Order-Sorted Algebra I:

Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations[1]

Joseph A. Goguen
Programming Research Group, University of Oxford
SRI International, Menlo Park CA 94025

José Meseguer
SRI International, Menlo Park CA 94025
Center for the Study of Language and Information
Stanford University, Stanford CA 94305

Abstract

This paper generalizes many-sorted algebra (hereafter, MSA) to order-sorted algebra (hereafter, OSA) by allowing a partial ordering relation on the set of sorts. This supports abstract data types with multiple inheritance (in roughly the sense of object-oriented programming), several forms of polymorphism and overloading, partial operations (as total on equationally defined subsorts), exception handling, and an operational semantics based on term rewriting. We give the basic algebraic constructions for OSA, including quotient, image, product and term algebra, and we prove their basic properties, including Quotient, Homomorphism, and Initiality Theorems. The paper's major mathematical results include a notion of OSA deduction, a Completeness Theorem for it, and an OSA Birkhoff Variety Theorem. We also develop conditional OSA, including Initiality, Completeness, and McKinsey-Malcev Quasivariety Theorems, and we reduce OSA to (conditional) MSA, which allows lifting many known MSA results to OSA. Retracts, which intuitively are left inverses to subsort inclusions, provide relatively inexpensive run-time error handling. We show that it is safe to add retracts to any OSA signature, in the sense that it gives rise to a conservative extension. A final section compares and contrasts many different approaches to OSA. This paper also includes several examples demonstrating the flexibility and applicability of OSA, including some standard benchmarks like `STACK` and `LIST`, as well as a much more substantial example, the number hierarchy from the naturals up to the quaternions.

[1]Supported in part by Office of Naval Research Contracts N00014-82-C-0333, N00014-85-C-0417, and N00014-86-C-0450, National Science Foundation Grant MCS8201380, and a gift from the System Development Foundation.

1 Introduction

The essence of order-sorted algebra (hereafter, OSA) is a partial ordering $\leq$ on a set S of sorts; this **subsort** relation imposes the restriction on an S-sorted algebra A that if $s \leq s'$ in S then $A_s \subseteq A_{s'}$ where A_s denotes the set of elements of sort s in A. A major motivation is to correctly handle erroneous and meaningless expressions, such as the `top` of an empty stack or division by zero. This has been an important problem from the earliest days of the algebraic approach to abstract data types [35]. Error algebra was a first try at a more elegant solution [17], but unfortunately error algebra specifications do not always have initial algebras [65]. OSA, which originated in [18], provides what now seems a fully satisfactory and very flexible approach that provides:

1. Several forms of polymorphism and overloading;
2. Error definition, detection and recovery;
3. Multiple inheritance;
4. Selectors when there are multiple constructors;
5. Retracts, which (intuitively) are left inverses to subsort inclusions;
6. Partial operations made total on equationally defined subsorts;
7. An operational semantics that executes equations as (left-to-right) rewrite rules; and
8. A rigorous model-theoretic semantics for all these features.

The research reported here supports OBJ, a programming language with mathematical semantics given by order-sorted algebra, and operational semantics given by order-sorted term rewriting [19, 14, 15, 23]. Our experience with OBJ shows that subsorts are enormously helpful in practice, since they can greatly improve both expressivity and readability.

1.1 Type Disciplines

A type discipline for a programming language has two main benefits:

1. it facilitates conceptual clarity by making explicit the restrictions on the arguments and results of operations, and
2. it allows simple checks at program entry time that can catch many errors before compilation or execution are attempted.

The most obvious type discipline is **strong typing**, where each operation has a fixed sequence of argument types and a fixed result type. Many-sorted algebra (hereafter, MSA) formalizes this for first-order operations, by interpreting strongly typed syntax in many-sorted algebra. However, traditional strong typing is both too rigid and too inexpressive. Order-sorted algebra overcomes both limitations by combining two key ideas: inheritance and subsort polymorphism.

1.1.1 Inheritance and Polymorphism

Inheritance as a programming language feature developed from the Simula language [12], and intuitively corresponds to inclusion of concepts, as found in natural language. For example, we say that every hound is a dog and that every dog is a mammal, because our concept of mammal includes that of dog which in turn includes that of hound. If we

associate an *extension* to each concept, the set of objects that fall under it (e.g., the set of all hounds, or the set of all rational numbers), then inclusion of concepts appears as set-theoretic inclusion of the corresponding extensions. The obvious way to formalize this kind of inclusion is by a *partial ordering*, that is, a reflexive, transitive, antisymmetric relation. For example, the *names* `Natural`, `Integer`, and `Rational` satisfy the relation

```
Natural < Integer < Rational
```

and their extensions, denoted **N**, **Z**, and **Q** respectively, satisfy the corresponding subset inclusions, $\mathbf{N} \subseteq \mathbf{Z} \subseteq \mathbf{Q}$. In order-sorted algebra, names such as `Natural` and `Rational` are called **sorts**, and belong to the syntax, while the extensions **N**, **Z**, **Q** belong to an *interpretation* of the syntax, that is, to an (order-sorted) algebra. The syntax is called a **signature** and consists of a family of sorts, ordered by a partial order relation of inheritance, plus a family of operation symbols with appropriate type information as discussed below.

A very attractive feature of standard mathematical notation is that it allows using one symbol for several different but related operations, so that in applying this symbol we may not even realize that we are moving about within the type hierarchy in a quite free way. This is nicely illustrated by the number hierarchy. We can add 2 + 2 (two naturals), or -7 + 2/3 (an integer and a rational), or 1/5 + 7/9 (two rationals), or 2 + 3/29 (a natural and a rational). This flexibility comes from combining the "overloading" of the + symbol for addition with inheritance among naturals, integers and rationals, in such a way that no matter which addition is used, we get the same result from the same arguments, whenever they make sense. We summarize this situation by saying that + is **subsort polymorphic**. As discussed in the next subsection, this is only one of several different ways that the word "polymorphic" is used.

1.1.2 Polymorphism is Polymorphic

The term "polymorphism" was introduced by Christopher Strachey to express the use of a single operation symbol with different meanings in a programming language. He distinguished two main forms of polymorphism, which he called *ad hoc* and parametric. In his own words [75]:

> In *ad hoc* polymorphism there is no simple systematic way of determining the type of the result from the type of the arguments. There may be several rules of limited extent which reduce the number of cases, but these are themselves *ad hoc* both in scope and content. All the ordinary arithmetic operations and functions come into this category. It seems, moreover, that the automatic insertion of transfer functions by the compiling system is limited to this class.
>
> Parametric polymorphism is more regular and may be illustrated by an example. Suppose f is a function whose argument is of type α and whose result is of type β (so that the type of f might be written $\alpha \to \beta$), and that L is a list whose elements are all of type α (so that the type of L is α**list**). We can imagine a function, say Map, which applies f in turn to each member of L and makes a list of the results. Thus Map[f,L] will produce a β**list**. We would like Map to work on all types of list provided f was a suitable function, so that Map would have to be polymorphic. However its polymorphism is of a particularly simple parametric type which could be written $(\alpha \to \beta, \alpha\textbf{list}) \to \beta\textbf{list}$, where α and β stand for any types.

Strachey's distinction is based on the kind of semantic relationship that holds between the different meanings of an operation symbol, and it suggests a spectrum of possible styles for

the multiple use of an operation symbol, in which the more "regular" the relationship is, the easier it is to do type inference, and the closer it is to parametric polymorphism:

- **ad hoc** in its strongest sense indicates semantically unrelated uses, such as + for both integer addition and Boolean disjunction. (Even in such an extreme case, there is still the tenuous connection that both instances of + are associative, commutative, and have an identity element.)
- **multiple representation** when the uses are related semantically, but their representations may be different, as with Strachey's arithmetic system.
- **subsort polymorphism** where the different instances of an operation symbol are related by inheritance (interpreted as subset inclusion) such that the result does not depend on the instance used, as with + for natural, integer, and rational numbers.
- **parametric polymorphism**, as in Strachey's Map function; this is implemented in higher order functional programming languages such as Hope [5], ML [40] and Miranda [77].

OSA distinguishes and supports all four styles of polymorphism. *Ad hoc* polymorphism is supported by signatures in which the same symbol is used for sorts that are unrelated in the inheritance hierarchy; subsort polymorphism is inherent in the nature of OSA, as already explained. The implementation of arithmetic described by Strachey involves "transfer functions" (which might now be called "coercions") to change the representation of numbers. But coercions are not needed for subsort polymorphic operations, since inheritance appears as *subset inclusion* of the data elements; also, for regular signatures (Definition 2.3 below), any expression involving subsort polymorphism has a smallest sort. OSA also nicely accomodates coercions and multiple representation polymorphism, as discussed in [29] and briefly reviewed in Section 1.5 below, while parametric polymorphism is provided by parameterized ordered-sorted algebras such as `LIST[X]` that provide higher-order capabilities in a first-order setting [21]. These are called **parameterized objects** in the OBJ language [14, 15, 19], and their semantics will be treated in Part III of this paper.

1.2 Logical and Operational Semantics

The original vision of "logic programming" called for using pure first order predicate calculus directly as a programming language [49]. As has been well argued by Prolog advocates (e.g., [73]), this confers some important benefits, including: program simplicity and clarity (which can greatly ease program understanding, reusability, debugging and maintenance); separation of logic and control; and identity of program logic with proof logic. In such a language, a high level description of what a program does is actually a program, and can be executed. Prolog [10, 9] only partially realizes this vision, since it has many features with no corresponding feature in logic (e.g., cut, `is` and `assert`), and also lacks some important features of logic (e.g., semantic equality and true negation).

We believe that the many advantages claimed for logic programming are all compromised to the extent that it fails to realize a pure logic. Consequently, a major goal of our research has been to create powerful programming languages that are based upon pure logics and yet still support truly practical programming. An important advantage of logic-based languages is that they are more convenient for parallel machines, since the compiler and operating system can exploit whatever concurrency is actually available in the program and the particular target machine, because programs are not tied down to particular control strategies (sequential control in traditional imperative languages, and tasking, *rendezvous*,

etc. in imperative languages providing explicit concurrency). To this end, we have taken the broad view[2] that a **logical programming language** $\mathcal{L}$ consists of:

- a well-understood[3] logical system $\mathcal{I}$ together with two subclasses of sentence called **statements** and **queries**,

such that

- an $\mathcal{L}$ **program** $\mathcal{P}$ is a finite set of statements,
- every program has an **initial model**[4], which gives its denotational semantics,
- operational semantics is a (reasonably efficient) form of *deduction* in $\mathcal{I}$, and
- a query is satisfied in an initial model of $\mathcal{P}$ if and only if it can be proven from $\mathcal{P}$ (this is a form of completeness).

We can now define an **answer** to a query to be some property of a proof of the query; for example, we might extract a value for each variable that occurs in the query. This definition of logical programming explicates the perhaps more familiar notion of *declarative programming*, in which programs tell what properties the result should have, rather than how to calculate it. We claim that programs in logical programming languages are easier to read, understand, write, debug, reuse, modify, maintain, and verify; we also claim that it is easier to build environments to support such languages; in particular, it is easier to build debuggers (see [63] for a discussion of some serious difficulties that arise in trying to implement a debugger that can handle Prolog's cut). Logical programming in this general sense includes:

- **functional programming**, where the logic is some kind of equational logic, i.e., a logic of the substitution of equals for equals; for example, OBJ is based on first order order-sorted equational logic, and the usual higher order functional programming languages can be seen as based upon higher order equational logic.
- **relational** (i.e., predicate, or Horn clause, or "logic") **programming**, where the logic is first order Horn clause logic (without equality), as in pure Prolog [52].
- **multiparadigm programming**, by combining the underlying logical systems, for example, to get combined relational and functional programming from Horn clause logic with equality as in Eqlog [26], combined functional and object-oriented programming from reflective equational logic as in FOOPS [30], and all three paradigms together from a reflective Horn clause logic with equality as in FOOPlog [30].

Logical programming can be given a precise grounding using the notions of institution [22] and logical system [54], and this is in part responsible for the cleanliness and simplicity of the various languages that we have designed. A logical programming language "wears its semantics on its sleeve" and does not need the complex machinery of Scott-Strachey-style "denotational" semantics [70, 74] or of Hoare-style "axiomatic" semantics [43]. In fact, we would claim that a language that can *only* be given a semantics in one of these styles, and

[2] The basic intuitions for this view were expressed in [26] and formalized using institutions in [20]. The definition below is an informal exposition of the more recent formalization in [54].

[3] In particular, there should be reasonably simple notions of sentence, deduction, model and satisfaction, preferably with a **completeness theorem**, saying that the notion of deduction is fully adequate for the notion of model, in the sense that given any set $\mathcal{P}$ of sentences, another sentence s can be deduced from $\mathcal{P}$ if and only if every model of $\mathcal{P}$ satisfies s.

[4] In some sense, initial models are "standard" or "most prototypical" models; see below for more detail.

thus is not a logical programming language, is just too complex. Strictly speaking, most functional programming languages are not logical programming languages in our sense, since they have features which are not consistent with any simple deductive or model-theoretic semantics[5].

Although equational deduction by undirected replacement of equals by equals can be very inefficient, *directed* replacement (i.e., **term rewriting**) can be much faster. For example, [62] claims speeds comparable to compiled Lisp on sequential machines for a (restricted) class of equations, and the Rewrite Rule Machine Project at SRI is developing a parallel architecture on which term rewriting promises to be much more efficient than conventional languages on conventional machines [51, 31]; see also [46] for a survey of efficient implemention techniques for higher order functional programming. Term rewriting provides a complete deductive system for equality, and any expression reduces to a unique "canonical form" (one that cannot be further rewritten), provided certain simple conditions are satisfied[6]. Thus, the proof theory of order-sorted equational logic developed in this paper gives efficient term rewriting in two different ways, yielding two different OBJ systems:

- OBJ2 [23] reduces order-sorted rewriting to many-sorted rewriting using results in Section 4 and [23].
- OBJ3 uses a more efficient operational semantics that does order-sorted term rewriting directly [48].

1.3 Retracts

In a strongly typed programming language, certain expressions may fail strong type checking, even though intuitively they have a meaningful value. For example, if the factorial function is only defined for natural numbers, then the expression `((- 6)/(- 2))!` is not well-formed, since the argument of the factorial function is a rational number. However, we would like to give such an expression the "benefit of the doubt" at run-time, since it might actually evaluate to a natural (in this case, it evaluates to 3). **Retracts** provide this flexibility by lowering the sort of a subexpression to the required subsort. In this example, the parser inserts the retract function symbol,

$$\mathtt{r}_{\mathtt{Rational,Natural}} \ : \ \mathtt{Rational \ -> \ Natural}$$

to fill the gap, yielding the expression $(\mathtt{r}_{\mathtt{Rational,Natural}}\mathtt{((- 6)/(- 2)))!}$. Retracts only disappear if their argument has the required sort. This is accomplished by "retract equations" of the form

$$r_{s,s'}(x) = x$$

where $s' \leq s$ and x is a variable of sort s'. Otherwise, the retract remains, providing an error message that pinpoints exactly where the problem occurred. For example, the expression `7 + (((- 3)/(- 9))!)` evaluates to $\mathtt{7 + (r}_{\mathtt{Rational,Natural}}\mathtt{(1 / 3))!}$. The basic result about retracts asserts its soundness, in the sense that adding retracts and retract equations to an order-sorted specification is a *conservative extension*, i.e., the original equational deduction and standard model are not disturbed. Retracts combine the flexibility of untyped languages with the discipline of strong typing.

[5] For example, ML has assignments and exceptions, while Miranda has *ad hoc* coercions among various kinds of numbers, as well as lazy pattern matching.

[6] These conditions are that the equations, when viewed as rules, are terminating and Church-Rosser; in the order-sorted case, one must also assume that the rules are sort-decreasing.

1.4 Exceptions and Partial Operations

It is very difficult to handle exceptional expressions, such as division by zero or the top of an empty stack, within a strong typing discipline. For example, there is no satisfactory way to specify a type as simple as stack of natural number, because `top(empty)` should be a natural number but isn't. Rational numbers are even worse, because avoiding division by zero requires heavy use of "hidden functions" and "error constants." However, OSA provides very simple solutions to all these problems. For stacks, it suffices to specify a subsort of nonempty stacks, `NeStack < Stack`, such that `top` and `pop` have `NeStack` as their argument sort. Similarly, for rational numbers, it suffices to specify a subsort `NzRational < Rational` of nonzero rationals such that division has `NzRational` as its second (divisor) argument sort.

OSA supports in a natural way many different styles for dealing with errors and partial operations. The two examples discussed above make the operations well-defined by specifying an appropriate domain subsort. More generally, the domain of a partial operation may be specified by a condition; for example, to compose two paths in a graph, the end vertex of the first path should coincide with the source vertex of the second. Such conditions are called **sort constraints**. In other cases, the best approach may be to provide an **error supersort**. For example, an operation to read the value, of sort `Value`, of an array in a given position could have value sort `Value?`, a supersort of `Value`, that contains error messages for attempting to read at positions where no value is stored. Part II of this paper will cover all these different approaches and their semantics, also discussing how they relate to other solutions, such as partial algebras and error algebras.

1.5 Constructors, Selectors, Multiple Representations and Coercions

Structured data are generally composed by *constructors* and decomposed by *selectors*. The inadequacy of strong typing for the stack example is a special case of what we call the **constructor-selector problem**: for a given constructor, to define operations that retrieve its components. Although this problem is insoluble in MSA (many-sorted algebra), it has a simple solution in OSA [29].

There are also many problems where one wants to represent data in more than one way, and then convert freely among the representations, using whichever is more convenient or efficient in a given context. This is **multiple representation**; for example, consider Cartesian and polar coordinates for points. There are other problems where one wants to convert from one sort of data to another in an irreversible way; for example, to apply integer addition to two rational numbers, one might first truncate them; this illustrates **coercions**. Multiple representation is a special case of coercion, since the selectors for one representation applied to data of another can be considered mediated by coercions that change the representation. The difference is that conversions between multiple representations are necessarily reversible, i.e., are isomorphisms. OSA also provides an initial algebra semantics for all these constructions [29].

1.6 About this Paper

After introducing the basic concepts of OSA, this paper gives a detailed account of order-sorted equational deduction, including a completeness theorem and an initial algebra construction for conditional equations. This machinery is then applied to show that adding retracts is a conservative extension. A reduction theorem shows that encoding order-sorted algebras as many-sorted algebras yields an equivalence of categories, which can then be exploited to prove a general existence theorem for initial algebras (it applies even when terms do not have a least sort) as well as simple proofs of OSA McKinsey-Malcev Quasivariety

and Birkhoff Variety Theorems. A final section compares our notion of order-sorted algebra to others in the literature. To help the reader's intuition and illustrate the expressive ease of OSA, a number of examples are given using an OBJ-like syntax. Appendix A gives a more ambitious example, OBJ code for a number hierarchy from the naturals up to the quaternions.

This paper has been a long time in gestation. The first paper on order-sorted algebra [18] was written in 1978, but never published because it seemed so possible and desirable to simplify and generalize its approach. The present paper finally fulfills the promise of [18], with suitable simplifications and generalizations, and it also treats some new topics, including order-sorted equational deduction and model-theoretic results about varieties and quasi-varieities. Several versions of the present paper have been circulated fairly widely; their titles are slight variations of the current title, and their dates include 2 March 1985, 22 October 1986, and 17 May 1988. The last of these reflects our decision to split the paper into three parts, as further discussed in the subsection below. In the meantine, a rather large literature has grown up around order-sorted algebra and its applications, and trying to take proper account of it has slowed us down further.

1.6.1 Brief Overview of Subsequent Parts

Part II of this paper will consider exception handling and sort constraints in detail, including several error recovery and error specification disciplines and their soundness, and comparing retracts, error supersorts and strict and unsafe operations. It will also discuss the very important topic of sort constraints, which permit defining subsorts by equational conditions. The main theorem for sort constraints is an initial algebra construction reducing the problem to order-sorted equational logic. Part III will give an algebraic semantics for parameterized order-sorted abstract data types with the related concepts of theory, view and module expression, as in OBJ [14, 15] and Clear [3, 4]. This supports the effective integration of the programming and assertional aspects of OBJ, which make it a "wide spectrum" language.

1.7 Acknowledgements

We thank Prof. Jean-Pierre Jouannaud for help with several of the topics discussed here. In particular, he participated in formulating the definition of regularity and in the initial stages of the number hierarchy example; our joint work on the operational semantics of OSA [23] was also helpful for the treatments of deduction and retracts given here. We also thank Dr. Kokichi Futatsusgi, who worked out much of the OBJ2 implementation [14, 15], Prof. David Plaisted, who worked out much of the OBJ1 implementation [32], and Dr. Joseph Tardo, who did the original OBJT implementation [34]. Drs. Christoph Walther, Gert Smolka and Harald Ganzinger deserve thanks for pointing out a bug in the definition of direct deduction in an earlier draft, and Smolka has also made many other useful suggestions. Drs. Claude and Hélène Kirchner, and especially Timothy Winkler, deserve special thanks for their work on theoretical and practical aspects of our latest version of OBJ, the OBJ3 system that is available from SRI International. We also thank Winkler for his comments and for his help with the number hierarchy example, and Mr. Narciso Marti-Oliet for his detailed comments on one of the latest drafts. Finally, thanks to Mr. Malcolm Harper for translating this paper from Scribe to LaTeX; without this effort, it might never have been completed.

2 Order-Sorted Algebra

This section contains the most basic definitions and results of OSA, including signature, algebra, homomorphism, term, least sort of a term, initiality, equation, satisfaction, subalgebra, quotient, congruence, image, the Homomorphism Theorem, and product.

2.1 Signatures

The notation of sorted (also called "indexed") sets greatly facilitates the technical development of both MSA and OSA. Given a "sort set" S, an S-**sorted set** A is just a family of sets A_s for each "sort" $s \in S$; we will write $\{A_s \mid s \in S\}$. Similarly, given S-sorted sets A and B, an S-**sorted function** $f\colon A \to B$ is an S-sorted family $f = \{f_s\colon A_s \to B_s \mid s \in S\}$.

In the order-sorted case, S is a **partially ordered set**, or **poset**, i.e., there is a binary relation $\leq$ on S that is reflexive, transitive, and antisymmetric, in the sense that $x \leq y$ and $y \leq x$ imply $x = y$. Every poset also has an associated relation $<$, defined by $x < y$ iff $x \leq y$ and $x \neq y$, that is transitive and antireflexive in the sense that $\neg(x < x)$. We will often use the extension of the ordering on S to strings of equal length in S^* by $s_1 \ldots s_n \leq s'_1 \ldots s'_n$ iff $s_i \leq s'_i$ for $1 \leq i \leq n$. Similarly, $\leq$ extends to pairs $\langle w, s\rangle$ in $S^* \times S$ by $\langle w, s\rangle \leq \langle w', s'\rangle$ iff $w \leq w'$ and $s \leq s'$. (These are the orderings that arise from poset products.)

Definition 2.1 A **many-sorted signature** is a pair (S, Σ), where S is called the **sort set** and Σ is an $S^* \times S$-sorted family $\{\Sigma_{w,s} \mid w \in S^* \text{and } s \in S\}$. Elements of (the sets in) Σ are called operation (or function) symbols, or for short, **operations**. An **order-sorted signature** is a triple $(S, \leq, \Sigma)$ such that (S, Σ) is a many-sorted signature, $(S, \leq)$ is a poset, and the operations satisfy the following **monotonicity condition**,

$$\sigma \in \Sigma_{w1,s1} \cap \Sigma_{w2,s2} \text{ and } w1 \leq w2 \text{ imply } s1 \leq s2.$$

When the sort set S is clear, we write Σ for (S, Σ), and when the poset (S,$\leq$) is clear, we write Σ for $(S, \leq, \Sigma)$. When $\sigma \in \Sigma_{w,s}$ we say that σ has **rank** $\langle w, s\rangle$, **arity** w, and (value, or result, or coarity) **sort** s.

We may write $\sigma\colon w \to s$ for $\sigma \in \Sigma_{w,s}$ to emphasize that σ denotes a function with arity w and sort s. An important special case is $w = \lambda$, the empty string; then $\sigma \in \Sigma_{\lambda,s}$ denotes a **constant** of sort s. Notice that the monotonicity condition excludes overloaded constants, because $\lambda = w1 = w2$ implies $s1 = s2$. □

Example 2.2 (Lists of Integers) We give an order-sorted signature for lists of integers, assuming that the sort `Int` of integers is already defined. The subsort `NeList` of nonempty lists is introduced so that the (traditionally partial) `head` and `tail` operations can be total on this subsort. The notation used in this example (and in subsequent examples) supports a powerful and flexible "mixfix" operation syntax; in particular, it allows prefix, postfix, infix and "outfix" (as in {_} for singleton set formation). Here the kth underbar character (_) is a placeholder in an operation form that shows where to put an expression whose sort is less than or equal to the kth sort in the sort list (which occurs between the `:` and the `->` signs); the value sort follows the `->`. Also, $\leq$ is written `<` for typographic convenience. All these syntactic conventions follow OBJ.

```
sorts NeList List .
subsorts Int < NeList < List .
op nil : -> List .
op _ _ : List List -> List .
op _ _ : NeList List -> NeList .
op head : NeList -> Int .
op tail : NeList -> List .
```

The double underbar operation form defines a juxtaposition notation for concatenation of lists. This concatenation operation is subsort polymorphic, and would be ambiguous in an ordinary many-sorted signature. To fully describe the intended model, we need more than just a signature, we also need equations, algebras, and initiality; these are introduced in the subsections below. □

Given an operation symbol σ and a lower bound $w0$ for the sorts of its arguments, we can consider the following three conditions:

(1) There is a least arity for σ that is $\geq w0$.

(2) There is a least rank for σ among those with arity $\geq w0$.

(3) There is a least sort for σ among those with arity $\geq w0$.

It turns out that (1) and (2) are equivalent because of monotonicity, and that both imply (3). Signatures satisfying (1) are quite basic to our exposition, and are called **regular**. Regular signatures both support a least sort for terms, and extend the usual word (or term) algebra construction to OSA (see Section 2.3). Signatures satisfying (3) are called **preregular**, and are discussed further in Section 5.2 below.

Definition 2.3 An order-sorted signature Σ is **regular** iff given σ in $\Sigma_{w1,s1}$ and given $w0 \leq w1$ in S^* there is a least rank $\langle w, s\rangle \in S^* \times S$ such that $w0 \leq w$ and $\sigma \in \Sigma_{w,s}$. □

Regularity allows a strong form of subsort polymorphism "locally," while still permitting *ad hoc* polymorphism "globally" (Section 1.1.1 explained these terms); for example, `+` can denote addition over the complex numbers and its many subtypes with subsort polymorphism, as well as Boolean exclusive or with *ad hoc* polymorphism. The signature in Example 2.2 above is regular, but it would not be if an operation `_ _` of rank $\langle$`ListNeList`, `NeList`$\rangle$ were added to it. We now give a more precise statement of some relations among the three conditions above:

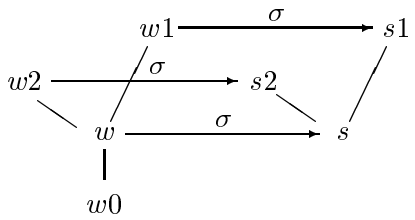
Fact 2.4 An order-sorted signature Σ is regular iff given σ in $\Sigma_{w1,s1}$ and given $w0 \leq w1$ in S^* there is a least arity $w \in S^*$ such that $w0 \leq w$ and $\sigma \in \Sigma_{w,s}$ for some $s \in S$. Moreover, if Σ is regular then given σ in $\Sigma_{w1,s1}$ with $w0 \leq w1$ there is a least sort $s \in S$ such that $\sigma \in \Sigma_{w,s}$ for some $w \in S^*$ with $w0 \leq w$, and this s is the same one that appears in the least rank $\langle w, s\rangle$ for σ with $w \geq w0$; thus, regularity implies preregularity.

Proof: The "only if" is immediate, while "if" follows from monotonicity. The other assertions are also easy. □

When the poset of sorts satisfies a descending chain condition (and thus in particular, when it is finite), there is a combinatorial condition that is equivalent to regularity. (Figure 1 illustrates the relations among the arities and sorts in this result.)

Definition 2.5 A poset $(S, \leq)$ satisfies the **ascending chain condition**, or is **Noetherian**, iff there is no strictly increasing infinite chain $s_1 < s_2 < ... < s_n < ...$ in $(S, \leq)$. Similarly, $(S, \leq)$ satisfies the **descending chain condition**, or is **coNoetherian**, iff there is no strictly decreasing infinite chain $s_1 > s_2 > ... > s_n > ...$ in $(S, \leq)$. □

Lemma 2.6 An order-sorted signature Σ over a coNoetherian poset $(S, \leq)$ is regular if and only if whenever $\sigma \in \Sigma_{w1,s1} \cap \Sigma_{w2,s2}$ and there is some $w0 \leq w1, w2$ then there is some $w \leq w1, w2$ such that $\sigma \in \Sigma_{w,s}$ and $w0 \leq w$.



Note: Diagonal and vertical lines indicate sort inclusions, while horizontal arrows indicate instances of the operation symbol σ.

Figure 1: Visualizing Lemma 2.6

Proof: The "only if" part is easy. For the "if" part, let us say that a pair $\langle w, s\rangle$ "satisfies condition P" iff $\sigma \in \Sigma_{w,s}$ and $w0 \leq w$. Then Σ is regular iff $\langle w1, s1\rangle$ satisfies P implies there is a least $\langle w, s\rangle$ satisfying P. So we now suppose that there is some $\langle w1, s1\rangle$ satisfying P but there is no least $\langle w, s\rangle$ satisfying P. Then in particular, $\langle w1, s1\rangle$ cannot be least for P, and so there is some $\langle w1', s1'\rangle$ satisfying P such that $\langle w1', s1'\rangle \not\geq \langle w1, s1\rangle$. Then by assumption, there is some $\langle w2, s2\rangle < \langle w1, s1\rangle$ satisfying P. Iterating this process yields an infinite descending chain $\langle w1, s1\rangle > \langle w2, s2\rangle > ... > \langle wn, sn\rangle > ...$, which contradicts the coNoetherian assumption. (This last step uses the easy to check fact that any finite product of coNoetherian posets is coNoetherian.) □

2.2 Algebras

We now turn to the models that provide actual functions to interpret the operation symbols in a signature.

Definition 2.7 Let (S, Σ) be a many-sorted signature. Then an (S, Σ)-**algebra** A is a family $\{A_s \mid s \in S\}$ of sets called the **carriers** of A, together with a function $A_\sigma\colon A_w \to A_s$ for each σ in $\Sigma_{w,s}$ where $A_w = A_{s1} \times ... \times A_{sn}$ when $w = s1...sn$ and where A_w is a one point set when $w = \lambda$.

Let $(S, \leq, \Sigma)$ be an order-sorted signature. Then an $(S, \leq, \Sigma)$-**algebra** is an (S, Σ)-algebra A such that

1. $s \leq s'$ in S implies $A_s \subseteq A_{s'}$ and
2. $\sigma \in \Sigma_{w1,s1} \cap \Sigma_{w2,s2}$ and $w1 \leq w2$ imply $A_\sigma\colon A_{w1} \to A_{s1}$ equals $A_\sigma\colon A_{w2} \to A_{s2}$ on A_{w1}.

Both of these are **monotonicity conditions**. When the sort set S is clear, (S, Σ)-algebras may be called **many-sorted** Σ-**algebras**; similarly, when $(S, \leq)$ is clear, $(S, \leq, \Sigma)$-algebras may be called **order-sorted** Σ-**algebras**. Also, we may write $A_\sigma^{w,s}$ for $A_\sigma\colon A_w \to A_s$. □

Many different ways to define order-sorted algebras have by now appeared in the literature. However, most of them are either less general (for example, they fail to admit overloading) or else are more complex, as discussed in Section 5 in much more detail.

Example 2.2: (continued) If we let $\mathbf{Z}$ denote the set of all integers, then the algebra that we have in mind for the List of Integers signature has $A_{\texttt{List}} = \mathbf{Z}^*$ (all lists of integers), $A_{\texttt{NeList}} = \mathbf{Z}^+$ (the non-empty lists), $A_{\texttt{Int}} = \mathbf{Z}$ (the lists of length 1), $\texttt{nil} = \lambda$ (the empty list), $_\,_$ as concatenation, and `head` and `tail` as expected. Note that $\mathbf{Z} \subseteq \mathbf{Z}^+ \subseteq \mathbf{Z}^*$. □

Stacks can be described in a very similar way, with `pop` and `top` partial operations defined only on the non-empty stacks; see Example 2.15 in Section 2.4.

Definition 2.8 Let (S, Σ) be a many-sorted signature, and let A and B be (S, Σ)-algebras. Then an (S, Σ)-**homomorphism** $h\colon A \to B$ is an S-sorted function $h = \{h_s\colon A_s \to B_s \mid s \in S\}$ satisfying the following **homomorphism condition**

(1) $h_s(A_\sigma^{w,s}(a)) = B_\sigma^{w,s}(h_w(a))$ for each $\sigma \in \Sigma_{w,s}$ and $a \in A_w$

where $h_w(a) = \langle h_{s1}(a1), ..., h_{sn}(an)\rangle$ when $w = s1...sn$ and $a = \langle a1, ..., an\rangle$ with $ai \in A_{si}$ for i=1,...,n when $w \neq \lambda$. If $w = \lambda$, condition (1) specializes to

(1′) $h_s(A_\sigma^{\lambda,s}) = B_\sigma^{\lambda,s}$.

(S, Σ)-algebras and (S, Σ)-homomorphisms form a category that we denote $\mathbf{Alg}_\Sigma$. When the sort set S is clear, (S, Σ)-homomorphism may be called just (many-sorted) Σ-homomorphisms.

Let $(S, \leq, \Sigma)$ be an order-sorted signature, and let A, B be order-sorted $(S, \leq, \Sigma)$-algebras. Then an $(S, \leq, \Sigma)$-**homomorphism** $h\colon A \to B$ is an (S, Σ)-homomorphism satisfying the following **restriction condition**

(2) $s \leq s'$ and $a \in A_s$ imply $h_s(a) = h_{s'}(a)$.

When the poset $(S, \leq)$ is clear, $(S, \leq, \Sigma)$-homomorphisms are also called (**order-sorted**) Σ-**homomorphisms**. The $(S, \leq, \Sigma)$-algebras and $(S, \leq, \Sigma)$-homomorphisms form a category that we denote $\mathbf{OSAlg}_\Sigma$. □

Since, by definition, every $(S, \leq, \Sigma)$-algebra is an (S, Σ)-algebra and every $(S, \leq, \Sigma)$-homomorphism is an (S, Σ)-homomorphism, there is a "forgetful" functor from $\mathbf{OSAlg}_\Sigma$ to $\mathbf{Alg}_\Sigma$. Notice the slight abuse of language whereby Σ denotes two different signatures: an order-sorted signature $(S, \leq, \Sigma)$ in $\mathbf{OSAlg}_\Sigma$ and a many-sorted signature (S, Σ) in $\mathbf{Alg}_\Sigma$. Also notice that OSA *properly generalizes* MSA, in the sense that any many-sorted (S, Σ)-algebra is an order-sorted $(S, \leq, \Sigma)$-algebra for $\leq$ the trivial ordering on S with $s \leq s'$ iff $s = s'$. Indeed, with this ordering on S we have that $\mathbf{OSAlg}_\Sigma = \mathbf{Alg}_\Sigma$ and the forgetful functor $\mathbf{OSAlg}_\Sigma \to \mathbf{Alg}_\Sigma$ is the identity.

Injective and surjective are defined for an order-sorted Σ-homomorphism $f\colon A \to B$ just as for the many-sorted case: f is **injective** iff f_s is an injective function for each $s \in S$, and f is **surjective** iff f_s is surjective for each $s \in S$. Similarly, f is an **isomorphism** iff f is both injective and surjective. Just as in the many-sorted case we have

Lemma 2.9 An order-sorted Σ-homomorphism $f\colon A \to B$ is an isomorphism iff there is an order-sorted Σ-homomorphism $f^{-1}\colon B \to A$ such that $f^{-1} \circ f = 1_A$ and $f \circ f^{-1} = 1_B$.

Proof: Since the "if" part is easy, we will just show the "only if" part, using the well-known fact that the desired result holds for many-sorted algebra. This gives a many-sorted Σ-homomorphism $f^{-1}\colon B \to A$ satisfying the desired two equations. Now we only need to check that f^{-1} satisfies the restriction condition of Definition 2.8. Let $b \in B_s$ and let $s \leq s'$. Then $b = f_s(a)$ for some $a \in A_s$ and also $b = f_{s'}(a)$ since f is order-sorted. Thus $f_s^{-1}(b) = a = f_{s'}^{-1}(b)$. □

2.3 Terms

This subsection shows that terms over regular signatures have a well-defined least sort, and also that the standard MSA term algebra construction gives an initial order-sorted algebra. We first review the inductive construction of the many-sorted term algebra T_Σ using the same notation as in [57], except that we will be more pedantic, using $\underline{(}$ and $\underline{)}$ to denote parentheses used as formal syntactic symbols; however, this pedantry is only temporary. If Σ is a many-sorted signature with sort set S, then:

- $\Sigma_{\lambda,s} \subseteq T_{\Sigma,s}$;
- if $\sigma \in \Sigma_{w,s}$ and if $ti \in T_{\Sigma,si}$ for $i = 1, ..., n$ where $w = s1...sn$ with $n > 0$, then (the string) $\sigma\underline{(}t1...tn\underline{)}$ is in $T_{\Sigma,s}$.

Now given an order-sorted signature Σ, we similarly construct the order-sorted Σ-term algebra $\mathcal{T}_\Sigma$ as the least family $\{\mathcal{T}_{\Sigma,s} \mid s \in S\}$ of sets satisfying the following conditions:

- $\Sigma_{\lambda,s} \subseteq \mathcal{T}_{\Sigma,s}$ for $s \in S$;
- $\mathcal{T}_{\Sigma,s'} \subseteq \mathcal{T}_{\Sigma,s}$ if $s' \leq s$;
- if $\sigma \in \Sigma_{w,s}$ and if $ti \in \mathcal{T}_{\Sigma,si}$ where $w = s1...sn \neq \lambda$, then (the string) $\sigma\underline{(}t1...tn\underline{)} \in \mathcal{T}_{\Sigma,s}$.

Also,

- for $\sigma \in \Sigma_{w,s}$ let $\mathcal{T}_\sigma$: $\mathcal{T}_w \to \mathcal{T}_s$ send $t1, ..., tn$ to (the string) $\sigma\underline{(}t1...tn\underline{)}$.

Thus we can write $\sigma(t1, ..., tn)$ for $\sigma\underline{(}t1...tn\underline{)}$.

Clearly $\mathcal{T}_\Sigma$ is an order-sorted Σ-algebra. Notice that $\mathcal{T}_{\Sigma,s}$ is not in general equal to $T_{\Sigma,s}$ or even to $\bigcup_{s' \leq s} T_{\Sigma,s'}$. Also notice that it is quite possible that $\mathcal{T}_{\Sigma,s} = \emptyset$ for some s, i.e., that there are no ground terms of sort s. $\mathcal{T}_\Sigma$ is a kind of order-sorted Herbrand universe construction; unfortunately, some authors insist on adding a constant if none is otherwise provided, thus destroying the initiality of their construction.

A given term t in an order-sorted term algebra can have many different sorts. In particular, if $t \in \mathcal{T}_\Sigma$ has sort s, then it also has sort s' for any $s' \geq s$; and because an operation symbol σ may have different ranks, a term $\sigma(t1, ..., tn)$ can even have sorts that are not directly comparable. One unfortunate consequence of such ambiguity is that $\mathcal{T}_\Sigma$ may fail to be initial, just as in the many-sorted case T_Σ may fail to be initial if Σ is ambiguous. However, this problem disappears for regular signatures.

Proposition 2.10 Given a regular order-sorted signature Σ, for every[7] $t \in \mathcal{T}_\Sigma$ there is a least $s \in S$, called the **least sort** of t and denoted $LS(t)$, such that $t \in \mathcal{T}_{\Sigma,s}$.

Proof: We proceed by induction on the depth of terms in $\mathcal{T}_\Sigma$. If $t \in \mathcal{T}_\Sigma$ has depth 0, then $t = \sigma$ for some $\sigma \in \Sigma_{\lambda,s1}$ and so by regularity with $w0 = w1 = \lambda$, there is a least $s \in S$ such that $\sigma \in \Sigma_{\lambda,s}$; this is the least sort of σ. Now consider a well-formed term $t = \sigma\underline{(}t1...tn\underline{)} \in \mathcal{T}_{\Sigma,s}$ of depth $n + 1$. Then each ti has depth $\leq n$ and therefore by the induction hypothesis, has a least sort, say si; let $w0 = s1...sn$. Then $\sigma \in \Sigma_{w',s'}$ for some w', s' with $s' \leq s$ and $w0 \leq w'$, and by regularity, there are least w' and s' such that $\sigma \in \Sigma_{w',s'}$ and $w' \geq w0$; this least s' is the desired least sort of t. □

This result can be generalized by weakening the notion of regularity to preregularity. In fact, preregularity is actually equivalent to the existence of a least sort for each term (by Proposition 5.2 in Section 5.2). We now turn to the important issue of initial algebras.

[7] By convention, for A a Σ-algebra, $a \in A$ means $a \in A_s$ for some $s \in S$.

Definition 2.11 Let Σ be an order-sorted signature. Then an order-sorted Σ-algebra is **initial** in the class of all order-sorted Σ-algebras iff there is a unique order-sorted Σ-homomorphism from it to any other order-sorted Σ-algebra. $\square$

Theorem 2.12 Let Σ be a regular order-sorted signature. Then $\mathcal{T}_\Sigma$ is an initial order-sorted Σ-algebra.

Proof: In this proof we write $\mathcal{T}$ for $\mathcal{T}_\Sigma$. Let A be an order-sorted Σ-algebra; then we must show that there is a unique order-sorted Σ-homomorphism h: $\mathcal{T} \to A$. We will (1) construct h, then (2) show it is an order-sorted Σ-homomorphism, and finally (3) show it is unique.

(1) We construct h by induction on the depth of terms in $\mathcal{T}$. There are two cases:

(1a) If $t \in \mathcal{T}$ has depth 0, then $t = \sigma$ for some constant σ in Σ. By regularity, σ has a least sort s. Then for any $s' \geq s$ we define

$$h_{s'}(\sigma) = A_\sigma^{\lambda,s}.$$

(1b) If $t = \sigma(t1...tn) \in \mathcal{T}$ has depth $n+1$, then by regularity there are least w and s with $\sigma \in \Sigma_{w,s}$ where $w = s1...sn \neq \lambda$ and $LS(ti) \leq si$ for $i = 1, ..., n$. Then for any $s' \geq s$ we define

$$h_{s'}(t) = A_\sigma^{w,s}(h_{s1}(t1), ..., h_{sn}(tn)),$$

noting that $h_{s1}(t1), ..., h_{sn}(tn)$ are already defined.

(2) We now show that h is an order-sorted Σ-homomorphism. By construction h satisfies the restriction condition ((2) of Definition 2.8). To see that it also satisfies the homomorphism condition ((1) of Definition 2.8), we again consider two cases:

(2a) $\sigma \in \Sigma_{\lambda,s}$ is a constant. By regularity and monotonicity, s is the least sort of σ, and we have already defined

$$h_s(\sigma) = A_\sigma^{\lambda,s}$$

as needed.

(2b) We now consider a term t of depth greater than 0, and let $\sigma \in \Sigma_{w',s'}$ with $w' = s'1...s'n \neq \lambda$ be such that $t = \sigma(t1...tn) = \mathcal{T}_\sigma^{w',s'}(t1, ..., tn)$. By regularity and Proposition 2.10 there are least $w = s1...sn$ and $s = LS(t)$ such that $t = \sigma(t1...tn) = \mathcal{T}_\sigma^{w,s}(t1, ..., tn)$. Then $w \leq w'$ and $s \leq s'$ so that (2) of Definition 2.7 gives $A_\sigma^{w',s'} = A_\sigma^{w,s}$ on A^w. Thus, using the already established fact that h satisfies the restriction condition, we have

$$h_{s'}(\sigma(t1...tn)) = A_\sigma^{w,s}(h_{s1}(t1), ..., h_{sn}(tn)) = A_\sigma^{w',s'}(h_{s'1}(t1), ..., h_{s'n}(tn))$$

as needed.

(3) Finally, we show the uniqueness of h. In fact, we will show that if h': $\mathcal{T} \to A$ is an order-sorted Σ-homomorphism, then $h = h'$, by induction on the depth of terms. For depth 0 consider $\sigma \in \Sigma_{\lambda,s}$. Then s is the least sort of σ, and for any $s \geq s'$, we must have

$$h'_{s'}(\sigma) = h'_s(\sigma) = A_\sigma^{\lambda,s} = h_s(\sigma) = h_{s'}(\sigma),$$

as desired. Now assuming the result for depth $\leq n$, consider a term $t = \sigma(t1...tn) = \mathcal{T}_\sigma^{w',s'}(t1, ..., tn)$ of depth $n+1$ with $\sigma \in \Sigma_{w',s'}$ and $w' = s'1...s'n$. As in (2b), there are least $w = s1...sn$ and $s = LS(t)$ such that $t = \sigma(t1...tn) = \mathcal{T}_\sigma^{w,s}(t1, ..., tn)$ and $A_\sigma^{w',s'} = A_\sigma^{w,s}$ on A^w. Then

$$
\begin{aligned}
h'_{s'}(t) &= A_\sigma^{w',s'}(h'_{s'1}(t1), ..., h'_{s'n}(tn)) \\
&= A_\sigma^{w',s'}(h_{s'1}(t1), ..., h_{s'n}(tn)) \text{ (by induction hypothesis)} \\
&= A_\sigma^{w,s}(h_{s1}(t1), ..., h_{sn}(tn)) \\
&= h_{s'}(t)
\end{aligned}
$$

as needed. □

The terms considered above are **ground terms**, in the sense that they involve no variables. We can extend the above result to so-called free algebras by considering instead terms that may involve variables. In fact, terms with variables can be seen as a special case of ground terms, by enlarging the signature with new constants that correspond to the variable symbols. Let us assume that each variable comes with a given sort, so that we have an S-sorted family $X = \{X_s \mid s \in S\}$ of *disjoint* sets that we shall call a **variable set**. Given an order-sorted signature $(S, \leq, \Sigma)$ and an S-sorted variable set X that is disjoint from Σ, we define the new order-sorted signature $(S, \leq, \Sigma(X))$ by $\Sigma(X)_{\lambda,s} = \Sigma_{\lambda,s} \cup X_s$ and $\Sigma(X)_{w,s} = \Sigma_{w,s}$ for $w \neq \lambda$. It is easy to see that $\Sigma(X)$ is regular if Σ is. Now form $\mathcal{T}_{\Sigma(X)}$ and view it as an order-sorted Σ-algebra just by forgetting about the constants in X; let us denote this algebra $\mathcal{T}_\Sigma(X)$. The following result and proof are entirely analogous to the MSA case [57].

Theorem 2.13 Given a regular order-sorted signature $(S, \leq, \Sigma)$, let A be a Σ-algebra and let a: $X \to A$ be an S-sorted function; hereafter we call such a function an **assignment**. Then there is a unique order-sorted Σ-homomorphism a^*: $\mathcal{T}_\Sigma(X) \to A$ such that $a^*(x) = a(x)$ for each $x \in X$.

Proof: Σ-algebras A with an assignment a: $X \to A$ are in bijective correspondence with $\Sigma(X)$-algebras A. Now the initiality of $\mathcal{T}_\Sigma(X)$ among all $\Sigma(X)$-algebras A (Theorem 2.12) gives the desired result. □

2.4 Equations

Order-sorted algebra would be very impoverished without equations. We first give two simple examples of what equations can do, and then we give the formal definitions; these are somewhat more subtle than might be expected. In the examples, the keyword pair `obj...endo` delimits an **object** and indicates that *initial algebra semantics* is intended.

Example 2.14 (Bits)

```
obj BITS is
  sorts Bit ErrBit List ErrList .
  subsorts Bit < List < ErrList .
  subsorts Bit < ErrBit < ErrList .
  ops 0 1 : -> Bit .
  op nil : -> List .
  op _ _ : List List -> List .
  op head : List -> ErrBit .
  op tail : List -> ErrList .
  vars L L' L'' : List .
  var B : Bit .
  eq nil L = L .
  eq L nil = L .
  eq L (L' L'') = (L L') L'' .
  eq head(B L)= B .
  eq tail(B L)= L .
endo
```

What is interesting here is the way the "error supersorts" `ErrBit` and `ErrList` are used in `head` and `tail`; in the intended interpretation, elements that are in `ErrList` but not in `List` serve as error messages. An alternative approach follows Example 2.2 by defining a subsort `NeList` as the domain for `head` and `tail`; this would have made `ErrBit` and `ErrList` unnecessary. Also note that `Bit` is a subsort of the non-empty lists purely for syntactic convenience, allowing us to say that `0` is itself a list. □

Example 2.15 (Stack of Integers) This example is interesting primarily because it has previously been treated in so many different formalisms, so that comparison between formalisms is facilitated. We believe that no other formalism gives so simple and natural a description as the following:

```
obj STACK-OF-INT is
  protecting INT .
  sorts Stack NeStack .
  subsort NeStack < Stack .
  op empty : -> Stack .
  op push : Int Stack -> NeStack .
  op top_ : NeStack -> Int .
  op pop_ : NeStack -> Stack .
  var E : Int .
  var S : Stack .
  eq top(push(E,S)) = E .
  eq pop(push(E,S)) = S .
endo
```

□

The above examples are actually executable OBJ3 code [36]. Of course, our development of OSA is fully general and considers arbitrary models for sets of equations over an order-sorted signature. OBJ uses this 'loose' or 'theory' semantics to describe requirements on actual parameters for parameterized objects. For example, a parameterized sorting object should allow any partially ordered set as actual parameter, and a parameterized polynomial object should allow any commutative ring for its coefficients. Initiality modulo a set of equations is discussed in Section 3 below, but parameterization and requirement theories are deferred to Part III of this paper.

We now develop the formalities concerning equations. Recall that by the freeness of $\mathcal{T}_\Sigma(X)$ (Theorem 2.13), an assignment a of values in an order-sorted Σ-algebra A to elements from a variable set X that is disjoint from Σ extends to an order-sorted Σ-homomorphism a^* to A from the Σ-terms with variables in X. The OSA definition of equations is similar to that for MSA [57], in that equations are triples $\langle X, t, t' \rangle$ with t and t' in $\mathcal{T}_\Sigma(X)$, and an order-sorted algebra A satisfies such an equation iff $a^*(t) = a^*(t')$ for each assignment $a\colon X \to A$. However, before actually giving such a definition we need to consider what sorts to allow for the terms t and t'. In MSA, we are forced to require that t and t' have the *same* sort, but OSA allows more flexibility. For example, in the `BITS` example above, the equation `head(B L) = B` has a lefthand side whose least sort is `ErrBit` and a righthand side whose least sort is `Bit`. The following example will help to motivate a general restriction on the form of equations.

Example 2.16

```
obj ABCD is
  sorts A B C D .
```

```
  subsort A C < B .
  subsort C < D .
  op a : -> A .
  op b : -> B .
  op c : -> C .
  op d : -> D .
  eq a = b .
  eq b = c .
  eq c = d .
endo
```

These equations do not involve any variables. To say that an algebra H satisfies them presumably means that $h_A(a) = h_B(b) = h_C(c) = h_D(d)$ for h: $\mathcal{T}_\Sigma \to H$ the unique order-sorted homomorphism (where Σ is the signature of the example). Given these equations, one expects to be able to "replace equals by equals" and deduce that the equation `a = d` holds, even though the sorts `A` and `D` are not comparable in the sort ordering[8]. In fact, under the notion of satisfaction suggested above, the equation `a = d` is satisfied by any algebra H that satisfies the original equations. This might suggest that we only require that the sorts of the terms t and t' in an equation lie in the same connected component[9] of the poset $(S, \leq)$. □

Definition 2.17 For $(S, \leq, \Sigma)$ a regular order-sorted signature, a Σ-**equation** is a triple $\langle X, t, t' \rangle$ where X is a variable set and t, t' are in $\mathcal{T}_{\Sigma(X)}$ with $LS(t)$ and $LS(t')$ in the same connected component of $(S, \leq)$. We will use the notation $(\forall X)\ t = t'$. An order-sorted Σ-algebra A **satisfies** a Σ-equation $(\forall X)\ t = t'$ iff $a^*_{LS(t)}(t) = a^*_{LS(t')}(t')$ in A for every assignment a: $X \to A$. Similarly, A **satisfies** a set Γ of Σ-equations iff it satisfies each member of Γ; in this case, we say that A is a (Σ, Γ)-**algebra**. When the variable set X can be deduced from the context (for example, if X contains just the variables that occurr in t and t', with sorts that are uniquely determined or else have been previously declared) we allow it to be omitted; that is, we allow *unquantified* notation for equations[10].

Order-sorted conditional equations generalize order-sorted equations in the usual way, i.e., they are expressions of the form $(\forall X)\ t = t'$ if C, where the **condition** C is a finite set of unquantified Σ-equations involving only variables in X (when $C = \emptyset$, conditional Σ-equations are regarded as ordinary Σ-equations). An order-sorted Σ-algebra A **satisfies** the equation $(\forall X)\ t = t'$ if C iff for each assignment a: $X \to A$ such that $a^*_{LS(v)}(v) = a^*_{LS(v')}(v')$ in A for each equation $v = v'$ in C, then also $a^*_{LS(t)}(t) = a^*_{LS(t')}(t')$ in A.

Given a signature Σ and a set Γ of (possibly conditional) Σ-equations, we let $\mathbf{OSAlg}_{\Sigma,\Gamma}$ denote the category of all (Σ, Γ)-algebras, with all Σ-homomorphisms among them. □

Although these notions of equation and satisfaction seem quite reasonable for OSA, and in particular seem general enough to support equational deduction, there is a subtle difficulty: equational satisfaction is not closed under isomorphism, i.e., an order-sorted algebra A may satisfy an equation that is not satisfied by an isomorphic algebra B. The following exhibits this curious phenomenon:

Example 2.18

[8] But notice that the sorts *are* comparable for each equation in the `BITS` and `STACK` examples.

[9] Given a poset $(S, \leq)$, let $\equiv$ denote the transitive and symmetric closure of $\leq$. Then $\equiv$ is an equivalence relation whose equivalence classes are called the **connected components** of $(S, \leq)$.

[10] However, the reader should be aware that satisfaction of an equation depends crucially on its variable set [57].

```
obj ABC is
  sorts A B C .
  subsorts B < A C .
  op a : -> A .
  op b : -> B .
  op c : -> C .
  eq a = c .
endo
```

Letting Σ be the signature of this example, the term algebra $\mathcal{T}_\Sigma$ has $(\mathcal{T}_\Sigma)_A = \{a, b\}$, $(\mathcal{T}_\Sigma)_B = \{b\}$ and $(\mathcal{T}_\Sigma)_C = \{b, c\}$ does *not* satisfy the equation `a = c`. However, the order-sorted Σ-algebra H with $H_A = H_C = \{b, d\}$ and $H_B = \{b\}$, with the constants a, b, c interpreted as d, b, d (respectively) *does* satisfy `a = c`, even though the unique order-sorted Σ-homomorphism h: $\mathcal{T}_\Sigma \to H$ is a Σ-isomorphism. □

The desire to be rid of this analomy motivates the following:

Definition 2.19 A poset $(S, \leq)$ is (upward) **filtered** iff for any two elements $s, s' \in S$ there is an element $s'' \in S$ such that $s, s' \leq s''$. A partially ordered set S is **locally filtered** iff each of its connected components is filtered. An order-sorted signature $(S, \leq, \Sigma)$ is **locally filtered** iff $(S, \leq)$ is locally filtered, and is **coherent** iff it is locally filtered and regular. □

We will show below that for coherent signatures, satisfaction is "abstract" in the sense of being closed under isomorphism. Coherence guarantees that all sorts in a connected component "cohere" in the sense that any finite set of them can always be reconciled by appeal to a bigger sort; "incoherence" causes the trouble in Example 2.18. Any many-sorted signature is coherent, since the trivial ordering ($s \leq s'$ iff $s = s'$) is always locally filtered and regular. In many examples, the sort poset is Noetherian.

Proposition 2.20 A Noetherian poset is locally filtered if and only if each connected component has a maximum element.

Proof: The "if" part is obvious. For the "only if" part, assume that there is no maximum element in a given connected component C and pick any element $s_1 \in C$. Since s_1 is not a maximum, there must be an element $s_1' \in C$ such that $s_1' \not\leq s_1$. Since S is locally filtered, we get an element $s_2 \geq s_1, s_1'$ such that $s_1 < s_2$. We can now iterate this process to get a strictly increasing sequence $s_1 < s_2 < ... < s_n < ...$ that contradicts the Noetherian assumption. □

Proposition 2.21 Given a coherent signature Σ and isomorphic Σ-algebras A and B, then A satisfies an equation $(\forall X)\ t = t'$ if and only if B does.

Proof: By symmetry of the isomorphism relation, it is enough to prove the "only if" part. Assume that A satisfies $(\forall X)\ t = t'$ and let f: $A \to B$ be an isomorphism. Then any assignment b: $X \to B$ can be written $b = f \circ a$ for some assignment a: $X \to A$. Initiality now implies that $b^* = f \circ a^*$. Let $s \geq LS(t), LS(t')$. Then

$$b^*_s(t) = f_s(a^*_s(t)) = f_s(a^*_s(t')) = b^*_s(t')$$

as desired. □

This result generalizes easily to the satisfaction of conditional equations.

How restrictive is coherence? In practice, not at all. In fact, coherence can be automatically ensured by a computer implementation, just by adding some new top elements to the signature given by a user: given a regular order-sorted signature Σ, extend it to a coherent signature $coh(\Sigma)$ identical to Σ except for adding a new sort u_C for each nonfiltered connected component C. Note that for each sort s in the original set of sorts we then have $\mathcal{T}_{coh(\Sigma),s} = \mathcal{T}_{\Sigma,s}$ and for the new sorts u_C we have $\mathcal{T}_{coh(\Sigma),u_C} = \bigcup_{s \in C} \mathcal{T}_{\Sigma,s}$. This approach is even more flexible and general than requiring a universal maximum of all sorts as in [23]. Intuitively, the sorts in a connected component form a semantically related "local universe" of discourse.

One benefit of requiring signatures to be coherent is a great simplicity and flexibility in the treatment of equality, since we can always assume that t and t' have the same sort whenever they appear in an equation $t = t'$ by going to a common supersort. This does require that t and t' lie in the same connected component, but we do not consider equations across different components to be meaningful; moreover, even this condition could be dropped by adding a universal sort, as discussed in Section 5.

2.5 Subalgebras, Congruences, Quotients and Products

This subsection gives OSA forms of some familiar MSA concepts, including subalgebra, congruence relation, quotient algebra, kernel, image, and product algebra. It also proves the Homomorphism Theorem and the universal properties of quotients and products.

Definition 2.22 For (S, Σ) a many-sorted signature and for A a many-sorted Σ-algebra, a **many-sorted Σ-subalgebra** B of A is an S-sorted family of subsets $B_s \subseteq A_s$ for each $s \in$S such that

(1) given $\sigma \in \Sigma_{w,s}$ with $w = s1...sn$ and $bi \in B_i$ for $i = 1, ..., n$, then $A_\sigma(b1, ..., bn) \in B_s$; in particular, when $w = \lambda$ then $A_\sigma \in B_s$.

For $(S, \leq, \Sigma)$ an order-sorted signature and A an order-sorted Σ-algebra, an **order-sorted Σ-subalgebra** B of A is a many-sorted Σ-subalgebra B of A such that

(2) $B_s \subseteq B_{s'}$ whenever $s \leq s'$.

□

Definition 2.23 For (S, Σ) a many-sorted signature and A a many-sorted Σ-algebra, a **many-sorted Σ-congruence** $\equiv$ on A is a S-sorted family $\{\equiv_s |\ s \in S\}$ of equivalence relations $\equiv_s$ on A_s such that

(1) given $\sigma \in \Sigma_{w,s}$ with $w = s1...sn$ and given $ai, a'i \in A_{si}$ for $i = 1, ..., n$ such that $ai \equiv_{si} a'i$, then

$$A_\sigma(a1, ..., an) \equiv_s A_\sigma(a'1, ..., a'n).$$

For $(S, \leq, \Sigma)$ an order-sorted signature and A an order-sorted Σ-algebra, an **order-sorted Σ-congruence** $\equiv$ on A is a many-sorted Σ-congruence $\equiv$ such that

(2) given $s \leq s'$ in S and $a, a' \in A_s$ then $a \equiv_s a'$ iff $a \equiv_{s'} a'$.

□

Proposition 2.24 Let Σ be an order-sorted signature. Then

1. The order-sorted Σ-subalgebras of an order-sorted Σ-algebra A form a complete lattice under the inclusion ordering.

2. The order-sorted Σ-congruences on an order-sorted Σ-algebra A form a complete lattice under the inclusion ordering.

Moreover, in these lattices greatest lower bound is computed by set intersection. (These results are well known for MSA.)

Proof: By the following lemma, it suffices to show that any intersection of Σ-algebras or of Σ-congruences is a Σ-congruence, which is easy in this case. □

Lemma 2.25 A class $\mathcal{C}$ of subsets of a set C is a complete lattice under set inclusion if it is closed under arbitrary set-theoretic intersections, including intersection over the empty family of subsets, which by convention is the maximum element of $\mathcal{C}$; moreover, greatest lower bound is then computed by set intersection. □

Definition 2.26 Let $f\colon A \to B$ be a many-sorted Σ-homomorphism. Then the **kernel** of f is the S-sorted family of equivalence relations $\equiv_f$ defined by $a \equiv_{f,s} a'$ iff $f_s(a) = f_s(a')$; it will be denoted $ker(f)$. □

Proposition 2.27 A kernel is a many-sorted congruence. If $f\colon A \to B$ is an order-sorted Σ-homomorphism, then $ker(f)$ is an order-sorted Σ-congruence.

Proof: Given an S-indexed function $f\colon A \to B$, then each $\equiv_{f,s}$ is an equivalence relation. To prove the congruence property (1), let $\sigma \in \Sigma_{w,s}$ with $w = s1...sn$ and assume that $ai \equiv_{f,s} a'i$, i.e., that $f_s(ai) = f_s(a'i)$ for $i = 1, ..., n$. Then

$f_s(A_\sigma(a1, ..., an)) = B_\sigma(f_{s1}(a1), ..., f_{sn}(an)) =$

$B_\sigma(f_{s1}(a'1), ..., f_{sn}(a'n)) = f_s(A_\sigma(a'1, ..., a'n))$

so that

$A_\sigma(a1, ..., an) \equiv_{f,s} A_\sigma(a'1, ..., a'n)$

as desired. When f is order-sorted, we have to check the congruence property (2). This follows from the fact that $f_s(a) = f_{s'}(a)$ and $f_s(a') = f_{s'}(a')$ whenever $s \leq s'$ in S and $a, a' \in A_s$. □

Definition 2.28 The **image** of a Σ-homomorphism $f\colon A \to B$ is the subalgebra $f(A)$ with $f(A)_s = f(A_s)$ for each $s \in S$. □

Fact 2.29 If $f\colon A \to B$ is an order-sorted Σ-homomorphism, then $f(A)$ is an order-sorted subalgebra.

Proof: To check condition (1) of the definition of subalgebra, let $\sigma \in \Sigma_{w,s}$ with $w = s1...sn$, let $bi \in f(A)_{si}$ for $i = 1, ..., n$, and let $ai \in A_{si}$ such that $bi = f_{si}(ai)$ for $i = 1, ..., n$. Then $B_\sigma(b1, ..., bn) \in f(A)_s$ since $B_\sigma(b1, ..., bn) = f_s(A_\sigma(a1, ..., an))$. For the order-sorted case, we have to check condition (2), but this is an easy set-theoretic consequence of the fact that f is order-sorted. □

We now define the quotient of an order-sorted algebra by a congruence relation and (more generally) by a set of relations. This construction is simpler for locally filtered signatures, but it can be generalized to arbitrary signatures.

Definition 2.30 For $(S, \leq, \Sigma)$ a locally filtered order-sorted signature, A an order-sorted Σ-algebra, and $\equiv$ an order-sorted Σ-congruence on A, the **quotient** of A by $\equiv$ is the order-sorted Σ-algebra $A/\equiv$ defined as follows: for each connected component C, let $A_C = \bigcup_{s \in C} A_s$ and define the congruence relation $\equiv_C$ by $a \equiv_C a'$ iff there is a sort $s \in C$ such

that $a \equiv_s a'$. Then $\equiv_C$ is clearly reflexive and symmetric. It is transitive since $a \equiv_s a'$ and $a' \equiv_{s'} a''$ yield $a \equiv_{s''} a''$ for $s'' \geq s, s'$. The inclusion $A_s \subseteq A_C$ induces an injective map $A_s/\equiv_s \to A_C/\equiv_C$ since for $a, a' \in A_s$ we have $a \equiv_s a'$ implies $a \equiv_C a'$ by construction, and conversely $a \equiv_C a'$ implies $a \equiv_{s'} a'$ for some $s' \in C$, and taking $s'' \geq s, s'$ it also implies $a \equiv_{s''} a'$ and therefore it implies $a \equiv_s a'$ by property (2) of the definition of order-sorted congruence. Denoting by q_C the natural projection q_C: $A_C \to A_C/\equiv_C$ of each element a into its $\equiv_C$-equivalence class, we define the carrier $(A/\equiv)_s$ of sort s in the quotient algebra to be $q_C(A_s)$. The order-sorted algebra $A/\equiv$ comes equipped with a surjective order-sorted Σ-homomorphism q: $A \to A/\equiv$ defined by restriction of the q_C to each of the sorts, called the **quotient map** associated to the congruence $\equiv$. The operations are defined by $(A/\equiv)_\sigma([a1], ..., [an]) = [A_\sigma(a1, ..., an)]$, which is well defined since $\equiv$ is an order-sorted Σ-congruence. $\square$

Fact 2.31 Under the assumptions of Definition 2.30, $ker(q) = \equiv$. $\square$

Fact 2.32 Again under the assumption of Definition 2.30, any S-sorted family R of binary relations R_s on A_s for $s \in S$ is contained in a smallest order-sorted Σ-congruence on A.

Proof: This congruence can be expressed as the intersection in the lattice of congruences of all order-sorted congruences that contain R. $\square$

Definition 2.33 Given an arbitrary S-sorted family R of binary relations R_s on A_s for $s \in S$, then the **quotient** of A by R, denoted A/R, is the quotient of A by the smallest order-sorted Σ-congruence on A containing R. $\square$

Proposition 2.34 (Universal Property of Quotient) If Σ is a locally filtered order-sorted signature, if A is an order-sorted Σ-algebra, and if R is an S-sorted family of binary relations R_s on A_s for $s \in S$, then the quotient map q: $A \to A/R$ satisfies the following:

(1) $R \subseteq ker(q)$, and

(2) if f: $A \to B$ is any order-sorted Σ-homomorphism such that $R \subseteq ker(f)$, then there is a unique Σ-homomorphism v: $A/R \to B$ such that $v \circ q = f$ (see Figure 2).

Proof:

(1) follows from $ker(q)$ being the smallest congruence containing R.

For (2), let f: $A \to B$ be an order-sorted Σ-homomorphism such that $R \subseteq ker(f)$. Then $ker(q) \subseteq ker(f)$ and both are congruences so that for each connected component C we have $ker(q)_C \subseteq ker(f)_C$ and there is a unique function v_C: $(A/R)_C \to B_C$ such that $v_C \circ q_C = f_C$ for f_C: $A_C \to B_C$ defined by $f_C(a) = f_s(a)$ if $a \in A_s$ (this is well defined by local filtering). It remains only to check that, restricting v_C to each one of the sorts $s \in C$, the family $\{v_s \mid s \in S\}$ thus obtained is an order-sorted Σ-homomorphism. Property (2) for order-sorted homomorphisms follows by construction. Let $\sigma \in \Sigma_{w,s}$ with $w = s1...sn$ and let $ai \in A_{si}$ for $i = 1, ..., n$. Then (omitting sort qualifications throughout) we have

$v((A/R)_\sigma([a1], ..., [an])) = v([A_\sigma(a1, ..., an)]) =$
$f(a_\Sigma(a1, ..., an)) = B_\sigma(f(a1), ..., f(an)) =$
$B_\sigma(v([a1]), ..., v([an]))$.

We leave the case $w = \lambda$ for the reader to check. $\square$

We remark that this universal property characterizes the quotient map uniquely up to isomorphism. The following is now an easy consequence of Proposition 2.34:

Proposition 2.35 (Homomorphism Theorem) Let Σ be a locally filtered order-sorted signature and let f: $A \to B$ be an order-sorted Σ-homomorphism. Then $A/ker(f) \cong f(A)$ (isomorphism as order-sorted Σ-algebras).

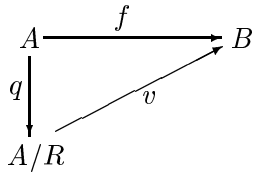


Figure 2: Condition (2) of Proposition 2.34

Proof: Let f': $A \to f(A)$ denote the corestriction of f to $f(A)$. Then by the universal property of the quotient with $R = ker(f') = ker(f)$, there is a (unique) v: $A/ker(f) \to f(A)$ such that $v \circ q = f'$. Then v is surjective since f' is, and it remains to show that v is injective. To this end (omitting sort qualifications again), suppose that $v([a1]) = v([a2])$. Then $f(a1) = f(a2)$, so $[a1] = [a2]$. □

We shall say that an order-sorted algebra C is an **homomorphic image** of another order-sorted algebra A iff there is an order-sorted Σ-homomorphism f: $A \to B$ such that $C = f(A)$. By the Homomorphism Theorem (for Σ locally filtered), C is a homomorphic image of A iff C is Σ-isomorphic to $A/\equiv$ for some order-sorted Σ-congruence $\equiv$.

Definition 2.36 Let (S, Σ) be an order-sorted signature and let A and B be many-sorted Σ-algebras. Then we define their **product** $A \times B$ to be the many-sorted Σ-algebra with carriers $(A \times B)_s = A_s \times B_s$ for each $s \in S$, and with $(A \times B)_\sigma(\langle a_1, b_1\rangle, ..., \langle a_n, b_n\rangle) = \langle A_\sigma(a_1, ..., a_n), B_\sigma(b_1, ..., b_n)\rangle$ for each σ: $s_1...s_n \to s$ in Σ, where each a_i and b_i are of sort s_i for $i = 1, ..., n$. We now define the two **projections** $p1$: $A \times B \to A$ and $p2$: $A \times B \to B$ to be $\{p1_s \mid s \in S\}$ and $\{p2_s \mid s \in S\}$ respectively, where $p1_s$: $A_s \times B_s \to A_s$ and $p2_s$: $A_s \times B_s \to B_s$ are the first and second projection functions from the Cartesian product $A_s \times B_s$. Notice that $p1$ and $p2$ are Σ-homomorphisms. If A and B are order-sorted algebras, then so is $A \times B$, and the projection functions are order-sorted homomorphisms. Similarly, we can define the product $\prod_i A_i$ of a family $\{A_i \mid i \in I\}$ of many-sorted or order-sorted Σ-algebras, with projection homomorphisms p_j: $\prod_i A_i \to A_j$. □

Proposition 2.37 (Universal Property of Product) Let A, B, C be order-sorted (or many-sorted) Σ-algebras, and let $q1$: $C \to A$ and $q2$: $C \to B$ be order-sorted (or many-sorted) Σ-homomorphisms. Then there is a unique order-sorted (or many-sorted) Σ-homomorphism v: $C \to A \times B$ such that $p1 \circ v = q1$ and $p2 \circ v = q2$. This result also generalizes to products of arbitrary families. □

3 Order-Sorted Equational Deduction

This section gives rules of deduction for OSA with conditional equations, and proves their completeness. This yields a construction for initial and free order-sorted algebras as quotients of term algebras by the congruence generated by the rules of deduction from the given equations, in a way that parallels MSA.

Before turning to the rules, we consider order-sorted term substitution. Given a coherent order-sorted signature $(S, \leq, \Sigma)$ and two S-sorted variable sets X and Y, a **substitution** is an S-sorted map θ: $X \to \mathcal{T}_\Sigma(Y)$; note that this is a special case of the *assignment* concept given earlier (Theorem 2.13) in which the values assigned to the variables are terms. We adopt the convention that the unique order-sorted Σ-homomorphism θ^*: $\mathcal{T}_\Sigma(X) \to \mathcal{T}_\Sigma(Y)$ induced by θ is also denoted θ.

3.1 The Rules of Order-sorted Equational Deduction

Given an order-sorted signature Σ and a set Γ of conditional Σ-equations, we consider each unconditional equation in Γ to be **derivable**. The following rules allow deriving further (unconditional) equations:

(1) *Reflexivity.* Each equation of the form
$(\forall X)\ t = t$
is derivable.
(2) *Symmetry.* If
$(\forall X)\ t = t'$
is derivable, then so is
$(\forall X)\ t' = t$.
(3) *Transitivity.* If the equations
$(\forall X)\ t = t'$, $(\forall X)\ t' = t''$
are derivable, then so is
$(\forall X)\ t = t''$.
(4) *Congruence.* If θ, θ': $X \to \mathcal{T}_\Sigma(Y)$ are substitutions such that for each $x \in X$, the equation
$(\forall Y)\ \theta(x) = \theta'(x)$
is derivable, then given $t \in \mathcal{T}_\Sigma(X)$, the equation
$(\forall Y)\ \theta(t) = \theta'(t)$
is also derivable.
(5) *Substitutivity.* If
$(\forall X)\ t = t'$ if C
is in Γ, and if θ: $X \to \mathcal{T}_\Sigma(Y)$ is a substitution such that for each $u = v$ in C, the equation
$(\forall Y)\ \theta(u) = \theta(v)$
is derivable, then so is
$(\forall Y)\ \theta(t) = \theta(t')$.

When the equations in Γ are unconditional, rule (5) takes the form

(5′) *Unconditional Substitutivity.* If
$(\forall X)\ t = t'$
is in Γ, and if θ: $X \to \mathcal{T}_\Sigma(Y)$ is a substitution, then
$(\forall Y)\ \theta(t) = \theta(t')$
is derivable.

Although these rules are rather compactly formulated, they correspond exactly to intuitions that we feel should be expected for equational deduction. Of course, there are many possible variations on this rule set; for example, see [72]. Also, order-sorted Horn clause logic is discussed in [28], and [27] gives an overview of the equational case.

3.2 Completeness and Initiality Theorems

We now show that the above rules are sound and complete for deriving all the *unconditional* equations that hold in the class of all algebras that satisfy Γ. We then obtain initial and free algebras for a set Γ of conditional equations as a corollary. While the structure of our proof is fairly traditional, it is more succinct than traditional proofs, because it exploits the machinery of algebra rather than relying on purely syntactic arguments; for example, it uses initiality to prove commutativity of a diagram.

Theorem 3.1 (Completeness) Given a coherent order-sorted signature Σ, given t, t' in $\mathcal{T}_\Sigma(X)$, and given a set Γ of conditional Σ-equations, the following assertions are equivalent:

(C1) $(\forall X)\ t = t'$ is derivable from Γ using rules (1)-(5).

(C2) $(\forall X)\ t = t'$ is satisfied by every order-sorted Σ-algebra that satisfies Γ.

When all equations in Γ are unconditional, the same holds replacing rule (5) by rule (5′).

Proof: We leave the reader to check *soundness*, i.e., that (C1) implies (C2); this follows as usual by induction from the soundness of each rule of deduction separately. Here we show *completeness*, i.e., that (C2) implies (C1). The structure of this proof is as follows: We are given a Σ-equation $e = (\forall X)\ t = t'$ that is satisfied by every Σ-algebra that satisfies Γ, and we wish to show that e is derivable from Γ; to this end, we construct a Σ-algebra $\mathcal{A}$ such that if $\mathcal{A}$ satisfies e then e is derivable from Γ; then we show that $\mathcal{A}$ satisfies Γ.

First, we show that the following property of terms $t, t' \in \mathcal{T}_\Sigma(X)_s$ for some sort s, defines an order-sorted Σ-congruence on $\mathcal{T}_\Sigma(X)$:

(D) $(\forall X)\ t = t'$ is derivable from Γ using rules (1)-(5).

Let us denote this relation $\sim_{\Gamma(X)}$. Then rules (1)-(3) say that $\sim_{\Gamma(X)}$ is an equivalence relation on $\mathcal{T}_\Sigma(X)_s$ for each sort s. By applying rule (4) to terms t of the form $\sigma(x_1, ..., x_n)$ for $\sigma \in \Sigma$, we see that $\sim_{\Gamma(X)}$ is a many-sorted Σ-congruence. Finally, $\sim_{\Gamma(X)}$ is also an order-sorted Σ-congruence, because property (D) does not depend upon s.

Now we can form the order-sorted quotient of $\mathcal{T}_\Sigma(X)$ by $\sim_{\Gamma(X)}$, which we denote by $\mathcal{T}_{\Sigma,\Gamma}(X)$, or within this proof, just $\mathcal{A}$ for short. Then by the construction of $\mathcal{A}$, for each $t, t' \in \mathcal{T}_\Sigma(X)$ we have

(*) $[t] = [t']$ in $\mathcal{A}$ iff (D) holds,

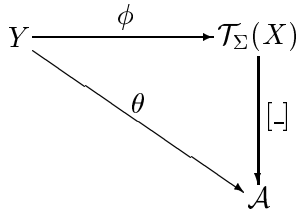
where $[t]$ denotes the $\sim_{\Gamma(X)}$-equivalence class of t.

We next show the key property of $\mathcal{A}$, that

(**) $(\forall X)\ t = t'$ satisfied in $\mathcal{A}$ implies that (D) holds.

Since the equation $(\forall X)\ t = t'$ is satisfied in $\mathcal{A}$, we can use the inclusion $i_X\colon X \to \mathcal{A}$ sending x to $[x]$ as an S-sorted assignment to get that $[t] = [t']$ in $\mathcal{A}$; then (D) holds by (*).

We now prove that $\mathcal{A}$ satisfies Γ. Let $(\forall Y)\ t = t'$ if C be a conditional equation in Γ, and let $\theta\colon Y \to \mathcal{A}$ be an S-sorted assignment such that $\theta(u) = \theta(v)$ for each $u = v$ in C. Then for each $s \in S$ and each $y \in Y_s$ we can choose a representative $t_y \in \mathcal{T}_\Sigma(X)_s$ such that $\theta(y) = [t_y]$ in $\mathcal{A}$. Now let $\phi\colon Y \to \mathcal{T}_\Sigma(X)$ be the substitution sending y to t_y. Then $\theta(y) = [\phi(y)]$ for each $y \in Y$, and therefore $\theta(t) = [\phi(t)]$ in $\mathcal{A}$ for any $t \in \mathcal{T}_\Sigma(Y)$, by the freeness of $\mathcal{T}_\Sigma(Y)$ over Y.



Therefore, $[\phi(u)] = [\phi(v)]$ holds in $\mathcal{A}$, and by the property (*), the equation $(\forall X)\ \phi(u) = \phi(v)$ is derivable from Γ using (1)-(5) for each $u = v$ in C. Therefore by rule (5), the equation $(\forall X)\ \phi(t) = \phi(t')$ is derivable from Γ, and hence by (*), $\theta(t) = \theta(t')$ holds in $\mathcal{A}$, and thus the conditional equation $(\forall Y)\ t = t'$ if C holds in $\mathcal{A}$.

Since an unconditional equation is just a conditional equation whose set C of conditions is empty, when every equation in Γ is unconditional we are reduced to the simplified special case of the above argument where only the rule (5′) is needed. □

It is interesting to notice that this theorem also gives the Completeness Theorems for ordinary MSA, and of course for unsorted algebra, as special cases. Now the initiality and freeness results.

Corollary 3.2 (Initiality I) Given a coherent order-sorted signature Σ and a set Γ of conditional Σ-equations, then $\mathcal{T}_{\Sigma,\Gamma}(\emptyset)$ (henceforth denoted $\mathcal{T}_{\Sigma,\Gamma}$) is an initial (Σ,Γ)-algebra, and $\mathcal{T}_{\Sigma,\Gamma}(X)$ is a free (Σ,Γ)-algebra on X.

Proof: First notice that the freeness of $\mathcal{T}_{\Sigma,\Gamma}(X)$ specializes to the initiality of $\mathcal{T}_{\Sigma,\Gamma}$ when $X = \emptyset$, so that it suffices to show the freeness of $\mathcal{T}_{\Sigma,\Gamma}(X)$. Let A be an order-sorted algebra satisfying Γ, and let a: $X \to A$ be an assignment for A. Then we have to show that there is a unique order-sorted Σ-homomorphism $a^\&$: $\mathcal{T}_{\Sigma,\Gamma}(X) \to A$ extending a, i.e., such that $a^\&(q(x)) = a(x)$ for each $x \in X$, where q denotes the quotient homomorphism q: $\mathcal{T}_\Sigma(X) \to \mathcal{T}_{\Sigma,\Gamma}(X)$. The existence of $a^\&$ follows from the Completeness Theorem, because the fact that A satisfies Γ implies that $a^*(t) = a^*(t')$ for every equation $(\forall X)\ t = t'$ that is derivable from Γ with the rules (1)-(5), and this implies that $\sim_{\Gamma(X)} \subseteq ker(a^*)$, and thus by the universal property of quotients (Proposition 2.34), there is a unique order-sorted homomorphism $a^\&$: $\mathcal{T}_{\Sigma,\Gamma}(X) \to A$ with $a^* = a^\& \circ q$.

The uniquenes of $a^\&$ now follows by combining the universal property of $\mathcal{T}_\Sigma(X)$ as a free order-sorted algebra on X with the universal property of q as a quotient, as follows: Let h: $\mathcal{T}_{\Sigma,\Gamma}(X) \to A$ be another order-sorted homomorphism such that $h(q(x)) = a(x)$ for each $x \in X$. Since $\mathcal{T}_\Sigma(X)$ is a free order-sorted algebra on X, we have $a^* = h \circ q$, and by the universal property of q as a quotient we have $h = a^\&$ as desired. □

It is also worth explicitly drawing out the following consequence of our proof of the Completeness Theorem:

Corollary 3.3 Given a coherent order-sorted signature Σ and a set Γ of (conditional) Σ-equations, an equation $(\forall X)\ t = t'$ is satisfied by every Σ-algebra that satisfies Γ iff it is satisfied by $\mathcal{T}_{\Sigma,\Gamma}(X)$. □

3.3 Retracts

We have already shown in the Introduction that strong typing is not flexible enough in practice, and suggested that OSA can provide the necessary flexibility with retracts. For example, a term such as `head(tail(0 1 0 0))` is not well-formed according to the syntax of Example 2.14 (`BITS`), because `head`'s arguments should have sort `NeList` but the term `tail(0 1 0 0)` only has sort `List`, even though we know that it will evaluate to the nonempty list `1 0 0`. One might think that this is "just run-time type checking," and should therefore be handled by the operational semantics. However, retracts have a very nice, purely semantic treatment as a *conservative extension* (see below); of course, there is also an operational semantics, developed in joint work with Jean-Pierre Jouannaud [23].

The basic construction extends an order-sorted signature Σ to another order-sorted signature $\Sigma^\otimes$ having the same sorts as Σ, and having the same operation symbols as Σ plus some new ones called **retracts** of the form $r_{s',s}$: $s' \to s$ for each pair s', s with $s' > s$. The semantics of retracts is then given by new **retract equations** of the form

$$(\forall x)\ r_{s',s}(x) = x$$

where x is a variable of sort s.

The OBJ implementation inserts retracts to transform ill-formed Σ-terms, such as `head(tail(0 1 0 0))`, that might become well-formed after reduction, into $\Sigma^\otimes$-terms.

This has the effect of giving them the benefit of the doubt at parse time, by filling gaps between actual sorts and required sorts with retracts. For example,

`head(tail(0 1 0 0))`

is replaced by

$\texttt{head}(\texttt{r}_{\texttt{List,NeList}}(\texttt{tail(0 1 0 0 0)}))$

and is then reduced to `1` by applying the rules in `BITS` and a retract rule; thus the original term is vindicated during reduction. On the other hand, the term

`head(tail(tail(1)))`

is temporarily accepted as the term

$\texttt{head}(\texttt{r}_{\texttt{List,NeList}}(\texttt{tail}(\texttt{r}_{\texttt{List,NeList}}(\texttt{tail(1)}))))$

and is then reduced to

$\texttt{head}(\texttt{r}_{\texttt{List,NeList}}(\texttt{tail}(\texttt{r}_{\texttt{List,NeList}}(\texttt{nil}))))$

which serves as a very informative error message. This kind of run-time typechecking is relatively inexpensive, and together with the polymorphism provided by subsorts and by parameterized modules[11], combines the syntactic flexibility of untyped languages with the advantages of strong typing. In fact, unlike the untyped case, truly nonsensical expressions can be detected at compile time and rejected, whereas any expression that could possibly recover is allowed to be evaluated. By "truly nonsensical" we mean expresions such as `factorial(false)` that contain subexpressions in the wrong connected component (assuming that booleans and natural numbers are in different connected components of the sort poset) and therefore cannot be parsed by inserting retracts.

We now show that adding retracts is safe. Suppose that we begin with an order-sorted signature Σ and a set Γ of conditional Σ-equations. By adding the retract operations we extend Σ to a signature $\Sigma^{\otimes}$, and by adding the retract equations we extend Γ to a set of equations $\Gamma^{\otimes}$. Our requirement for retracts to be well-behaved is that the extension $(\Sigma, \Gamma) \subseteq (\Sigma^{\otimes}, \Gamma^{\otimes})$ should be **conservative** in the sense that

$$t \sim_{\Gamma(X)} t' \text{ iff } t \sim_{\Gamma^{\otimes}(X)} t', \text{ for all } t, t' \in \mathcal{T}_{\Sigma}(X).$$

In model-theoretic terms, this is equivalent to requiring that the unique order-sorted Σ-homomorphism $\psi_X \colon \mathcal{T}_{\Sigma,\Gamma}(X) \to \mathcal{T}_{\Sigma^{\otimes},\Gamma^{\otimes}}(X)$ which leaves the elements of X fixed, is injective. We will prove this under the following very natural assumption on the algebras $\mathcal{T}_{\Sigma,\Gamma}(X)$: given $X \subseteq X'$, then the unique Σ-homomorphism $\iota_{X,X'} \colon \mathcal{T}_{\Sigma,\Gamma}(X) \to \mathcal{T}_{\Sigma,\Gamma}(X')$ induced by the composite map $X \hookrightarrow X' \to \mathcal{T}_{\Sigma,\Gamma}(X')$ (first inclusion, then the natural mapping of each variable to the class of terms equivalent to it) is injective. We will say that a presentation (Σ, Γ) is **faithful** if it satisfies this injectivity condition. Although they are pathological, unfaithful presentations do exist, and for them the extension with retracts is not conservative, as shown by the following example from [25]:

Example 3.4 Let Σ have sorts a, b, u with $a, b \leq u$, have an operation $f \colon a \to b$, have no constants of sort a, have constants $0, 1$ of sort b, plus $+, \&$ binary infix and $\neg$ unary prefix of sort b. Let Γ have the equations $\neg(f(x)) = f(x)$, $y + y = y$, $y \& y = y$, $y + (\neg y) = 1$, $(\neg y) + y = 1$, $y \& (\neg y) = 0$, $(\neg y) \& y = 0$, $\neg 0 = 1$, $\neg 1 = 0$. Then $(\forall x)\, 1 = 0$ is deducible from Γ, where x is a variable of sort a, although $(\forall \emptyset)\, 1 = 0$ is *not* deducible from Γ. Thus (Σ, Γ) is not faithful. Note that $\mathcal{T}_{\Sigma,\Gamma}$ has $1 \neq 0$ (because of the second equation) but $\mathcal{T}_{\Sigma^{\otimes},\Gamma^{\otimes}}$ has $1 = 0$ because of the first equation and the presence of constants of sort a such as $r_{u,a}(0)$ and $r_{u,a}(1)$. Thus, the extension $(\Sigma, \Gamma) \subseteq (\Sigma^{\otimes}, \Gamma^{\otimes})$ is not conservative. □

There are simple conditions on both the signature Σ and on the equations Γ that guarantee faithfulness of a presentation (Σ, Γ). For arbitrary Γ, it is necessary and sufficient

[11] Parameterized modules will be the main subject of the forthcoming Part III of this paper.

that Σ has no **quasi-empty** models, which are algebras A such that $A_s = \emptyset$ for some s but $A_{s'} \neq \emptyset$ for some other sort s' [25]. For arbitrary Σ, it is sufficient that Γ is a set of confluent rewrite rules [56].

The following model-theoretic proof of the conservative extension result for faithful presentations uses naturality of the family ψ_X of morphisms, which in particular gives commutativity of the following diagram for $X \subseteq X'$, where $\mu_{X,X'}$ is the unique $\Sigma^{\otimes}$-homomorphism induced by the composite map $X \hookrightarrow X' \to \mathcal{T}_{\Sigma^{\otimes},\Gamma^{\otimes}}(X')$:

$$\begin{array}{ccc} \mathcal{T}_{\Sigma,\Gamma}(X) & \xrightarrow{\psi_X} & \mathcal{T}_{\Sigma^{\otimes},\Gamma^{\otimes}}(X) \\ \downarrow{\scriptstyle \iota_{X,X'}} & & \downarrow{\scriptstyle \mu_{X,X'}} \\ \mathcal{T}_{\Sigma,\Gamma}(X') & \xrightarrow[\psi_{X'}]{} & \mathcal{T}_{\Sigma^{\otimes},\Gamma^{\otimes}}(X') \end{array}$$

Theorem 3.5 If Σ is coherent and (Σ, Γ) is faithful, then the extension $(\Sigma, \Gamma) \subseteq (\Sigma^{\otimes}, \Gamma^{\otimes})$ is conservative.

Proof: We have to show that ψ_X: $\mathcal{T}_{\Sigma,\Gamma}(X) \to \mathcal{T}_{\Sigma^{\otimes},\Gamma^{\otimes}}(X)$ is injective. By the above naturality diagram plus faithfulness, it suffices to show that $\psi_{X'}$: $\mathcal{T}_{\Sigma,\Gamma}(X') \to \mathcal{T}_{\Sigma^{\otimes},\Gamma^{\otimes}}(X')$ is injective, where $X' \supseteq X$ is obtained from X by adding a new variable symbol of sort s for each sort s with $X_s = \emptyset$. Now pick an arbitrary variable symbol $x^0_s \in X_s$ for each $s \in S$. The key step is to make the (Σ, Γ)-algebra $\mathcal{T}_{\Sigma,\Gamma}(X')$ into a $(\Sigma^{\otimes}, \Gamma^{\otimes})$-algebra by defining $r_{s',s}$: $\mathcal{T}_{\Sigma,\Gamma}(X')_{s'} \to \mathcal{T}_{\Sigma,\Gamma}(X')_s$ to be the function that sends $[t] \in \mathcal{T}_{\Sigma,\Gamma}(X')_s$ to $[t]$, and otherwise sends it to x^0_s. It is now easy to see that the retract equations are satisfied. Thus the freeness of $\mathcal{T}_{\Sigma^{\otimes},\Gamma^{\otimes}}(X)$ implies that the natural inclusion $X' \to \mathcal{T}_{\Sigma,\Gamma}(X')$ induces a unique $\Sigma^{\otimes}$-homomorphism q: $\mathcal{T}_{\Sigma^{\otimes},\Gamma^{\otimes}}(X') \to \mathcal{T}_{\Sigma,\Gamma}(X')$ such that $q \circ \psi_{X'}$ is the identity. Therefore $\psi_{X'}$ is injective. $\square$

4 Reduction to Many-Sorted Algebra

This section reduces OSA to conditional MSA, thus providing a systematic way to import OSA analogues of known MSA results. The difference is essentially one of viewpoint; mathematically, it is an "equivalence of categories" (this notion is defined below). This result also implies that MSA rewriting can be used as the operational semantics of a logical programming language based on order-sorted algebra (as in OBJ2 [14, 15]); see [23] for details. Next, we relate OSA and MSA equational satisfaction, and get less direct proofs of the existence of initial and free order-sorted algebras for conditional equations than those in Section 3.2 above. We also lift the Birkhoff Variety Theorem and the McKinsey-Malcev Quasivariety Theorem from MSA to OSA.

4.1 Reduction Theorem

The basic idea is to provide for each locally filtered order-sorted signature Σ a corresponding many-sorted signature $\Sigma^{\#}$ with a set J of $\Sigma^{\#}$-equations such that being an order-sorted Σ-algebra is "essentially the same" (i.e., up to isomorphism) as being a many-sorted $\Sigma^{\#}$-algebra satisfying J.

Given a locally filtered order-sorted signature Σ with sort poset $(S, \leq)$, the corresponding $\Sigma^{\#}$ has the same sort set S, has an operation symbol $\sigma_{w,s} \in \Sigma^{\#}_{w,s}$ for each $\sigma \in \Sigma_{w,s}$ (including constants, where $w = \lambda$), and has additional operation symbols $c_{s,s'} \in \Sigma^{\#}_{s,s'}$ whenever $s \leq s'$ in S, called **inclusion** operations. The conditional equations in J are the following (omitting the obvious quantifier and sort information):

1. (identity) $c_{s,s}(x) = x$, for each $s \in S$;
2. (injectivity) $x = y$ <u>if</u> $c_{s,s'}(x) = c_{s,s'}(y)$, for each $s \leq s'$ in S;
3. (transitivity) $c_{s',s''}(c_{s,s'}(x)) = c_{s,s''}(x)$, for each $s \leq s' \leq s''$ in S;
4. (homomorphism) whenever σ: $s1...sn \to s$ and σ: $s'1...s'n \to s'$ are in Σ with $si \leq s'i$ and therefore (by monotonicity) $s \leq s'$ in S, then

$$c_{s,s'}(\sigma_{s1...sn,s}(x_1, ..., x_n)) = \sigma_{s'1...s'n,s'}(c_{s1,s'1}(x_1), ..., c_{sn,s'n}(x_n)).$$

(Note that the injectivity equation is conditional.)

We can view an order-sorted Σ-algebra A as a many-sorted $\Sigma^\#$-algebra $A^\#$ by letting $A^\#_s = A_s$ for each $s \in S$, with $A^\#_{c_{s,s'}}$ the inclusion $A^\#_s \subseteq A^\#_{s'}$ for each $s \leq s'$ and with $A^\#_{\sigma_{w,s}} = A_\sigma$: $A_w \to A_s$ for $\sigma \in \Sigma_{w,s}$. Then $A^\#$ satisfies J by construction. Moreover, this construction of $A^\#$ from A extends naturally to homomorphisms, since an order-sorted Σ-homomorphism f: $A \to B$ is also a many-sorted $\Sigma^\#$-homomorphism $f^\#$: $A^\# \to B^\#$ with $f^\#_s = f_s$: $A^\#_s \to B^\#_s$. This follows because the operations $\sigma_{w,s}$ satisfy condition (1) of Definition 2.8 by construction, while for the operations $c_{s,s'}$ this is just condition (2) for f to be an order-sorted homomorphism. In this way we get a functor

$$(_)^\#\text{: } \mathbf{OSAlg}_\Sigma \to \mathbf{Alg}_{\Sigma^\#,J}$$

where $\mathbf{Alg}_{\Sigma^\#,J}$ is the category of many-sorted $\Sigma^\#$-algebras satisfying J. The Reduction Theorem below shows that this functor is an equivalence of categories.

Our proof of the Reduction Theorem needs some facts about filtered colimits of sets. A **filtered diagram** of sets is a functor D: $(S, \leq) \to \mathbf{Set}$ where $(S, \leq)$ is a filtered poset; i.e., D is a collection of sets $\{D_s \mid s \in S\}$ together with functions $d_{s,s'}$: $D_s \to D_{s'}$ for each $s \leq s'$ in $(S, \leq)$, with $d_{s,s}$ the identity on D_s for each s, and such that $d_{s,s''} = d_{s',s''} \circ d_{s,s'}$ whenever $s \leq s' \leq s''$. The **colimit** of such a filtered diagram D, written $colim(D)$, can be computed as a quotient of the coproduct $\bigsqcup_{s \in S} D_s$ (which we represent as the disjoint union $\bigcup_{s \in S} D_s \times \{s\}$) by the equivalence relation $\equiv$ defined by $(a, s) \equiv (a', s')$ iff for some $s'' \geq s, s'$ in S, $d_{s,s''}(a) = d_{s',s''}(a')$. Reflexivity and symmetry of the relation $\equiv$ are obvious, and transitivity follows from filtration. For each D_s there is a map j_s: $D_s \to colim(D)$ defined as the composition of the coproduct injection $D_s \to \bigsqcup_{s \in S} D_s$ with the natural projection into equivalence classes $\bigsqcup_{s \in S} D_s \to colim(D)$, and the j_s commute with the $d_{s,s'}$ in the natural way by construction. Moreover, one can now check that $colim(D)$ with the maps j_s has the following universal property of a colimit in **Set** of the diagram D: given maps $\{f_s$: $D_s \to A \mid s \in S\}$ such that $f_s = f_{s'} \circ d_{s,s'}$ whenever $s \leq s'$, then there is a unique map f: $colim(D) \to A$ such that $f \circ j_s = f_s$ for each $s \in X$. We need the following result about this construction:

Lemma 4.1 If all the $d_{s,s'}$ of a filtered diagram D: $(S, \leq) \to \mathbf{Set}$ are injective, then the j_s are also injective.

Proof: $(a, s) \equiv (a', s)$ iff $d_{s,s'}(a) = d_{s,s'}(a')$ for some $s' \geq s$ iff (since the $d_{s,s'}$ are injective) $a = a'$. □

Theorem 4.2 (<u>Reduction</u>) Given a coherent order-sorted signature Σ, then the functor $(_)^\#$: $\mathbf{OSAlg}_\Sigma \to \mathbf{Alg}_{\Sigma^\#,J}$ is an equivalence of categories, in the sense that there is another functor $(_)^\bullet$: $\mathbf{Alg}_{\Sigma^\#,J} \to \mathbf{OSAlg}_\Sigma$ such that for each A in $\mathbf{OSAlg}_\Sigma$ and B in $\mathbf{Alg}_{\Sigma^\#,J}$ there ~~are isomorphisms $A \simeq A^{\#\bullet}$ and $B \simeq B^{\bullet\#}$~~ that are natural[12] in A and B, respectively.

[12] The condition for an isomorphism to be natural is spelled out in the body of this proof; see also [50], Theorem IV.4.

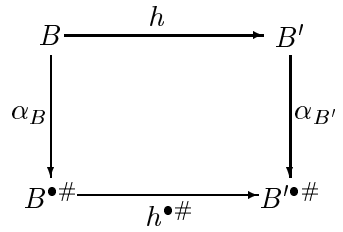
Proof: Given B in $\mathbf{Alg}_{\Sigma^\#,J}$ we define $B^\bullet$ as follows: First, notice that, for each connected component C of S, the sets $\{B_s \mid s \in C\}$ together with their $c_{s,s'}$ form a filtered diagram, and the maps j_s into the filtered colimit $B^\bullet_C$ are injective by Lemma 4.1. Now define $B^\bullet_s = j_s(B_s)$, and given $\sigma \in \Gamma_{s1\ldots sn}$ define $B^\bullet_\sigma$: $B^\bullet_{s1\ldots sn} \to B^\bullet_s$ by

$$B^\bullet_\sigma(j_{s1}(b_1), ..., j_{sn}(b_n)) = j_s(B_\sigma^{s1\ldots sn,s}(b_1, ..., b_n)).$$

Checking that $B^\bullet$ in fact satisfies the conditions of an order-sorted algebra is an exercise in the use of the equations J and the commutation of the j_s with the $c_{s,s'}$.

This construction becomes a functor as follows: first notice that given a connected component C of S and given h: $A \to B$ in $\mathbf{Alg}_{\Sigma^\#,J}$ the maps h_s constitute a natural transformation between two diagrams on the poset C, and therefore they induce a map $h^\bullet_C$: $A^\bullet_C \to B^\bullet_C$ between their colimits: on elements, the map $h^\bullet_C$ is defined by $h^\bullet_C([(a,s)]) = [(h_s(a), s)]$. Therefore, we can define maps $h^\bullet_s$ by restricting $h^\bullet_C$ to domain $A^\bullet_s$ and codomain $B^\bullet_s$. It follows from the definitions of $(_)^\#$ and $(_)^\bullet$ that for any order-sorted Σ-algebra A one has $A^{\#\bullet} \simeq A$; indeed, in this case we can compute the colimits $A^{\#\bullet}_C$ as unions $\bigcup_{s\in C} A_s$ and get an actual equality $A^{\#\bullet} = A$.

By using the equations in J it is also easy to check that the bijections j_s: $B_s \to B^{\bullet\#}_s$ define an isomorphism α_B: $B \simeq B^{\bullet\#}$. We now have to show that the isomorphism α_B is natural in B. This just means that when B varies over $\mathbf{Alg}_{\Sigma^\#,J}$ the α_B's are compatible with the functor $(_)^{\bullet\#}$, i.e., for any h: $B \to B'$ in $\mathbf{Alg}_{\Sigma^\#,J}$ the diagram



commutes. This follows from the definition of $h^{\bullet\#}$ and is left as an exercise. (The identity $A^{\#\bullet} = A$ that we got computing the colimits involved as unions is already natural in A, since $(_)^{\#\bullet}$ is the identity functor on $\mathbf{OSAlg}_\Sigma$.) □

4.2 Semantic Consequences of the Reduction Theorem

The Reduction Theorem is also useful for lifting other MSA results to OSA. Because an equivalence of categories preserves initial objects (for example, by the general result that an equivalence of categories preserves colimits, e.g., [50], Theorem V.5.1), the Reduction Theorem implies that $(_)^\#$ sends any initial order-sorted Σ-algebra to an initial $(\Sigma^\#, J)$-algebra whenever Σ is coherent, and so we get the isomorphism $\mathcal{T}^\#_\Sigma \simeq T_{\Sigma^\#,J}$. Similarly, when Σ is a (not necessarily regular) locally filtered signature, $(_)^\bullet$ sends the initial $(\Sigma^\#, J)$-algebra $T_{\Sigma^\#,J}$ to an initial order-sorted algebra, because $(_)^\bullet$ is an equivalence of categories. Thus initial order-sorted algebras exist even when Σ is not regular (of course, there is an isomorphism $T^\bullet_{\Sigma^\#,J} \simeq \mathcal{T}_\Sigma$ when Σ is coherent). By the equivalence of categories, the existence of an initial order-sorted algebra now follows directly from the well-known existence of many-sorted initial algebras for conditional equations. However, the explicit construction of $\mathcal{T}_\Sigma$ given in Theorem 2.12 when Σ is regular is fairly simple, helps to develop intuitions about OSA, and does not require local filtering.

Corollary 4.3 (Initiality II) Given a locally filtered order-sorted signature Σ with sorts S, and an S-sorted set X disjoint from Σ, then $T^\bullet_{\Sigma^\#,J}$ is an initial order-sorted Σ-algebra and

$(T_{\Sigma^\#,J}(X))^\bullet$ is a free order-sorted Σ-algebra on X; if Σ is coherent then $\mathcal{T}_\Sigma^\#$ is an initial $(\Sigma^\#, J)$-algebra and $\mathcal{T}_\Sigma(X)^\#$ is a free $(\Sigma^\#, J)$-algebra on X. $\square$

This corollary could also be obtained by noticing that there is a (right adjoint) forgetful functor U: $\mathbf{Alg}_{\Sigma^\#,J} \to \mathbf{Set}^S$ (where $\mathbf{Set}^S$ is the category of S-sorted sets) with $U(B) = \{B_s \mid s \in S\}$ and using the facts that an equivalence of categories is an adjoint and that the composition of adjoints is an adjoint (see [50], Theorems IV.8.1 and IV.4.1).

This corollary is useful in connection with parsing order sorted terms, through the unique $\Sigma^\#$-homomorphism from the initial $\Sigma^\#$-algebra, h: $T_{\Sigma^\#} \to \mathcal{T}_\Sigma^\#$ or, if we want terms with variables, h_X: $T_{\Sigma^\#}(X) \to \mathcal{T}_\Sigma(X)^\#$. The set $P(t) = \{t' \in T_{\Sigma^\#}(X) \mid h_X(t') = t\}$ for $t \in \mathcal{T}_\Sigma(X)$ is the set of all **disambiguated parses** of t as a $\Sigma^\#$-term; let $P(t)_s$ denote the set of parses of t of sort s, i.e., $P(t) \cap T_{\Sigma^\#}(X)_s$. Proposition 2.10 showed that for Σ coherent, there is a least sort s with $t \in \mathcal{T}_{\Sigma,s}$ and therefore with $P(t)_s$ nonempty; this sort s was denoted $LS(t)$. For Γ a set of order-sorted equations, conditional or not, let $P(\Gamma)$ denote the set of all possible parses for each equation in Γ.

In fact, the construction of Proposition 2.10 can be adapted as follows to find the least sort parse $LP(t)$ of t: First, for x a variable symbol, let $LP(x) = x$; next, for $t = \sigma(t1...tn)$ with $n \geq 0$ and with $si = LS(ti)$, let $\langle w, s\rangle$ be the least pair such that $s1...sn \leq w$ and $\sigma \in \Sigma_{w,s}$ (which exists because Σ is regular); then $LP(t) = \sigma_{w,s}(LP(t1), ..., LP(tn))$.

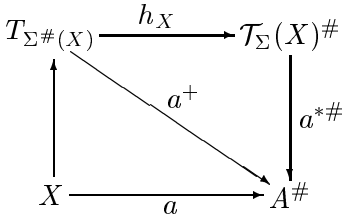
The results of this section are also useful in reducing the satisfaction of equations in OSA to the satisfaction of equations in MSA. The main theorem is:

Theorem 4.4 (Satisfaction) For Σ a coherent order-sorted signature:

(1) A Σ-algebra A satisfies a conditional equation $(\forall X)\ t = t'$ if C iff the $\Sigma^\#$-algebra $A^\#$ satisfies any conditional equation (say of sort s) $(\forall X)\ t_1 = t_1'$ if C_1 such that $h_{X,s}(t_1) = t, h_{X,s}(t_1') = t'$, and $h_X(C_1) = C$.

(2) Conversely, a $(\Sigma^\#, J)$-algebra B satisfies a conditional equation $(\forall X)\ t_1 = t_1'$ if C_1 (of sort s) iff the order-sorted algebra $B^\bullet$ satisfies the conditional equation $(\forall X)\ h_X(t_1) = h_X(t_1')$ if $h_X(C_1)$.

Proof: For any assignment a: $X \to A$, let a^+: $T_{\Sigma^\#(X)} \to A^\#$ and a^*: $\mathcal{T}_\Sigma(X) \to A$ denote the unique homomorphisms induced by a. By definition of satisfaction, to prove (1) it is enough to show that for any $t \in \mathcal{T}_\Sigma(X)$ and $t_1 \in T_{\Sigma^\#}(X)_s$ such that $h_{X,s}(t_1) = t$, one has $a_s^+(t_1) = a^*(t)$. This follows from the initiality of $T_{\Sigma^\#(X)}$ by noting that the diagram



gives $a^{*\#}(h_X(x)) = a^{*\#}(x) = a(x) = a^+(x)$, and thus $a^{*\#} \circ h_X$ equals the homomorphism a^+: $T_{\Sigma^\#(X)} \to A^\#$. Next, we reduce the proof of (2) to the proof just given for (1) by noticing that since B is isomorphic to $B^{\bullet\#}$, it satisfies exactly the same equations as $B^{\bullet\#}$ and (using (1)) $B^{\bullet\#}$ satisfies an equation $(\forall X)\ t_1 = t_1'$ iff $B^\bullet$ satisfies the equation $(\forall X)\ h_{X,s}(t_1) = h_{X,s}(t_1')$. $\square$

This theorem shows that for Γ a set of conditional order-sorted Σ-equations and for $P(\Gamma)$ the set of all possible parses of the equations in Γ as conditional $\Sigma^\#$-equations, the functor $(_)^\#$ restricts as expected:

Corollary 4.5 For Σ a coherent signature and Γ a set of conditional Σ-equations, there is an equivalence of categories

$$(_)^{\#}\colon \mathbf{OSAlg}_{\Sigma,\Gamma} \to \mathbf{Alg}_{\Sigma^{\#},J\cup P(\Gamma)}.$$

□

As before, this means that initial algebras, and more generally free algebras, are preserved by the equivalence of categories. Therefore we can prove in a different way, without appeal to order-sorted deduction, the existence of initial and free order-sorted algebras.

Corollary 4.6 (Initiality III) Given a coherent signature Σ, the class $\mathbf{OSAlg}_{\Sigma,\Gamma}$ of Σ-algebras satisfying a set Γ of conditional equations has an initial algebra, and for any variable set X, also a free (Σ,Γ)-algebra over X. In particular, $(T_{\Sigma^{\#},J\cup P(\Gamma)})^{\bullet}$ is an initial (Σ,Γ)-algebra, and $(T_{\Sigma^{\#},J\cup P(\Gamma)}(X))^{\bullet}$ is a free (Σ,Γ)-algebra on X. □

Two important consequences of the Satisfaction Theorem are order-sorted versions of the McKinsey-Malcev Quasivariety and the Birkhoff Variety Theorems. Since the case when the set of sorts S is infinite requires some additional developments (for which see [25]), we treat the case of a finite set of sorts. The MSA McKinsey-Malcev Theorem states that a class of many-sorted algebras is definable by conditional equations iff it is closed under products, subalgebras, and filtered colimits (for example, see [38] 63.3, where the statement is one-sorted; note that our formulation considers limits and colimits up to isomorphism, so we do not need closure under isomorphisms). The Birkhoff Variety Theorem [1] characterizes classes of algebras definable by unconditional equations as those classes closed under products, subalgebras, and homomorphic images (Birkhoff's original formulation was one-sorted; see [42] for the first many-sorted formulation, and [25] for a corrected statement regarding quantification of variables and a discussion of infinitely many sorts). Our aim is to use the equivalence of categories to lift these two theorems from MSA to OSA. However, first we need to *relativize* the MSA McKinsey-Malcev and Birkhoff Theorems to a subclass defined by conditional equations, due to the presence of the conditional $\Sigma^{\#}$-equations J.

First some notation: For $\mathcal{C}$ a class of order-sorted Σ-algebras, let $P(\mathcal{C})$, $S(\mathcal{C})$, $H(\mathcal{C})$, $F(\mathcal{C})$ denote the closure of $\mathcal{C}$ under products, subalgebras, homomorphic images, and filtered colimits, respectively. Similarly, for $\mathcal{C}_1$ a class of many-sorted $\Sigma^{\#}$-algebras, let $P'(\mathcal{C}_1)$, $S'(\mathcal{C}_1)$, $H'(\mathcal{C}_1)$, $F'(\mathcal{C}_1)$ denote the corresponding many-sorted closures.

Lemma 4.7 Given a many-sorted signature Ω with a finite sort set, a set for Γ_1 of conditional Ω-equations, and a class $\mathcal{C}_1$ of algebras contained in $\mathbf{Alg}_{\Omega,\Gamma_1}$ then the following hold:

(1) $\mathcal{C}_1$ is of the form $\mathbf{Alg}_{\Omega,\Gamma_1\cup\Gamma_2}$ for some set Γ_2 of conditional equations iff it is closed in $\mathbf{Alg}_{\Omega,\Gamma_1}$ under products, subalgebras and filtered colimits.

(2) $\mathcal{C}_1$ is of the form $\mathbf{Alg}_{\Omega,\Gamma_1\cup\Gamma_2}$ for some set Γ_2 of unconditional equations iff it is closed in $\mathbf{Alg}_{\Omega,\Gamma_1}$ under products, subalgebras and homomorphic images.

Proof: The first statement follows from the well known (and easily shown) fact that classes of equations and classes of algebras form a Galois connection, and the closures under products, subalgebras, and filtered colimits of any class in $\mathbf{Alg}_{\Omega,\Gamma_1}$ and in $\mathbf{Alg}_{\Omega}$ coincide precisely by virtue of the McKinsey-Malcev Theorem.

The second statement follows by remarking that $\mathbf{Alg}_{\Omega,\Gamma_1}$ is closed under products and subalgebras, so that those two closures coincide in $\mathbf{Alg}_{\Omega,\Gamma_1}$ and in $\mathbf{Alg}_{\Omega}$. The closure

under homomorphic images of $\mathcal{C}_1$ in $\mathbf{Alg}_{\Omega,\Gamma_1}$ is just the intersection $H'(\mathcal{C}_1) \cap \mathbf{Alg}_{\Omega,\Gamma_1}$. Since $\mathcal{C}_1$ is assumed closed under products and subalgebras, and since the closure under homomorphic images of a class closed under products and subalgebras is also so closed, the Birkhoff Variety Theorem implies that $H'(\mathcal{C}_1)$ is of the form $\mathbf{Alg}_{\Omega,\Gamma_2}$ for some set Γ_2 of unconditional equations, and so we have $\mathcal{C}_1 = \mathbf{Alg}_{\Omega,\Gamma_1} \cap \mathbf{Alg}_{\Omega,\Gamma_2} = \mathbf{Alg}_{\Omega,\Gamma_1 \cup \Gamma_2}$ as desired. The converse is now easy. □

Corollary 4.8 (McKinsey-Malcev Quasivariety and Birkhoff Variety) For $(S, \leq, \Sigma)$ a coherent order-sorted signature with S finite:

- A class of order-sorted Σ-algebras is definable by some set of conditional equations Γ (i.e., is of the form $\mathbf{OSAlg}_{\Sigma,\Gamma}$ for some set Γ of conditional equations) iff it is closed under products[13], subalgebras, and filtered colimits.
- A class of order-sorted Σ-algebras is definable by some set of (unconditional) equations Γ (i.e., is of the form $\mathbf{OSAlg}_{\Sigma,\Gamma}$ for some Γ of unconditional equations) iff it is closed under products, subalgebras, and homomorphic images.

Proof: Notice that by the Satisfaction Theorem, any class of order-sorted algebras of the form $\mathbf{OSAlg}_{\Sigma,\Gamma}$ for Γ a set of conditional equations, can be written as $(\mathbf{Alg}_{\Sigma^\#,J \cup P(\Gamma)})^\bullet$. Similarly, for Γ_1 a set of conditional $\Sigma^\#$-equations, one has $(\mathbf{Alg}_{\Sigma^\#,J \cup \Gamma_1})^\bullet = \mathbf{OSAlg}_{\Sigma,h_X(\Gamma_1)}$ (where X is a set of variables that contains all those declared in the equations of Γ_1). This means that a class of order-sorted Σ-algebras $\mathcal{C}$ is definable by conditional equations (respectively, unconditional equations) iff it is of the form $(\mathbf{Alg}_{\Sigma^\#,J \cup \Gamma_1})^\bullet$ for Γ_1 some set of conditional (respectively, unconditional) $\Sigma^\#$-equations. Note also that if $\mathcal{C}_1$ is a class of many-sorted algebras contained in $\mathbf{Alg}_{\Sigma^\#,J}$ and closed under isomorphisms, then $(\mathcal{C}_1)^\bullet$ is also closed under isomorphisms; in particular, equationally definable classes of order-sorted Σ-algebras are closed under isomorphisms (this was the motivation for defining coherent signatures). Now consider the following identities that hold for $\mathcal{C}_1$ a class of many-sorted algebras contained in $\mathbf{Alg}_{\Sigma^\#,J}$ and closed under isomorphisms:

(1) $P((\mathcal{C}_1)^\bullet) = (P'(\mathcal{C}_1))^\bullet$
(2) $S((\mathcal{C}_1)^\bullet) = (S'(\mathcal{C}_1))^\bullet$
(3) $H((\mathcal{C}_1)^\bullet) = (H'(\mathcal{C}_1))^\bullet$
(4) $F((\mathcal{C}_1)^\bullet) = (F'(\mathcal{C}_1))^\bullet$.

Since equivalences of categories preserve all limits and colimits, (1) and (4) are immediate. (2) and (3) follow from $\mathcal{C}_1$ (and thus $(\mathcal{C}_1)^\bullet$) being closed under isomorphisms, by remarking that the functors $(_)^\#$ and $(_)^\bullet$ both preserve injections and surjections. The OSA McKinsey-Malcev Theorem now follows from Lemma 4.7 and (1), (2), (4), while the OSA Birkhoff Theorem follows from Lemma 4.7 and (1), (2), (3). □

5 Variations on the Theme

Many different ways to define order-sorted algebra have appeared in the literature. However, most are less general than our approach; for example, they may fail to admit many-sorted algebra as a special case, or to provide a semantic account of overloading.

[13] I.e., products $\prod_i A_i$ of families $\{A_i \mid i \in I\}$ over arbitrary index sets I.

5.1 Preregularity

Let us begin with a variation of our own invention, a weakening of regularity that is needed for the discussions which follow:

Definition 5.1 An order-sorted signature Σ is **preregular** iff given $w0 \leq w1$ in S^* and given σ in $\Sigma_{w1,s1}$ there is a least sort $s \in S$ such that $w0 \leq w$ and $\sigma \in \Sigma_{w,s}$ for some $w \in S^*$; we call s the **least sort** of σ with arguments (arity) **over** $w0$, and denote it $LS(\sigma, w0)$. Notice that $\Sigma(X)$ is preregular if Σ is. $\square$

Proposition 5.2 The following are equivalent[14] for an order-sorted signature Σ:

1. Σ is preregular.
2. Each $t \in \mathcal{T}_\Sigma$ has a least $s \in S$ such that $t \in \mathcal{T}_{\Sigma,s}$ called the **least sort** of t and denoted $LS(t)$.
3. Given $S' \subseteq S$ and a variable set X that is disjoint from Σ, then $\bigcap_{s \in S'} \mathcal{T}_\Sigma(X)_s = \bigcup_{s' \leq S'} \mathcal{T}_\Sigma(X)_{s'}$ (where $s' \leq S'$ means that $s' \leq s''$ for all $s'' \in S'$).

Proof:
(1)⇒(2) may be proved essentially the same way as Proposition 2.10.
(2)⇒(3): Since preregularity is preserved by adding constants, we need only consider ground terms, and since the opposite containment is obvious, it is enough to show that $\bigcap_{s \in S'} \mathcal{T}_{\Sigma,s} \subseteq \bigcup_{s' \leq S'} \mathcal{T}_{\Sigma,s'}$. For any $t \in \bigcap_{s \in S'} \mathcal{T}_{\Sigma,s}$ we have LS(t)≤S′; thus, $t \in \bigcup_{s' \leq S'} \mathcal{T}_{\Sigma,s'}$ as desired.
(3)⇒(1): Suppose that Σ is not preregular. Then there are $w0$ and σ such that σ is in $\Sigma_{w1,s1}$ with $w0 \leq w1$ for some $w1 \in S^*$ but $LS(\sigma, w0)$ does not exist. Let $w0 = s1...sn$, and let X consist of the variables $x1, ..., xn$ of sorts $s1, ..., sn$. Then the set S' of all possible sorts for the term $\sigma(x1, ..., xn)$ is such that any s' with $\sigma \in \Sigma_{w',s'}$ and $w0 \leq w'$ is in S' and any s'' in S' is of the form $s'' \geq s'$ for one such s'; thus, the set S' cannot have a least element. Since $\sigma(x1, ..., xn)$ belongs to $\bigcap_{s \in S'} \mathcal{T}_\Sigma(X)_s$ and by hypothesis we have $\bigcap_{s \in S'} \mathcal{T}_\Sigma(X)_s = \bigcup_{s' \leq S'} \mathcal{T}_\Sigma(X)_{s'}$ we can conclude that S' has a least element, which is a contradiction. $\square$

The least parse $LP(t)$ of a term t discussed in Section 4.2 also generalizes to preregular signatures.

5.2 Related Work

The approach to order-sorted algebra given in this paper generalizes the one given in [23], and differs from others in the literature [16, 67, 71, 72]. This section gives a precise comparison of our approach with these others, and concludes that the approaches are close enough that they can simulate each other; on the other hand, it also concludes that there are substantial advantages, both in generality and in the pragmatics of language design, that support our choice. Our main goals in choosing definitions have been:

- To be as general and simple as reasonably possible.
- To insure that MSA is a special case of OSA.
- To give a semantic account of overloading.

[14] We first proved this result assuming that the poset S of sorts satisfied the descending chain condition; we thank Gert Smolka for pointing out that this restriction is unnecessary.

All authors seem to agree on the notion of order-sorted signature (except perhaps for an inessential restriction in [16]). However, significant differences arise in the notions of algebra and homomorphism. From more to less general we have the following:

1. $\mathbf{OSAlg}_\Sigma$ is the notion given in this paper, which in particular involves the following monotonicity condition in the definition of order-sorted algebra:

   (2) $\sigma \in \Sigma_{w1,s1} \cap \Sigma_{w2,s2}$ and $w1 \leq w2$ imply that $A_\sigma^{w1,s1}(a) = A_\sigma^{w2,s2}(a)$ for all $a \in A_{w1}$.

2. $\mathbf{OSAlg}'_\Sigma$ replaces our condition (2) by the condition

   (2′) if $\sigma \in \Sigma_{w1,s1} \cap \Sigma_{w2,s2}$ and if there is a $w0 \leq w1, w2$, then $A_\sigma^{w1,s1}(a) = A_\sigma^{w2,s2}(a)$ for all $a \in A_{w0}$.

   The definition of homomorphism is exactly the same in these two cases.

3. $\mathbf{OSAlg}''_\Sigma$ is the category proposed by [16, 67, 71, 72]. It replaces condition (2) by

   (2″) if $\sigma \in \Sigma_{w1,s1} \cap \Sigma_{w2,s2}$ and if $a \in A^{w1} \cap A^{w2}$, then $A_\sigma^{w1,s1}(a) = A_\sigma^{w2,s2}(a)$.

   The notion of homomorphism $f\colon A \to B$ adds to ours the requirement

   (H) if $a \in A_s \cap A_{s'}$ then $f_s(a) = f_{s'}(a)$.

The differences in generality are reflected by inclusions of categories,

$$\mathbf{OSAlg}''_\Sigma \subseteq \mathbf{OSAlg}'_\Sigma \subseteq \mathbf{OSAlg}_\Sigma$$

where the inclusion $\mathbf{OSAlg}'_\Sigma \subseteq \mathbf{OSAlg}_\Sigma$ is full, whereas the inclusion $\mathbf{OSAlg}''_\Sigma \subseteq \mathbf{OSAlg}_\Sigma$ is not full in general (i.e., there are homomorphisms in our sense that are not homomorphisms in $\mathbf{OSAlg}''_\Sigma$). The discussion below will show that:

- If Σ is regular then $\mathbf{OSAlg}'_\Sigma = \mathbf{OSAlg}_\Sigma$.
- If Σ is preregular then $\mathcal{T}_\Sigma$ is initial in $\mathbf{OSAlg}'_\Sigma$.
- Any preregular signature Σ can be extended to a regular signature Σ' such that $\mathbf{OSAlg}'_\Sigma = \mathbf{OSAlg}_{\Sigma'}$.

Thus the difference between $\mathbf{OSAlg}'_\Sigma$ and $\mathbf{OSAlg}_\Sigma$ is not very substantial and, since regularity is nicer than preregularity, the main parts of this paper stick to regularity.

Condition (2) may seem surprisingly general, because it admits some possibly unexpected behavior. For example, consider (S, Σ) where $S = \{s1, s2, s3\}$ with $s1 \leq s2, s3$, where $a \in \Sigma_{\lambda,s1}$ and $\sigma \in \Sigma_{s2,s2} \cap \Sigma_{s3,s3}$. Then there are order-sorted Σ-algebras A such that

$A_\sigma^{s2,s2}(a) \neq A_\sigma^{s3,s3}(a)$.

For example, one such algebra has $A_{s1} = \{a\}, A_{s2} = \{a, b\}, A_{s3} = \{a, c\}$ with

$A_\sigma^{s2,s2}(a) = A_\sigma^{s2,s2}(b) = b$

and

$A_\sigma^{s3,s3}(a) = A_\sigma^{s3,s3}(b) = c$.

Condition (2′) excludes this kind of behavior, but condition (2) is technically easier to work with, as well as more general; moreover, it is needed for one of the main results of this paper, Theorem 4.2.

Although preregularity may seem very natural, it fails to ensure the equivalence of conditions (2) and (2′) in the definition of order-sorted algebra. For example, consider a signature Σ with sorts $S = \{s0, s1, s2, s3\}$, subsort relations $s0 \leq s1, s2$, and operations σ: $s1 \to s3$ and σ: $s2 \to s3$. Then Σ is preregular, but the order-sorted algebra $\mathbf{N}$ with $\mathbf{N}_{si} = \mathbf{N}$, the natural numbers, for $i = 0, 1, 2, 3$, and with $\mathbf{N}_\sigma$: $\mathbf{N}_{s1} \to \mathbf{N}_{s3}$ the identity function and $\mathbf{N}_\sigma$: $\mathbf{N}_{s2} \to \mathbf{N}_{s3}$ the constant function mapping all the natural numbers to 0, fails to satisfy condition (2′). One can rule out such bizarre models by accepting only algebras in the subcategory $\mathbf{OSAlg}'_\Sigma$ of $\mathbf{OSAlg}_\Sigma$ containing algebras that satisfy condition (2′). Since Lemma 5.4 below shows that any preregular signature can be extended to a regular signature and, since regular signatures ensure condition (2′), this paper emphasizes regularity and the simpler, more general condition (2). Moreover, we have

Fact 5.3 If Σ is a regular order-sorted signature, then a Σ-algebra A satisfies condition (2) iff it satisfies condition (2′).

Proof: Clearly (2′) implies (2). Conversely, assume that A satisfies (2), let $\sigma \in \Sigma_{w1,s1} \cap \Sigma_{w2,s2}$ and let $w0 \leq w1, w2$. Then there is a least $\langle w, s\rangle$ with $\sigma \in \Sigma_{w,s}$ and $w0 \leq w$. In particular, $\langle w, s\rangle \leq \langle w1, s1\rangle$,$\langle w2, s2\rangle$. Therefore, $A_\sigma^{w1,s1}$ and $A_\sigma^{w2,s2}$ are equal to $A_\sigma^{w,s}$ on A_w. Thus, if $a \in A_{w0}$ then also $a \in A_w$ and $A_\sigma^{w1,s1}(a) = A_\sigma^{w2,s2}(a)$. □

Lemma 5.4 Given a preregular signature Σ, there is a regular signature Σ' on the same sort poset such that $\Sigma \subseteq \Sigma'$ and there is an isomorphism of categories $\mathbf{OSAlg}'_\Sigma \cong \mathbf{OSAlg}_{\Sigma'}$.

Proof: Let Σ' be the signature containing Σ (with the same sort poset S) and for each $w \in S^*$ such that $\sigma \in \Sigma_{w',s'}$ for some $w' \geq w$ a new operation σ: $w \to LS(\sigma, w)$. Since Σ satisfies the monotonicity condition, and $w \leq w'$ implies $LS(\sigma, w) \leq LS(\sigma, w')$ whenever this is defined, the signature Σ' also satisfies the monotonicity condition. Also, for each $w \in S^*$ such that $\sigma \in \Sigma_{w',s'}$ for some $w' \geq w$ the least rank for σ with arity greater than or equal to w is precisely $\langle w, LS(\sigma, w)\rangle$. Therefore Σ' is regular.

The functor $\mathbf{OSAlg}_{\Sigma'} \to \mathbf{OSAlg}'_\Sigma$ of the claimed isomorphism just forgets about the new operations introduced in Σ', noting that condition (2′) is satisfied because Σ' is regular. Showing that there is an inverse functor is tantamount to showing that each A in $\mathbf{OSAlg}'_\Sigma$ can be extended in a unique way to an algebra A' in $\mathbf{OSAlg}_{\Sigma'}$ identical with A for operations in Σ in such a way that if f: $A \to B$ is in $\mathbf{OSAlg}'_\Sigma$ then f: $A' \to B'$ is in $\mathbf{OSAlg}_{\Sigma'}$. Since for each new operation σ: $w \to LS(\sigma, w)$ in Σ' there is an operation σ: $w' \to LS(\sigma, w)$ in Σ with $w \leq w'$, the extension A', if it exists, must clearly be unique and then Σ-homomorphisms must preserve the new operations since these are just restrictions of already existing operations. But existence of A' is guaranteed by restriction of the already existing operations, precisely by condition (2′). □

Corollary 5.5 For a preregular signature Σ the algebra $\mathcal{T}_\Sigma$ is initial in the full subcategory $\mathbf{OSAlg}'_\Sigma$ of $\mathbf{OSAlg}_\Sigma$ defined by those algebras satisfying condition (2′).

Proof: This can be proved directly, by minor modifications of the proof of Theorem 2.12, but it follows more abstractly from the isomorphism of categories $\mathbf{OSAlg}'_\Sigma \cong \mathbf{OSAlg}_{\Sigma'}$ that maps $\mathcal{T}_\Sigma$ to $\mathcal{T}_{\Sigma'}$, since isomorphisms of categories preserve all limits and colimits and in particular preserve initial objects. □

A nice property of term algebras is that they automatically satisfy condition (2″); moreover, in the smaller category $\mathbf{OSAlg}''_\Sigma$ the term algebra $\mathcal{T}_\Sigma$ is initial for *any* order-sorted signature Σ. However, there are good pragmatic reasons to require regularity in any case. Poigné [67] perceived regularity does not actually disappear in the category $\mathbf{OSAlg}''_\Sigma$; it is hidden in condition (2″) in a sense to be made precise below. However, the category $\mathbf{OSAlg}''_\Sigma$ has some serious drawbacks, including the following:

- Condition ($2''$) rules out the convenient flexibility of *ad hoc* polymorphism. For example, one cannot have an algebra in which 0 and 1 are both Booleans and naturals, and in which + is both addition of naturals and exclusive or of Booleans.

- Conditions ($2''$) and (H) radically exclude many-sorted algebra as a particular case of order-sorted algebra. In a many-sorted algebra, two different sorts may have elements in common, but homomorphisms may map the same element to different images depending on the sort.

This lack of compatability between the many-sorted and order-sorted approaches associated with $\mathbf{OSAlg}''_\Sigma$ is unfortunate, since order-sorted logic is in principle a refinement of many-sorted logic, and since the previous literature on abstract data types has, almost entirely, been developed in the many-sorted framework.

It is also unfortunate that overloading is so severely limited in this approach, because *ad hoc* polymorphism is such a pervasive and important part of ordinary mathematical notation that it would be a great pity, either to entirely rule it out in the design of programming languages, or to relegate it to the realm of "mere syntax," without the backing of a proper semantic theory, so that one cannot know in advance whether or not some proposed feature might work. Discussions about overloading are difficult, and sometimes even acrimonious, for languages as diverse as Ada [13] and Haskell [44], precisely because of the lack of an underlying semantic basis for these discussions.

We also wish to mention that requiring signatures to be coherent allows a very simple and flexible treatment of equality, since we can always assume that t and t' have the same sort whenever they appear in an equation[15] $t = t'$ by going to a common supersort. By contrast, Smolka [71] introduces special equality predicates of the form $=_{s,s'}$ and requires closure under certain properties of such predicates (so-called "balanced" signatures) in order to obtain a completeness theorem. This seems somewhat unnatural.

A one-sorted "universe" view of order-sorted algebra lurks within conditions ($2''$) and (H). Defining $A = \bigcup_{s \in S} A_s$ to be the "universe", then condition ($2''$) is equivalent to the existence for each operation σ with n arguments of a *partial* operation A_σ: $A^n \to A$ whose domain of definition is the union of the A^w such that σ: $w \to s$ in Σ satisfies appropriate sort conditions for the results. Similarly, condition (H) is equivalent to the existence of a set-theoretic function between universes that preserves sorts and operations. Therefore, one way to reconcile our view with that of [16, 67, 71] is to make the universe explicit. This has also the advantage of showing how an order-sorted "universe" view can easily be embedded into an unsorted view where one gets for free (in both the categorical and the pragmatic senses!) informative error messages for ill typed expressions that take the form of terms whose only sort is the entire universe. The idea is very simple. Take *any* order-sorted signature Σ and extend it to asignature Σ^u by adding to it a new sort u such that $s \leq u$ for any old sort s, and also adding operations σ: $u^n \to u$ for all σ: $s1...sn \to s$ in Σ (but note that Σ^u need not be regular when Σ is). We then have:

Theorem 5.6

- $\mathbf{OSAlg}''_{\Sigma^u} = \mathbf{OSAlg}_{\Sigma^u}$ and in particular, $\mathcal{T}_{\Sigma^u}$ is the initial algebra for all three categories.

- The forgetful functor $(_|_\Sigma)$: $\mathbf{OSAlg}_{\Sigma^u} \to \mathbf{OSAlg}_\Sigma$ that forgets about the universe sort lands inside $\mathbf{OSAlg}''_\Sigma$ and sends one term algebra to the other, i.e., $\mathcal{T}_{\Sigma^u}|_\Sigma = \mathcal{T}_\Sigma$.

[15] Recall that we require t and t' to lie in the same connected component, since we do not consider equations across different components meaningful. However, even this restriction could be dropped by adding a universal sort.

- There is a functor $(_^u)$: $\mathbf{OSAlg}''_\Sigma \to \mathbf{OSAlg}_{\Sigma^u}$ left adjoint to $(_|_\Sigma)$ with a natural (unit) identity $A = A^u|_\Sigma$ and with a very simple description, namely $A^u_\sigma(x1, ..., xn) =$ if $xi \in A_{si}$ and $\sigma \in \Sigma_{s1...sn,s}$ then $A_\sigma(x1, ..., xn)$ else the term $\sigma(x1, ..., xn)$ of sort u.

□

There is another way of relating the two different approaches to order-sorted algebra that has the advantage of making explicit in what sense the regularity assumption is hidden in conditions (2″) and (H). This has also been noticed by Poigné [67], although his statement of the facts seems to be inaccurate because he claims a full subcategory inclusion rather than an isomorphism of categories. The idea is to complete the sort poset S by finite intersections into a poset $I(S)$: The elements of $I(S)$ can be represented as finite expressions $s1\&...\&sn$ for $s1, ..., sn \in S$ and with $s1\&...\&sn \leq s'1\&...\&s'm$ iff for each $s'j$ there is an si such that $si \leq s'j$; and of course, two representations $s1\&...\&sn$ and $s'1\&...\&s'm$ are equal iff $s1\&...\&sn \leq s'1\&...\&s'm$ and $s1\&...\&sn \geq s'1\&...\&s'm$. For a general justification of why this construction of $I(S)$ works and makes the inclusion map $S \to I(S)$ universal, see for example Corollary 3.2 (dualized) of [53].

We can extend an arbitrary order-sorted signature Σ on S to a regular signature $I(\Sigma)$ on $I(S)$ if Σ satisfies the following reasonable finiteness condition[16]: for any σ in Σ having n arguments, and for any word $w0$ of length n in $I(S)^*$, the set $\{w \in S^* \mid \sigma \in \Sigma_{w,s}$ and $w0 \leq w\}$ has a finite set of minimal elements, say $w1, ..., wn$, with σ: $wi \to si$. When such a set is nonempty, we introduce in $I(\Sigma)$ an operation σ: $w0 \to s1\&...\&sn$. This makes $I(\Sigma)$ regular by construction. Now notice that every algebra A in $\mathbf{OSAlg}''_\Sigma$ can be extended to an $I(\Sigma)$-algebra $I(A)$ by defining $I(A)_{s1\&...\&sn} = A_{s1} \cap ... \cap A_{sn}$ with operations extended to intersection sorts in the natural way, i.e., suppose that we have σ: $w0 \to s1\&...\&sn$ as above, obtained from σ: $wi \to si$. Then $a \in A^{w0}$ implies that $a \in A^{wi}$ and by condition (2″), $A_\sigma(a)$ is uniquely defined and belongs to A_{si} for each $i = 1, ..., n$ and therefore to $A_{s1\&...\&sn}$. Since the homomorphisms h: $A \to B$ in $\mathbf{OSAlg}''_\Sigma$ are functions on the universes that preserve the sorts and the operations, they also preserve intersection of sorts. In other words, there is a functor I: $\mathbf{OSAlg}''_\Sigma \to \mathbf{OSAlg}_{I(\Sigma)}$ that is faithful and injective on objects and preserves initial algebras; I is also full, since many-sorted functions that agree on intersections glue together to give a function on the universes. Therefore, we can regard $\mathbf{OSAlg}''_\Sigma$ as a full subcategory of $\mathbf{OSAlg}_{I(\Sigma)}$. However, the category $\mathbf{OSAlg}_{I(\Sigma)}$ can have other objects B such that there is a proper inclusion $B_{s1\&...\&sn} \subset B_{s1} \cap ... \cap B_{sn}$ rather than an equality. Actually, $\mathbf{OSAlg}''_\Sigma$ can be nicely axiomatized by *sort constraints* of the form

```
as x : s1 &...& sn if x : s1 and ...and x : sn .
```

Further details on sort constraints must wait for Part II of this paper; however, see [23] for a very brief introduction. In summary, we have

Theorem 5.7 The functor I: $\mathbf{OSAlg}''_\Sigma \to \mathbf{OSAlg}_{I(\Sigma)}$ is full, faithful and injective on objects, and therefore makes $\mathbf{OSAlg}''_\Sigma$ isomorphic to a full subcategory of $\mathbf{OSAlg}_{I(\Sigma)}$. Moreover, I preserves initial algebras, i.e., $I(\mathcal{T}_\Sigma) = \mathcal{T}_{I(\Sigma)}$. □

Pragmatically, it is very helpful to have a least sort for each term t in an order-sorted term algebra. This makes the task of parsing much easier and also supports good programming and specification practice. Our experience with many examples indicates that this very natural property is generally satisfied in practice, and moreover, nonsatisfaction is often connected with conceptual errors. Of course, it is also easy to check this condition

[16] We could actually do it without assuming this condition, but then infinite intersections of sorts would need to be added.

syntactically. The above subcategory inclusion tells us that in a sense, regularity is always present, but we prefer to make it explicit, since this gives a much simpler approach to the syntactic aspects of order-sorted algebra that any programming language based on these ideas must necessarily address. Moreover, as already mentioned, our choice is the only one that makes the logic a natural extension of many-sorted logic. For all these reasons, as well as for its being simpler and more general, we prefer our approach to the alternatives in [16, 67, 71, 72]. Another reason that has been implicit in our choice, and therefore should also be mentioned, is that our approach is intimately connected with the Cartesian algebraic theories of categorical logic and (with the addition of sort constraints) it actually gives a very convenient way to specify Cartesian theories that avoids many of their shortcomings; this will also be explained in Part II of this paper.

5.2.1 Summary

We may summarize the above discussion with the following points:

- There is basic agreement among all authors about the concept of order-sorted signature; also, the approaches in [16, 67, 71, 72] are all equivalent (except perhaps for an inessential restriction in [16]).
- Our approach is more general in the sense that, for each signature, the algebras and homomorphism of the alternative approaches form a subcategory of our algebras and homomorphisms.
- Only our approach provides a natural extension of many-sorted algebra as the particular case where the sort poset has the discrete order.
- Only our approach permits the convenient flexibility of *ad hoc* polymorphism.
- All approaches can be reconciled, yielding identical categories, by adding a universe sort and extending the operation symbols at the universe level.
- The approach of [16, 67, 71, 72] has the advantage that term algebras are initial in general, whereas we must require that each term have a least sort; however, initial algebras exist in our approach even without this requirement, as shown by Initiality Theorem II (Theorem 4.3 of Section 4.2). The requirement that a least sort exist for each term is implicit in the other approaches in a sense made explicit by a full subcategory embedding. We believe that the least sort requirement is very natural, and that it supports simpler implementations and better programming practice.

5.3 Further Literature

There is by now such a vast amount of related work that we can hardly do more than cite examples almost at random, including the following:

1. Implementations of inheritance in Simula [12] as further developed in Smalltalk [37] and other object-oriented languages.
2. Overloading and subtypes in Ada [13].
3. The theory of (higher order) polymorphism as developed in [58], [68], [8] and [55], among many others.
4. There has been recent work on adding subtypes to higher order calculi [7, 2].

5. Work on "classified algebras" [78] and on "multi-target operation" algebra [39].

6. Work on the semantics of natural and artificial languages, including: [45], which shows how Montague grammar [59] (a formal system for natural language semantics) can be treated with a version of initial algebra semantics with subsorts; [33], which uses error algebras to define programming languages (and thus compilers); and [41], which uses partial algebras to give a semantics for subscripted variables.

7. There is some explicit theory of multiple inheritance in the context of object-oriented programming, including [6] and [76], and we have ourselves applied order-sorted algebra to this problem [30].

8. There is also some work giving operational semantics for subsorts by rewriting, e.g., [11] and [80]; [23] and [48] give details of two different operational semantics that implement precisely the framework given in this paper.

9. Peter Mosses has generalized order-sorted algebra to "unified algebra" [61], which treats elements and subsorts in a uniform way, and thus can handle non-determinism in an algebraic setting. Mosses developed this formalism to support his "action semantics," an algebraic approach to denotational semantics [60].

There are many interesting relationships among these papers: for example, [16] follows [18] in using signatures Σ that are "fully overloaded" in the sense that if $\sigma\colon w \to s$ is in Σ and $w' \leq w$ and $s \leq s'$, then $\sigma\colon w' \to s'$ is in Σ. Our weaker notion of regular signature is intended to capture *ad hoc* polymorphism. Reynolds has subsequently abandoned the algebraic approach of [69], since (he says) it fails to handle the higher-order case. However, higher-order abstract data types have been treated by [64], even with a notion of subsort; see [66] for some corrections to [64]. In fact, the approach of [69] can be seen as arising from taking the so-called tensor product of one algebraic theory with another that consists entirely of subsort inclusions.

The extremes to which one might be driven by the difficulties of partial algebras are illustrated in [41], which models a state change by a change of algebra, and thus models a computation by a sequence of algebras. The "classified algebra" of [78] seems to be a version of OSA, and the "multi-target operation" approach of [39] combines aspects of the partial algebra and the explicit error sort approaches.

There is also now much interesting work on unification for order-sorted algebra, including [11] and [80], who discuss algorithms for unification, and [79], who argues for the utility of subsorts in connection with resolution and paramodulation. [18] gives a systematic treatment of order sorted unification that is consistent with the present paper, and includes a linear time unification algorithm for signatures satisfying some simple conditions.

Kamin and Archer [47] argue that total algebras are unsuitable for treating errors, for reasons like the following:

- the error messages from various abstractions that use (say) the integers, `Int`, cannot be kept separate;

- you have to specify all the error behavior of a module in advance of implementing it;

- and thus, to show correctness of an implementation, all this behavior must also be verified.

None of these objections is valid against the full power of OSA. The first objection is met in an elegant and simple manner by permitting each different abstraction that uses `Int` to have its own supersort of `Int` containing its own error behavior; these supersorts need have

no intersection outside of `Int`. The force of the second and third objections arise from the fact that the error behavior of an abstraction is often determined by the context in which you want to use it. OSA again saves the day, although some concepts not discussed in this paper are needed: the notion of behavioral equivalence of abstract machines [24, 57] can be slightly generalized to consider only certain designated subsorts, e.g., those that exclude the error messages; behavior outside these subsorts is not specified, and thus need not be verified. The method is flexible enough to permit specifying error messages when required by a problem; for example, in specifying a compiler, one might well want to require that certain specific error messages are produced for certain kinds of erroneous input. Kamin and Archer [47] also argue that error features like the finite bound of a stack or array should not be specified, but should be determined by the implementation; but we think this is wrong, since one often wants to specify that *at least* a certain amount of storage must be available. The "implementation" with no storage capacity at all is not useful. These issues are discussed in more detail in [57]. It is perhaps worth emphasizing that OSA can be used in connection with both abstract machines (which have internal states) and data constraints, which together give much more power for applications than we have been able to illustrate in the present paper.

A A Number Hierarchy

This appendix illustrates the expressiveness of order-sorted algebra by constructing the number hierarchy from scratch, all the way from the naturals to the quaternions with rational coefficients. Figure 3 displays the highly nontrivial sort structure of this example.

The actual OBJ3 code consists of modules `NAT`, `INT`, `RAT`, `CPX-RAT` and `QUAT-RAT`, plus some test cases that use a module `TEST` defining decimal digits as shorthand for the Peano notation with zero and successor given in the code. Since the Peano notation is clumsy and inefficient, OBJ3 provides built in modules `NAT`, `INT` and `RAT` that satisfy the specifications given here, but with efficient implementations of the usual decimal notation. However, the code below does not make any use of these built in data types.

It is worth noting that standard many-sorted algebra *cannot* satisfactory specify an example like this. Since `RAT`, `CPX-RAT` and `QUAT-RAT` are fields, one sinks into the murky water of division by zero, and the resulting code is inevitably embarrassingly complex, or even wrong. By contrast, providing subsorts for nonzero elements makes division by zero a nonproblem. Moreover, subsort polymorphism for the arithmetic operators allows using the same function symbol for operations like addition throughout the hierarchy, as is usual in mathematical notation.

```
---> this file is /users/goguen/obj/num/quat.obj
---> number hierarchy up to the quarternions

obj NAT is sorts Nat NzNat Zero .
  subsorts Zero NzNat < Nat .
  op 0 : -> Zero .
  op s_ : Nat -> NzNat .
  op p_ : NzNat -> Nat .
  op _+_ : Nat Nat -> Nat [assoc comm] .
  op _*_ : Nat Nat -> Nat .
  op _*_ : NzNat NzNat -> NzNat .
  op _>_ : Nat Nat -> Bool .
  op d : Nat Nat -> Nat [comm] .
  op quot : Nat NzNat -> Nat .
```

Figure 3: Sort Structure for the Number Hierarchy

```
  op gcd : NzNat NzNat -> NzNat [comm] .
  vars N M : Nat .
  vars N' M' : NzNat .
  eq p s N = N .
  eq N + 0 = N .
  eq (s N)+(s M) = s s(N + M) .
  eq N * 0 = 0 .
  eq 0 * N = 0 .
  eq (s N)*(s M) = s(N +(M +(N * M))) .
  eq 0 > M = false .
  eq N' > 0 = true .
  eq s N > s M = N > M .
  eq d(0,N) = N .
  eq d(s N, s M) = d(N,M) .
  eq quot(N,M') = if ((N > M')or(N == M')) then s quot(d(N,M'),M')
                     else 0 fi .
  eq gcd(N',M') = if N' == M' then N' else (if N' > M' then
                     gcd(d(N',M'),M') else gcd(N',d(N',M'))fi)fi .
endo

obj INT is sorts Int NzInt .
  protecting NAT .
  subsort Nat < Int .
  subsorts NzNat < NzInt < Int .
  op -_ : Int -> Int .
  op -_ : NzInt -> NzInt .
  op _+_ : Int Int -> Int [assoc comm] .
  op _*_ : Int Int -> Int .
  op _*_ : NzInt NzInt -> NzInt .
  op quot : Int NzInt -> Int .
  op gcd : NzInt NzInt -> NzNat [comm] .
  vars I J : Int .
  vars I' J' : NzInt .
  vars N' M' : NzNat .
  eq - - I = I .
  eq - 0 = 0 .
  eq I + 0 = I .
  eq M' +(- N') = if N' == M' then 0 else
                    (if N' > M' then - d(N',M') else d(N',M')fi)fi .
  eq (- I)+(- J) = -(I + J) .
  eq I * 0 = 0 .
  eq 0 * I = 0 .
  eq I *(- J) = -(I * J) .
  eq (- J)* I = -(I * J) .
  eq quot(0,I') = 0 .
  eq quot(- I',J') = - quot(I',J') .
  eq quot(I',- J') = - quot(I',J') .
  eq gcd(- I',J') = gcd(I',J') .
endo
```

```
obj RAT is sorts Rat NzRat .
  protecting INT .
  subsort Int < Rat .
  subsorts NzInt < NzRat < Rat .
  op _/_ : Rat NzRat -> Rat .
  op _/_ : NzRat NzRat -> NzRat .
  op -_  : Rat -> Rat .
  op -_  : NzRat -> NzRat .
  op _+_ : Rat Rat -> Rat [assoc comm] .
  op _*_ : Rat Rat -> Rat .
  op _*_ : NzRat NzRat -> NzRat .
  vars I' J' : NzInt .
  vars R S : Rat .
  vars R' S' : NzRat .
  eq R /(R' / S') = (R * S')/ R' .
  eq (R / R')/ S' = R /(R' * S') .
  ceq J' / I' = quot(J',gcd(J',I'))/ quot(I',gcd(J',I'))
                 if gcd(J',I') =/= s 0 .
  eq R / s 0 = R .
  eq 0 / R' = 0 .
  eq R /(- R') = (- R)/ R' .
  eq -(R / R') = (- R)/ R' .
  eq R +(S / R') = ((R * R')+ S)/ R' .
  eq R *(S / R') = (R * S)/ R' .
  eq (S / R')* R = (R * S)/ R' .
endo

obj CPX-RAT is sorts Cpx Imag NzImag NzCpx .
  protecting RAT .
  subsort Rat < Cpx .
  subsort NzRat < NzCpx .
  subsorts NzImag < NzCpx Imag < Cpx .
  subsorts Zero < Imag .
  op _i : Rat -> Imag .
  op _i : NzRat -> NzImag .
  op -_ : Cpx -> Cpx .
  op -_ : NzCpx -> NzCpx .
  op _+_ : Cpx Cpx -> Cpx [assoc comm] .
  op _+_ : NzRat NzImag -> NzCpx [assoc comm] .
  op _*_ : Cpx Cpx -> Cpx .
  op _*_ : NzCpx NzCpx -> NzCpx .
  op _/_ : Cpx NzCpx -> Cpx .
  op _# : Cpx -> Cpx .
  op |_|^2 : Cpx -> Rat .
  op |_|^2 : NzCpx -> NzRat .
  vars R S : Rat .
  vars R' R" S' S" : NzRat .
  var A B C : Cpx .
  eq 0 i = 0 .
  eq C + 0 = C .
```

```
  eq (R i)+(S i) = (R + S)i .
  eq -(R' +(S' i)) = (- R')+((- S')i) .
  eq -(S' i) = (- S')i .
  eq R *(S i) = (R * S)i .
  eq (S i)* R = (R * S)i .
  eq (R i)*(S i) = -(R * S) .
  eq C *(A + B) = (C * A)+(C * B) .
  eq (A + B)* C = (C * A)+(C * B) .
  eq R # = R .
  eq (R' +(S' i))# = R' +((- S')i) .
  eq (S' i) # = ((- S') i) .
  eq | C |^2 = C * (C #) .
  eq (S' i)/ R" = (S' / R")i .
  eq (R' +(S' i))/ R" = (R' / R")+((S' / R")i) .
  eq A /(R' i) = A *(((- s 0)/  R')i) .
  eq A /(R" +(R' i)) =
     A *((R" / |(R" +(R' i))|^2)+(((- R')/ |(R" +(R' i))|^2)i)).
endo

obj QUAT-RAT is sorts Quat NzQuat J NzJ .
  protecting CPX-RAT .
  subsorts NzJ Zero < J < Quat .
  subsorts NzCpx < NzQuat Cpx < Quat .
  subsort NzJ < NzQuat .
  op _j : Cpx -> J .
  op _j : NzCpx -> NzJ .
  op -_ : Quat -> Quat .
  op _+_ : Quat Quat -> Quat [assoc comm] .
  op _+_ : Cpx NzJ -> NzQuat [assoc comm] .
  op _*_ : Quat Quat -> Quat .
  op _*_ : NzQuat NzQuat -> NzQuat .
  op _/_ : Quat NzQuat -> Quat .
  op _# : Quat -> Quat .
  op |_|^2 : Quat -> Rat .
  op |_|^2 : NzQuat -> NzRat .
  var O P Q : Quat .
  vars B C : Cpx .
  vars C' : NzCpx .
  eq O j = O .
  eq Q + O = Q .
  eq -(C +(B j)) = (- C)+((- B)j ) .
  eq (C j)+(B j) = (C + B)j .
  eq C *(B j) = (C * B)j .
  eq (B j)* C = (B *(C #))j .
  eq (C j)*(B j) = -(C *(B #)) .
  eq Q *(O + P) = (Q * O)+(Q * P) .
  eq (O + P)* Q = (O * Q)+(P * Q) .
  eq (P + Q)# = (P #)+(Q #) .
  eq (C j)# = (- C)j .
  eq | Q |^2 = Q *(Q #) .
```

```
  eq Q /(C' j) = Q *((s 0 /(- C'))j) .
  eq Q /(C +(C' j)) = Q *(((C #)/ |(C +(C' j))|^2) +
                          (((- C')/ |(C +(C' j))|^2)j)) .
endo

*** now some test cases, preceded by some helpful notation
obj TST is protecting QUAT-RAT .
  ops 1 2 3 4 5 6 7 8 9 : -> NzNat [memo] .
  eq 1 = s 0 .
  eq 2 = s 1 .
  eq 3 = s 2 .
  eq 4 = s 3 .
  eq 5 = s 4 .
  eq 6 = s 5 .
  eq 7 = s 6 .
  eq 8 = s 7 .
  eq 9 = s 8 .
endo

reduce 3 + 2 .
reduce 3 * 2 .
reduce p p 3 .
reduce 4 > 8 .
reduce d(2,8) .
reduce quot(7,2) .
reduce gcd(9,6) .
reduce (- 4)+ 8 .
reduce (- 4)* 2 .
reduce 8 /(- 2) .
reduce (1 / 3)+(4 / 6) .
reduce | 1 +(2 i)|^2 .
reduce |(1 +(3 i))+(1 +((- 2) i))|^2 .
reduce (3 +((3 i)+((- 2) i))) /((2 i)+ 2) .
reduce (2 +((3 i)j)) *((5 i)+(7 j)) .
reduce (1 +((1 i)j))/(2 j) .
```

References

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

[2] Kim Bruce and Guiseppe Longo. A modest model of records. In *Proceedings, Symposium on Logic in Computer Science*, pages 38–50. IEEE Computer Society, 1988.

[3] Rod Burstall and Joseph Goguen. Putting theories together to make specifications. In Raj Reddy, editor, *Proceedings, Fifth International Joint Conference on Artificial Intelligence*, pages 1045–1058. Department of Computer Science, Carnegie-Mellon University, 1977.

[4] Rod Burstall and Joseph Goguen. The semantics of Clear, a specification language. In Dines Bjorner, editor, *Proceedings, 1979 Copenhagen Winter School on Abstract Software Specification*, pages 292–332. Springer, 1980. Lecture Notes in Computer Science, Volume 86; based on

unpublished notes handed out at the Symposium on Algebra and Applications, Stefan Banach Center, Warsaw, Poland.

[5] Rod Burstall, David MacQueen, and Donald Sannella. Hope: an experimental applicative language. In *Proceedings, First LISP Conference*, volume 1, pages 136–143. Stanford University, 1980.

[6] Luca Cardelli. A semantics of multiple inheritance. In Giles Kahn, David MacQueen, and Gordon Plotkin, editors, *Semantics of Data Types*, pages 51–68. Springer, 1984. Lecture Notes in Computer Science, Volume 173.

[7] Luca Cardelli. Structural subtyping and the notion of power type. In *Proceedings, Symposium on Principles of Programming Languages*, pages 70–79. Association for Computing Machinery, 1988.

[8] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, 1985.

[9] William F. Clocksin and Christopher Mellish. *Programming in Prolog.* Springer, 1981.

[10] Alan Colmerauer, H. Kanoui, and M. van Caneghem. Etude et réalisation d'un système Prolog. Technical report, Groupe d'Intelligence Artificielle, U.E.R. de Luminy, Université d'Aix-Marseille II, 1979.

[11] R. J. Cunningham and A. Jeremy Dick. Rewrite systems on a lattice of types. Technical report, Imperial College, Department of Computing, 1983.

[12] Ole-Johan Dahl, B. Myhrhaug, and Kristen Nygaard. The SIMULA 67 common base language. Technical report, Norwegian Computing Center, Oslo, 1970. Publication S–22.

[13] Department of Defense. Reference manual for the Ada programming language. United States Government, Report ANSI/MIL-STD-1815A, 1983.

[14] Kokichi Futatsugi, Joseph Goguen, Jean-Pierre Jouannaud, and José Meseguer. Principles of OBJ2. In Brian Reid, editor, *Proceedings, Twelfth ACM Symposium on Principles of Programming Languages*, pages 52–66. Association for Computing Machinery, 1985.

[15] Kokichi Futatsugi, Joseph Goguen, José Meseguer, and Koji Okada. Parameterized programming in OBJ2. In Robert Balzer, editor, *Proceedings, Ninth International Conference on Software Engineering*, pages 51–60. IEEE Computer Society, March 1987.

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

[17] Joseph Goguen. Abstract errors for abstract data types. In Eric Neuhold, editor, *Proceedings, First IFIP Working Conference on Formal Description of Programming Concepts*, pages 21.1–21.32. MIT, 1977. Also in *Formal Description of Programming Concepts*, Peter Neuhold, Ed., North-Holland, pages 491–522, 1979.

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

[19] Joseph Goguen. Parameterized programming. *Transactions on Software Engineering*, SE-10(5):528–543, September 1984.

[20] Joseph Goguen. One, none, a hundred thousand specification languages. In H.-J. Kugler, editor, *Information Processing '86*, pages 995–1003. Elsevier, 1986. Proceedings of 1986 IFIP Congress.

[21] Joseph Goguen. Higher-order functions considered unnecessary for higher-order programming. In David Turner, editor, *Research Topics in Functional Programming*, pages 309–352. Addison-Wesley, 1990. University of Texas at Austin Year of Programming Series; preliminary version in SRI Technical Report SRI-CSL-88-1, January 1988.

[22] 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, as Report ECS-LFCS-90-106, Computer Science Department, University of Edinburgh, January 1990; an ancestor is "Introducing Institutions," in *Proceedings, Logics of Programming Workshop*, Edward Clarke and Dexter Kozen, editors, Springer Lecture Notes in Computer Science, Volume 164, pages 221-256, 1984.

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

[24] Joseph Goguen and José Meseguer. Universal realization, persistent interconnection and implementation of abstract modules. In M. Nielsen and E.M. Schmidt, editors, *Proceedings, 9th International Conference on Automata, Languages and Programming*, pages 265–281. Springer, 1982. Lecture Notes in Computer Science, Volume 140.

[25] 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.

[26] Joseph Goguen and José Meseguer. Eqlog: Equality, types, and generic modules for logic programming. In Douglas DeGroot and Gary Lindstrom, editors, *Logic Programming: Functions, Relations and Equations*, pages 295–363. Prentice-Hall, 1986. An earlier version appears in *Journal of Logic Programming*, Volume 1, Number 2, pages 179-210, September 1984.

[27] Joseph Goguen and José Meseguer. Remarks on remarks on many-sorted equational logic. *Bulletin of the European Association for Theoretical Computer Science*, 30:66–73, October 1986. Also in *SIGPLAN Notices*, Volume 22, Number 4, pages 41-48, April 1987.

[28] Joseph Goguen and José Meseguer. Models and equality for logical programming. In Hartmut Ehrig, Giorgio Levi, Robert Kowalski, and Ugo Montanari, editors, *Proceedings, 1987 TAPSOFT*, pages 1–22. Springer, 1987. Lecture Notes in Computer Science, Volume 250.

[29] Joseph Goguen and José Meseguer. Order-sorted algebra solves the constructor selector, multiple representation and coercion problems. In *Proceedings, Second Symposium on Logic in Computer Science*, pages 18–29. IEEE Computer Society, 1987. Also Report CSLI-87-92, Center for the Study of Language and Information, Stanford University, March 1987, and revised version to appear in *Information and Computation*.

[30] Joseph Goguen and José Meseguer. Unifying functional, object-oriented and relational programming, with logical semantics. In Bruce Shriver and Peter Wegner, editors, *Research Directions in Object-Oriented Programming*, pages 417–477. MIT, 1987. Preliminary version in *SIGPLAN Notices*, Volume 21, Number 10, pages 153-162, October 1986.

[31] Joseph Goguen and José Meseguer. Software for the Rewrite Rule Machine. In Hideo Aiso and Kazuhiro Fuchi, editors, *Proceedings, International Conference on Fifth Generation Computer Systems 1988*, pages 628–637. Institute for New Generation Computer Technology (ICOT), 1988.

[32] Joseph Goguen, José Meseguer, and David Plaisted. Programming with parameterized abtract objects in OBJ. In Domenico Ferrari, Mario Bolognani, and Joseph Goguen, editors, *Theory and Practice of Software Technology*, pages 163–193. North-Holland, 1983.

[33] Joseph Goguen and Kamran Parsaye-Ghomi. Algebraic denotational semantics using parameterized abstract modules. In J. Diaz and I. Ramos, editors, *Formalizing Programming Concepts*, pages 292–309. Springer, 1981. Lecture Notes in Computer Science, Volume 107.

[34] Joseph Goguen and Joseph Tardo. An introduction to OBJ: A language for writing and testing software specifications. In Marvin Zelkowitz, editor, *Specification of Reliable Software*, pages 170–189. IEEE, 1979. Reprinted in *Software Specification Techniques*, Nehan Gehani and Andrew McGettrick, editors, Addison-Wesley, 1985, pages 391–420.

[35] 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.

[36] Joseph Goguen and Timothy Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, SRI International, Computer Science Lab, August 1988. Revised version to appear with additional authors José Meseguer, Kokichi Futatsugi and Jean-Pierre Jouannaud, in *Applications of Algebraic Specification using OBJ*, edited by Joseph Goguen, Cambridge, 1992.

[37] Adele Goldberg and David Robson. *Smalltalk-80: The Language and its Implementation.* Addison-Wesley, 1983.

[38] George Gratzer. *Universal Algebra.* Springer, 1979.

[39] Gerard Guiho. Multioperator algebras. In *Proceedings, Second Workshop on Theory and Applications of Abstract Data Types.* Universität Passau, Germany, 1983.

[40] Robert Harper, David MacQueen, and Robin Milner. Standard ML. Technical Report ECS-LFCS-86-2, Department of Computer Science, University of Edinburgh, 1986.

[41] R. Hartwig. An algebraic approach to the syntax and semantics of languages with subscripted variables. *Periordica Mathematica Hungarica*, 15(1):61–71, 1984.

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

[43] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the Association for Computing Machinery*, 12(10):576–580, October 1969.

[44] Paul Hudak, Philip Wadler, Arvind, et al. Report on the functional programming language Haskell. Technical Report YALEU/DCS/RR-666, Computer Science Department, Yale University, December 1988. Draft Proposed Standard.

[45] Theo Janssen. *Foundations and Applications of Montague Grammar.* PhD thesis, University of Amsterdam, 1983.

[46] Simon Peyton Jones. *The Implementation of Functional Programming Languages.* Prentice-Hall, 1987.

[47] Samuel Kamin and Mila Archer. Partial implementations of abstract data types: A dissenting view on errors. In Giles Kahn, David MacQueen, and Gordon Plotkin, editors, *Semantics of Data Types*, pages 317–336. Springer, 1984. Lecture Notes in Computer Science, Volume 173.

[48] Claude Kirchner, Hélène Kirchner, and José Meseguer. Operational semantics of OBJ3. In T. Lepistö and Aarturo Salomaa, editors, *Proceedings, 15th International Colloquium on Automata, Languages and Programming, Tampere, Finland, July 11-15, 1988*, pages 287–301. Springer, 1988. Lecture Notes in Computer Science No. 317.

[49] Robert Kowalski. Logic for problem solving. Technical Report DCL Memo 75, Department of Artificial Intelligence, University of Edinburgh, 1974. Also in the Artificial Intelligence Series, North-Holland, 1979.

[50] Saunders Mac Lane. *Categories for the Working Mathematician.* Springer, 1971.

[51] Sany Leinwand, Joseph Goguen, and Timothy Winkler. Cell and ensemble architecture of the Rewrite Rule Machine. In Hideo Aiso and Kazuhiro Fuchi, editors, *Proceedings, International Conference on Fifth Generation Computer Systems 1988*, pages 869–878. Institute for New Generation Computer Technology (ICOT), 1988.

[52] John Lloyd. *Foundations of Logic Programming.* Springer, 1984.

[53] José Meseguer. Order completion monads. *Algebra Universalis*, 16:63–82, 1983.

[54] José Meseguer. General logics. In H.-D. Ebbinghaus et al., editors, *Proceedings, Logic Colloquium, 1987.* North-Holland, 1989.

[55] José Meseguer. Relating models of polymorphism. In *Proceedings, Symposium on Principles of Programming Languages*, pages 228–241. Association for Computing Machinery, 1989. Longer version in Technical Report SRI-CSL-88-13, Computer Science Lab., SRI International, October 1988.

[56] José Meseguer and Joseph Goguen. Deduction with many-sorted rewrite rules. Technical Report CSLI-85-42, Center for the Study of Language and Information, Stanford University, December 1985. To appear in *Theoretical Computer Science*.

[57] 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.

[58] Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348–375, 1978.

[59] Richard Montague. *Formal Philosophy: Selected Papers of Richard Montague*. Yale, 1974. Edited and with an introduction by Richard Thomason.

[60] Peter Mosses. A basic semantic algebra. In Giles Kahn, David MacQueen, and Gordon Plotkin, editors, *Proceedings, International Symposium on the Semantics of Data Types*, pages 87–107. Springer, 1985. Lecture Notes in Computer Science, Volume 173.

[61] Peter Mosses. Unified algebras and institutions. Technical Report DAIMI PB-274, Computer Science Department, Aarhus University, 1989.

[62] Michael O'Donnell. *Equational Logic as a Programming Language*. MIT, 1985.

[63] Richard O'Keefe. Source level tools for logic programming. In *Symposium on Logic Programming*, pages 68–72. IEEE Computer Society, 1985.

[64] Kamran Parsaye-Ghomi. *Higher Order Data Types*. PhD thesis, UCLA, Computer Science Department, January 1982.

[65] David Plaisted. An initial algebra semantics for error presentations. SRI International, Computer Science Laboratory, 1982.

[66] Axel Poigné. On semantic algebras: Higher order structures. Informatik II, Universität Dortmund, 1983.

[67] Axel Poigné. Parameterization for order-sorted algebraic specification. *Journal of Computer and System Sciences*, 40(3):229–268, 1990.

[68] John Reynolds. Towards a theory of type structure. In *Colloquium sur la Programmation*, pages 408–423. Springer, 1974. Lecture Notes in Computer Science, Volume 19.

[69] John Reynolds. Using category theory to design implicit conversions and generic operators. In Neal Jones, editor, *Semantics Directed Compiler Generation*, pages 211–258. Springer, 1980. Lecture Notes in Computer Science, Volume 94.

[70] Dana Scott and Christopher Strachey. Towards a mathematical semantics for computer languages. In *Proceedings, 21st Symposium on Computers and Automata*, pages 19–46. Polytechnic Institute of Brooklyn, 1971. Also Technical Monograph PRG 6, Oxford University, Programming Research Group.

[71] Gert Smolka. Order-sorted horn logic: Semantics and deduction. Technical Report SEKI Report SR-86-17, Fachbereich Informatik, Universitat Kaiserslautern, 1986.

[72] 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.

[73] Leon Sterling and Ehud Shapiro. *The Art of Prolog*. MIT, 1986.

[74] Joseph Stoy. *Denotational Semantics of Programming Languages: The Scott-Strachey Approach to Programming Language Theory*. MIT, 1977.

[75] Christopher Strachey. Fundamental concepts in programming languages. Lecture Notes from International Summer School in Computer Programming, Copenhagen, 1967.

[76] D. S. Touretzky. *The Mathematics of Inheritance Systems*. PhD thesis, Carnegie-Mellon University, 1984.

[77] David Turner. Miranda: A non-strict functional language with polymorphic types. In Jean-Pierre Jouannaud, editor, *Functional Programming Languages and Computer Architectures*, pages 1–16. Springer, 1985. Lecture Notes in Computer Science, Volume 201.

[78] William Wadge. Classified algebras. Technical Report 46, University of Warwick, October 1982.

[79] Christoph Walther. A many-sorted calculus based on resolution and paramodulation. In *Eighth International Joint Conference on Artificial Intelligence*, pages 882–891. W. Kaufman (Los Altos CA), 1983.

[80] Christoph Walther. A classification of many-sorted unification theories. In *Proceedings, 8th International Conference on Automated Deduction*, pages 525–537. Springer, 1986. Lecture Notes in Computer Science, Volume 230.

Contents

1 Introduction **1**
- 1.1 Type Disciplines . . . 1
  - 1.1.1 Inheritance and Polymorphism . . . 1
  - 1.1.2 Polymorphism is Polymorphic . . . 2
- 1.2 Logical and Operational Semantics . . . 3
- 1.3 Retracts . . . 5
- 1.4 Exceptions and Partial Operations . . . 6
- 1.5 Constructors, Selectors, Multiple Representations and Coercions . . . 6
- 1.6 About this Paper . . . 6
  - 1.6.1 Brief Overview of Subsequent Parts . . . 7
- 1.7 Acknowledgements . . . 7

2 Order-Sorted Algebra **8**
- 2.1 Signatures . . . 8
- 2.2 Algebras . . . 10
- 2.3 Terms . . . 12
- 2.4 Equations . . . 14
- 2.5 Subalgebras, Congruences, Quotients and Products . . . 18

3 Order-Sorted Equational Deduction **21**
- 3.1 The Rules of Order-sorted Equational Deduction . . . 22
- 3.2 Completeness and Initiality Theorems . . . 22
- 3.3 Retracts . . . 24

4 Reduction to Many-Sorted Algebra **26**
- 4.1 Reduction Theorem . . . 26
- 4.2 Semantic Consequences of the Reduction Theorem . . . 28

5 Variations on the Theme **31**
- 5.1 Preregularity . . . 32
- 5.2 Related Work . . . 32
  - 5.2.1 Summary . . . 37
- 5.3 Further Literature . . . 37

A A Number Hierarchy **39**