Programming Research Group

CLASSIFICATIONS IN ALGEBRAIC SPECIFICATIONS OF ABSTRACT DATA TYPES

Simone Veglioni
Programming Research Group
University of Oxford
Oxford, U.K.
e-mail: `veglioni@comlab.ox.ac.uk`

PRG-TR-7-96

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD

Classifications in Algebraic Specifications of Abstract Data Types

Simone Veglioni*
Programming Research Group
University of Oxford
Oxford, U.K.
e-mail: veglioni@comlab.ox.ac.uk

keywords: algebraic specification, sort, strong and weak overloading, many-sorted and universal structures.

Abstract

In this paper we describe the rôle classifications play in algebraic approaches to specification of Abstract Data Types (ADTs) and show how they influence expressivity and mechanizability.

In the last two decades many different logical systems for the algebraic specification of ADTs have been developed, each one pursuing its own principles and goals. Since a careful analysis of these logical systems, taking into account not only their expressivity, but also their mechanizability, is important to clarify and organize a rather rich and controversial subject — for example overloading is a topic that needs to be clarified — we believe our contribution will be useful.

Mechanizability of logical systems is in fact as important as expressivity, and becomes crucial for those logical systems which are ment to define the semantics of declarative and rapid prototyping languages.

We have taken great care to present all arguments in this paper in as elementary a way as possible, in order to make the paper accessible to non-expert readers, even though a minimum knowledge of category theory and institutions might be of help in few points.

1 Introduction

There is a natural way of considering an intended universe of discourse: classifying it into sorted sub-universes, so that one can deal with entities of the same nature (sort). This is a very natural process for human beings, who are used to classifying the world about them according to some common properties. Indeed we are used to talking about plants, animals, dogs ... . Avoiding emphasis on philosophical considerations, we just say that this is a fundamental principle of abstraction, which tries to put some order on the otherwise chaotic universe. It is extremely helpful in case of mechanized reasoning, because it allows to discover many meaningless expression at compile time, and to dramatically reduce the search space of solutions (e.g., see [52]).

A clear classification of the various algebraic methods for the specification of ADTs can be done according to the use they make of sorts. This classification allows us to group together structures having important characteristics in common, in order to facilitate comparison and to discover their main qualities and deficiencies.

To reiterate, we consider a classification of structures based on sorts, i.e., on the notion of sort they embody. We distinguish between a *semantic* (or *dynamic*) notion and a *syntactic* (or

*Supported by European Community, Human Capital and Mobility Program, Contract: ERBCHBICT941002.

static) notion of sort (the reason for using these names will be clearer later). The semantic notion corresponds to the universal view: a single universe is considered, where an element — or a function acting on this universe – even though it may be part of different classifications, is unambiguously determined by its name, without considering its classification. For example, all the books in a library can be classified according to the subject. A particular copy of a book may have a code (that is a name) *CS-332*, a sort *science*, a (sub-)sort *computer science*, a (sub-)sort *concurrency* and so on. Similarly, a particular function to update the code of the books in the library will have a name — *updt* for example — and many classifications: (*all-books* $\rightarrow$ *all-books*); (*history* $\rightarrow$ *history*); (*concurrency* $\rightarrow$ *concurrency*) and so on. If we think of *updt : all-books* $\rightarrow$ *all-books*; *updt : history* $\rightarrow$ *history*; *updt : concurrency* $\rightarrow$ *concurrency* as different functions with the same overloaded name *updt*, we have that all these functions will agree on common elements. Therefore, names *CS-332* and *updt* will determine entities unambiguously.

This view might appear clearer if put in contrast with the many-sorted view, expressed by syntactic classifications. A typical many-sorted classification of numbers may include a sort *nat* for the "world" of naturals (many-sorted = many "worlds") and a sort *bool* for that of booleans. An important phenomenon allowed by this view is that a name may refer to different elements or operation belonging to different worlds. For example, the name 1 may indicate both the natural 1 and the boolean 1; and the name + both addition on naturals and disjunction on booleans — which do not agree on (1,1). It emerges that, in general, a name does not unambiguously determine an entity: the sort (world) considered is also required.

The classification of structures we propose enables us to examine the notion of overloading by relating so-called strong overloading with the syntactic view of sorts, and weak overloading with the semantic one. We believe that this way of presenting overloading may clarify some controversial aspects.

We have concentrated our effort to the analysis of the following structures: *Many-Sorted Algebra* [27, 2, 24, 22, 10], *universal Order-Sorted Algebra* [11, 42, 47], *overloaded Order-Sorted Algebra* [15, 23, 18], *Unified Algebra* [37, 36, 38] and *Heterogeneous Unified Algebra* [41], a novel framework holding, in a sense, both a syntactic and a semantic notion of sort.

We found that all issues concerning classifications raised by other models, including partial ones, are covered by the structures considered. For example, *Equational Type Logic* [30], *Type Horn Logic* [44] and other universal structures, from a classification point of view, are entirely similar to Unified Algebra; while the interesting relation between partiality and dynamicity emerges in Section 5.4.

Remark 1 The purpose of this paper is not to give assessements, but rather to analyze the kind of classification supported. But the choice for a particular notion of sort in a logical system may have many consequences (in terms of expressivity and mechanizability, for example), whose importance will depend on the particular task these systems have to perform. However, we can say that, in general, the existence of initial algebras and completeness results or the ability to express partial computable functions are important properties, but are far from being sufficient. Other aspects like the verification of the variety and quasi-variety theorem [6], the possibility of expressing parameterization and modularization are important in specification languages, as argued by Poigné in [43]: "...Any specification method will and should be judged for the parameterization facilities provided...". Whether the underlying logic forms a (liberal) institution [17] must be considered. Finally, mechanizability is a fundamental aspect, particularly for the definition of declarative and rapid prototyiping languages. Following Meseguer [31], we can say that in order for a logical system to be a (logic) programming language, "... in some pragmatic sense the implementation in the machine should be reasonably *efficient* so that for a broad enough class of applications the language can in fact be used in practice; otherwise such a system should be better described as a *theorem prover*".

In the course of this paper, we show that, often, structures allowing only static classifications have problems about expressivity, while structures allowing only dynamic classifications have problems about machanizability. It will emerge that structures allowing both kinds of classifications might be considered as a successful compromise.

We conclude this introduction by considering another couple of possible interpretations for a sort: *intensional* and *extensional* [40], where the extension of a sort is the set of elements it classifies, while the intension is its meaning. A sort has only one extension but two sorts with the same extension may have more than one intension: the sort of all integers greater than zero has the same extension as the sort of all natural numbers with a well defined reciprocal, and these two sorts may be regarded as different. We will realize how the intensional notion of sort is often related to the semantic one, while the extensional to the syntactic.

In Section 2, 3 and 4 we review basic concepts of Many-Sorted Algebra, and show main concequences due to the syntactic notion of sort embodied. In particular, we will show that strong overloading is not expressible by many-sorted specifications. In Section 5 we introduce order-sortedness and review two main approaches to Order-Sorted Algebra, based on two different notions of sort embodied. Then we show that the choice for a particular notion determines relevant consequences, but not with respect to overloading. In Section 6 we deal with universal structures, and show that having a pure semantic notion of sort gives significative problems concerning modularization and above all mechanizability. In Section 7 we show that even though strong overloading in specifications is not expressed either with static or dynamic classifications, this does not preclude the specification of systems with non-strict inheritance. Finally, in the last two sections, we show how the kind of classification chosen may influence call-by-need techniques and non-determinism.

We will take the liberty of writing MS and OS instead of many-sorted and order-sorted respectively; MSA and OSA instead of MS Algebra(s) and OS Algebra(s) respectively, as well as unOSA and ovOSA instead of universal and overloaded OSA respectively. Finally, we may write ETL for Equational Type Logic, UA for Unified Algebra(s) and HUA for Heterogeneous Unified Algebra(s).

As told, we use in few points very basic categorical notions such has category, initial object and functor, and unfamiliarity with these notions should not discourage the reader. All categorical concepts, in fact, except initiality, are irrelevant to the main subject of the paper. The initial algebra can be simply thought of as the "minimal model" in the sense of Herbrand.

Acknowledgement

I am particularly grateful to Grant Malcolm for his help in smoothing the presentation of an earlier version of this paper, and to Paola Inverardi for very helpful discussions. Also many thanks are due to Joseph Goguen and Peter Mosses for their invaluable comments, which vastly improved the quality of the paper. Finally, I take the occasion to express my gratitude to Franceso Parisi-Presicce for unveiling me the wonderful world of algebraic specifications.

2 Many-Sorted Algebra

The purpose of this section is mainly to introduce notation and basic concepts. Many readers may wish to skip to the next section.

MSA seems to have been first studied by Higgins [27], and then by Benabou [2] who gave a categorical formalization.

Definition 2 A many-sorted signature without overloading is a pair (S, Σ) where S is a set (of sorts) and Σ is an indexed set (of operations) $\{\Sigma_{w,s} \mid w \in S^* \text{ and } s \in S\}$, with $\Sigma_{w,s} \cap \Sigma_{w',s'} = \emptyset$ if $w \neq w'$ or $s \neq s'$. Generally, the rank $\langle w, s \rangle$ is called the functionality of σ if $\sigma \in \Sigma_{w,s}$. We may write $\sigma : w \to s$, $\sigma_{w,s}$ or simply σ for $\sigma \in \Sigma_{w,s}$. We may also write Σ instead of (S, Σ) if the context permits.

Notice that many-sorted signatures without overloading could be formalized without the subscript on Σ by defining a signature as a function from the unstructured set of symbol names Σ to $S^* \times S$. The formalization we adopted is due to Goguen (first appeared in print in [13]), and is preferred in this paper because it is designed to handle overloading, which will be addressed soon later.

Definition 3 Given two MS signatures (S, Σ) and (S', Σ'), a many-sorted signature morphism $\phi : (S, \Sigma) \to (S', \Sigma')$ is a pair (ϕ_1, ϕ_2), where $\phi_1 : S \to S'$ maps sorts in S to sorts in S', and ϕ_2 maps the operation names of Σ to operation names in Σ' in such a way that for $\sigma \in \Sigma_{w,s}$ we have $\phi_2(\sigma) \in \Sigma'_{\phi_1^*(w),\phi_1(s)}$, where ϕ_1^* is the obvious extension of ϕ_1 to strings.

MS signatures and MS signature morphisms form a category.

Definition 4 Given a MS signature (S, Σ), a many-sorted Σ-algebra (Σ-MSA) is an indexed set (of carriers) $\{A_s \mid s \in S\}$ with an indexed family $i_{w,s} : \Sigma_{w,s} \to [A_w \to A_s]$, where A_w is the product $A_1 \times ... \times A_{s_n}$ when $w = s_1 ... s_n$, and $[A \to B]$ is the set of all functions from A to B. We may write $A_\sigma^{w,s}$ or σ_A instead of $i_{w,s}(\sigma)$ when the context permits.

Definition 5 Given a MS signature Σ and two Σ-MSA A and B, a many-sorted Σ homomorphism $h : A \to B$ is an indexed set of functions $\{h_s : A_s \to B_s \mid s \in S\}$ such that: $h_s(\sigma_A(a_1, ..., a_n)) = \sigma_B(h_{s_1}(a_1), ..., h_{s_n}(a_n))$ for $\sigma \in \Sigma_{w,s}$, $a_i \in A_{s_i}$ and $w = s_1 ... s_n$.

For a MS signature Σ, the set of Σ-MSA and the set of Σ-MS homomorphisms form a category that we call MSA(Σ).

Given any signature, we can construct an algebra whose carriers are sets of terms built up from the given operation names viewed as symbols of an alphabet.

Definition 6 Given a many sorted signature Σ, the term-algebra T_Σ is constructed as follows. Let $\cup\Sigma$ be the set of all operation names in Σ; then T_Σ is the least S-sorted set of strings over the alphabet $(\cup\Sigma) \cup \{\underline{(}, \underline{)}\}$ such that:

— for each constant symbol $\sigma \in \Sigma_{[],s}$, the string $\sigma \in (T_\Sigma)_s$;

— for each non-empty list $w = s_1 \ldots s_n \in S^*$, and each $\sigma \in \Sigma_{w,s}$, and all $t_i \in (T_\Sigma)_{s_i}$ for $i = 1, \ldots, n$, the string $\sigma\underline{(}t_1 \ldots t_n\underline{)} \in (T_\Sigma)_s$.

The special symbols '$\underline{(}$' and '$\underline{)}$' are used to emphasise that the carriers of T_Σ are sets of strings; from now on we usually write $\sigma(t_1, \ldots, t_n)$ for $\sigma\underline{(}t_1 \ldots t_n\underline{)}$.

We give T_Σ the structure of a Σ-algebra by interpreting each operation name of Σ: for each $\sigma \in \Sigma_{[],s}$ the constant $(T_\Sigma)_\sigma$ is the string $\sigma \in (T_\Sigma)_s$; for each non-empty list $w = s_1 \ldots s_n \in S^*$ and operation name $\sigma \in \Sigma_{w,s}$ the operation $(T_\Sigma)_\sigma : (T_\Sigma)_w \to (T_\Sigma)_s$ maps a tuple of strings $t_1, \ldots, t_n$ to the string $\sigma(t_1, \ldots, t_n)$.

We may write $(T_\Sigma)_s$ as $T_{\Sigma,s}$ or simply as T_s when the context permits.

Proposition 7 T_Σ is an initial Σ-algebra. For any Σ-algebra A, the unique Σ-homomorphism $h : T_\Sigma \to A$ is defined recursively as follows:

— for each constant symbol $\sigma \in \Sigma_{[],s}$, let $h_s(\sigma) = A_\sigma$;

— for each non-empty list $w = s_1 \dots s_n$ and $\sigma \in \Sigma_{w,s}$ and $t_i \in (T_\Sigma)_{s_i}$ for $i = 1, \dots, n$, let $h_s(\sigma(t_1, \dots, t_n)) = (A_\sigma)(h_{s_1}(t_1), \dots, h_{s_n}(t_n))$.

The homomorphism h assigns values in A to Σ-terms by interpreting the operation names in Σ as the corresponding operations on A.

Definition 8 Given a many sorted signature (S, Σ) and an S-sorted set of variables $X = (x_s)_{s \in S}$ disjoint from the set of constants in Σ, the Σ-term-algebra with variables from X is the term-algebra $T_{\Sigma \cup X}$. We view $T_{\Sigma \cup X}$ as a Σ-algebra by forgetting about the added constant operations, and write it as $T_\Sigma(X)$.

This prepares us for the definition of equations.

Definition 9 A (S, Σ)-equation is a triple (X, l, r) where X is a set of variables and l and r are terms in $T_\Sigma(X)$ of the same sort; i.e., $l, r \in T_\Sigma(X)_s$ for some $s \in S$. We write equations in the form $(\forall X)\ l = r$. Conditional equations generalize equations in the usual way, i.e. they œare expressions of the form $(\forall X)\ l = r$ if C, where the condition C is a finite set of pairs of terms involving only variables in X (when $C = \emptyset$ a conditional equation is regarded as an ordinary equation).

A specification is a triple (S, Σ, E) where (S, Σ) is a signature and E is a set of Σ-equations. We usually abbreviate (S, Σ, E) to just (Σ, E). An algebra A satisfies a given equation iff the left and right-hand sides of the equation are equal under all interpretations of the variables, and a conditional equation $(\forall X)\ l = r$ if C iff l and r are equal under all interpretations of the variables that satisfy the condition.

A model of a specification (Σ, E) is a Σ-algebra satisfying all the equations in E. The models of a specification (Σ, E), together with their homomorphisms, form a category called $\mathrm{MSA}(\Sigma, \mathrm{E})$ — full sub-category of $\mathrm{MSA}(\Sigma)$ — and represents the loose semantics of the specification.

Just as each signature has an initial algebra, each specification has an initial model. The initial model is constructed from the term algebra by identifying exactly those terms that are equal as a consequence of the given equations. We call this term-algebra $T_{\Sigma,\mathrm{E}}$. In practice this is T_Σ quotiented by the congruence relation given by E.

This logic has very important characteristics: it admits initial semantics and allows parameterization and modularization in a very natural way.

But, probably, its main quality is that it has an efficient operational semantics (with respect to what said in Remark 1): each presentation (Σ, E) can be considered as a many-sorted term-rewriting system (TRS). If the TRS obtained is Church-Rosser and terminating[1] the algebra of normal forms is isomorphic to the initial algebra. Therefore we can reason about the initial algebra by using term rewriting. The efficiency is due to these reasons:

— the extension of unsorted TRS towards many-sorted TRS does not present any complication ([33] among others). It results in a very tractable tool,

— this logic allows static-type checking, i.e. checking at compile time of the type of an expression,

[1]In a Church-Rosser and terminating TRS every term has a unique normal form.

— by using sorts, which allow a total treatment of many operations, the space searching of axioms to be matched with a part of term to be evaluated is dramatically reduced,

— the extension of unconditional TRS towards equational conditional TRS does not present any complication ([33] among others).

For these reasons many systems based on many-sorted rewriting have been developed, including commercial ones, such as SOFTEX [53].

However, despite all these qualities, MSA presents also important deficiencies, mainly concerning partial operations [24]. If we insert a predecessor operation in the specification of natural numbers: $pred : Nat \to Nat$, $pred(succ(n)) = n$, we encounter the problem of how to interpret $pred(0)$. This is a serious problem because well-definedness of terms cannot be statically checked.

The problem of partiality has been tackled pursuing different principles and, probably, different goals. A considerable effort has been made on partial models, where operations are partial and errors are represented by undefinedness ([7, 46, 1] among many others). Another research direction focused on many-sorted total models with error elements (see [14, 12, 4, 3] among others), sometimes without success. Direction which evolved towards order-sorted techniques (see section 5 and 6), which are more elegant and more efficient.

3 Overloading

When an operation has more that one functionality we say that (the name of) the operation is overloaded.

Especially in programming we may give different operations the same name, in order to improve the writeability and readability of programs. A common example is the symbol + used to denote the sum of naturals, positives, reals, strings...; or the operation *if_then_else_* whose functionality is in general $bool\, s\, s :\to s$ for all sorts s in the signature.

Overloading is classified as *strong* if an overloaded symbol can be interpreted as two functions which disagree on some common element — thus realizing so-called *ad hoc polymorphism*. A typical example is the symbol + interpreted as both the addition on natural numbers and the addition on (the sub-sort) integers modulo 2 ($1 + 1 = 2$; $1 + 1 = 0$). Overloading is classified as *weak* if all the interpretations of an overloaded symbol must agree on common elements. The polimorphism realized can be called *universal polimorphism*, or *subsort polimorphism* if subsorts are involved (subsorts will be introduced in Section 5). A typical example is the addition on naturals which agrees with the addition on (the sub-sort) positives.

Strong overloading is in principle more powerful than weak overloading in terms of expressivity and freedom of design. However, there is not a general agreement on the efficiency of overloading expressed by some algebraic structures. In order to be clear and fair on this subject let us introduce some requirements that structures providing strong overloading features should match.

Remark 10 A structure providing strong overloading should also offer the possibility of expressing weak overloading, otherwise the advantage would be dubious. A structure would be undoubtly advantageous if it allows to impose a weak overloading interpretation to some function, and a strong overloading interpretation to some other function. Or, even better, to define a function realizing strong overloading with respect to some elements, and weak overloading with respect to some other elements.

On investigating the relationships between the degree of overloading allowed in a structure and the notion of sort embodied, one could soon argue that semantic classifications only allow weak overloading. This is true because, according to the arguments given in the introduction, every overloaded function is an instance (a restriction) of a common universal function. However, strong overloading could be regarded as compatible with the universal view, by considering a strong overloaded function as a non-deterministic function (see [50] for details).

On the other hand, the many-sorted view clearly appears well suited to handle strong overloading.

4 Overloaded Many-Sorted Algebra

The introduction of overloaded operations in a specification is not painless: it gives us more expressive power, but also some complication, as shown afterwards.

The definition of an overloaded MS signature is the definition of a MS signature without the condition: $\Sigma_{w,s} \cap \Sigma_{w',s'} = \emptyset$ if $w \neq w'$ or $s \neq s'$.

The definition of overloaded MS algebra and homomorphism is that given for MS algebra and homomorphism.

Unfortunately, overloading may prevent the term-algebra being initial.

Example 11 Let Σ be $(\{s_1, s_2\}, \{a :\to s_1, a :\to s_2\})$. We have $T_{\Sigma,s_1} = T_{\Sigma,s_2} = \{a\}$, and if we consider the Σ-algebra A with $A_{s_1} = \{1\}$ and $A_{s_2} = \{2\}$ we have that the homomorphism $h : T_\Sigma \to A$ such that $h_{s_1}(a) = 1$ and $h_{s_2}(a) = 2$ can distinguish the two sorts of term a. The term-algebra, in fact, is initial. But if we add to Σ the sort s' and two operations $f : s_1 \to s'$ and $f : s_2 \to s'$, the term-algebra is not initial. In fact the carrier of the term-algebra is: $T_{\Sigma,s_1} = T_{\Sigma,s_2} = \{a\}$, $T_{\Sigma,s'} = \{f(a)\}$, and there is no homomorphism from the term-algebra to the algebra A whose carrier is: $A_{s_1} = \{1\}$, $A_{s_2} = \{2\}$ $A_{s'} = \{f(1), f(2)\}$. This is because the two parsings of the term $f(a)$ cannot be distinguished.

The problem remains even if overloading on constants is forbidden, as the signature whose sorts are s_1, s_2, s_3, s_4, and whose operations are $a :\to s_1$, $f : s_1 \to s_2$, $f : s_1 \to s_3$, $g : s_2 \to s_4$ $g : s_3 \to s_4$ shows.

Fortunately, there is a syntactic way to regain initiality of the term algebra.

Definition 12 An overloaded MS signature (S, Σ) is regular if for all $w \in S^*$; $s, s' \in S$; $\sigma : w \to s$, $\sigma : w \to s'$ implies $s = s'$.

Notice that a regular signature cannot have overloaded constants.

Given a regular signature Σ, the term-algebra is initial [16]. The proof is immediate by structural induction, and the cause is that regularity implies that every term can be parsed in only one way (i.e., carriers of the term-algebra do not overlap).

However, if we allow uncontrolled overloading, the initiality property is not lost, in fact the parsed term algebra[2] is initial. But, this means that, in general, we have to write sentences using parsed terms, or at least giving (minimal) information to disambiguate terms: we eliminate overloading (the previous example may become: $1 +_{Nat} 1 = 2; 1 +_{Z_2} 1 = 0$).

In order to evaluate the degree of overloading expressed by MSA over regular signatures we notice that given a regular signature Σ, a Σ-MSA can interpret the overloading in a strong sense. Also an initial one can do it. But the term-algebra cannot, because, as we have seen, its carriers do not overlap. Furthermore, let us consider a signature Σ whose sorts are s and s', and whose operations are $f : s \to s$ and $f : s' \to s'$. By no means can we impose a weak overloading

[2] It is the algebra of parsed terms. A parsed term is a term equipped, using a dot quantification, with its parse. For example if we have $a :\to s_1$; $a :\to s_2$; $f : s_1 \to s_1$; $f : s_2 \to s_2$; $f : s_1 \to s_2$ then $a.s_1$ and $f.s_2(a.s_1)$ are parsed terms.

interpretation to the symbol f: any Σ-MSA — even an initial one — may have $A_f^{s,s}(a) \neq A_f^{s',s'}(a)$ for all $a \in A_s \cap A'_s$ (cfr. Remark 10).

In assessing the overloading expressed by MSA in general, we should consider many cases: with respect to the loose semantics, w.r.t. the initial semantics, w.r.t. the term-algebra and so on. However, the most important case is that concerning the term-algebra, which reveals the overloading expressible "by design"[3]: we can, therefore, conclude that, as far as this notion of design is considered,

MS specifications do not support strong overloading.

After this discussion it appears that even though many sorted classifications can handle strong overloading, in the algebraic specification setting they fail to do it because the term algebra has a universal nature. This is even clearer when order-sorted specifications (see below) are considered.

It has already been argued that a view expressed by a classification concerns both elements and operations. We now coherently extend the view of a classification to include also homomorphisms, as it emerges from this and the following sections.

5 Order-Sorted Algebra

We have already mentioned order-sortedness as being an elegant and efficient technique to express partial operations in a total framework. In this section we review the structure of Order-Sorted Algebra, which represents the first attempt to address sort-inclusion.

Definition 13 A non overloaded order-sorted signature is a triple $(S, \leq, \Sigma)$ where $(S, \leq)$ is a partial order relation and (S, Σ) is a non overloaded MS signature.

As for MS signature we may denote $(S, \leq, \Sigma)$ simply by Σ. An OS signature Σ can be considered as a MS signature by forgetting the partial order relation; when the context permits we also call the induced MS signature Σ.

Definition 14 Given an OS signature $(S, \leq, \Sigma)$, an order-sorted Σ-algebra (Σ-OSA) A is a Σ-MSA such that: $s \leq s'$ implies $A_s \subseteq A_{s'}$ for all $s, s' \in S$.

In order to define order-sorted homomorphisms we cannot simply inherit the definition given for MS homomorphisms. Let us consider, for example, that if $h : A \to B$ is a MS homomorphism, the image $h(A)$ may not be an OSA, because, for example, from $s \leq s'$ it does not necessarily follow $h(A_s) \subseteq h(A_{s'})$. This is due to the fact that MS homomorphisms are not sensible to the sorts of elements: an element having two sorts is considered as two different elements (because of the purely syntactic notion of sort held). An adequate definition of OS homomorphism depends on which meaning we want to give to sorts. This means that we cannot simply impose the condition that h_s must agree with $h_{s'}$ on the same elements if $s \leq s'$: we have to choose, before, the notion of sort to adopt. In fact, the principle according to which an element with two sorts must be considered as a unique element only if the sorts are related by $\leq$ needs some motivation.

If we decide on a more syntactic approach, an element having two sorts will be considered as two different elements, unless the sorts are related by $\leq$; while, if we pursue a purely semantic approach the element will be a unique individual, and its parses will not be distinguishable by any homomorphism.

[3]For the same reason we say that MSA cannot express sort inclusion even though carriers may be included in other carriers. In fact this inclusion is "accidental" not "by design".

These two alternatives are both sensible, and, based on them, two different approaches to OSA arose during the 80's: *universal OSA* and *overloaded OSA*.

Overloading is very much influenced by the notion of sort supported. As we can imagine, the semantic approach only allows weak overloading, while the syntactic one, as for MSA, only allows a weak form of strong overloading.

Definition 15 An (overloaded) OS signature is as in Definition 13 except that it is built over an overloaded MS signature.

5.1 Universal OSA

The first approach to OSA we analyze is the semantic one [11, 42, 47]. It is based on the fact that what happens for an element having two sorts related by $\leq$ (i.e. it is a unique element) must not be an exception: every element having more than one sort is to be considered as a unique individual. As an immediate consequence we have that homomorphisms cannot distinguish these elements, and that overloaded operations must agree on elements having more than one sort[4].

Definition 16 Given an OS signature $(S, \leq, \Sigma)$, a universal Σ-OSA A is an overloaded Σ-MSA such that for $s, s' \in S$ and $w, w' \in S^*$ we have:
- $s \leq s'$ implies $A_s \subseteq A_{s'}$ and
- $\sigma \in \Sigma_{w,s} \cap \Sigma_{w',s'}$ implies $A_\sigma^{w,s} = A_\sigma^{w',s'}$ in $A_w \cap A_{w'}$.

According to this definition, the various overloaded operations are instances (i.e. restrictions) of a universal operation being their union. This is a universal vision of overloading and the definition of homomorphism is coherent with this view.

Definition 17 Given an OS signature Σ and two universal Σ-OSA A and B, a universal OS homomorphism $h : A \to B$ is a MS homomorphism $(h_s : A_s \to B_s)_{s \in S}$ such that: $h_s(a) = h_{s'}(a)$ if $a \in A_s \cap A_{s'}$.

This means that a universal homomorphism can be considered as a (universal) function $h : \bigcup_{s \in S} A_s \to \bigcup_{s \in S} B_s$, together with the various h_s, instances of h.

An algebra whose carriers overlap is no more isomorphic to one whose carriers do not overlap. In fact an algebra may be seen as $(A, (A_s)_{s \in S}, (A_\sigma)_{\sigma \in \Sigma})$ where $A = \bigcup_{s \in S} A_s$ and $A_\sigma : A_{w_1} \cup ... \cup A_{w_n} \to A$, if $\sigma \in \Sigma_{w_1,s_1} \cap ... \cap \Sigma_{w_n,s_n}$, giving more precisely the flavor of a universal structure.

The view taken with universal OSA is sometimes called *sort-independent semantics*: a legal term should always have the same value, regardless of what sorts are ascribed to its subterms.

The category of universal OSA over a signature Σ — unOSA(Σ) — admits initial algebra, and this is quite obvious if we consider that the condition in Definition 16 imposes the behaviour of the term-algebra on all algebras (if the term algebra has overlapping carriers than also the other algebras must overlap), i.e., overloading is weak. In fact, notice that parsed terms do not form a universal OSA.

Definition 18 Equations of a universal OS specification are as in Definition 9, except that they can involve terms without restriction on their sorts (in coherence with the universality of the structure).

[4] We emphasize that it is a consequence, because this does not clearly emerge from other papers.

Important results have been shown for the logic underlying universal OSA, such as the fact the it forms a liberal institution; however we cannot hope to inherit or generalize the theory developed for MSA because there is no compatibility between universal OSA and MSA. This is an obvious consequence of their different notions of sort. There is no need to show this by examples: it suffices to consider that a MS homomorphism can distinguish an element with two sorts, while a universal OS homomorphism cannot. Notice, that the incompatibility is not determined by overloading.

Concerning overloading, universal OSA can express only weak overloading, in accordance with the semantic view of sort adopted. This should not be seen as a deficiency: we claim that all other algebraic structures cannot express a broader class of overloading. In a sense we also claim that strong overloading is not necessary, as explained later. Furthermore, notice that overloading of constants is not forbidden.

Finally, this section confirms what already realized in Section 3: semantic classifications induce homomorphisms to be universal, and overloading to be weak.

5.2 Overloaded OSA

Overloaded[5] OSA (ovOSA) [15, 23, 18] corresponds to the choice for a more syntactic notion of sort made in order to define OS homomorphisms, as discussed in Section 5.

Definition 19 An ovOSA over a signature $(S, \leq, \Sigma)$ is a Σ-MSA such that $s \leq s'$ implies $A_s \subseteq A_{s'}$. An ovOS homomorphism $h : A \to B$, where A and B are Σ-ovOSA, is a MS homomorphism such that $s \leq s'$ implies $h_s = h_{s'}$ on A_s.

The syntactic choice allows not to compromise the compatibility between MSA and OSA, but the extension of MSA to OSA poses some problem.

The first relevant problem is to obtain initiality of the term-algebra, and is solved by putting some constraints on signatures, as in the following definitions.

For the same reason we put a regularity condition on MS signatures, we put a similar condition on OS signatures. This condition, interpreted in the OS setting, becomes:

Definition 20 An OS signature $(S, \leq, \Sigma)$ is monotone if $\sigma : w \to s$; $\sigma : w' \to s'$ and $w \leq w'$ implies $s \leq s'$.

This condition does not guarantee initiality of the term-algebra, and shows that some constraints must also be put on algebras, so to insert semantics aspects in the notion of sort supported.

Example 21 Let $s \leq s'$; $a :\to s$; $f : s \to s$ and $f : s' \to s'$. Then $T_{s_1} = T_{s_2} = \{a, f(a), ff(a), ...\}$. Let us consider now the algebra $A = ((Z_2, N), 0, succ_2, succ)$, with $succ_2$ successor modulo 2. A homomorphism $h : T_\Sigma \to A$ should have $h_{s_1}(f(a)) = 1 \neq 2 = h_{s_2}(f(a))$. Therefore, no homomorphism exist.

Thus, the definition of overloaded OS homomorphisms requires the following definition of algebra:

Definition 22 A Σ-ovOSA is monotone if $\sigma : w \to s$, $\sigma : w' \to s'$, $w \leq w'$ and $s \leq s'$ implies $A_\sigma^{w,s} = A_\sigma^{w',s'}$ on A_w.

The monotonicity constraints are necessary, as seen, but still not sufficient:

[5]Some author prefers *Goguen-Meseguer* OSA, because the name *overloading* might induce the reader to erroneously believe that unOSA does not support overloading.

Example 23 $s_1 \leq s_2 \leq s_4$; $s_1 \leq s_3 \leq s_4$; $a : \to s_1$; $f : s_2 \to s_2$; $f : s_3 \to s_3$. We have $T_{s_1} = \{a\}$, $T_{s_2} = T_{s_3} = T_{s_4} = \{a, f(a), ff(a), ...\}$. An algebra A may not have the intersection that the term-algebra has. For example $f(a)$ can be interpreted as two different elements of sort s_2 and s_3. Again, there is no homomorphism from the term-algebra to A.

The term-algebra is initial only when there is no "accidental" intersection between its carriers. This means that every carrier must be named by a sort, that is every term has a least sort. Fortunately, sufficient syntactic conditions, exist.

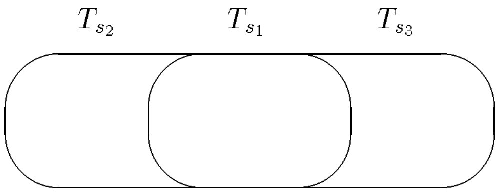
Definition 24 An OS signature $(S, \leq, \Sigma)$ is regular iff it is monotone and, given $\sigma : w_1 \to s_1$, $w_0 \leq w_1$, there exists a least rank $\langle w, s\rangle$ such that $w_0 \leq w$ and $\sigma : w \to s$.

It has been shown that the term-algebra of a regular signature is initial[6]. Regular signatures allow equations to be defined for the overloaded approach.

Definition 25 Equations of an overloaded OS specification are as in Definition 9, except that they can involve terms whose least sorts are in the same connected component, where a connected component of $(S, \leq)$ is an equivalence class of the equivalence closure of $\leq$.

Now that the goal of having the term-algebra initial is reached, let us see what kind of overloading is expressed by monotone ovOSA over regular signatures.

Every intersection $T_{s_2} \cap T_{s_3}$ is a carrier T_{s_1}, with $s_1 \leq s_2$; $s_1 \leq s_3$:



By initiality of the term-algebra we have that if the term-algebra has an intersection, then all other algebras with the same signature will have the same intersection. However, one of these algebra — even an initial one — can have intersections that the term-algebra cannot. Overloading on these parts can be interpreted by the algebra in a strong manner, as the following example shows.

Example 26 Let $s_1 \leq s_2$; $s_1 \leq s_3$; $a : \to s_1$; $b : \to s_2$; $c : \to s_3$; $f : s_1 \to s_1$; $f : s_2 \to s_2$; $f : s_3 \to s_3$. An algebra A must have $A_a^{s_2} = A_a^{s_3}$ and $A_f^{s_2,s_2}(A_a^{s_2}) = A_f^{s_3,s_3}(A_a^{s_3}) \in A_{s_1}$, but may also have $A_b^{s_2} = A_c^{s_3} = d \in A_{s_2} \cap A_{s_3}$ and $A_f^{s_2,s_2}$ may disagree with $A_f^{s_3,s_3}$ on d.

However, the liberty of a generic ovOSA to interpret this overloading as strong is lost because of another constraint to be imposed on signatures. This constraint is necessary because the notion of equality is not abstract for ovOSA, that is, isomorphic algebras do not necessarily verify the same equalities.

Example 27 Let $s_1 \leq s_2$; $s_1 \leq s_3$; $a : \to s_1$; $b : \to s_2$; $c : \to s_3$, and let $b = c$ be an equation. We have $T_{s_1} = \{a\}$, $T_{s_2} = \{a, b\}$ and $T_{s_3} = \{a, c\}$. The term-algebra, which does not satisfy the equation, is isomorphic to the algebra A with $A_{s_1} = \{1\}$, $A_{s_2} = A_{s_3} = \{1, 2\}$ (considering the obvious interpretation of constants) which does satisfy the equation.

[6]There are weaker conditions and also necessary and sufficient conditions [23, 51]. Regularity is generally preferred because of its simplicity.

The reader may notice that the problem is completely independent from overloading, and is essentially due to the fact that the semantic conditions of homomorphisms (Definition 19) imposed for related sorts must be extended to all connected sorts. This is because equations involve terms whose sorts are in the same connected component.

Fortunately, a purely syntactic constraint solving the problem has been found.

Definition 28 An OS signature is coherent if it is regular and locally filtered.

A partial order $(S, \leq)$ is filtered iff for $s, s' \in S$ there exists $s'' \in S$ such that $s, s' \leq s''$. $(S, \leq)$ is locally filtered iff each connected component of $(S, \leq)$ is filtered. We extend those properties to signatures in the obvious way. When referred to an algebra, the connected component means the set of carriers whose set of sorts is a connected component.

We have that for coherent signatures the notion of equality is abstract, and that the category of ovOSA over a signature verifies the McKinsey-Malcev quasi-variety and the Birckoff variety theorems[7] [23].

Local filtering is the most important constraint: all standard constructions and results of ovOSA carry through for any locally filtered signature. Even the monotonicity constraint on algebras can be avoided [18]. This is because all constraints except local filtering are determined by overloading, and can be avoided if overloading is disambiguated.

To understand the overloading expressible with all these constraints we have to consider that local filtering gives a more universal flavor to a connected component. In fact homomorphisms cannot distinguish an element having more than one connected sort. This means that two isomorphic algebras have the same intersections inside the connected components. Consequently, an initial algebra can have only weak overloading inside a connected component, as it is isomorphic to the term-algebra.

Definition 28 does not restrict overloading in non-initial algebras. It just implies that an algebra with strong overloading inside a connected component cannot be isomorphic to an algebra having only weak overloading.

Example 29 Let $s_1, s_2 \leq s_3$; $\sigma : s_1 \to s_1$; $\sigma : s_2 \to s_2$. An algebra A may have $A_{s_1} = \{a, b\}$, $A_{s_2} = \{a, c\}$ with $A_{\sigma_{s_1,s_1}}(a) = b$ and $A_{\sigma_{s_2,s_2}}(a) = c$

The fact that, inside a connected component an element with more than one sort cannot be distinguished by homomorphisms but can be distinguished by overloaded operations, may be considered incoherent (see end of Section 4 and footnote 4), and in fact it is reflected in the operational semantics (order sorted rewriting as we will see) which, in order to be complete, requires consistency [28].

Definition 30 An algebra A is consistent iff $\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$ with w_1, s_1 connected to w_2, s_2 respectively, implies $A_{\sigma_{w_1,s_1}} = A_{\sigma_{w_2,s_2}}$ on $A_{w_1} \cap A_{w_2}$. A homomorphism $h : A \to B$ is consistent iff $h_s = h_{s'}$ in $A_s \cap A_{s'}$ for any connected sorts s, s'.

Fortunately, syntactic conditions which can guarantee consistency of algebras and homomorphisms again exist. In fact consistency of homomorphisms is guaranteed in locally filtered signatures, while consistency of algebras is guaranteed in the following signatures:

Definition 31 A signature $(S, \leq, \Sigma)$ is consistent[8] iff it is locally filtered and such that for $\sigma \in \Sigma_{w_1,s_1} \cap \Sigma_{w_2,s_2}$, with w_1, s_1 respectively connected to w_2, s_2 there is a rank $\langle w, s \rangle$ such that $\sigma \in \Sigma_{w,s}$ with $w_1, w_2 \leq w$ and $s_1, s_2 \leq s$.

[7] In [23] finiteness of the set of sorts is required.

[8] Names used in this paper such as regular and consistent may have different meaning in other papers.

We can say that consistency "completes" local filtering. Notice that regularity is no more needed to guarantee initiality of the term-algebra, in fact if two terms with different sort are equal in the term algebra, then they will be equal in all algebras with same signature, because they will be both equal to a same term with greater sort. Notice also that overloading on constants is allowed only inside a connected component.

This version of OSA is considered as the "right" one by Meseguer in [32], and the view it expresses is very clear: a specification represents a many-sorted world, where each connected component represents a single world. A single world has a universal interpretation: strong overloading is not allowed. Strong overloading is in principle allowed on operations acting on different worlds, but as seen for the many-sorted case, in the term-algebra different worlds do not intersect; and even though an initial algebra may have this form of strong overloading, it is isomorphic to the term-algebra which does not allow strong overloading. Consequently, for the same argument used for the many sorted case, we can conclude that

overloaded OS specifications do not support strong overloading.

This result is obvious because, as told, the term-algebra has an intrinsic universal nature and, consequently, interprets overloading weakly. No constraint can change its universal nature; as seen, constraints can only change the nature of loose semantics, thus reconciling the many-sorted nature of the loose semantics and the universal nature of the term-algebra.

Notice also that, as a connected component describes a single world, the categories of unOSA and ovOSA over a filtered consistent signature are isomorphic. Based on this equivalence Poigné has developed some theory generalizing unOSA and consistent OSA [43] (see also [23] for categorical relations between unOSA and ovOSA).

5.3 Comparison between unOSA and ovOSA

What emerges from the previous analysis is that a unOS specification describes a single world where each classification is semantic, whereas an ovOS specification still describes a single world, but divided into non overlapping parts, and hence some classification is syntactic.

A summarizing example can be the ADT of Stacks of Natural Numbers where the classification of natural numbers and stacks, denoted by two sorts generally called *Nat* and *Stack*, are statically determined, while other classifications like those of positives and non-empty stacks, denoted by two sorts generally called *Pos* and *NeStack*, are dynamically determined.

Of course the second view generalizes the first one, and is certainly more powerful because, as we will see, it can capture some static aspects that the first view cannot.

Trying to show that one version of OSA is more powerful than another is probably misleading, in fact unOSA over a signature Σ can be simulated by ovOSA by defining a consistent and filtered signature specifying essentially the same target. But also the contrary is possible because unOSA can model a single world with non overlapping carriers when all syntactic restrictions developed for ovOSA are applied. This substantial equivalence should be acceptable, once we have shown the pragmatic equivalence of the overloading expressed by these two approaches.

But if we associate the structured view to ovOSA and the universal view to unOSA, we may conclude that the first approach is more powerful because it managed to capture semantic aspects without having lost its syntactic nature, where the second has lost it.

Property 32 Static-type checking of an element of an ovOSA over a coherent signature is possible, where the type represents a connected component[9].

[9] Checking the particular sort of an element is a dynamic problem and can be done in the same way as equalities between elements is proved.

This important result does not hold for universal OSA because of the lack of compatibility with respect to MSA, and probabily justifies the doubts someone has about the real benefits of universal order-sorted logic.

Finally, we want to remark that

Remark 33 The view supported by consistent OS specifications *is* that of a sort-independent semantics.

5.4 Mechanizability and Partiality for OSA

The extension of MSA to OSA determines an extension of many-sorted term rewriting (MSTR) to order-sorted term rewriting (OSTR). This extension is not as straightforward as that from unsorted to many-sorted rewriting; however order-sorted rewriting has been widely and extensive studied, and the result is a rather efficient tool, even though many properties of MSTR can be hardly extended to OSTR. Some restrictions are also needed.

Definition 34 An (S, Σ)-equation $(\forall X)\ l = r$ (which determines the rewrite rule $(\forall X)\ l \to r$) is sort decreasing if for all sorts $s \in S$ and all substitutions $a : X \to T_\Sigma$ we have that $a(l) \in T_{\Sigma,s}(X)$ implies $a(r) \in T_{\Sigma,s}(X)$. The equation is sort-preserving if the implication is an equivalence.

For sort-decreasing specifications, order-sorted rewriting, unification and completion works properly, as shown in [51], among others.

The other well-known problem of terms like $pred\,pred\,succ\,succ\,0$ which are not well-formed but have a semantic meaning ($succ\,succ\,pred\,pred\,0 = 0$) can be solved by considering some dynamic mechanism to check whether an element of a sort is also an element of a subsort. The most important work in this direction is probabily represented by *retracts*, introduced in [20] for overloaded OSA.

Roughly speaking, to an OS specification (Σ, E) we add a retract operation $r_{s,s'} : s \to s'$ and a retract equation $r_{s,s'}(x) = x$ (where x has sort s') for all sorts $s' \leq s$. This forms a new specification $(\Sigma^{\otimes}, E^{\otimes})$ which, under mild assumptions, is a conservative extension of (Σ, E). This means that adding retracts is safe, because the original specification is "preserved" (i.e., no new equality between terms without retracts can be proved).

Example 35 If the specification of Natural Numbers contains the signature:
$pos \leq nat$; $0 :\to nat$; $succ : nat \to pos$; $pred : pos \to nat$; ...., we can simply add the retract $r_{nat,pos} : nat \to pos$ and the axiom $r_{nat,pos}(x) = x$, where x has sort *Pos*.

In this view the denotational semantics of a specification can be defined as the family of the Σ-homomorphic extensions $\psi_X : T_{\Sigma,E}(X) \to T_{\Sigma^{\otimes},E^{\otimes}}(X)$, which are injective because of the ceservative extension property. Therefore, instead of a single initial algebra, we have a collection of free algebras with retracts. Notice that this does not lose the 'no junk and no confusion' characteristic of initial algebra semantics. Furthermore, adding retracts does not compromise parameterization and modularization of overloaded OS specifications [25].

The operational meaning of retracts is rather natural: for terms like $pred\,r\,pred\,succ\,succ\,0$ they disappear, whereas for terms like $pred\,r\,0$ the computation is stopped because of their irreducibility, thus giving an error message.

Finally, we want to remark that the final specification of an OSA considered by the operational semantics involves a considerably greater number of axioms, operations and sorts than initially appears, and that checking the coherence and sort decreasingness of a specification is an NP-problem (so, pragmatically, it is simply assumed, thus increasing the risk of having non-correct specifications and wrong results). Furthermore, for the parametric or modular case, syntactic constraints have to be imposed (and checked) on signature morphisms, otherwise pushouts might not preserve coherence of specifications [26].

All this, together with the fact that adding retract requires an error handling technique, may justifies the exploration of other approaches to order-sortedness, such as MS algebras with embeddings or universal structures.

We have seen that MSA does not directly support subsorts, but this notion is achievable by coercion: in the case of *pred* we introduce the sort *Pos* of positives and relate it to *Nat* by means of an injective operation: $u : Pos \to Nat; \quad n = m \quad if \quad u(n) = u(m)$, so to have a subsort corrispondence betwen the two carriers. At this point one might believe that a dynamic notion of sort can be transformed into a static one, but, unfortunately, the dynamic relation between *Pos* and *Nat* cannot be avoided: in order to allow terms such as $pred(pred(n))$ one needs a partial (!) function from *Nat* to *Pos*. Notice the similarity with retracts, showing their partial nature.

MSA with embeddings and partial functions is the (current) way adopted by the CoFI group [35] to express sort-inclusion in CASL [8]. If this approach radically avoids the complications raised by order-sorted logic, it loses the benefits, for consistent and sort-decreasing specifications, of order-sorted rewriting, which is more efficient (as shown for example in [52]).

6 Universal structures

We have seen that MSA and OSA are based, in principle, on a syntactic notion of sort, even though the use of partial functions or the imposition of semantic constraints allows to deal with dynamic phenomena.

The structures we review here are universal in exactly the same way as unOSA are universal: a single universe is modeled (as a consequence, overloading is weak).

What differs with universal OSA is that here the semantic way is pursued from the beginning: the fact that an element has a particular sort is expressed with axioms.

Equational Type Logic, Type Horn Logic and Unified Algebra are just a few of such universal structures, and for the economy of this paper we only examine UA. This is for at least two reasons:

— for what concerns sorts and classifications, UA, ETL, THL and other universal structures are very similar, and the analysis we make for UA is valid also for the other two structures;

— UA is easy to describe, and allows a better introduction of the structure of HUA (based on UA) which differs from the other universal structures for supporting both static and dynamic classifications.

6.1 Unified Algebra

Unified Algebra [37, 36, 38], originally developed by Mosses for his action semantics [39], is based on an illuminating idea: elements and sorts are treated in a unified way — they are values. It turns out that sorts have an intensional meaning, in fact if elements and sorts are treated in a unified way we have that operations can be applied to elements as well as to sorts. A sort is considered as a non-deterministic choice of elements.

Definition 36 A unified signature is an unsorted signature with two predicates and three fixed operations giving the signature of a lattice with minimum (written in mix-fix notation): $\Omega = \langle (\Omega_n)_{n \in N}; \Pi^0 \rangle$ where Ω contains $\Omega^0 = \{nothing;\ \_|\_;\ \_\&\_\}$, and Π^0 is $\{\_ : \_;\ \_ \leq \_\}$.

A unified signature morphism is an unsorted signature morphism preserving the fixed part. When the context permits we omit the fixed part.

Definition 37 Unified axioms are Horn-clauses with equality.

Definition 38 A unified algebra A over a signature Ω is an Ω-homogeneous algebra over the (Ω-dependent) theory T^0 given in the following table:

$x \leq y; y \leq x \Longrightarrow x = y$ $x \leq x$ $x \leq y; y \leq z \Longrightarrow x \leq z$	$x : x; x \leq y \Longrightarrow x : y$ $x : y \Longrightarrow x \leq y$ $x : y \Longrightarrow x : x$
$x\|y = y\|x$ $x \leq x\|y$ $x \leq z; y \leq z \Longrightarrow x\|y \leq z$	$x \& y = y \& x$ $x \& y \leq x$ $z \leq x; z \leq y \Longrightarrow z \leq x \& y$
$nothing \leq x$ $(x\|y)\&z = (x\&z)\|(y\&z) \qquad (x\&y)\|z = (x\|z)\&(y\|z)$ For all $\sigma \in \Omega_n$, with $n \geq 1$, the n clauses: $x_i \leq x_i' \Longrightarrow \sigma(x_1, ..., x_i, ..., x_n) \leq \sigma(x_1, ..., x_i', ..., x_n)$	

It emerges that an axiom $t_1 \leq t_2$ induces a sort-inclusion interpretation (a typical example would be $Pos \leq Nat$ or $succ(Nat) \leq Nat$), while $t_1 : t_2$ means that t_1 is an element of sort t_2, thus recovering the notions of element and sort.

In practice, all values of a UA may be thought of as sorts, with the elements represented by singleton sorts (*Hobson's choices*).

The high expressivity of UA does not compromise the existence of initial algebras, as it is essentially a Horn clause specification.

Property 39 Unified specifications support initial semantics [37], and unified algebras verify variety and quasi-variety theorems [18].

However, in order to match the requirements in Remark 1 we need to investigate whether UA supports parameterization and modularization, and how it can be implemented. And as one may expect, the purely semantic notion of sort held creates some problems.

The starting problem is the definition of the forgetful functor, because in a homogeneous structure it is not straightforward which sorts it should forget about. The solution given by Mosses is as follows.

Definition 40 Given a signature morphism $\phi : \Omega \to \Omega'$, the (more) forgetful functor $\_ \ddagger \phi$: $\mathrm{UA}(\Omega') : \to \mathrm{UA}(\Omega)$ regards the image of Ω-ground terms (along ϕ) as sorts to retain, considers all elements related (by $\_ : \_$) with these sorts, and yields an algebra generated by only these elements. Formally, given an Ω'-algebra A' it gives the algebra $A' \ddagger \phi = A$ defined by:

— $E_A = \{x \in A' \mid x :_{A'} \phi(t)_{A'}$ for some ground Ω-term $t\}$;

— the carrier $|A|$ is the least set that includes E_A such that when $\sigma \in \Omega_n$ and $x_1, ..., x_n \in |A|$ then also $\phi(\sigma)_{A'}(x_1, ..., x_n) \in |A|$;

— $\sigma_A = \phi(\sigma)_{A'}$ for each $\sigma \in \Omega$.

Furthermore, given a unified Ω'-homomorphism $h' : A' \to B'$, the homomorphism $h' \ddagger \phi$ is just the restriction of h' to $A' \ddagger \phi$ ($A \ddagger \phi$ is a sub-algebra of A).

The motivation for this choice is not very clear (for example it is not very clear why any sort in $A \ddagger \phi$ must be generated by elements) and gives some trouble:

— the forgetful of the identity signature morphism is not the identity functor;

— it does not allow the verification of one side of the satisfaction condition: $A' \ddagger \phi \models ax \not\Rightarrow A' \models \phi(ax)$ with $\phi : \Sigma \to \Sigma'$ (however the other side, saying that the functor sends theories to theories, is satisfied).

This means that on this forgetful functor an institution cannot be defined (not even a specification logic, as defined in [9]). And, what is worse, is that even if unified specifications support initial algebra semantics they do not support free semantics, almost essential to define the semantics of parameterization and modularization. In fact the fascinating theory of modularization developed by Mosses [38], based on initial semantics does not match some semantic requirements like, for example, that the semantics of an imported module be preserved (in order to allow the importation of meaning instead of a mere importation of code) [41, 50].

However, these criticisms must not be extended to all universal structures, in fact Equational Type Logic and Type Horn Logic are examples of liberal institutions, thanks to a great effort made in order to define a more suitable forgetful functor, (but paying a price in terms of clarity and simplicity). But sufficient syntactic conditions which can guarantee the persistency of the free functor — necessary in order to realize importation of meaning instead of mere importation of code — have never been given. And the complexity of the forgetful functors [30, 44] suggests that this is hardly achievable. This is another critical point.

Our main concern about these universal structures is their pure semantic treatment of sorts, which does not allow any relevant static reasoning, and seems incoherent with the evolution from unsorted to many-sorted algebras. This assumes even more importance if we think that universal structures are particularly weak for what concerns mechanizability, because of their many more values expressed and their heavy use of conditional equations, which require a very careful use of sorts.

In conclusion, a pure semantic approach to sorts does not match the requirement concerning mechanizability in Remark 1 (even for structures like ETL where a more appropriate notion of rewriting is considered).

6.2 Heterogeneous Unified Algebra

Heterogeneous Unified Algebra is the last structure analyzed. It corresponds to the compromise between many-sorted structures and universal structures. The view expressed by HU specifications is that of a universe in which some disjoint parts are statically classified, while others are dynamically classified, in order to be able to express static as well as dynamic aspects: exactly the view of overloaded OS specifications, which according to the arguments given so far seems to be the most powerful.

HUA shares the principles underlying ovOSA, but it has been developed starting from an alternative point: while the theory developed for ovOSA starts from a static notion of sort (many-sorted view) and then, by putting some constraints, it also achieves a dynamic notion of sort, the theory developed for HUA starts from a semantic notion of sort (universal view) and then, by putting some constraints, it can also capture a static notion of sort.

Of course, we may expect that the "essentially" syntactic notion of sort held in ovOSA favors mechanizability, while the "essentially" semantic notion of sort held in HUA favors expressivity.

The theory of HUA is based, as the name suggests, on the theory of UA. In brief, HUA comes out by considering a MS signature, and by defining a UA in which the (major) sorts denoted in the signature are maximal (this is expressible by axioms like $x \leq s$) and do not share values (this is expressible by axioms like $s_1 \& s_2 = nothing$). This allows to define a forgetful functor in a more natural way.

The naturalness of this idea is confirmed by important results like the possibility to define a liberal institution, and, mainly, the compatibility with MSA [41], whose importance has already been discussed.

Definition 41 HU signatures and signature morphisms are MS signatures and signature morphisms respectively.

A HUA can be described as follows:

— the algebra is homogeneous, (genuinely) total and partially ordered. Some values represent elements, others, sorts — interpreted as non-deterministic choices of elements. The partial order is interpreted as the sub-sort relation. Operations are applied both on elements and on sorts, thus forming many classifications;

— it has a minimum (*nothing*), representing a vacuous sort, useful to express partial operations; and a maximum (*everything*) representing the entire universe;

— the partial order is a lattice, where the operations sup ($\_|\_$) and inf ($\_\&\_$) represent the union and the intersection of sorts;

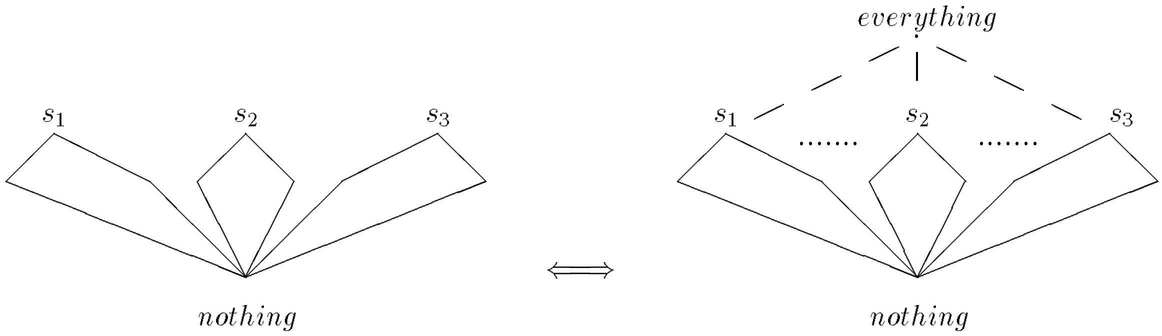
— it has a heterogeneous flavour, in view of particular values - major sorts - denoted in the signature, whose intersection is the minimum;

— the functionality of operations in the signature is interpreted as follows: if an operation is applied on values out of its domain, then the result is the minimum value;

— non-strict operations are allowed. For example we can define the operation $if\_ then\_ else\_$ which applied to a term like $(true, 1, nothing)$ returns 1;

— the classification of an element is possible thanks to a binary predicate ($\_ : \_$). For example a typical classification is $0 : Nat$;

— from the structure it is possible to single out a core made up of "sections" (as in the figure), sharing only the minimum, where the maximal values are the major sorts. Each section may be seen as the carrier of the sort. There is an isomorphism between the core and the whole algebra: the values in the remaining part are the sup of values of the core part, with "no junk and no confusion". Thanks to this isomorphism, we can reason about the core instead of the whole algebra without loosing generality.



<table>
<tr><th colspan="2">$\Sigma = (S, OP) \quad \longrightarrow \quad \Omega_\Sigma$</th></tr>
<tr><td>$x \le y; y \le x \Longrightarrow x = y$
$x \le x$
$x \le y; y \le z \Longrightarrow x \le z$</td><td>$x : x; x \le y \Longrightarrow x : y$
$x : y \Longrightarrow x \le y$
$x : y \Longrightarrow x : x$</td></tr>
<tr><td>$x|y = y|x$
$x \le x|y$
$x \le z; y \le z \Longrightarrow x|y \le z$</td><td>$x\&y = y\&x$
$x\&y \le x$
$z \le x; z \le y \Longrightarrow z \le x\&y$</td></tr>
<tr><td colspan="2">$nothing \le x, \qquad x \le everything, \qquad everything = s_1|s_2|...|s_n$
$(x|y)\&z = (x\&z)|(y\&z) \qquad (x\&y)|z = (x|z)\&(y|z)$
For all $\sigma \in \Omega_n$, $n \ge 1$, the n axioms:
$x_i \le x'_i \Longrightarrow \sigma(x_1, ..., x_i, ..., x_n) \le \sigma(x_1, ..., x'_i, ..., x_n)$</td></tr>
<tr><td colspan="2">For all $s_1, s_2 \in S$, $s_1 \neq s_2$, the axiom: $s_1\&s_2 = nothing$
For all $\sigma \in OP_{s_1...s_n,s}$, the axioms: $\sigma(s_1, ..., s_n) \le s$
For all $\sigma \in OP_{s_1...s_n,s}$, the axioms: $\sigma(x_1, ..., x_n) = \sigma(x_1\&s_1, ..., x_n\&s_n)$</td></tr>
<tr><td colspan="2">other axioms expressing the whole algebra as directely determined by its core[10]</td></tr>
</table>

Table 1: T^0 is the closure of the axioms in the table.

Formally, this is realized by transforming a many-sorted signature into an unsorted one in the way described by the following example, and considering a fixed theory over this unsorted signature.

Example 42 The many-sorted signature of lists of natural numbers $\Sigma = (\{Nat, List\};\ 0 \in \Sigma_{[],Nat};\ succ, pred \in \Sigma_{Nat,Nat};\ empty \in \Sigma_{[],List};\ head \in \Sigma_{List,Nat};\ tail \in \Sigma_{List,List})$ is transformed into the signature $\Omega_\Sigma = (0, empty, Nat, List \in \Omega_0;\ succ, pred, head, tail \in \Omega_1)$, with Ω having the fixed part $\langle \Omega^0 = \{nothing, everything, \_|\_, \_\&\_\}; \Pi^0 = \{\_ : \_, \_ \le \_\}\rangle$.

Definition 43 Given a MS signature Σ, the category HUA(Σ) is the category of homogeneous algebras $Alg(\Omega_\Sigma, \mathrm{T}^0)$, where T^0 is in Table 1.

Property 44 HUA allows static-type checking.

In fact the type of a term (its major sort) can be established at the signature level, by considering the functionality of operations.

The forgetful functor is defined as in the many-sorted case: it forgets entire carriers of sorts not in the image of a signature morphism. This allows a perfect compatibility with the many-sorted institution.

HUA is a generalization of UA, which is expressed when a signature consists of only one sort[11]. Of course, as it happened between ovOSA and unOSA, a UA can be seen as a HUA when the same constraints used for HUA are adopted. However, as for the OS case, we are mainly interested in

[10]These axioms are pure first-order axioms, but this does not compromise the existence of initial algebras or the possibility of having rewriting implementations, because of an isomorphism between HU specifications and MS specifications [41, 50].

[11]Actually this is only partially true because UA do not foresee a top sort.

the view expressed, and the view expressed by HUA is more powerful than that expressed by UA, and, in general, by other universal structures.

Finally, notice that HUA, as defined here (and in [41]) do not support overloading. Intuitively, overloading can be expressed ([50]) for regular many-sorted signatures by substituting last axiom in Table 1 with:

$$\sigma(x_1, ..., x_n) = \sigma(x_1\&s_1, ..., x_n\&s_n)|\sigma(x_1\&s'_1, ..., x_n\&s'_n)$$

for all $\sigma \in OP_{s_1...s_n,s} \cap OP_{s'_1...s'_n,s'}$.

We hope to develop an appropriate notion of rewriting for heterogeneous unified specifications in the near future, following, probabily, [29] and [30].

7 Again on overloading

We have seen that overloading is strong when an overloaded function is interpreted as two functions which disagree on some common element, thus provoking an ambiguity. If this ambiguity is 'by design', i.e., in the term algebra, a mechanical implementation cannot resolve it (if it could, there would be no ambiguity, and overloading would be weak). This forces us to manually disambiguate terms (otherwise, the implementation would identify the different results of the overloaded operation), using for example a dot qualification which *de facto* eliminates overloading.

In [18] Goguen and Diaconescu maintain that strong overloading is needed to formalize and mechanize the 'dynamic binding' arising in the object paradigm when methods are redefined (non-strict inheritance).

Even though we have shown that algebraic specification frameworks do not support strong overloading, we believe the use of these structures in the specification of object-oriented systems is not comprimised. This is for at least three reasons:

- redefinition of methods is not recommended in programs (because correctness of development is no more guaranteed), and can/should be avoided [34];
- inheritance does not necessarily have to be expressed by subsorts;
- in modeling non-strict inheritance by subsorts we can adopt a dot qualification and let the system do the binding dynamically, as advocated in [45].

8 Call-by-need

Call-by-need is an operational technique to improve the efficiency of the evaluation of expressions [5].

Example 45 Let us suppose that an algebraic specification (whatever kind) include on operation *first* : $ss \to s$ which simply chooses its first argument. The expression *first*(t_1, t_2) may have many possible reduction paths, all getting to its normal form t_1 (assuming that every term has one). All these paths are semantically equivalent, but it appears that reducing the term t_2 is useless: an efficient evaluation strategy only reduces t_1.

Implementations which resolve all subterms of an expression are called *call-by-value*. Implementations which resolve only needed subterms are called *call-by-need*. These neatly increase the efficiency of resolution, and every operational semantic should involve it. However, they are difficult to handle as shown below.

Call-by-need resolution is strongly related to errors and strictness: in the presence of errors (or partial functions) we have to know whether non-strictness is allowed. For example $first(t, error)$ can be resolved in a call-by-need way only if the operation *first* is non-strict. Otherwise, in order to resolve a generic term $first(t_1, t_2)$ we have to check that t_2 is not an error. Strictness is incompatible with call-by-need techniques (to the extent that in a strict structure any subterm needs to be evaluated).

The presence of subtypes may further limit the use of call-by-need strategies, because the well-definedness of terms, in general, cannot be checked statically. Therefore, the evaluation of all subterms might be required.

In conclusion, static classifications may support, in principle, call-by-need techniques, but the difficulty in handling errors and (mainly) non-strictness poses a problem.

Dynamic classifications are not well suited to support call-by-need techniques, because well definedness of terms, in general, cannot be checked statically. However, UA and HUA represent an exception. In fact they are technically unsorted: operations are technically defined in all the universe, where partiality is recovered thanks to the bottom, representing undefinedness. Because non-strictness is allowed in these structures, it turns out that call-by-need techniques can in principle be defined for them [50].

9 Nondeterminism

Nondeterminism is an essential aspect of systems specification. Even though every real system is deterministic, we often reach a level of abstraction where part of a state is hidden, so to virtually specify nondeterministic systems.

Typically, non-determinism in algebraic specifications is captured by underspecification. Here, a nondeterministic operator, i.e., an operator which applied to a value may give rice to more that one value, is represented by a number of different deterministic implementations. For example, an operation f which applied to a constant a may return any natural number less than 2 will be implemented in two different (deterministic) ways: some models will have $f(a) = 0$, others $f(a) = 1$.

In an algebraic setting, this can be achieved by considering the loose semantics, thus paying the price of renouncing at initiality. However, this might not be a problem: among others, recent works on the *hidden paradigm* [19, 21, 48, 49] show that initial semantics is very useful to specify data, but is not required for the specification of states[12].

It emerges that underspecification is compatible both with static and dynamic classifications.

An alternative approach, based on dynamic classifications, is due to Mosses with his unified treatment of elements and sorts. Here, nondeterminism is computational: instead of admitting a choice among different (deterministic) models, a choice within one model is admitted. Let us see an example of natural numbers with choices:

Example 46 Let NAT-CH be the following heterogeneous unified specification whose only sort is Nat:

$$0, Pos :\to Nat;$$
$$succ\_, pred\_, [0..\_] : Nat \to Nat;$$
$$add : Nat, Nat \to Nat; \qquad \text{with axioms } (n, m, N, M, P \text{ variables}):$$

[12]Data are immutable and visible (2, 5, *Monday*, *blue*,...). States are mutable and hidden. They can be observed through the application of methods, changing the state, and attributes, giving information of the state in terms of data. Semantics for data is equational (initial), semantics for states is behavioural (loose or final).

$$
\begin{array}{l}
0 : Nat; \quad Pos = succ\, Nat; \\
n : Nat \Rightarrow succ\, n : Nat; \\
pred\, 0 = nothing; \quad pred\, N = pred(N \& Pos); \\
pred\, succ\, N = N; \quad N \leq Pos \Rightarrow succ\, pred\, N = N; \\
succ(N|M) = (succ\, N)|(succ\, M); \quad pred(N|M) = (pred\, N)|(pred\, M); \\
pred\, nothing = nothing; \quad succ\, nothing = nothing; \\
[0..N] = [0..pred\, N]|N \\
[0..(M|N)] = [0..M]|[0..N]; \\
[0..nothing] = nothing; \\
add(0, N) = N; \\
add(N, M) = add(M, N); \\
N \leq Pos \Rightarrow add(N, M) = add(pred\, N, succ\, M); \\
n : Nat \Rightarrow add(Nat, add(n, Nat)) = add(n, Nat); \\
add(N|M, P) = add(N, P)|add(M, P); \\
add(nothing, N) = nothing; \\
n : Nat \Rightarrow [0..n] \& add(succ\, n, Nat) = nothing; \\
n : Nat \Rightarrow n|add(succ\, n, Nat) = add(n, Nat); \\
n : n; \; m : m; \; n {=}/{=} m \Rightarrow n \& m = nothing
\end{array}
$$

Of course we could easily add constants like *Even* and *Odd*, and operations like *double*, with the obvious meaning.

In the initial term-algebra *nothing* and *Nat* are respectively the bottom and the top of the lattice. Elements are 0, $succ\, 0$, ..., and are "just above" the bottom. All other values are choices of elements and represent the sets of elements below them (two values are equal iff the set of elements below them is the same), with | and & being, respectively, the union and the intersection operations. All finite choices are in the initial models, where those of consecutive elements are intervals. Infinite choices, generable by *Nat*, *succ* and | are represented: $succ^n Nat$, $0|succ^2 Nat$, etc. Notice that in this example the infinite choices are all the co-finite sets of natural numbers.

Remark 47

— An equivalent specification of NAT-CH can be given by using unified specifications, in fact the difference between UA and HUA mainly arises when more than one (major) sort is considered;

— the built-in boolean valued operation =/= has meaning only in the initial term algebra of Church-Rosser and Terminating TRS (see [33, 25]). However, last axiom could also be expressed by $n : n; \; m : Pos \Rightarrow n \& add(n, m) = nothing$;

— the initial algebras of NAT-CH are c.p.o.'s. In particular, they are algebraic, that is, they are domains. They can also be regarded as boolean algebras, by introducing a complement operation, in fact every value has a complement. For example, the complement of an element n is $[0..pred\, n]|add(succ\, n, Nat)$. See the forthcoming paper [50] for a reading on nondeterminism and continuity in HUA.

10 Conclusions

In the course of this paper we have looked at the view expressed by algebraic approaches to ADTs. We have distinguished between three possible views to which correspond three different interpretations of sorts. The many-sorted view expresses a number of (overlapping) worlds, and holds a static (or syntactic) notion of sort; the universal view expresses a single universe, and holds a dynamic (semantic) notion of sort; while the more general view of a single world divided into non-overlapping parts has elements of both.

We have seen that in a specification setting, where we need both expressivity and mechanizability, the two extreme views are hardly acceptable because the pure many-sorted view does not match the expressivity requirement, while the universal view does not match the requirement concerning mechanizability. But, the difference between these views is not influenced by overloading, because both can express only weak overloading. This should not be considered as a deficiency, because the effects of strong overloading can be obtained by weak overloading plus dynamic binding, as discussed in Section 7.

The third view has the potential to match all requirements, but, according to the particular technical approach chosen, it can privilege mechanizability — like overloaded Order-Sorted Algebra — or expressivity — like Heterogeneous Unified Algebra. This view is also supported by two new frameworks: CASL [8] and Membership Algebra [32].

Relations between static or dynamic classifications and call-by need techniques have been investigated.

The paper ends by showing the nondeterminism expressed by algebraic specifications when static classifications (underspecification) or dynamic classifications (computational nondeterminism) are supported.

References

[1] Egidio Astesiano and Maura Cerioli. Non-strict don't care algebras and specifications. *Mathematical Structures in Computer Science*, 6(1), 1996.

[2] Jean Benabou. Structures algébriques dans les catégories. *Cahiers de Topologie et Géometrie Différentiel*, 10:1–126, 1968.

[3] Gilles Bernot, Michel Bidoit, and Christine Choppy. Abstract data types with exception handling: an initial approach based on a distiction between exceptions and errors. *Theoretical Computer Science*, 46:13–46, 1986.

[4] Michel Bidoit. Algebraic specification of exception handling and error recovery by means of declarations and equations. In *Proceedings, 14th International Conference on Automata, Languages and Programming*, pages 95–108. Springer, 1984. Lecture Notes in Computer Science, Volume 172.

[5] Richard Bird and Philip Wadler. *Introducing Functional Programming.* Prentice-Hall International, 1988.

[6] Garrett Birkhoff. On the structure of abstract algebras. *Proceedings of the Cambridge Philosophical Society*, 31:433–454, 1935.

[7] Manfred Broy and Martin Wirsing. Partial abstract types. *Acta Informatics*, 18:47–64, 1982.

[8] CoFI. CASL: The CoFI algebraic specification language, tentative design: Language Summary. Technical Report Notes Series NS-96-15, BRICS, Department of Computer Science, University of Aarhus, 1996.
URL: `http://www.brics.dk/Projects/CoFI/Documents/Tentative/LanguageSummary/`.

[9] Harmut Ehrig, Michael Baldamus, and Fernando Orejas. New concepts for amalgamation and extension in the framework of specification logics. Technical Report 91/05, Technical University Berlin, 1991.

[10] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. Springer, 1985. EATCS Monographs on Theoretical Computer Science, Volume 6.

[11] Martin Gogolla. Partially ordered sorts in algebraic specifications. In Bruno Courcelle, editor, *Proceedings, 9th CAAP (Bordeaux)*, pages 139–153. Cambridge, 1984. Also Forschungsbericht Nr. 169, Universität Dortmund, Abteilung Informatik, 1983.

[12] Martin Gogolla, K. Drosten, Udo Lipeck, and Hans-Dieter Ehrich. Algebraic and operational semantics of specifications allowing exceptions and errors. *Theoretical Computer Science*, 34:289–313, 1984.

[13] Joseph Goguen. Semantics of computation. In Ernest G. Manes, editor, *Proceedings, First International Symposium on Category Theory Applied to Computation and Control*, pages 234–249. University of Massachusetts at Amherst, 1974. Also in Lecture Notes in Computer Science, Volume 25, Springer, 1975, pages 151–163.

[14] Joseph Goguen. Abstract errors for abstract data types. In *Proceedings, 1977 IFIP Working Conference on the Formal Description of Programming Concepts*. North-Holland, 1978.

[15] Joseph Goguen. Order sorted algebra. Technical Report 14, UCLA Computer Science Department, 1978. Semantics and Theory of Computation Series.

[16] Joseph Goguen. *Theorem Proving and Algebra*. MIT, to appear 1997.

[17] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, January 1992. Draft appears as Report ECS-LFCS-90-106, Computer Science Department, University of Edinburgh, January 1990; an early ancestor is "Introducing Institutions," in *Proceedings, Logics of Programming Workshop*, Edward Clarke and Dexter Kozen, Eds., Springer Lecture Notes in Computer Science, Volume 164, pages 221–256, 1984.

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

[19] Joseph Goguen and Răzvan Diaconescu. Towards an algebraic semantics for the object paradigm. In Hartmut Ehrig and Fernando Orejas, editors, *Proceedings, Recent Trends in Data Type Specification*. Springer, 1994. Lecture Notes in Computer Science, Volume 785.

[20] Joseph Goguen, Jean-Pierre Jouannaud, and José Meseguer. Operational semantics of order-sorted algebra. In Wilfried Brauer, editor, *Proceedings, 1985 International Conference on Automata, Languages and Programming*. Springer, 1985. Lecture Notes in Computer Science, Volume 194.

[21] Joseph Goguen and Grant Malcolm. Proof of correctness of object representations. In A. W. Roscoe, editor, *A Classical Mind: essays dedicated to C.A.R. Hoare*, chapter 8, pages 119–142. Prentice-Hall International, 1994.

[22] Joseph Goguen and José Meseguer. Completeness of many-sorted equational logic. *Houston Journal of Mathematics*, 11(3):307–334, 1985. Preliminary versions have appeared in: *SIGPLAN Notices*, July 1981, Volume 16, Number 7, pages 24–37; SRI Computer Science Lab, Report CSL-135, May 1982; and Report CSLI-84-15, Center for the Study of Language and Information, Stanford University, September 1984.

[23] Joseph Goguen and José Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Computer Science*, 105(2):217–273, 1992. Also Programming Research Group Technical Monograph PRG–80, Oxford University, December 1989, and Technical Report SRI-CSL-89-10, SRI International, Computer Science Lab, July 1989; originally given as lecture at *Seminar on Types*, Carnegie-Mellon University, June 1983; many draft versions exist, from as early as 1985.

[24] Joseph Goguen, James Thatcher, and Eric Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. Technical Report RC 6487, IBM T.J. Watson Research Center, October 1976. In *Current Trends in Programming Methodology, IV*, Raymond Yeh, editor, Prentice-Hall, 1978, pages 80–149.

[25] Joseph Goguen, Timothy Winkler, José Meseguer, Kokichi Futatsugi, and Jean-Pierre Jouannaud. Introducing OBJ. In Joseph Goguen and Grant Malcolm, editors, *Software Engineering with OBJ: Algebraic Specification in Practice*. Cambridge University Press, 1996. Also available as a technical report from SRI International.

[26] Anne E. Haxthausen and Friederike Nickl. Pushouts of order-sorted algebraic specifications. In *Proceedings, Algebraic Methodology and Software Technology, 5th International Conference, AMAST'96*. Springer, 1996. Lecture Notes in Computer Science, Volume 1101.

[27] Phillip J. Higgins. Algebras with a scheme of operators. *Mathematische Nachrichten*, 27:115–132, 1963.

[28] Tomas Isakowitz. *Theorem Proving Methods for Order Sorted Logic*. PhD thesis, University of Pennsylvania, 1989. Appeared in *Logic in Computer Science '89*, pages 304–312.

[29] Stéphane Kaplan. Rewriting with a nondeterministic choice operator. *Theoretical Computer Science*, 56:37–57, 1988.

[30] Vincenzo Manca, Antonio Salibra, and Giuseppe Scollo. Equational type logic. *Theoretical Computer Science*, 77:131—159, 1990.

[31] José Meseguer. General logics. In H.D. Ebbinghaus et al., editors, *Proceedings, Logic Colloquium'87*, pages 275–329. North-Holland, 1989.

[32] José Meseguer. Membership algebra. 1996. Unpublished manuscript.

[33] José Meseguer and Joseph Goguen. Initiality, induction and computability. In Maurice Nivat and John Reynolds, editors, *Algebraic Methods in Semantics*, pages 459–541. Cambridge, 1985.

[34] Bertrand Meyer. *Object-Oriented Software Construction*. Prentice-Hall International, 1988.

[35] Peter D. Mosses. CoFI: The Common Framework Initiative for algebraic specifications. In *Bullettin of the EATCS, June 1996*. URL: `http://www.brics.dk/Projects/CoFI/`.

[36] Peter D. Mosses. Unified algebras and action semantics. In *Proceedings, Symposium on Theoretical Aspects of Computer Science.* Springer, 1989. Lecture Notes in Computer Science, Volume 349.

[37] Peter D. Mosses. Unified algebras and institutions. In *Proceedings, Fourth Annual Conference on Logic in Computer Science*, pages 304–312. IEEE, 1989.

[38] Peter D. Mosses. Unified algebras and modules. In *Proceedings 16th Annual ACM Symposium on Principles of Programming Languages*, pages 329–343. ACM, 1989.

[39] Peter D. Mosses. *Action Semantics.* Cambridge, 1992. Tracts in Theoretical Computer Science, number 26.

[40] Peter D. Mosses. The use of sorts in algebraic specifications. In *Proceedings, 8th Workshop on Abstract Data Types*, pages 66–92. Springer, 1993. Lecture Notes in Computer Science, Volume 665.

[41] Francesco Parisi-Presicce and Simone Veglioni. Heterogeneous unified algebras. In *Proceedings, Mathematical Foundations of Computer Science '93.* Springer, 1993. Lecture Notes in Computer Science, Volume 711.

[42] Axel Poigné. Another look at parametrization using algebraic specifications with subsorts. In *Proceedings, Mathematical Foundations of Computer Science '84.* Springer, 1984. Lecture Notes in Computer Science, Volume 176.

[43] Axel Poigné. Once more on order-sorted algebra. In *Proceedings, Mathematical Foundations of Computer Science '91*, pages 397–405. Springer, 1990. Lecture Notes in Computer Science, Volume 520.

[44] Axel Poigné. Typed horn logic. In *Proceedings, Mathematical Foundations of Computer Science '90*, pages 470–477. Springer, 1990. Lecture Notes in Computer Science, Volume 452.

[45] Lucia Rapanotti and Adolfo Socorro. Introducing FOOPS. Technical Report TR-28-92, Programming Research Group, Oxford University, 1992.

[46] Horst Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras.* Oxford, 1987. The International series of monographs on computer science, Volume 2.

[47] Gert Smolka, Werner Nutt, Joseph Goguen, and José Meseguer. Order-sorted equational computation. In Maurice Nivat and Hassan Aït-Kaci, editors, *Resolution of Equations in Algebraic Structures, Volume 2: Rewriting Techniques*, pages 299–367. Academic, 1989. Preliminary version in *Proceedings*, Colloquium on the Resolution of Equations in Algebraic Structures, held in Lakeway, Texas, May 1987, and SEKI Report SR-87-14, Universität Kaiserslautern, December 1987.

[48] Simone Veglioni. *Algebraic Semantics for the Development of Distributed Systems.* PhD thesis, Programming Research Group, Oxford University. To appear 1997.

[49] Simone Veglioni. Objects as Abstract Machines. In *Proceedings, Formal Methods for Open, Object-based, Distributed Systems, 2nd International Conference, FMOODS'97.* Chapman & Hall, 1997.

[50] Simone Veglioni and Francesco Parisi-Presicce. Heterogeneous unified algebras: static and dynamic classifications, institutions, non-strictness and continuity. To appear 1997.

[51] Uwe Waldmann. Semantics of order-sorted specifications. *Theoretical Computer Science*, 94(1):1–35, 1992.

[52] Christoph Walther. Many-sorted inferences in automated theorem proving. In *Sorts and Types in Artificial Intelligence*, pages 18–48. Springer, 1990. Lecture Notes in Artificial Intelligence, Volume 418.

[53] Toru Yamanouchi, Akiyoshi Sato, Minoru Tomobe, Hiroshi Takeuchi, Jun Takamura, and Masanobu Watanabe. Software Synthesis Shell SOFTEX/S. In *Proceedings of the 7th Knowledge-Based Software Engineering Conference (KBSE-92)*, pages 28–37, 1992.