)$ may be a hard task. Nevertheless, the above proposition gives us a general way of finding initial models irrespectively of the form of specification. For instance, we may construct an initial model for the following specification of binary nondeterministic choice $\sqcup$ which uses disjunction.

Example 6.13 *Initial model for nondeterministic choice.*

$$
\begin{array}{lll}
\multicolumn{3}{l}{\textbf{spec NDChoice}^{\mathcal{MA}} =} \\
 & \mathbf{S}: & s \\
 & \Omega: & a :\to s \\
 & & b :\to s \\
 & & c :\to s \\
 & & d :\to s \\
 & & \sqcup : s \to s \\
\textbf{axioms}: & 1. & a \doteq a \\
 & 2. & b \doteq b \\
 & 3. & c \doteq c \\
 & 4. & d \doteq d \\
 & 5. & x \prec x \sqcup y \\
 & 6. & x \sqcup y \prec y \sqcup x \\
 & 7. & z \prec x \sqcup y \Rightarrow z \prec x, z \prec y
\end{array}
$$

$T(SP)/_{\cong}$ is initial for the above specification, since it is a multialgebra and satisfies all axioms, especially axiom 7.

$$
\begin{array}{l}
s^{T(SP)/_{\cong}} = \{[a],[b],[c],[d]\},\ \textit{where}\ [a] = a, a \sqcup a, \ldots \\
a^{T(SP)/_{\cong}} = [a] \\
b^{T(SP)/_{\cong}} = [b] \\
c^{T(SP)/_{\cong}} = [c] \\
d^{T(SP)/_{\cong}} = [d] \\
(x \sqcup y)^{T(SP)/_{\cong}} = \{x, y\}\ \textit{where}\ x, y \in s^{T(SP)/_{\cong}}
\end{array}
$$

$T(SP)/_{\cong}$ has the following property of standard initial models of deterministic specifications.

Proposition 6.14 *If $T(SP)/_{\cong} \in \mathsf{Mod}(SP)$ then for any ground terms $s, t \in T(\Sigma)$:*

$$T(SP)/_{\cong} \models s \doteq t \iff SP \models s \doteq t.$$

Proof. $\Leftarrow$ is trivial by the assumption. So assume that $T(SP)/_{\cong} \models s \doteq t$. This means, in particular, that $[s], [t] \in |T(SP)/_{\cong}|$, i.e., that $SP \models s \doteq s$ and $SP \models t \doteq t$. For any $A \in \mathsf{Mod}(SP)$, we have a unique homomorphism $\phi_A : T(SP)/_{\cong} \to A$ by lemma 6.11. Thus $A \models s \doteq t$. □

6.2 Initial models for atomic specifications

We show that for an atomic DET-additive specification SP (with only atomic formulae as axioms), $T(SP)/_{\cong} \in \mathsf{Mod}(SP)$, and hence that it is an initial model. We will end up this section with a counter example demonstrating that the conditions for atomic specifications are not sufficient for ensuring the existence of initial models for specifications involving also conditional axioms.

Lemma 6.15 *If $SP \models s \doteq t$, for $s, t \in T(\Sigma, X)$, then we have that: $T(SP)/_{\cong} \models s \doteq t$.*

Proof. Note that for an $A \in \mathsf{Mod}(SP)$, each assignment $\alpha : X \to T(SP)/_{\cong}$ induces a unique assignment $\alpha' : X \to A$, defined by $\alpha' = \alpha; \phi_A$, where ϕ_A is the unique homomorphism $\phi_A : T(SP)/_{\cong} \to A$.

If $SP \models s \doteq t$, than for any given assignment $\alpha : X \to T(SP)/_{\cong}$ we have that $\phi_A(\alpha(s)) = \phi_A(\alpha(t))$ for all $A \in \mathsf{Mod}(SP)$, where ϕ_A is as above. But also $|\phi_A(\alpha(s))| = 1 = |\phi_A(\alpha(t))|$ i.e $\alpha(s) \cong \alpha(t)$, by definition 6.10. Thus, since α was arbitrary, $T(SP)/_{\cong} \models s \doteq t$ □

Lemma 6.16 *[14] For an assignment $\alpha : X \to T(SP)/_{\cong}$ and a term $t \in T(\Sigma, X)$ we have that:* $\alpha(t)^{T(SP)/_{\cong}} = \{[t'] : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(t)\}$

Proof. We have three cases to prove.

- $t = x, x \in X$:

$$
\begin{aligned}
& \alpha(t)^{T(SP)/_{\cong}} \\
= \quad & \{ \text{ assumption}\} \\
& \alpha(x)^{T(SP)/_{\cong}} \\
= \quad & \{ \text{ def. } T(SP)/_{\cong}\text{, lemma 6.15 and } SP \models \alpha(x) \doteq \alpha(x) \ \} \\
& \{[t'] : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(x)\}
\end{aligned}
$$

- $t = c$, c constant:

$$
\begin{aligned}
& \alpha(t)^{T(SP)/_{\cong}} \\
= \quad & \{ \text{ assumption}\} \\
& \alpha(c)^{T(SP)/_{\cong}} \\
= \quad & \{ \text{ no assignment for constants}\} \\
& c^{T(SP)/_{\cong}} \\
= \quad & \{ \text{ def. } T(SP)/_{\cong}\} \\
& \{[t'] : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec c\} \\
= \quad & \{ \text{ no assignment for constants}\} \\
& \{[t'] : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(c)\}
\end{aligned}
$$

- $t = f(t_1, \ldots, t_n)$:

$$
\begin{aligned}
& \alpha(t)^{T(SP)/_{\cong}} \\
= \quad & \{ \text{ assumption}\} \\
& \alpha(f(t_1, \ldots, t_n))^{T(SP)/_{\cong}} \\
= \quad & \{ \text{ assignment}\} \\
& \textstyle\bigcup_{a_i \in \alpha(t_i)} f(a_1, \ldots, a_n)^{T(SP)/_{\cong}} \\
= \quad & \{ \text{ def. } T(SP)/_{\cong}\} \\
& \{[t'] : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \textstyle\bigcup_{a_i \in \alpha(t_i)} f(a_1, \ldots, a_n)^{T(SP)/_{\cong}}\} \\
= \quad & \{ \text{ assignment}\} \\
& \{[t'] : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(f(t_1, \ldots, t_n))\}
\end{aligned}
$$

□

Lemma 6.17 *If $s, t \in T(\Sigma, X)$ and $SP \models s \prec t$ then: $T(SP)/_{\cong} \models s \prec t$:*

Proof. If $SP \models s \prec t$ and α an assignment we have that:

$$
\begin{aligned}
& \alpha(s)^{T(SP)/_{\cong}} \\
= \quad & \{ \text{ lemma 6.16}\} \\
& \{[t'] : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(s)\} \\
\subseteq \quad & \{ \ SP \models s \prec t\} \\
& \{[t'] : t' \in T(\Sigma) \wedge SP \models t' \doteq t' \wedge SP \models t' \prec \alpha(t)\} \\
= \quad & \{ \text{ lemma 6.16}\} \\
& \alpha(t)^{T(SP)/_{\cong}}
\end{aligned}
$$

□

Combining lemma 6.11, lemma 6.15 and lemma 6.17 we get the sufficient condition for the existence of initial models for atomic specifications:

Proposition 6.18 *[14] If SP is an atomic DET-additive specification then $T(SP)/_{\simeq}$ is an initial model for SP.*

One might hope that the conditions of proposition 6.18 will carry over to conditional axioms, but the following example shows that this it not the case.

Example 6.19 *Counter example: DET-additivity does not ensure the existence of initial model for Horn formulae.*

$$\begin{array}{lll}
\multicolumn{3}{l}{\textbf{spec Counterex}^{\mathcal{MAH}} =} \\
\quad \mathbf{S}: & & s \\
\quad \Omega: & & c :\to s \\
 & & d :\to s \\
 & & e :\to s \\
\textbf{axioms}: & 1. & \Rightarrow c \doteq c \\
 & 2. & \Rightarrow d \doteq d \\
 & 3. & \Rightarrow c \prec e \\
 & 4. & \Rightarrow c \doteq e \Rightarrow d \doteq e
\end{array}$$

The following multialgebra A satisfies the specification:

$$\begin{array}{l}
s^A = \{c, d, j\} \\
c^A = c \\
d^A = d \\
e^A = \{c, j\}
\end{array}$$

But $T(SP)/_{\simeq}$ will not satisfy the specification. It looks as follows:

$$\begin{array}{l}
s^{T(SP)/_{\simeq}} = \{[c], [d]\}, \text{ where } [c] = \{c, e\} \text{ and } [d] = \{d\} \\
c^{T(SP)/_{\simeq}} = [c] \\
d^{T(SP)/_{\simeq}} = [d] \\
e^{T(SP)/_{\simeq}} = [c]
\end{array}$$

So $T(SP)/_{\simeq}$ does not satisfy axiom 4. On the other hand, if we take the least congruence on $T(SP)/_{\simeq}$ that satisfies the axioms, we would obtain $[c] = [d]$, but then we have no homomorphism to A – $T(SP)/_{\simeq'}$ satisfies now the equation $c \doteq e$ that is not satisfied by A.

At the moment, we do not have general syntactic conditions ensuring the existence of initial models for conditional specifications and we have to live with the proposition 6.12 as the strongest general result.

7 Partial algebras and multialgebras

In this section we will study the relationships between the institution of partial algebras and $\mathcal{MA}$. We start with the recap of partial algebras. Subsection 7.2 illustrates by example the intended way of reusing partial algebra specifications in our framework. Subsection 7.3 shows an embedding of the institution of partial algebras into $\mathcal{MA}$ and, finally, subsection 7.4 describes another way of relating institutions, namely *institution transormation* from [16], which formalizes the intentions of the example from 7.2.

7.1 Partial algebras

We recall the basic definitions from partial algebras, for a survey see [5, 7, 19].

Partial algebra signatures are, as before, pairs $(\mathbf{S}, \Omega)$ and their category is the same as for multialgebras, **Sign** from definition 2.1.

Definition 7.1 *A partial algebra A for Σ is given by:*

- *A set s^A for each sort $s \in S$*

- *A partial function* $\omega^A : s^A \times \cdots \times s_n^A \to s^A$, *for each symbol* $\omega : s_1 \times \cdots \times s_n \to s \in \Omega$

A (weak) homomorphism between partial algebras A *and* B *is a total function* $h : |A| \to |B|$ *such that* $h(\omega^A(x_1,\ldots,x_n)) = \omega^B(h(x_1),\ldots,h(x_n))$ *for each operation* $\omega \in \Omega$ *and arguments* $x_1,\ldots,x_n$ *where* $\omega^A(x_1,\ldots,x_n)$ *is defined.*

The total functions are special cases of the partial ones. Weak homomorphism h can be equivalently described as an ordinary homomorphism such that: for each operation $\omega \in \Omega$: $h(\mathbf{dom}(\omega^A)) \subseteq \mathbf{dom}(\omega^B)$, where **dom** identifies ω's definition domain in a given algebra A.

Definition 7.2 *Formulae:*

- *Atomic* $t \stackrel{e}{=} t'$ *– existential equalities*
- *Formulae are conditional existential equalities:* $a_1 \wedge \cdots \wedge a_n \Rightarrow a$ *with* $n \geq 0$.

Definition 7.3 *Let* $\alpha : X \to |A|$ *be an assignment (total function):*

- $A \models_\alpha t \stackrel{e}{=} t'$ *iff* $\overline{\alpha}(t)$ *and* $\overline{\alpha}(t')$ *are defined and* $\overline{\alpha}(t) = \overline{\alpha}(t')$
- $A \models_\alpha a_1 \wedge \cdots \wedge a_n \Rightarrow a$ *iff* $\exists i : 1 \leq i \leq n : A \not\models_\alpha a_i$ *or* $A \models_\alpha a$
- $A \models \phi$ *iff* $A \models_\alpha \phi$ *for all* α

Example 7.4 *Following is an example of a partial algebra specification:*

$$
\begin{array}{rll}
\textbf{spec Nat}^{\mathcal{PA}} = & & \\
\textbf{S}: & & Nat \\
\Omega: & & zero : \to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
\textbf{axioms}: & 1. & zero \stackrel{e}{=} zero \\
& 2. & succ(x) \stackrel{e}{=} succ(x) \\
& 3. & pred(succ(x)) \stackrel{e}{=} x \\
& 4. & pred(x) \stackrel{e}{=} pred(x) \Rightarrow succ(pred(x)) \stackrel{e}{=} x
\end{array}
$$

The first two axioms make zero and succ total operations. The third axioms ensures that pred(succ(x)) is defined and equal to x. The last axiom means that if pred(x) is defined then also succ(pred(x)) is defined and equal to x.

We will from now on use **tot** as an abbreviation; axioms 1. and 2. above will be abbreviated as **tot**: $zero, succ$.

7.2 Partial algebras and multialgebras

To relate multialgebras with partial algebras we recall the similarities between the element equalities $\doteq$ from multialgebras and the existential equalities $\stackrel{e}{=}$ in partial algebras. Both equalities hold when both sides of the equality sign denote the same element. In a multialgebra, the equality does not hold if one side of the equality sign is interpreted by a set with cardinality greater than one or by the empty set. The existential equality does not hold in a partial algebra if one side of the equality sign is undefined. (Besides equality does not hold, in both multialgebra and partial algebra, when the elements are different on each side of the equality sign.) These similarities suggest the straightforward relation between multialgebras and partial algebras: replace **tot** by **det** and $\stackrel{e}{=}$ by $\doteq$.

Example 7.5 *The multialgebra specification corresponding to the specification from example 7.4 is as follows:*

$$
\begin{array}{rll}
\textbf{spec Nat}^{\mathcal{MA}} = & & \\
\textbf{S}: & & Nat \\
\Omega: & & zero : \to Nat \\
& & succ : Nat \to Nat \\
& & pred : Nat \to Nat \\
\textbf{det}: & & zero, succ \\
\textbf{axioms}: & 1. & pred(succ(x)) \doteq x \\
& 2. & pred(x) \doteq pred(x) \Rightarrow succ(pred(x)) \doteq x
\end{array}
$$

The partial algebra models and the multialgebra models for the specifications from the above two examples will coincide on their total parts but the difference concerns the undefined parts. Among the partial algebra models, there will be ones where the term $pred(zero)$ has no interpretation – is undefined. Similarly, among the multialgebraic models there will be ones where this term denotes empty set. But we also have the possibility that this term may be nondeterministic, even completely nondeterministic.

If, instead of axiom 2., one writes the axioms $succ(pred(x)) \stackrel{e}{=} x$, respectively $succ(pred(x)) \doteq x$, one forces $succ(pred(x))$ to be always defined. In the partial algebra case, due to strictness of all operations, this forces also $pred(x)$ to be defined. In the multialgebra case this corresponds to a particular error recovery – even if $pred(zero)$ is undefined, the subsequent application of $succ$ will yield $zero$. The model class of the multialgebraic specification has "more" models.

The above example shows our intended goal of allowing reuse of partial algebra specifications in the context where additional flexibility is offered by the extension of the model class. This will be our topic in subsection 7.4. As a preliminary step we show first a direct embedding of $\mathcal{PA}$ into $\mathcal{MA}$.

7.3 Embedding $\mathcal{PA}$ 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.

Definition 7.6 *The functor* $\beta^- : \mathsf{Mod}_{\mathcal{PA}}(\Sigma) \to \mathsf{Mod}_{\mathcal{MA}}(\Sigma)$ *maps a partial algebra A to a multialgebraby given by:*

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

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

For a multialgebra M where all nondeterministic operations return empty set, $\beta(M)$ will denote the corresponding partial algebra, i.e., $\beta^-(\beta(M)) = M$.

Saying that a multialgebra and a partial algebra are "essenitally the same", we will mean that they are obtained from each other by means of $\beta(\_)$, resp. $\beta(\_)$.

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 our, generalized, context need explicit axioms.

For an operation $f(\overline{x})$, the axiom forcing it to be empty or deterministic is of the form $y \prec f(\overline{x}) \Rightarrow f(\overline{x}) \doteq f(\overline{x})$, where y is a fresh variable. We first show that this is the case – i.e., that multialgebras satisfying such axioms (for all operations) are "essentially" partial algebras.

Lemma 7.7 *Let $SP = (S, \Omega, \Gamma)$ be a specification in $\mathcal{MA}$ such that, for each operation $f \in \Omega : SP \models y \prec f(\overline{x}) \Rightarrow f(\overline{x}) \doteq f(\overline{x})$, where y is distinct from all $\overline{x}$. Then in any $M \in \mathsf{Mod}(SP)$ we have that for all $\overline{x} \in |M| : f(\overline{x})^M = \emptyset$ or $f(\overline{x})^M$ is deterministic.*

Proof. Let $M \in \mathsf{Mod}(SP)$ and $\alpha : \{y, \overline{x}\} \to M$ be an assignment. If $M \not\models_\alpha f(\overline{x}) \doteq f(\overline{x})$ then, since $M \models_\alpha y \prec f(\overline{x}) \Rightarrow f(\overline{x}) \doteq f(\overline{x})$, we must have that $M \models_\alpha y \prec f(\overline{x}) \Rightarrow$, i.e., $\alpha(f(\overline{x}))^M = \emptyset$. □

Proposition 7.8 *There is an embedding (Ψ, α, β) of institutions from $\mathcal{PA}$ to $\mathcal{MA}$; the embedding is a simple map.*

Proof.

- 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. $\alpha(t \stackrel{e}{=} t') \equiv t \doteq t'$ for each axiom $t \stackrel{e}{=} t'$

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 β's from definition 7.6, i.e.:
  - $|\beta_\Sigma(M')| = |M'|$
  - $f(x_1, \ldots, x_n)^{\beta_\Sigma(M')} = \begin{cases} \textit{undefined} \text{ if } f(x_1, \ldots, x_n)^{M'} = \emptyset \\ x \text{ such that } f(x_1, \ldots, x_n)^{M'} = \{x\} \text{ otherwise} \end{cases}$

  This is well defined partial algebra by lemma 7.7. For a homomorphism: $h \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \Gamma))$, we define $\beta_\Sigma(h) = h$.

We verify the "thruth" condition: for every $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \emptyset))$ and $\phi \in \mathsf{Sen}_{\mathcal{PA}}(\Sigma)$:

$$M' \models_{\mathcal{MA}} \alpha_\Sigma(\phi) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') \models_{\mathcal{PA}} \phi \tag{1}$$

By the definition of β, for any $\Psi(\Sigma, \emptyset)$-multialgebra M', the Σ-partial algebra $\beta(M')$ has the same carrier, $|M| = |\beta(M')|$. Thus every assignment $\nu' : X \to M'$ is also an assignment $\nu' : X \to \beta(M')$, and vice versa. Given a partial algebra signature $\Sigma = (S, \Omega)$, $\Psi(\Sigma, \emptyset)$ adds an axiom $y \prec \omega(\overline{x}) \Rightarrow \omega(\overline{x}) \doteq \omega(\overline{x})$, for every $\omega \in \Omega$ – thus for any ground term $t \in T(\Sigma)$, $\Psi(t) = t$ will be interpreted as an element or the empty set in every $\Psi(\Sigma, \emptyset)$-multialgebra M'. We show that for every atom $a \in \mathsf{Sen}_{\mathcal{PA}}(\Sigma)$ and for any assignment $\nu : X \to |M'|$:

$$M' {\models_{\mathcal{MA}}}_\nu \alpha_\Sigma(a) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') {\models_{\mathcal{PA}}}_\nu a \tag{2}$$

Let $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \emptyset))$: $a \equiv s \stackrel{e}{=} t$:

$M' \models_\nu \alpha(s \stackrel{e}{=} t)$
$\iff$ { def. α}
$M' \models_\nu s \doteq t$
$\iff$ { satisfaction}
$\nu(s)^{M'} = \nu(t)^{M'} = e, e \in |M'|$
$\iff$ { def. β}
$\nu(s)^{\beta(M')} = \nu(t)^{\beta(M')} = e \in |\beta(M')|$
$\iff$ { satisfaction}
$\beta(M') \models_\nu s \stackrel{e}{=} t$

The general statement (1) follows easily from the above fact (2). □

We also have an immediate consequence of the above proof:

Fact 7.9 *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 $\mathsf{Mod}_{\mathcal{PA}}(\Sigma, \Gamma)$.*

Proof. The inverse functor $\beta^-_{(\Sigma,\Gamma)}$ sends a partial algebra $P \in \mathsf{Mod}_{\mathcal{PA}}(\Sigma, \Gamma)$ onto a multialgebra $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \Gamma))$ such that $\beta(M') = P$, i.e., it is β^- from definition 7.6. One verifies easily the isomorphism condition. □

As Mossakowski showed in [19] $\mathcal{PA}$ allows to specify exactly the finitely locally presentable categories [1], i.e. we have identified the subinstitution of $\mathcal{MA}$ allowing to specify thes classes of models.

7.4 Extending the model class of $\mathcal{PA}$ specifications

The intention of passing from partial algebra to multilagebra specification is, on the one hand, to be able to reuse specifications which have been written in the former framework and, on the other, to allow for their furhter development with explicit error handling. The embedding of institutions from proposition 7.8 does not address this latter issue since it merely yields essentially the same model class.

What we will do now is to merely import the partial algebra specifications *without* augmenting them with the additional axioms. This will allow us to make the desired transition:

given a partial algebra specification, we translate it trivially into a multialgebraic specification. This transition results in a larger model class, where in addition to essentially the same partial models, we also have the models where operations are non-strict. Further development can now take place in the multialgebraic framework, allowing one to refine the specification to the level of explicit error treatement.

The isomorphism of respective model classes stated in fact 7.9 gives us the following relation called *institution transformation* in [16].

Proposition 7.10 *There is an institution transformation* $(\Psi^*, \alpha, \beta^-)$ *from* $\mathcal{PA}$ *to* $\mathcal{MA}$.

Proof.

- The functor $\Psi^* : \mathbf{Sign}_{\mathcal{PA}} \to \mathbf{Th}_{0\mathcal{MA}}$ is given by: $\Psi^*(S, \Omega) = (S, \Omega, \emptyset)$. 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 $\Psi^*(\Sigma, \Gamma) = (\Sigma, \alpha_\Sigma(\Gamma))$.
- The components of the natural transformation $\beta^- : \mathsf{Mod}_{\mathcal{PA}} \to \mathsf{Mod}_{\mathcal{MA}} \circ (\Psi^*)^{op}$ are β^- from definition 7.6:

From the embedding in proposition 7.8 above we have the "thruth" condition: for every $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \emptyset))$ and $\phi \in \mathsf{Sen}_{\mathcal{PA}}(\Sigma)$ (Ψ is as in proposition 7.8):

$$M' \models_{\mathcal{MA}} \alpha_\Sigma(\phi) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') \models_{\mathcal{PA}} \phi \tag{3}$$

Since $\beta_{(\Sigma,\Gamma)}$ is an isomorphism of categories (fact 7.9) we get that i) for each $P \in \mathsf{Mod}_{\mathcal{PA}}(\Sigma, \emptyset)$ there exists an $M \in \mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \emptyset))$, with $P = \beta(M)$ and $M = \beta^-(P)$. Now, the extension of the model class is "conservative": for a $\mathcal{PA}$ specification (Σ, Γ), $\Psi^*(\Sigma, \Gamma) \subset \Psi(\Sigma, \Gamma)$ and hence $\mathsf{Mod}_{\mathcal{MA}}(\Psi(\Sigma, \Gamma)) \subseteq \mathsf{Mod}_{\mathcal{MA}}(\Psi^*(\Sigma, \Gamma))$. Thus, the M existing by i) above is actually an element of $\mathsf{Mod}_{\mathcal{MA}}(\Psi^*(\Sigma, \Gamma))$, and we obtain the condition for the institution transformation: for all $P \in \mathsf{Mod}_{\mathcal{PA}}(\Sigma, \emptyset)$

$$\beta^-_{(\Sigma,\emptyset)}(P) \models_{\mathcal{MA}} \alpha_\Sigma(\phi) \text{ iff } P \models_{\mathcal{PA}} \phi \tag{4}$$

□

Notice that this proposition does not imply the existence of initial model for $\Psi^*(SP)$. For instance, let $f(t) \overset{e}{=} s$ be the only axiom of SP. The initial partial algebra model will have two elements $[s] = \{s, f(t)\}$ and $[t]$. The latter appears because of the underlying strictness assumption, $f(t) \overset{e}{=} s \Rightarrow t \overset{e}{=} t$. The corresponding assumption is absent in $\mathcal{MA}$ and the construction of $T(\Psi^*(SP))/_{\cong}$ will give only one element in the carrier $[s] = \{s, f(t)\}$, which does not yield a multialgebra.[2] Although this is, perhaps, bad news for the adherents of initial semantics, it seems to be the obvious price for dropping the strictness assumption. And getting rid of this assumption is necessary for a more detailed specification of error situations.

8 Membership algebras and multialgebras

We start by recalling the basic concepts from membership algebras and show, in subsection 8.2, that institution of membership algebras can be embedded into $\mathcal{MA}$.

8.1 Membership algebras

Membership algebras were introduced by Mesguer in [18]. We will use notation coresponding to the notation used elsewhere in the article, which slightly differs from the notation used in [18].

[2] In this trivial example, an initial multialgebra could be obtained by adding the axiom $t \doteq t$. This, however, is an option left to the specifier.

Definition 8.1 *(Membership signature) A (membership) signature Σ is a quadruple $\Sigma = (S, \Omega, P, \pi)$, where S is the names of the sorts and Ω the names of the $\overline{S} \times S$ indexed operations. P is s set of predicate names. π is a function $\pi : P \to S$ which for each predicate $p \in P$ assigns a corresponding sort $s \in S$. By P_s one denotes the predicates of sort s i.e. $\{\pi^{-1}(s) : s \in S\}$.*

The function π labels predicate symbols by sort – its intention is to identify a predicate p with $\pi(p) = s$ as a subset of sort s.

Definition 8.2 *A signature morphism between two membership signatures $\Sigma = (S, \Omega, P, \pi)$ and $\Sigma' = (S', \Omega', P', \pi')$ is a triple $\mu = (\mu_P, \mu_S, \mu_\Omega)$, where $\mu_P : P \to P'$ and $\mu_S : S \to S'$ are functions such that the following diagram*

$$\begin{tikzcd} P \arrow[r, "\mu_P"] \arrow[d, "\pi"'] & P' \arrow[d, "\pi'"] \\ S \arrow[r, "\mu_S"'] & S' \end{tikzcd}$$

commutes, and there exists an ω' for each ω such that:

$$\mu_\Omega(\omega : \overline{s} \to s) = \omega' : \mu_S(\overline{s}) \to \mu_S(s).$$

Definition 8.3 *(Membership algebra) A membership algebra for a signature Σ is:*

- *a many sorted (S, Ω) algebra A*
- *together with an assignment of a subset $p^A \subseteq \pi(p)^A$ for each predicate p.*

Definition 8.4 *(Homomorphism) A Homomorphism $h : A \to B$ is an ordinary total homomorphism which, in addition, satisfies: $h_{\pi(p)}(p^A) \subseteq p^B$*

Definition 8.5 *The axioms used for specifying classes of membership algebras are of the following form*

- *atomic formulas:*
  1. *equations of the form $t = t'$ where $t, t' \in T_\Sigma(X)_s$, terms of the same sort s.*
  2. *membership assertions of the form $t : p$, $p \in P$ and $t \in T_\Sigma(X)_{\pi(p)}$.*
- *universally quantified Horn clauses over atomic formulae, i.e. they have form:*
  1. $(\forall X) t = t' \Leftarrow u_1 = v_1 \wedge \ldots \wedge u_n = v_n \wedge w_1 : p_1 \wedge \ldots \wedge w_k : p_k$ *or*
  2. $(\forall X) t : p \Leftarrow u_1 = v_1 \wedge \ldots \wedge u_n = v_n \wedge w_1 : p_1 \wedge \ldots \wedge w_k : p_k$

  where X is a S sorted set containing all the variables appearing in the clause.

The assignments of values to variables are as usual. Then one defines satisfaction of formulae.

Definition 8.6 *(Satisfaction of formulae)*

1. $A \models_\alpha t = t'$ *iff* $\overline{\alpha}(t) = \overline{\alpha}(t')$
2. $A \models_\alpha t : p$ *iff* $\overline{\alpha}(t) \in p^A$
3. $A \models_\alpha a_1 \ldots a_n \Rightarrow a$ *iff* $\exists i : 1 \leq i \leq n : A \not\models_\alpha a_i$ *or* $A \models_\alpha a$
4. $A \models \varphi$ *iff* $A \models_\alpha \varphi \; \forall \alpha$

It is shown in [18] that the membership algebras with above formulae and satisfaction form a liberal institution $\mathcal{MEMB}$. Moreover is there embeddings both ways between $\mathcal{MEMB}$ and the institution of many sorted Horn logic with predicates and equalities, i.e. this to institutions can be viewed as sublogics of each other. Note that if there are no ground terms included in a predicate then the predicate is represented by the empty set in the initial model.

8.2 Embedding membership algebras into multialgebras

The embedding is based on the fact that nondeterministic constants play the same role as predicates. Hence the membership relation $s : p$ is naturally translated as $s \prec p$. Making, in addition, all operations deterministic, one obtains the straightforward translation of $\mathcal{MEMB}$ specifications into $\mathcal{MA}$.

Example 8.7 *A membership algebra specification of (a part of) natural numbers and the corresponding multialgebra specification using only nondeterminisitc constants:*

$\mathbf{specNat}^{\mathcal{MEMB}} =$		$\mathbf{specNat}^{\mathcal{MA}} =$	
$S:$	Nat	$S':$	Nat
$\Pi:$	$\pi(nat) = Nat$	$\Omega':$	$nat :\to Nat$
	$\pi(pos) = Nat$		$pos :\to Nat$
$\Omega:$	$zero \to Nat$		$zero \to Nat$
	$succ : Nat \to Nat$		$succ : Nat \to Nat$
	$pred : Nat \to Nat$		$pred : Nat \to Nat$
axioms		**axioms :**	$\mathbf{det} : zero, succ, pred$
	$zero : nat$		$zero \prec nat$
	$x : pos \Rightarrow x : nat$		$x \prec pos \Rightarrow x \prec nat$
	$x : nat \Rightarrow pred(succ(x)) = x$		$x \prec nat \Rightarrow pred(succ(x)) \doteq x$
	$x : pos \Rightarrow pred(x) : nat$		$x \prec pos \Rightarrow pred(x) \prec nat$
	$x : nat \Rightarrow succ(x) : pos$		$x \prec nat \Rightarrow succ(x) \prec pos$

The only difference is that the multialgebra specification needs to ensure determinacy of all the operations corresponding the the operations Ω from the membership specification (but not of the constants corresponding to the predicates Π).

Both of the above specifications will have the same initial model and, as a matter of fact, the same model class.

As the above example suggests, we can translate a membership algebra specification to a multialgebra specification, which will possess the same models.

Proposition 8.8 *There is an embedding (Φ, α, β) of institutions from $\mathcal{MEMB}$ to $\mathcal{MA}$; the embedding is a simple map of institutions.*

Proof.

- The functor $\Phi : \mathbf{Sign}_{\mathcal{MEMB}} \to \mathbf{Th}_{\mathcal{MA}}$ is given by: $\Phi(S, \Omega, \Pi)$ is the theory $(S, \Omega \uplus \Pi', \emptyset_\Sigma)$, for each $p \in \Pi$ is there a coresponding constant $p \to \pi(p)$ in Π' and where $\emptyset_\Sigma$ contains an axiom $\omega(\overline{x}) \doteq \omega(\overline{x})$, for each operation $\omega \in \Omega$. $\Phi(\mu_S, \mu_\Omega, \mu_P)$ is the signature morphism $(\mu_S, \mu_\Omega \uplus \mu_P)$
- The natural transformation $\alpha : \mathsf{Sen}_{\mathcal{MEMB}} \to \mathsf{Sen}_{\mathcal{MA}} \circ \Phi$ is given by:
  1. $\alpha(t : c) \equiv t \prec c$ for each atom $t : c$
  2. $\alpha(t = t') \equiv t \doteq t'$ for each atom $t = t'$
  3. $\alpha(a_1 \wedge \cdots \wedge a_n \Rightarrow a) \equiv \alpha(a_1) \wedge \cdots \wedge \alpha(a_n) \Rightarrow \alpha(a)$ for Horn clause $a_1 \wedge \cdots \wedge a_n \Rightarrow a$

  Φ is extended to a functor $\Phi : \mathbf{Th}_{0\mathcal{MEMB}} \to \mathbf{Th}_{0\mathcal{MA}}$ by $\Phi(\Sigma, \Gamma) = (\Sigma, \emptyset_\Sigma \cup \alpha_\Sigma(\Gamma))$.
- The natural transformation $\beta : \mathsf{Mod}_{\mathcal{MA}} \circ \Phi^{op} \to \mathsf{Mod}_{\mathcal{MEMB}}$ is essentially the identity on models and homomorphisms. For any multialgebra $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Phi(\Sigma, \Gamma))$
  - $|\beta(M')| = |M'|$
  - $f(x_1, \ldots, x_n)^{\beta(M')} = x$ such that $f(x_1, \ldots, x_n)^{M'} = \{x\}$
  - $p^{\beta(M')} = p^{M'}$

  For any homomorphisms: $h : M' \to B' \in \mathsf{Mod}(\Phi(\Sigma, \Gamma))$, we let $\beta(h) = h$.

We verify the "thruth" condition: for every $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Phi(\Sigma, \emptyset))$ and $\phi \in \mathsf{Sen}_{\mathcal{MEMB}}(\Sigma)$:

$$M' \models_{\mathcal{MA}} \alpha_\Sigma(\phi) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') \models_{\mathcal{MEMB}} \phi \tag{5}$$

By the definition of β, for any $\Phi(\Sigma, \emptyset)$-multialgebra M', the Σ-membership algebra $\beta(M')$ has the same carrier, $|M| = |\beta(M')|$. Thus every assignment $\nu' : X \to M'$ is also an assignment

$\nu' : X \to \beta(M')$, and vice versa. Given a membership signature $\Sigma = (S, \Omega, \Pi)$, $\Phi(\Sigma, \emptyset)$ adds an axiom $\omega(\overline{x}) \doteq \omega(\overline{x})$, for every $\omega \in \Omega$ – thus for any ground term $t \in T(\Omega)$, $\Phi(t) = t$ will be interpreted as an element in every $\Phi(\Sigma, \emptyset)$-multialgebra M'. We show that for every atom $a \in \mathsf{Sen}_{\mathcal{MEMB}}(\Sigma)$ and for any assignment $\nu : X \to |M'|$:

$$M' {\models_{\mathcal{MA}}}_{\nu} \alpha_{\Sigma}(a) \text{ iff } \beta_{(\Sigma,\emptyset)}(M') {\models_{\mathcal{MEMB}}}_{\nu} a \tag{6}$$

Let $M' \in \mathsf{Mod}_{\mathcal{MA}}(\Phi(\Sigma, \emptyset))$ – we have two possible atoms:

1. $a \equiv s = t$:

$$\begin{aligned}
& M' \models_{\nu} \alpha(s = t) \\
\Longleftrightarrow & \{ \text{ def. } \alpha\} \\
& M' \models_{\nu} s \doteq t \\
\Longleftrightarrow & \{ \text{ satisfaction}\} \\
& \nu(s)^{M'} = \nu(t)^{M'} = \{e\}, \; e \in |M'| \\
\Longleftrightarrow & \{ \text{ def. } \beta\} \\
& \nu(s)^{\beta(M')} = \nu(t)^{\beta(M')} = e \in |\beta(M')| \\
\Longleftrightarrow & \{ \text{ satisfaction}\} \\
& \beta(M') \models_{\nu} s = t
\end{aligned}$$

2. $a \equiv s : p$:

$$\begin{aligned}
& M' \models_{\nu} \alpha(s : p) \\
\Longleftrightarrow & \{ \text{ def. } \alpha\} \\
& M' \models_{\nu} s \prec p \\
\Longleftrightarrow & \{ \text{ satisfaction}\} \\
& \nu(s)^{M'} \subseteq \nu(p)^{M'} \\
\Longleftrightarrow & \{ \; M' \in \mathsf{Mod}(\Phi(\Sigma, \emptyset)) \text{ and } s \notin \Pi\} \\
& \nu(s)^{M'} \in \nu(p)^{M'} \\
\Longleftrightarrow & \{ \text{ def. } \beta\} \\
& \nu(s)^{\beta(M')} \in \nu(p)^{\beta(M')} \\
\Longleftrightarrow & \{ \text{ satisfaction}\} \\
& \beta(M') \models_{\nu} s : p
\end{aligned}$$

The general statement (5) follows easily from the above fact (6). □

Fact 8.9 *The functor $\beta_{(\Sigma,\Gamma)}$ is an equivalence (in fact, an isomorphism) of categories* $\mathsf{Mod}_{\mathcal{MA}}(\Phi(\Sigma, \Gamma))$ *and* $\mathsf{Mod}_{\mathcal{MA}}(\Sigma, \Gamma)$ *for every $\mathcal{MEMB}$ theory* (Σ, Γ).

Proof. The inverse functor $\beta^{-1}_{(\Sigma,\Gamma)}$ sends a membership algebra $M \in \mathsf{Mod}(\Sigma, \Gamma)$ onto a multialgebra $M' \in \mathsf{Mod}(\Phi(\Sigma, \Gamma))$ such that $\beta(M') = M$ (i.e., $|M'| = |M|$, $f^{M'}(\overline{x}) = \{f^M(\overline{x})\}$ and for $p \in \Pi : p^{M'} = p^M$.) One verifies easily the isomorphism condition. □

Given a membership algebra specification SP, we will call $\Phi(SP)$ a multialgebra specification *of the membership form.* Restricting the syntax of $\mathcal{MA}$ specifications in this way, we obtain a subinstitution $\mathcal{MAMB}$ of $\mathcal{MA}$ (and $\mathcal{MAH}$).

As a consequence of the above proposition (in particular, that $\beta_{(\Sigma,\Gamma)}$ is an equivalence), we get that the multialgebra models of specifications of membership form possess initial objects. We can verify this fact directly applying proposition 6.12.

Proposition 8.10 *If an $\mathcal{MA}$ specification SP is of the membership form, i.e., $SP = \Phi(SP_{\mathcal{MEMB}})$, then $T(SP)/_{\simeq}$ is a model of PS.*

Proof. 1) We define the multialgebra $D(SP)$, modifying slightly the construction used earlier (definitions 6.1 and 6.10). Let (S, Ω, Π) be the signature of the membership specification $SP_{\mathcal{MEMB}}$ for which we have $SP = \Phi(SP_{\mathcal{MEMB}})$. As the carrier we take

- $|D(SP)| = \{t \in T(\Omega)\} \subseteq \{t \in T(\Omega \uplus \Pi) : SP \models t \doteq t\}$, and we let

- $t^{D(SP)} = \{t\}$, for $t \in T(\Omega)$ and
- $p^{D(SP)} = \{t \in T(\Omega) : SP \models t \prec p\}$, for $p \in \Pi$
- for $p \in \Pi$, $f \in \Omega$, $f(p)$ is defined by pointwise extension: $f(p)^{D(SP)} = \bigcup_{t \in p^{D(SP)}} f(t)^{D(SP)}$.

2) By the same argument as for $T(SP)$ we get that there is a unique homomorphism $h_A : D(SP) \to A$ for each $A \in \mathsf{Mod}(SP)$, so let $\cong$ be the equivalence relation defined on $D(SP)$ by $\cong\ = \bigcap_A \sim_{h_A}$ (cf. 6.10).

3) We show now that $D(SP)/_{\cong} \in \mathsf{Mod}(SP)$. Let $\alpha : X \to D(SP)/_{\cong}$ be an arbitrary assignment – for a term $t \in T(\Omega, X)$, we will denote by $\alpha(t)$ a term obtained by substituting some representative $s' \in \alpha(x) = [s']$ for each occurrence of x in t. Let $s, t \in T(\Omega, X)$, and $p \in \Pi$, and consider possible axioms:

- $s \doteq t$: then, in particular, $SP \models \alpha(s) \doteq \alpha(t)$, and so for any $A \in \mathsf{Mod}(SP) : \alpha(s)^A = \alpha(t)^A$, i.e. $\alpha(s) {\cong} \alpha(t)$, and so $D(SP)/_{\cong} \models_\alpha s \doteq t$
- $s \prec t$: from $SP \models \alpha(s) \prec \alpha(p)$ we obtain that $\alpha(s)^A \in \alpha(p)^A = p^A$ for all $A \in \mathsf{Mod}(SP)$, and so $D(SP)/_{\cong} \models_\alpha s \prec p$.
- Horn formulae: $SP \models_\alpha a_1, \ldots a_n \to a$, where a_i, a atoms of the above form:
  1. If $A \models \alpha(a)$ for all $A \in \mathsf{Mod}(SP)$, then $D(SP)/_{\cong} \models_\alpha a$, by the above arguments.
  2. Otherwise, for some $A \in \mathsf{Mod}(SP) : A \not\models \alpha(a)$. Then we must also have $A \not\models \alpha(a_i)$ for some i. We show that this implies $D(SP)/_{\cong} \not\models_\alpha a_i$ – two cases:
     (a) $a_i \equiv s \doteq t$. It means that $\alpha(s)^A \neq \alpha(t)^A$, hence $\alpha(s) \not\cong \alpha(t)$. By the definition of Φ, both $s' \doteq s'$ and $t' \doteq t'$ are among the axioms of SP (where s, resp. t is obtained from s', resp. t' by substituting some determinisitc terms for some variables). But this means that $D(SP)/_{\cong} \not\models_\alpha s \doteq t$, i.e., $D(SP)/_{\cong} \not\models_\alpha a_i$.
     (b) $a_i \equiv s \prec p$. By the definition of Φ, $s' \doteq s'$ is among the axioms. Thus $\alpha(s)^A \notin \alpha(p)^A$ i.e. $SP \not\models \alpha(s) \prec \alpha(p)$, so $D(SP)/_{\cong} \not\models_\alpha s \prec p$.

This gives that $D(SP)/_{\cong} \in Mod(SP)$ and by 2) that it is initial model.

4) Finally, we show that $D(SP)/_{\cong} \simeq T(SP)/_{\cong}$.

The only difference in the construction was that in $D(SP)/_{\cong}$ we excluded all $p \in \Pi$ which would be taken into account in constructing the carrier of $T(SP)/_{\cong}$, i.e., for all $p \in \Pi : [p] \notin |D(SP)/_{\cong}|$. (We started with $|D(SP)| = \{t \in T(\Omega)\}$ instead of $|T(SP)| = \{t \in T(\Omega \uplus \Pi) : SP \models t \doteq t\}$.) Since $D(SP)/_{\cong} \in \mathsf{Mod}(SP)$, if $SP \models p \doteq p$, we can conclude that there must exist a $t \in T(\Omega)$ (in particular, $SP \models t \doteq t$) such that $p^{D(SP)/_{\cong}} = \{[t]\} = t^{D(SP)/_{\cong}}$ and $D(SP)/_{\cong} \models t \prec p$. But this also means, by the construction of $D(SP)/_{\cong}$, that $SP \models t \prec p$. Hence, in $T(SP)/_{\cong}$ we would obtain $[t] = [p]$. In other words, excluding the symbols Π from the construction does not change the result and $D(SP)/_{\cong} \simeq T(SP)/_{\cong}$. □

8.3 Extending the model class of $\mathcal{MEMB}$ specifications

We will now extend the membership algebras, by allowing functions to be partial.

Definition 8.11 *The institution $\mathcal{PMEMB}$ is as $\mathcal{MEMB}$ except that the equalities are changed to $\stackrel{e}{=}$ and operations are interpreted as partial functions; to relate algebras we use weak homomorphism, i.e. the usuall weak homomorphisms $h : A \to B$ for the operations and for the predicates satisfying the condition $x \in \pi_s^A \Rightarrow h(x) \in \pi_s^B$.*

This is an (obvious) subinstitution of what is called $\mathcal{RP}(\mathbf{R} \stackrel{e}{=}\!\!\to \mathbf{R} \stackrel{e}{=})$ in [19] – an institution where signatures, in addition to sort and function symbols, may contain n-ary predicates and have a distinguished set of total function symbols. There are embeddings both ways between $\mathcal{RP}(\mathbf{R} \stackrel{e}{=}\!\!\to \mathbf{R} \stackrel{e}{=})$ and $\mathcal{PA}$, and $\mathcal{PA}$ is an (obvious) subinstitution of $\mathcal{PMEMB}$. (All this institutions (and many more, see [19]) specify the same categories of models, the finitely locally presentable categories). Composition of the above embeddings is illustrated in the following diagram:

$$
\begin{tikzcd}
& \mathcal{RP}(\mathbf{R} \stackrel{e}{=}\!\!\to \mathbf{R} \stackrel{e}{=}) \arrow[d, hook] \\
\mathcal{PMEMB} \arrow[ur, hook] & \\
& \mathcal{PA} \arrow[ul, hook]
\end{tikzcd}
$$

Fact 8.12 *There is a simple embeding (Φ^*, α, β) from $\mathcal{PMEMB}$ to $\mathcal{MA}$. Moreover, each component of β is an isomorphism of model categories.*

- $\Phi^* : \mathbf{Sign}_{\mathcal{PMEMB}} \to \mathbf{Th}_{\mathcal{MA}}$ *is given by:* $\Phi^*(S, \Omega, \Pi) = (S, \Omega \uplus \Pi', \emptyset_\Sigma)$, *where*
  - *for each* $p \in \Pi$ *there is a coresponding constant* $p :\to \pi(p)$ *in* Π'
  - $\emptyset_\Sigma$ *contains an axiom* $y \prec \omega(\overline{x}) \to \omega(\overline{x}) \doteq \omega(\overline{x})$, *for each operation* $\omega \in \Omega$
  - $\Phi^*(\mu_S, \mu_\Omega, \mu_P)$ *is the signature morphism* $(\mu_S, \mu_\Omega \uplus \mu_P)$
- *The natural transformation* $\alpha : \mathsf{Sen}_{\mathcal{PMEMB}} \to \mathsf{Sen}_{\mathcal{MA}} \circ \Phi^*$ *is given by:*
  1. $\alpha(t : c) \equiv t \prec c$ *for each atom* $t : c$
  2. $\alpha(t \stackrel{e}{=} t') \equiv t \doteq t'$ *for each atom* $t = t'$
  3. $\alpha(a_1 \wedge \cdots \wedge a_n \Rightarrow a) \equiv \alpha(a_1) \wedge \cdots \wedge \alpha(a_n) \Rightarrow \alpha(a)$ *for Horn clause* $a_1 \wedge \cdots \wedge a_n \Rightarrow a$

  Φ^* *is extended to a functor* $\Phi^* : \mathbf{Th}_{0\mathcal{MEMB}} \to \mathbf{Th}_{0\mathcal{MA}}$ *by* $\Phi^*(\Sigma, \Gamma) = (\Sigma, \emptyset_\Sigma \cup \alpha_\Sigma(\Gamma))$.
- *The components of natural transformation* $\beta : \mathsf{Mod}_{\mathcal{MA}} \circ \Phi^{*op} \to \mathsf{Mod}_{\mathcal{PMEMB}}$ *are essentially the identities on models and homomorphisms (cf. β in definition 7.6). For an* $M \in \mathsf{Mod}_{\mathcal{MA}}(\Phi^*(\Sigma, \Gamma))$
  - $|\beta_\Sigma(M)| = |M|$
  - $f(x_1, \ldots, x_n)^{\beta_\Sigma(M)} = \begin{cases} x & \text{– such that } f(x_1, \ldots, x_n)^M = \{x\} \text{ if it exists} \\ undefined & \text{– otherwise} \end{cases}$
  - $p^{\beta_\Sigma(M)} = p^M$ *for all* $p \in \Pi$

  For any homomorphisms: $h : M \to B \in \mathsf{Mod}(\Phi(\Sigma, \Gamma))$, *we let* $\beta(h) = h$.

Using this fact we can construct an institution transformation from $\mathcal{PMEMB}$ to $\mathcal{MA}$ conservatively extending the modelclass of $\mathcal{PMEMB}$ in the same way as we did in proposition 7.10 for $\mathcal{PA}$. This means that we can use the syntax of membership algebra specifications directly in a multialgebra specification and introduce additional "partiality" axioms (corresponding to $\emptyset_\Sigma$ above) at an appropriate stage. Notice also that a "totality" axiom $\omega(\overline{x}) \doteq \omega(\overline{x})$ is a strengthening of the respective "partiality" axiom – thus a "totalization" of partial operation is always possible as a simple refinement step. In particular, one can obtain a membership algebra from a partial (membership) algebra by simply adding such a totality axiom (for each operation). As the embeddings between partial algebras (as in the diagram above and in [19]) show, predicates do not extend the specification power of partial algebras. Using predicates leads, however, to a more natural presentation.

9 Conclusions

We have presented a multialgebra based approach to developing specifications with partial operations. The novelty of the proposed framweork lies in thinking about and modelling 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 modelling 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.

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 also shown that other institutions, in particular, the institution of partial algebras and of membership algebras can be embedded into the institution of multialgebras $\mathcal{MA}$. We thus suggest that this last institution 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 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. We have stated a few preliminary results in the paper but this problem remains open.

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 [23, 24] 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] Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories.* Cambridge University Press, 1994.

[2] Gilles Bernot and Pascale Le Gall. Label algebras: a systematic use of terms. In Bidot and Choppy [4], pages 144–163.

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

[4] Michel Bidot and Christine Choppy, editors. *Recent Trends in Data Type Spesification*, volume 655 of *Lecture Notes in Computer Science.* Springer, 1993.

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

[6] Maura 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.

[7] Maura Cerioli, Till Mossakowski, and Horst Reichel. From total equational to partial first order. In Egidio Astesiano, Hans-Jörg Kreowski, and Bernd Krieg-Brückner, editors, *on Algebraic Foundations of Systems Specification*, chapter 3. Springer. to appear.

[8] Joseph Goguen and Răzvan Diaconescu. An oxford survey of order-sorted algebra. *Journal of Mathematical Structures in Computer Science*, 4:363–392, 1994.

[9] Joseph 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.

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

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

[12] Joseph A. Goguen and José Meseguer. Order-sorted algebra i: Equational deduction for multiple inheritance, overloading, exceptions, and partial operations. Technical Report SRI-CSL-89-10, Computer Science Laboratory SRI International, 1989.

[13] Magne Haveraaen and Eric G. Wagner. Guarded algebras and data type specification. Technical Report 108, Depeartment of Informatics, University of Bergen, 1995.

[14] Heinrich Hussmann. *Nondeterminism in Algebraic Specifications and Algebraic Programs.* Birkhäuser, 1993.

[15] J. A. Makowsky. Why horn formulas matter in computer science: Initial structures and generic examples. *Journal of Computer and System Sciences*, 34:266–292, 1987.

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

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

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

[19] Till 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.

[20] Peter D. Mosses. The use of sorts in algebraic spesifications. In Bidot and Choppy [4], pages 66–91.

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

[22] Michał Walicki and Marcin Białasik. Categories of relational structures. In Francesco Parisi Presicce, editor, *Recent Trends in Algebraic Development Techniques*, volume 1376 of *Lecture Notes in Computer Science*. Springer, 1998.

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

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

[25] Michał Walicki and Sigurd Meldal. Algebraic approaches to nondeterminism-an overview. *ACM Computing Surveys*, 29, 1997.