INFORMATION AND COMPUTATION 103, 114–158 (1993)

Order-Sorted Algebra Solves the Constructor-Selector, Multiple Representation, and Coercion Problems

JOSÉ MESEGUER

SRI International, Menlo Park, California 94025, and Center for the Study of Language and Information, Stanford University, Stanford, California 94305.

AND

JOSEPH A. GOGUEN

Oxford University, Programming Research Group, Oxford OX1 3QD, United Kingdom, and SRI International, Menlo Park, California 94025

Structured data are generally composed from constituent parts by *constructors* and decomposed by *selectors*. We show that the usual many-sorted algebra approach to abstract data types *cannot* capture this simple intuition in a satisfactory way. We also show that order-sorted algebra *does* solve this problem, and many others concerning partially defined, ill-defined, and erroneous expressions, in a simple and natural way. In particular, we show how order-sorted algebra supports and elegant solution to the problems of multiple representations and coercions. The essence of order-sorted algebra is that sorts have *subsorts*, whose semantic interpretation is the *subset* relation on the carriers of algebras.

1. INTRODUCTION

The goal of the research reported in this paper is to significantly expand the expressiveness and utility of abstract data types, both for programming languages, whether imperative or functional, and for specification languages, while preserving a rigorous initial algebra semantics and a simple first order equational proof theory. Although the many-sorted algebra (hereafter, MSA) approach to abstract data types has an attractive theory and many successful applications, it can lead to some very awkward specifications in practice, primarily because of its difficulties in handling erroneous expressions, such as dividing by zero in the rationals, or taking the top of an empty stack. In fact, there is no really satisfying MSA specification for stacks (see, for example, the two-stack specifications in (Ehrig and Mahr, 1985)). This is a special case of what we call the

0890-5401/93 $5.00

constructor–selector problem: for a given constructor, define functions that retrieve it components. This paper shows that the constructor–selector problem is insoluble in MSA, and that it can be solved with order-sorted algebra (hereafter, OSA).

There are many problems where one wants to represent data in more than one way, and convert freely between the representations, using whichever representation is most convenient or efficient. This is the problem of *multiple representations*; for example, think of 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 round them off; this illustrates the problem of *coercions*. The multiple representation problem is a special case of the coercion problem, since the selectors for one representation, when applied to a data item of another, can be thought of as mediated by coercions that change the representation. The difference is that conversions between multiple representations are necessarily reversible. OSA provides an initial algebra semantics for all these constructions.

The essence of OSA is that sorts have *subsorts*, semantically interpreted as subsets among the carriers of an algebra. OSA's rich type system supports many other features, including overloading, and total functions that would otherwise have to be partial by using sort constraints. The original paper on OSA was (Goguen, 1978), but see (Goguen and Meseguer, 1992) for the latest version, which simplifies, extends, and corrects (Goguen, 1978) in many ways. This paper begins with some basic OSA, including order-sorted deduction and a number of examples, using a syntax based on OBJ (Goguen *et al.*, 1988; Goguen *et al.*, 1993), which is a language having an initial OSA semantics for its (executable) "objects" and a loose semantics for (non-executable) "theories."

2. Order-Sorted Algebra

This section introduces the most basic definitions and results of OSA, including signature, algebra, homomorphism, term, the initial algebra construction, equation, satisfaction, subalgebra, congruence, quotient, and the Homomorphism Theorem. For a more detailed treatment, including aspects not treated here, see (Goguen and Meseguer, 1992). Although we give detailed definitions and prove a number of basic results, those few proofs which are omitted may be found in (Goguen and Meseguer, 1992).

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, one for each "sort" $s \in S$; we write $\{A_s \mid s \in S\}$. Similarly, given S-sorted sets A and B, then an S-sorted function $f: A \to B$ is an S-sorted family $f = \{f_s: A_s \to B_s \mid s \in S\}$.

In the order-sorted case, S is a *partially ordered set*, or *poset*; i.e., there is a binary relation $\leqslant$ on S that is reflexive, transitive, and antisymmetric in the sense that $x \leqslant y$ and $y \leqslant x$ implies $x = y$. Every poset also has an associated relation $<$, defined by $x < y$ iff $x \leqslant y$ 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 $s_1 \cdots s_n \leqslant s_1' \cdots s_n'$ iff $s_i \leqslant s_i'$ for $1 \leqslant i \leqslant n$. Similarly, $\leqslant$ extends to pairs $\langle w, s \rangle$ in $S^* \times S$ by $\langle w, s \rangle \leqslant \langle w', s' \rangle$ iff $w \leqslant w'$ and $s \leqslant s'$. (These are the orderings that arise from poset products.)

DEFINITION 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 $(S, \leqslant, \Sigma)$ such that (S, Σ) is a many-sorted signature, $(S, \leqslant)$ is a poset, and the operations satisfy the following *monotonicity condition*:

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

When the sort set S is clear, we write Σ for (S, Σ), and when the poset $(S, \leqslant)$ is clear, we write Σ for $(S, \leqslant, \Sigma)$. 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 $\sigma: w \to s$ for $\sigma \in \Sigma_{w,s}$ to emphasize that σ denotes a function with arity w and sort s. ▮

The monotonicity condition is very natural. In the intended semantics, subsorts are interpreted as subsets and overloaded function symbols related by the subsort relation must agree on their results; therefore, monotonicity means that if the argument sorts $w1$ of and instance $\sigma: w1 \to s1$ are all smaller than those of another instance $\sigma: w2 \to s2$, which at the semantic level is interpreted as subset containment, then $s1$ should be smaller then $s2$, i.e., it should be interpreted as a subset containment in the intended semantics.

2.2 *Algebras*

DEFINITION 2. 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 $A_\sigma: A_w \to A_s$ for each σ in $\Sigma_{w,s}$, where $A_w = A_{s1} \times \cdots \times A_{sn}$ when $w = s1 \cdots sn$ and where A_w is a one point set when $w = \lambda$.

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

(1) $s \leqslant s'$ in S implies $A_s \subseteq A_{s'}$ and

(2) $\sigma \in \Sigma_{w1,s1} \cap \Sigma_{w2,s2}$ and $w1 \leqslant w2$ imply $A_\sigma : A_{w1} \to A_{s1}$ equals $A_\sigma : A_{w2} \to A_{s2}$ 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 $(S, \leqslant)$ is clear, $(S, \leqslant, \Sigma)$-algebras may be called *order-sorted Σ-algebras*. Also, we may write $A_\sigma^{w,s}$ instead of $A_\sigma : A_w \to A_s$. ▮

Many different ways to define order-sorted algebras have appeared in the literature. However, most of them are either less general or else are more complex. As pointed out below, another nice feature of the present definition, not shared by other approaches, is that it contains many-sorted algebra as a special case. (Goguen and Meseguer, 1992) discusses this in much more detail.

Let us consider a very simple example of an order-sorted algebra, the Peano natural numbers in their simplest possible form, i.e., without any functions defined except zero and the successor functions

EXAMPLE 3 (Peano Naturals). We give the order-sorted signature of the natural numbers in Peano notation using the syntax of OBJ. The declaration `NzNat < Nat` indicated that the sort `NzNat` of nonzero naturals is a subsort of the sort `Nat` of naturals. We indicate "mix-fix" notation using the underbar character "_" to mark places where arguments go. Thus, `s_` for the successor function, and `if_then_else_fi` for an if-then-else operator. The first such place argument should have an argument of the first sort in the sort list between the : and −>; the second place should have an argument of the second sort; etc. The keywords `obj...endo` indicate that the initial model is the one intended in the semantics. The name `NAT` is a way of refering to the entire module just defined (called an *object* in the terminology of OBJ) and has in principle nothing to do with the sort `Nat` defined inside the module; e.g., we might just as well have called this object `PEANO`. Initiality and term algebras will be explained in detail later. However, the intuitive meaning for this example is quite clear; it corresponds to an order-sorted algebra **N** whose signature is specified by the object `NAT`, such that $\mathbf{N}_{\texttt{NzNat}}$ consists of the expressions `s0`, `ss0`, `sss0`, etc., and $\mathbf{N}_{\texttt{Nat}}$ is $\mathbf{N}_{\texttt{NzNat}}$ union with the constant `0`:

```
obj NAT is
  sorts NzNat Nat .
  subsorts NzNat < Nat .
  op 0 : -> Nat .
  op s_ : Nat -> NzNat .
endo
```

▮

DEFINITION 4. Let (S, Σ) be a many-sorted signature, and let A and B be (S, Σ)-algebras. Then a (S, Σ)-*homomorphism* $h: A \to B$ is an S-sorted function $h = \{h_s : A_s \to B_s \mid s \in S\}$ satisfying the *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 $w = s1 \ldots sn$ 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)-homomorphisms may be called just many-sorted Σ-homomorphisms.

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

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

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

Since, by definition, every $(S, \leqslant, \Sigma)$-algebra is an (S, Σ)-algebra and every $(S, \leqslant, \Sigma)$-homomorphism is an (S, Σ)-homomorphism, there is a "forgetful" functor from $\mathbf{OSAlg}_\Sigma$ to $\mathbf{Alg}_\Sigma$. Note the slight abuse of language, since Σ denotes two different signatures: an order-sorted signature $(S, \leqslant, \Sigma)$ in $\mathbf{OSAlg}_\Sigma$ and a many-sorted signature (S, Σ) in $\mathbf{Alg}_\Sigma$. Also note that OSA *properly generalizes* MSA, in the sense that any many-sorted (S, Σ)-algebra is an order-sorted $(S, \leqslant, \Sigma)$-algebra for $\leqslant$ the trivial ordering on S with $s \leqslant s'$ 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 $f: A \to B$ 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. The following lemma is easy to prove and is left as an exercise.

LEMMA 5. *An order-sorted* Σ-*homomorphism* $f: A \to B$ *is an isomorphism iff there is an order-sorted* Σ-*homomorphism* $f^{-1}: B \to A$ *such that* $f^{-1} \circ f = 1_A$ *and* $f \circ f^{-1} = 1_B$. ∎

2.3. *Terms*

This subsection shows that terms over regular signatures have a well-defined least sort, and also that the analogue of 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 (Meseguer and Goguen, 1985). 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 $w = s1 \ldots sn$ with $n > 0$, the expression $\sigma(t1 \ldots tn)$ is in $T_{\Sigma,s}$.

Also,

- for $\sigma \in \Sigma_{w,s}$ let $T_\sigma : T_w \to T_s$ send $t1, ..., tn$ to the expression $\sigma(t1 \ldots tn)$.

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

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

Also,

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

Clearly $\mathscr{T}_\Sigma$ is an order-sorted Σ-algebra. Note that $\mathscr{T}_{\Sigma,s}$ is not in general equal to $T_{\Sigma,s}$ or even to $\bigcup_{s' \leqslant s} T_{\Sigma,s'}$. Also note that it is quite possible that $\mathscr{T}_{\Sigma,s} = \varnothing$ for some s, if there are no ground terms of sort s. $\mathscr{T}_\Sigma$ is a kind of order-sorted Herbrand construction, but unfortunately, some authors insist on adding a constant if none is otherwise provided, thus destroying the initiality of this construction.

A given term t in an order-sorted term algebra can have many different sorts. In particular, if t in $\mathscr{T}_\Sigma$ has sort s, then it also has sort s' for any $s' \geqslant s$; 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 $\mathscr{T}_\Sigma$ may fail to be initial, just as in the many-sorted case T_Σ may fail to be initial if Σ is ambiguous. The following simple example shows this anomaly. Note the declaration `subsorts A < B C < D` which declares `A` smaller than `B` and `C`, and `B` and `C` smaller than `D`.

EXAMPLE 6.

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

```
  subsorts A < BC < D .
  op a : -> A .
  op f : B-> B .
  op f : C-> C .
endo
```

The problem is that the terms f(a) has sorts B and C. The order-sorted term algebra for this signature is not initial (see Definition 9 below). To see this, consider the order-sorted algebra R with $R_A = \{a\}$, $R_B = \{a, b\}$, $R_C = \{a, c\}$, $R_D = \{a, b, c\}$, and with $R_f^{B,B}(a) = R_f^{B,B}(b) = b$, $R_f^{C,C}(a) = R_f^{C,C}(c) = c$. The term f(a) has sorts B and C and therefore also sort D. If there were a homomorphism h, then f(a) should be mapped by h_D to both b and c, and therefore no such h exists. ∎

However, this problem disappears for regular signatures as defined below.

DEFINITION 7. An order-sorted signature Σ is *regular* iff given $w0 \leqslant w1$ in S^* and given σ in $\Sigma_{w1,s1}$ there is a least rank $\langle w, s \rangle \in S^* \times S$ such that $w0 \leqslant w$ and $\sigma \in \Sigma_{w,s}$. ∎

PROPOSITION 8. *Given a regular order-sorted signature Σ, for every*[1] *$t \in \mathcal{T}_\Sigma$ there is a least $s \in S$, called the* least sort *of t 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 $t \in \mathcal{T}_\Sigma$ is of 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 $t = (t1 \ldots tn) \in \mathcal{T}_{\Sigma,s}$ of depth $N+1$. Then each ti has depth $\leqslant N$ and therefore has a least sort, say si; let $w0 = s1 \ldots sn$. Then $\sigma \in \Sigma_{w',s'}$ for some w', s' with $s' \leqslant s$ and $w0 \leqslant w'$, and by regularity, there are least w' and s' such that $\sigma \in \Sigma_{w',s'}$ and $w' \geqslant w0$; this least s' is the desired least sort of t. ∎

DEFINITION 9. 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.

THEOREM 10. *Let Σ be a regular order-sorted signature. Then $\mathcal{T}_\Sigma$ is an initial order-sorted Σ-algebra.*

The proof is by induction on the depth of terms; see (Goguen and Meseguer, 1992) for the details.

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

The terms considered above are *gound terms*; i.e., they involve no variables. However, terms with variables can be seen as a special case of ground terms, by enlarging the signature with additional constants that correspond to the variables. We assume that variables come with a given sort, and thus form 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 $(S, \leqslant, \Sigma)$ and an S-sorted variable set X that is disjoint from Σ, we define the order-sorted signature $(S, \leqslant, \Sigma(X))$ by $\Sigma(X)_{\lambda,s} = \Sigma_{\lambda,s} = \Sigma_{\lambda,s} \cup X_s$ and $\Sigma(X)_{w,s} = \Sigma_{w,s}$ for $w \neq \lambda$; note that $\Sigma(X)$ is regular if Σ is. Now we can form $\mathscr{T}_{\Sigma(X)}$ and view it as an order-sorted Σ-algebra by forgetting about the constants in X; let us denote this algebra by $\mathscr{T}_\Sigma(X)$.

An assignment of values in an order-sorted algebra A to variables in a variable set X disjoint from Σ can be extended to terms by the initiality of $\mathscr{T}_\Sigma(X)$. This is entirely analogous to MSA (see (Meseguer and Coguen, 1985)), and is usually stated by saying that $\mathscr{T}_\Sigma(X)$ is the *free* order-sorted Σ-algebra generated by X, in the sense of the following consequence of Theorem 10:

THEOREM 11. *Given a regular order-sorted signature Σ, an order-sorted Σ-algebra A, and an S-sorted function $a: X \to A$, called an* assignment *from X to A, then there is a unique order-sorted Σ-homomorphism $a^*: \mathscr{T}_\Sigma(X) \to A$ that extends a (i.e., such that $a^*(x) = a(x)$ for all x in X).* ∎

2.4. *Equations*

In an MSA equation $t = t'$, we are forced to require that t and t' have the same sort. OSA allows more flexibility. Intuitively, it should be enough that the two terms in the equation share a common supersort. For equational deduction to work properly and for the models to behave well one is naturally led to ask that any two sorts in the same *connected component*[2] have a common supersort. This brings us to the concept of coherent signature.

DEFINITION 12. A poset $(S, \leqslant)$ is *filtered* iff for any two elements s and s' in S there is an element s'' in S such that $s, s' \leqslant s''$. A poset S is *locally filtered* iff each of its connected components is filtered. An order-sorted signature $(S, \leqslant, \Sigma)$ is *locally filtered* iff $(S, \leqslant)$ is locally filtered, and is *coherent* iff it is locally filtered and regular. ∎

[2] Given a poset $(S, \leqslant)$, let $\equiv$ denote the transitive and symmetric closure of $\leqslant$. Then $\equiv$ is an equivalence relation whose equivalence classes are called the *connected components* of $(S, \leqslant)$. For example, the poset of sorts of the `STACK-OF-INT` object in Example 26 has two connected components; one consists of the sorts {`NeStack`, `Stack`}, and the other of the sort `Int` and all its subsorts.

DEFINITION 13, For $(S, \leqslant, \Sigma)$ a coherent 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 $\mathscr{T}_\Sigma(X)$ with $LS(t)$ and $LS(t')$ in the same connected component of $(S, \leqslant)$. We 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 occurring 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.[3]

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 $C = \varnothing$, 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) equations, we let $\mathbf{OSAlg}_{\Sigma,\Gamma}$ denote the category of all (Σ, Γ)-algebras, with all Σ-homomorphisms among them. ∎

2.5. *Subalgebras, Congruences, and Quotients*

DEFINITION 14. 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 $\sigma \in \Sigma_{w,s}$ if $w = s1 \ldots sn$ and if $bi \in B_{si}$ for $i = 1, \ldots, n$, then $A_\sigma(b1, \ldots, bn) \in B_s$; in particular, when $w = \lambda$ then $A_\sigma \in B_s$.

For $(S, \leqslant, \Sigma)$ an order-sorted signature and A 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 $s \leqslant s'$. ∎

FACT 15. *If $f: A \to B$ is an order sorted Σ-homomorphism, then $f(A)$* with $f(A)_s = f_s(A_s)$ *for each $s \in S$ is an order-sorted subalgebra of B called the* image *of f.*

[3] However, the reader should be aware that satisfactions of an equation depends crucially on the variable set [Meseguer & Goguen 1985].

Proof. To check condition (1) of the definition of subalgebra, let $\sigma \in \Sigma_{w,s}$ with $w = s1 \ldots sn$, let $bi \in f(A)_{si}$ for $i = 1, \ldots, n$, and let $ai \in A_{si}$ such that $bi = f_{si}(ai)$ for $i = 1, \ldots, n$. Then $B_\sigma(b1, \ldots, bn) \in f(A)_s$ since $B_\sigma(b1, \ldots, bn) = f_s(A_\sigma(a1, \ldots, a_n))$. 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. ∎

DEFINITION 16. For (S, Σ) a many-sorted signature and A a many-sorted Σ-algebra, a *many-sorted Σ-congruence* $\equiv$ on A is an S-sorted family $\{\equiv_s \mid s \in S\}$ of equivalence relations $\equiv_s$ on A_s such that

(1) given $\sigma \in \Sigma_{w,s}$ for $w = s1 \ldots sn$ and $ai, a'i \in A_{si}$ for $i = 1, \ldots, n$ such that $ai \equiv_{si} a'i$, then $A_\sigma(a1, \ldots, an) \equiv_s A_\sigma(a'1, \ldots, a'n)$.

For $(S, \leqslant, \Sigma)$ 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 $s \leqslant s'$ in S and $a, a' \in A_s$ then $a \equiv_s a'$ iff $a \equiv_{s'} a'$. ∎

PROPOSITION 17. *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 of 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.

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 18. *A* class *$\mathscr{C}$ 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 $\mathscr{C}$; moreover, greatest lower bound is then computed by set intersection.* ∎

DEFINITION 19. Let $f: A \to B$ be a many-sorted Σ-homomorphism. Then the *kernel* of f is an 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 20. *A kernel is a many-sorted congruence. If $f: A \to B$ is an order-sorted Σ-homomorphism, then $\ker(f)$ is an order-sorted Σ-congruence.*

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

$$f_s(A_\sigma(a1, \ldots, an)) = B_\sigma(f_{s1}(a1), \ldots, f_{sn}(an))$$
$$= B_\sigma(f_{s1}(a'1), \ldots, f_{sn}(a'n)) = f_s(A_\sigma(a'1, \ldots, a'n))$$

so that

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

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

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

DEFINITION 21. For $(S, \leqslant, \Sigma)$ 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 $\equiv_C$ and *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 $s'' \geqslant s, s'$. 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 $s'' \geqslant s$, 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 $(A/\equiv)_\sigma([a1], \ldots, [an]) = [A_\sigma(a1, \ldots, an)]$, for $\sigma \in \Sigma_{s1,\ldots,sn,s}$ and $a_i \in A_{s_i}$, which is well defined since $\equiv$ is an order-sorted Σ-congruence. Note that $\ker(q) = \equiv$.

The quotient construction can be generalized by replacing $\equiv$ by an arbitrary S-sorted family R of binary relations R_s on A_s for each $s \in S$. We define the *quotient* of A by R, denoted A/R, as the quotient of A by the

smallest order-sorted Σ-congruence on A containing R. By Proposition 17, such congruence can be expressed as the intersection of all order-sorted congruences containing R in the lattice of congruences. ∎

PROPOSITION 22 (Universal Property of Quotient). *Let Σ be a locally filtered order-sorted signature, A an order-sorted Σ-algebra, and R 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 $u: A/R \to B$ such that $u \circ q = f$.*

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 $u_C: (A/R)_C \to B_C$ such that $u_C \circ q_C = f_C$ with $f_C: A_C \to B_C$ 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, when u_C is restricted to each one of the sorts s in C, the family $\{u_s \mid s \in S\}$ thus obtained is an order-sorted Σ-homomorphism. Property (2) for order-sorted homomorphisms follows by construction. Let $\sigma \in \Sigma_{w,s}$ with $w = s1 \ldots sn$ and let $ai \in A_{si}$ for $i = 1, \ldots, n$. Then (omitting sort qualifications throughout) we have $u((A/R)_\sigma([a1], \ldots, [an])) = u([A_\sigma(a1, \ldots, an)]) = f(A_\sigma(a1, \ldots, an)) = B_\sigma(f(a1), \ldots, f(an)) = B_\sigma(u[a1]), \ldots, u([an]))$. We leave the case $w = \lambda$ for the reader to check. ∎

We note that the universal property characterizes the quotient map uniquely up to isomorphism. Now the following is an easy consequence of Proposition 22:

PROPOSITION 23 (Homomorphism Theorem). *Let Σ be a locally filtered order-sorted signature and let $f: A \to B$ be an order-sorted Σ-homomorphism. Then $A/\ker(f)$ and $f(A)$ are isomorphic order-sorted Σ-algebras.*

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) $u: A/\ker(f) \to f(A)$ such that $u \circ q = f'$. Then u is surjective since f' is, and it remains to show that u is injective. To this end (omitting sort qualifications again), suppose that $u([a1]) = u([a2])$. Then $f(a1) = f(a2)$, so that $[a1] = [a2]$. ∎

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

3. Order-Sorted Equational Deduction

Before turning to the rules of deduction, we consider order-sorted term substitution. Given a coherent order-sorted signature $(S, \leqslant, \Sigma)$ and two S-sorted variable sets X and Y, a *substitution* is an S-sorted map $\theta: X \to \mathscr{T}_\Sigma(Y)$; note that this is the special case of the assignment concept given earlier (Theorem 11) in which the values assigned to the variables are terms. We adopt the convention that the unique order-sorted Σ-homomorphism $\theta^*: \mathscr{T}_\Sigma(X) \to \mathscr{T}_\Sigma(Y)$ 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 are the rules for 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 $\theta, \theta': X \to \mathscr{T}_\Sigma(Y)$ are substitutions such that for each x in X the equation

$$(\forall Y)\, \theta(x) = \theta'(x)$$

is derivable, then given $t \in \mathscr{T}_\Sigma(X)$,

$$(\forall Y)\ \theta(t) = \theta'(t)$$

is also derivable.

(5) *Substitutivity.* If

$$(\forall X) t = t' \text{ if } C$$

is in Γ, and if $\theta: X \to \mathscr{T}_\Sigma(Y)$ 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 $\theta: X \to \mathscr{T}_\Sigma(Y)$ is a substitution, then

$$(\forall Y)\ \theta(t) = \theta(t')$$

is derivable.

These rules 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, order-sorted Horn clause logic is discussed in (Goguen and Meseguer, 1987), and (Goguen and Meseguer, 1986) gives an overview for the many-sorted case.

These rules manipulate equations in which the variables that are universally quantified have been made explicit. This is not necessary for unsorted equational logic, but it is essential for many-sorted and order-sorted equational logic. In fact, the usual rules of unsorted equational deduction are *unsound* when used in a many-sorted or order-sorted context precisely for this reason (Meseguer and Goguen, 1985).

3.2. *Completeness and Initiality Theorems*

The above rules are sound and complete for deriving all the *unconditional* equations that hold in the class of all order-sorted Σ-algebras satisfying Γ. As a corollary, we obtain initial and free order-sorted algebras for a set Γ of conditional equations as quotients of term algebras modulo

the congruence generated by the rules of deduction. Detailed proofs of all these results may be found in (Goguen and Meseguer, 1992).

THEOREM 24 (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* (5) *by rule* (5'). ∎

The proof of the Completeness Theorem makes an essential use of the algebra $\mathcal{T}_{\Sigma,\Gamma}(X)$ whose construction we describe below. The key observation is 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 $\equiv_{\Gamma(X)}$. Then by rules (1)–(3), $\equiv_{\Gamma(X)}$ is an equivalence relation on $\mathcal{T}_\Sigma(X)_s$ for each sort s. Now applying rule (4) to terms t of the form $\sigma(x_1, \ldots, x_n)$ for σ in Σ, we see that $\equiv_{\Gamma(X)}$ is a many-sorted Σ-congruence. Finally, $\equiv_{\Gamma(X)}$ is also an order-sorted Σ-congruence, because property (D) does not depend on s.

The algebra $\mathcal{T}_{\Sigma,\Gamma}(X)$ is now defined as the order-sorted quotient of $\mathcal{T}_\Sigma(X)$ by the congruence $\equiv_{\Gamma(X)}$. The following corollary provides an abstract characterization, up to unique isomorphism, of the algebra $\mathcal{T}_{\Sigma,\Gamma}(X)$.

COROLLARY 25 (Initiality). *Given a coherent order-sorted signature* Σ *and a set* Γ *of conditional* Σ*-equations, then* $\mathcal{T}_{\Sigma,\Gamma}(\varnothing)$ (*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. ∎

This subsection concludes with three OSA specifications in the notation of OBJ. The first adds a predecessor function to the Peano naturals introduced in Example 3; the predecessor function is a very simple example of a *selector* (in this case, its associated *constructor* is the successor function). The initial model denoted by the example is therefore the Peano naturals with a predecessor function. The second example defines stacks of integers. Although stacks are a well known trade example, the problem of taking the top of an empty stack has plagued the algebraic data type literature with many specifications that are either incorrect or too complicated; this is unavoidable with MSA specifications, but it is easily resolved in OSA by

introducing a subsort NeStack of nonempty stacks. The operator push is a constructor, with associated slectors top_ and pop_. The initial model denoted by the third example is the rational numbers. As noted in Section 6.10 of (Ehrig and Mahr, 1986), there is no way to avoid heavy use of hidden functions and error constants in specifying the rational numbers, due to problems such as division by zero; a satisfactory MSA specification of the rationals is not possible. Note that in the RAT example the operations _/_, -_, and _*_ are "overloaded," and that [assoc comm] declares _+_ and _*_ to be associative and commutative. These are mathematically equivalent to the corresponding equations, but also have a useful operational semantics in OBJ.

All three examples contain the keyword protecting followed by the name of an object (NAT in the NAT-PRED example, INT (an object for the integers, which we assume already defined) in the STACK-OF-INT and RAT examples). Intuitively, protecting asserts that "no junk and no confusion" are added to the imported object when it is placed in the context of the new object being defined; that is, no new data is added to the old sorts, and no new identifications are imposed on the old data. We can make this intuitive idea precise by noting that the text of an OBJ object specifies an order-sorted equational theory (Σ, Γ) consisting of an order-sorted signature Σ and a set Γ of Σ-equations. The *mathematical semantics* of such an object is the initial order-sorted algebra[4] $\mathscr{T}_{\Sigma,\Gamma}$. Subobject[4] relations, such as INT being a subobject of RAT, correspond to an inclusion

$$(\Sigma, \Gamma) \subseteq (\Sigma', \Gamma')$$

of the corresponding equational theories. Such an inclusion induces a forgetful functor

$$-|_{\Sigma}: \mathbf{OSAlg}_{\Sigma',\Gamma'} \to \mathbf{OSAlg}_{\Sigma,\Gamma}$$

which regards a Σ'-algebra A as a Σ-algebra $A|_{\Sigma}$ by discarding the sorts and operations of Σ' that lie outside Σ. In particular, the initial (Σ', Γ')-algebra $\mathscr{T}_{\Sigma',\Gamma'}$ is now regarded as a (Σ, Γ)-algebra $\mathscr{T}_{\Sigma',\Gamma'}|_{\Sigma}$; therefore, there is a unique Σ-homomorphism

$$h: \mathscr{T}_{\Sigma,\Gamma} \to \mathscr{T}_{\Sigma',\Gamma'}|_{\Sigma}.$$

The assertion that the subobject relation $(\Sigma, \Gamma) \subseteq (\Sigma', \Gamma')$ is protecting states that this homomorphism h is an isomorphism.

[4] More precisely, the isomorphism class determined by such an algebra, which provides an abstract semantics entirely free of representation details (see [Meseguer & Goguen 1985] for a thorough discussion).

[5] Since executable OBJ modules are called objects, we used the word subobject to denote what in the standard software engineering terminology would be called a *submodule*.

In the examples at hand, the `protecting` assertion makes precise the intuition that the Peano naturals are not disturbed by adding a predecessor function (this is due to the existence of the subsort `NzNat`; otherwise, the naturals would indeed be disturbed by junk such as `p0`), and that the integers are not disturbed when used as values of a stack (again, this is due to the existence of the `NeStack` subsort; otherwise, junk such as `top empty` would be added to the integers), or when extended to the rationals (thanks to the existence of new supersorts that leave the old sorts unchanged; otherwise, new data would have been added to the integers).

EXAMPLE 26 (Predecessor, Stack, and Rationals).

```
obj NAT-PRED is
  protecting NAT .
  op p_ : NzNat -> Nat .
  var N : Nat .
  eq p s N = N .
endo

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

obj RAT is protecting INT .
  sorts NzRat Rat .
  subsorts 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 [assoc comm] .
  op _*_ : NzRat NzRat -> NzRat [assoc comm] .
```

```
  vars R S : Rat .
  vars R' S' T' : NzRat .
  eq R/(R'/S') = (R * S')/R' .
  eq (R/R')/S' = R/(R' * S') .
  eq (R' * T')/(S' * T') = R'/S' .
  eq R/1 = 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' .
endo
```


We note that an operational semantics has been given for OSA, using many-sorted rewriting (Goguen *et al.*, 1985). A simpler operational semantics that performs order-sorted rewriting has subsequently been developed in (Kirchner *et al.*, 1987) and has been implemented in the OBJ3 system; the basic concepts and results of order-sorted rewriting are presented in Section 3.3 below.

Regarding the operational semantics of conditional equations, OBJ3 does not implement conditional equations in the full generality of the definition given in Section 2.4 in which the condition is a finite set C of Σ-equations; it implements only the special case where there is just one equation in the condition set C and it has sort `Bool`, for the Boolean truth values, for which a decision procedure (using associative/commutative matching) is already provided in OBJ. More explicitly, a conditional equation has the form

```
eq t1 = t2 if b.
```

where `b` is of sort `Bool`, and this equation is applied iff `t1` matches and `b` evaluates to `true` using the substitution corresponding to that match. Given the existence of a built-in Boolean-valued binary operator "`_==_`" which evaluates to `true` when its two arguments evaluate to the same value, and given the existence of an "and" operator for the sort `Bool`, assuming that the equations have nice enough properties as rewrite rules, these restrictions on the operational treatment of conditional equations pose no problem in practice.

3.3. *Order-Sorted Rewriting*

Order-sorted rewriting is similar to standard term rewriting, but given that in the order-sorted case the sorts of terms on the left and on the right of an equation need not coincide, some care is needed to exclude rewritings

that would produce ill-formed terms; for example, this can happen by rewriting the term $f(a)$ with an equation $a = b$ involving two constants a and b of respective sorts A and B, with $A < B$, where the (only) rank of the function symbol f is $f: A \to A$. Rewriting with such an equation would yield the term $f(b)$ which is ill-formed. The definition of order-sorted rewrite rule given below excludes this bad behavior altogether; more general definitions can be given, but the present one (called "sort-decreasing rewrite rules" in (Kirchner *et al.*, 1987)) is the most common in practice, and has the nicest properties.

DEFINITION 27. Given a coherent order-sorted signature Σ and terms t, $t' \in \mathcal{T}_\Sigma(X)$, an unquantified Σ-equation $t = t'$ is an *order-sorted rewrite rule* if the set of variables appearing in t' is contained in the set of variables appearing in t and, in addition, for any order-sorted substitution θ we have $LS(\theta(t)) \geq LS(\theta(t'))$. ∎

Given a set $\mathscr{R}$ of order-sorted rewrite rules, we say that a term $u \in \mathcal{T}_\Sigma(Y)$ *rewrites in one step* to a term $v \in \mathcal{T}_\Sigma(Y)$ using $\mathscr{R}$ if there are a rewrite rule $t = t'$ in $\mathscr{R}$, an occurrence[6] α in u, and an order-sorted substitution $\theta: X \to \mathcal{T}_\Sigma(Y)$, for X the variables appearing in t, such that $u/\alpha = \theta(t)$ and $v = u[\alpha \Leftarrow \theta(t')]$. We then write $u \to_{\mathscr{R}} v$ to denote the one-step rewriting relation (we often drop the subscript $\mathscr{R}$ when the set of rules is understood). Similarly, we denote by $u \xrightarrow{*}_{\mathscr{R}} v$ the reflexive and transitive closure of the one-step rewriting relation and then say that u *rewrites* to v using $\mathscr{R}$. For a given $\mathscr{R}$, the relation $u \downarrow v$ holds between two terms in an order-sorted term algebra iff there is a term w such that $u \xrightarrow{*} w$ and $v \xrightarrow{*} w$. A set $\mathscr{R}$ of order-sorted rewrite rules is called *confluent* iff whenever we have $t \xrightarrow{*} u$ and $t \xrightarrow{*} v$, we then have $u \downarrow v$. Similarly, $\mathscr{R}$ is called *locally confluent* iff whenever we have $t \to u$ and $t \to v$, we then have $u \downarrow v$. $\mathscr{R}$ is called *terminating* iff there is no infinite chain of one step rewritings

$$t_0 \to t_1 \to t_2, \ldots \to t_n \to \ldots .$$

The following fact can be proved easily by induction on the number of rewrite steps using the definition of order-sorted rewrite rules.

FACT 28. For $\mathscr{R}$ a set of order-sorted rewrite rules, whenever $t \xrightarrow{*} t'$ *we have* $LS(t) \geq LS(t')$. ∎

The following two facts follow from general properties of rewriting relations (see (Huet, 1980)).

[6] We assume familiarity with the usual way of denoting occurrences in a tree by strings of natural numbers; if α is an occurrence in term u, then we denote by u/α the subterm of u rooted at that occurrence, and we denote by $u[\alpha \Leftarrow w]$ the result of replacing in u the subterm u/α by the term w at occurrence α.

FACT 29. *If a set $\mathscr{R}$ of order-sorted rewrite rules is confluent and terminating, then each term t has a unique term $\text{norm}_{\mathscr{R}}(t)$, called its $\mathscr{R}$-*normal form, *such that $t \xrightarrow{*}_{\mathscr{R}} \text{norm}_{\mathscr{R}}(t)$ and $\text{norm}_{\mathscr{R}}(t)$ cannot be further rewritten using the rules in $\mathscr{R}$.* ∎

FACT 30. *If a set $\mathscr{R}$ of order-sorted rewrite rules is locally confluent and terminating, then it is confluent.* ∎

The key result linking order-sorted rewriting and order-sorted equational deduction is the following theorem proved in (Kirchner *et al.*, 1987), and also appearing in (Smolka *et al.*, 1989):

THEOREM 31. *For $\mathscr{R}$ a set of confluent order-sorted rewrite rules and $u, v \in \mathscr{T}_\Sigma(Y)$, the following are equivalent:*

(i) $u \downarrow v$

(ii) $(\forall Y) u = v$ *is provable from the equations*[7] $\mathscr{R}$ *using order-sorted deduction.*

If in addition $\mathscr{R}$ is terminating, then (i) *and* (ii) *are equivalent to*

(iii) $\text{norm}_{\mathscr{R}}(u) = \text{norm}_{\mathscr{R}}(v)$. ∎

In practice, confluence of a set $\mathscr{R}$ of terminating order-sorted rewrite rules is checked by checking local confluence of its "critical pairs," which intuitively correspond to all the possible ways in which two rewrite rules in $\mathscr{R}$ can overlap. Technically, the key notion is that of an overlap between two rules. To simplify life, we assume that different rules have disjoint sets of variables, and in the case of "self-overlap" of a rule with itself we assume that a copy of the rule with a disjoint set of variables has been added to $\mathscr{R}$. An *overlap* is then a triple $(u = v, \alpha, w = r)$ with $u = v$, $w = r$ rules in $\mathscr{R}$ with disjoint sets of variables and α an occurrence in u such that u/α is not a variable and such that u/α and w are order-sorted unifiable; that is, there exists an order-sorted substitution θ (called a *unifier*) such that $\theta(u/\alpha) = \theta(w)$. In the case when $u = v$ and $w = r$ are "the same equation" up to an isomorphic renaming of variables we rule out the trivial overlap with $\alpha = \lambda$ (the empty string); i.e., we exclude from our definition "self-overlaps" at the top, A *critical pair* of an overlap $(u = v, \alpha, w = r)$ is a pair of the form $(\theta(v), \theta(u[\alpha \Leftarrow r]))$ for θ a unifier of u/α and w. The key result is the following (see (Smolka *et al.*, 1989)):

THEOREM 32. *A set $\mathscr{R}$ of order-sorted rewrite rules is locally confluent iff for all critical pairs (p, q) we have $p \downarrow q$.* ∎

[7] We regard each equation in $\mathscr{R}$ as quantified by the variables appearing in its righthand side.

In general, for order-sorted signatures there does not exist a most general unifier; however, under fairly mild conditions on a coherent signature, there exists a finite set of unifiers, called a *covering*, from which all other unifiers can be obtained by further substitution, and such a set can be effectively computed (Meseguer *et al.*, 1989, Smolka *et al.*, 1989). Under such circumstances, we can decide the confluence of a finite set of terminating order-sorted rewrite tules by restricting attention to those critical pairs whose substitution is in the corresponding covering. We refer to (Smolka *et al.*, 1989, Kirchner *et al.*, 1987; Meseguer *et al.*, 1989) for further details on order-sorted rewriting and order-sorted unification; in particular, (Kirchner *et al.*, 1987) discusses order-sorted rewriting with conditional equations and also order-sorted rewriting modulo axioms.

3.4. *Retracts*

Strong typing, even with overloading, has some disadvantages. For example, the term

```
top pop push(7,push(3,empty))
```

is not well formed according to the syntax of the `STACK-OF-INT` example, since `top`'s arguments should have sort `NeStack`, whereas the term

```
pop push(7,push(3,empty))
```

literally has sort `Stack`, even though we can see that

```
pop push(7,push(3,empty))
```

will evaluate to the nonempty stack `push(3,empty)`. A simple solution is to transform ill-formed Σ-terms into well-formed terms over an extended signature, denoted $\Sigma^{\otimes}$, that has the same sorts as Σ, plus some new operation symbols 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

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

where x is a variable of sort s.

The OBJ3 implementation uses retracts to handle ill-formed OSA terms, such as `top pop push(7,push(3,empty))`, that might become well-formed after reduction, giving them the benefit of the doubt at parse time by filling the gaps between actual sorts and required sorts with retracts. For example, the above term is parsed as

```
top r_Stack,NeStack(pop push(7,push(3,empty)))
```

and since it reduces to 3 under the rules in STACK-OF-INT plus a retract rule, it is vindicated by the reduction. Also, truly erroneous terms reduce to very informative error messages. This kind of runtime type checking, together with the polymorphism provided by subsorts and parameterization, gives most of the syntactic flexibility of untyped languages with all the advantages of strong typing. Moreover, adding retracts is *safe*, since, under extremely general hypotheses, it is a conservative extension (Goguen and Meseguer, 1992). There is also an operational semantics of retracts developed in joint work with Jean-Pierre Jouannaud (Goguen *et al.*, 1985). Such a semantics uses the retract equations as rewrite rules to attempt elminating retracts at runtime.

3.5. *Sort Constraints*

We have seen that subsorts provide a way to treat functions as total that would otherwise be partial or ill-defined; for example, a predecessor function for the NAT-PRED example, or rational division for the RAT example. In this section, we sketch how to further extend this way of dealing with "partial algebras as total algebras" by *constraining* the result of an operator to belong to a certain subsort under certain conditions. A much fuller treatment of this topic will be given in (Meseguer and Goguen, 1993). The key idea is that, instead of constraining the results of an operator $\sigma(x1, ..., xn)$ to always lie in a given subsort, we can constrain a *term* $t(x1, ..., xn)$ to lie in a given subsort, for any assignment of values to the variables $x1, ..., xn$ that satisfies a certain condition; for example, for a bounded stack an operator push can be so constrained if the length of the stack being pushed does not exceed a certain bound (this is called a *conditional sort constraint*).

We formalize conditional sort constraints as quadruples $\langle X, s, t, C \rangle$, where C is a condition (i.e., a finite set of equations). Such a formalization is sufficient for our purposes in this paper; an even more general definition is given in (Meseguer and Goguen, 1993).

DEFINITION 33 (Conditional Sort Constraints). For Σ a coherent signature, a *conditional sort constraint* is a quadruple $\langle X, s, t, C \rangle$ with X a finite S-sorted variable set, t an order-sorted Σ-terms on the variables X such that all the variables in X occur in t, s a sort strictly smaller than $LS(t)$, the lowest sort of t, and C a finite set of Σ-equations involving only variables in X.

An order-sorted algebra A is said to *satisfy* $\langle X, s, t, C \rangle$ iff for every S-sorted assignment $a: X \to A$ satisfying C, the element $a^*(t)$ belongs to A_s.

Some of the examples appearing in this paper involve sort constraints.

Typically, either the term t is of the form f(X1,...,Xn) for which the corresponding *OBJ* syntax is

op-as f:S1...Sn → S for f(X1,...,Xn) if c(X1,...,Xn).

where c(X1,...,Xn) is a term of sort Bool, which corresponds to the sort constraint

⟨{X1:S1,...,Xn:Sn},S,f(Xa,...,Xn),c(X1,...,Xn)=true⟩,

or t is just a variable which is constrained to be in a subsort if a condition is satisfied. This is expressed with the syntax

sort S < S' if c(X).

for c(X) a term of sort Bool, which both declares S as a subsort of S' and asserts the sort constraint

⟨X:S', S, X, c(X) = true⟩.

Sort constraints have not yet been implemented in OBJ. Next, we give an abstract data type that crucially uses a conditional sort constraint:

EXAMPLE 34 (Bounded Stack of Naturals).

```
obj BOUNDED-NAT-STACK is
  protecting NAT .
  sorts NeStack Stack Stack? .
  subsorts NeStack < Stack < Stack? .
  op empty : -> Stack .
  op overflow : ->Stack? .
  op bound : -> Nat .
  op top : NeStack -> Nat .
  op pop : NeStack -> Stack .
  op length : Stack -> Nat .
  op push : Nat Stack -> Stack? .
  op-as push : Nat Stack -> NeStack for push(N,S)
    if length(S) < bound .
  var N : Nat .
  var S : Stack .
  eq push(N,S) = overflow if length(S) == bound .
  eq bound = s s s s s s s s s s s s s s s s s s s s s s s s s s s s s 0 .
  eq length(empty) = 0 .
  eq length(push(N,S)) = s(length(S)) .
  eq top(push(N,S)) = N .
  eq pop(push(N,S)) = S .
endo
```

■

A result that amounts in some sense to a reduction of sort constraints to standard order-sorted algebra is proved in (Meseguer and Goguen, 1993). The result takes the form of a full coreflective subcategory, which permits reducing equational deduction for algebras with sort constraints to standard order-sorted equational deduction; there is an associated construction for the initial algebra satisfying given conditional sort constraints and equations. It is shown that such an initial algebra always exists.

4. Constructors and Selectors

Intuitively, *constructors* create data elements; thus, `push` creates `Stack` elements, `cons` creates (say) `List-of-Bool` elements, and `s` creates `Nat` elements. A constructor generally has associated *selectors* to "pull out" underlying components of the created data element; for example, `top` and `pop` do this for `Stack`, `head` and `tail` for `List-of-Bool`, and `p` for `Nat`. Such selectors should collectively act as an inverse to their constructor, in the sense of satisfying equations like the following (we adopt the usual notation `cons` for the list constructor that takes an element and a list as arguments)

```
eq top push(I,S) = I .
eq pop push(I,S) = S .

eq head(cons(B,L)) = B .
eq tail(cons(B,L)) = L .

eq p s N = N .
```

The problem is that expressions like `top(empty)`, `head(nil)`, and `p 0` are well-formed in MSA, but are intuitively erroneous, since there is no reasonable `Int` value for `top(empty)`, or `Bool` value for `head(nil)`. Thus, one must either make the selectors partial and abandon MSA, or else introduce new "erroneous" elements, say `error`. However, this gives rise to strange expression like `3 + error` and `push(3,push(error, empty))`. A desperate attempt the minimize the (generally infinite) number of erroneous elements by adding a few seemingly innocent equations can easily collapse all elements into a single point (Goguen *et al.*, 1976). In fact, the well-mentioned but hopeless attempt to solve the constructor–selector problem within MSA has produced many inconsistent specifications in the abstract data type literature. As shown in the `STACK-OF-INT` example, introducing a subsort like `NeStack` as the target sort of `push` removes these difficulties. Similarly, subsorts `NeList-of-Bool` < `List-of-Bool` and `NzNat` < `Nat` as targets for `cons` and `s` solve the

problem for lists of Booleans and for natural numbers. Theorem 43 below shows that the approach just sketched solves the constructor-selector problem in general. But first, we need some further concepts.

DEFINITION 35. A Σ-algebra A is *reachable* iff the unique Σ-homomorphism $\mathcal{T}_\Sigma \to A$ is surjective. ▮

DEFINITION 36. Let Σ be a coherent order-sorted signature and let Γ be a set of conditional Σ-equations. Then a *signature of canonical constructors* for Σ with respect to Γ is a coherent subsignature Π of Σ (i.e., $\Pi_{w,s} \subseteq \Sigma_{w,s}$ for all $w \in S^*$ and $s \in S$) with the same poset of sorts and the same least sort assignments for ground terms (i.e., for each $t' \in \mathcal{T}_\Pi$ we have $LS_\Pi(t') = LS_\Sigma(t')$) such that for each $t \in \mathcal{T}_\Sigma$ there is a *unique* $t' \in \mathcal{T}_\Pi$ such that the equation[8] $(\forall \varnothing)\ t = t'$ is derivable from Γ by OSA deduction and, in addition, $LS(t') \leqslant LS(t)$. We call elements of Π *canonical constructors*. ▮

LEMMA 37. *A coherent subsignature Π of Σ with the same poset of sorts and the same least sort assignments for ground terms as Σ is a subsignature of canonical constructors for Σ with respect to Γ iff the unique order-sorted Π-homomorphism $h: \mathcal{T}_\Pi \to \mathcal{T}_{\Sigma,\Gamma}|_\Pi$ is an isomorphism.*

Proof. By construction, the elements of $\mathcal{T}_{\Sigma,\Gamma}$ are equivalence classes of terms modulo the order-sorted congruence $\equiv_{\Gamma(\varnothing)}$ defined by

$$t \equiv_{\Gamma(\varnothing)} t' \text{ iff } (\forall \varnothing)\ t = t' \text{ is derivable from } \Gamma \text{ using rules (1)–(5).}$$

Therefore, the definition of signature of canonical constructors asserts that in each equivalence class $[t] \in (\mathcal{T}_{\Sigma,\Gamma})_s$ there is exactly one representative $t' \in \mathcal{T}_\Pi$ and, in addition, the least sort of t' is smaller than or equal to the least sort of any of the other terms in $[t]$. Therefore, $t' \in \mathcal{T}_{\Pi,s}$ and the unique Π-homomorphism $h: \mathcal{T}_\Pi \to \mathcal{T}_{\Sigma,\Gamma}|_\Pi$ is bijective for each $s \in S$, and thus an isomorphism. Conversely, if h is an isomorphism, h_s is bijective for each $s \in S$; therefore, each term $t \in \mathcal{T}_{\Sigma,s}$ has a unique $t' \in \mathcal{T}_{\Pi,s}$ such that $(\forall \varnothing)\ t = t'$ is derivable from Γ using rules (1)–(5). The term t' is unique not only along the elements of $\mathcal{T}_{\Pi,s}$ but actually among all the elements of $\mathcal{T}_\Pi$. Indeed, suppose that $t'' \in \mathcal{T}_\Pi$ has the same property and let $s' \geqslant s$ be a supersort such that $t, t'' \in \mathcal{T}_{\Sigma,s'}$ (note that such a supersort always exists by coherence). Then we have $t', t' \in \mathcal{T}_{\Pi,s'}$ and both terms must be mapped by $h_{s'}$ to $[t]$; therefore, the injectivity of $h_{s'}$ forces $t' = t''$. Note that $LS(t') \leqslant LS(t)$, since whenever $t \in \mathcal{T}_{\Sigma,s}$ we have $t' \in \mathcal{T}_{\Pi,s}$. ▮

[8] Note that, as we have already noted in our discussion of order-sorted equational deduction, explicit quantification is necessary. In this case, the equation involves ground terms and the set of variables is empty.

The conditions $LS(t') \leqslant LS(t)$ is needed for $h: \mathscr{T}_\Pi \to \mathscr{T}_{\Sigma,\Gamma}|_\Pi$ to be surjective. For example, for the signature Σ with two sorts $s' \leqslant s$ and just two constants, a' of sort s' and a of sort s, and with Γ consisting of the single equation $(\forall \varnothing)\ a = a'$, the subsignature Π with the same sorts and only the constant a induces $\mathscr{T}_\Pi \to \mathscr{T}_{\Sigma,\Gamma}|_\Pi$ not surjective, since we have $\mathscr{T}_{\Pi,s'} = \varnothing$, even though for each term $t \in \mathscr{T}_\Sigma$ there is a unique $t' \in \mathscr{T}_\Pi$ such that $(\forall \varnothing)$ $t = t'$ is derivable from Γ using rules (1)–(5).

Given a signature Π of canonical constructors for Σ with respect to Γ we have shown that there is a Π-isomorphism $h: \mathscr{T}_\Pi \to \mathscr{T}_{\Sigma,\Gamma}|_\Pi$. In other words, the algebra $\mathscr{T}_{\Sigma,\Gamma}|_\Pi$ is an initial Π-algebra and we have as well an inverse isomorphism $h^{-1}: \mathscr{T}_{\Sigma,\Gamma}|_\Pi \to \mathscr{T}_\Pi$ which is uniquely by initiality. Composing the quotient homomorphism $\mathscr{T}_\Sigma \to \mathscr{T}_{\Sigma,\Gamma}$ with h^{-1} we obtain a surjective Π-homomorphism

$$\text{can}: \mathscr{T}_\Sigma|_\Pi \to \mathscr{T}_\Pi$$

sending each term $t \in \mathscr{T}_\Sigma$ to the unique Π-term $\text{can}(t)$ in its equivalence class modulo the equations Γ. Composing the inclusion homomorphism $j: \mathscr{T}_\Pi \to \mathscr{T}_\Sigma|_\Pi$ with can, we get a Π-homomorphism $\text{can} \circ j: \mathscr{T}_\Pi \to \mathscr{T}_\Pi$, which by initiality of $\mathscr{T}_\Pi$ forces $\text{can} \circ j = 1_{\mathscr{T}_\Pi}$; i.e., the homomorphism can not only is surjective, but is actually a retract.

Note that by construction, the congruence $\equiv_{\text{can}}$ associated to can is exactly the congruence $t \equiv_{\Gamma(\varnothing)} t'$ that holds iff $(\forall \varnothing)\ t = t'$ is derivable from Γ using rules (1)–(5). Therefore, we have the important and very useful property

(*) $t \equiv_{\Gamma(\varnothing)} t'$ iff $(\forall \varnothing)\ t = t'$ is derivable from Γ using rules (1)–(5) iff $\text{can}(t) = \text{can}(t')$.

Having introduced the notion of constructor, we now define selectors. Our definition requires one of the properties that intuitively should hold for selectors, namely that if a data element is "pulled apart" by selectors and then "put together" again using the associated constructor, then the original data element is recovered. We shall see later how other properties follow as a consequence of this basic one.

DEFINITION 38. Let Π be a signature of canonical constructors for Σ with respect to Γ and let $\kappa \in \Pi_{w,s}$ be a canonical constructor with $w = s_1 \ldots s_n$ for $n \geqslant 1$. Then a family of operators $\sigma_i \in \Sigma_{s,s_i}$ for $1 \leqslant i \leqslant n$ is called a *family of selectors* for κ with respect to Π and Γ iff each reachable Σ-algebra that satisfies Γ also satisfies the equation

$$(\forall X)\ \kappa(\sigma_1(x), \ldots, \sigma_n(x)) = x$$

with X consisting of a single variable x of sort s; then σ_i is called the *ith selector* of κ. ∎

The following lemma shows that selectors are unique:

LEMMA 39. *Let $\kappa \in \Pi_{w,s}$ with $w = s_1 \ldots s_n$, be an element of a signature of canonical constructors for Σ with respect to Γ, and let $\sigma_i, \sigma_i' \in \Sigma_{s,s_i}$ for $1 \leqslant i \leqslant n$ be two families of canonical selectors for κ. Then, every reachable Σ-algebra that satisfies Γ also satisfies $(\forall X)\ \sigma_i(x) = \sigma_i'(x)$ for $1 \leqslant i \leqslant n$, with X consisting of a single variable x of sort s.*

Proof. By the definitions of satisfaction and quotient, if an order-sorted algebra A satisfies a given equation, then any quotient of A also satisfies that equation. The algebras mentioned in the lemma are the reachable Σ-algebras that satisfy the equations Γ. Of course, $\mathscr{T}_{\Sigma,\Gamma}$ is one such algebra, and any other such algebra is isomorphic to a quotient of it. Thus, it is enough to prove that $\mathscr{T}_{\Sigma,\Gamma}$ satisfies the equations. By (*), it is enough to show that for each $t \in \mathscr{T}_{\Sigma,s}$ the identity of Π-terms

$$\mathrm{can}(\sigma_i(t)) = \mathrm{can}(\sigma_i'(t))$$

holds for $1 \leqslant i \leqslant n$.

By hypothesis, since $\mathscr{T}_{\Sigma,\Gamma}$ is a reachable algebra satisfying Γ, we have for each $t \in \mathscr{T}_{\Sigma,s}$ that the equations

$$(\forall \varnothing)\ \kappa(\sigma_1(t), \ldots, \sigma_n(t)) = t$$

$$(\forall \varnothing)\ \kappa(\sigma_1'(t), \ldots, \sigma_n'(t)) = t$$

are provable from Γ using rules (1)–(5) for each $t \in \mathscr{T}_{\Sigma,s}$. Therefore, by transitivity and symmetry the equation

$$(\forall \varnothing)\ \kappa(\sigma_1(t), \ldots, \sigma_n(t)) = \kappa(\sigma_1'(t), \ldots, \sigma_n'(t))$$

is also provable from Γ using rules (1)–(5) for each $t \in \mathscr{T}_{\Sigma,s}$. Therefore, by (*), we have the term identity

$$\mathrm{can}(\kappa(\sigma_1(t), \ldots, \sigma_n(t))) = \mathrm{can}(\kappa(\sigma_1'(t), \ldots, \sigma_n'(t)))$$

which, since can is a Π-homomorphism, can be rewritten as

$$\kappa(\mathrm{can}(\sigma_1(t)), \ldots, \mathrm{can}(\sigma_n(t))) = \kappa(\mathrm{can}(\sigma_1'(t)), \ldots, \mathrm{can}(\sigma_n'(t)))$$

and, since κ is an injective operation in the term algebra $\mathscr{T}_\Pi$, this yields the identities

$$\mathrm{can}(\sigma_i(t)) = \mathrm{can}(\sigma_i'(t)),$$

for $1 \leqslant i \leqslant n$, as desired. ∎

Since MSA is the special case of OSA in which the poset structure on the sorts is the identity relation, the definition of constructors (and similarly for selectors) is just the specialization of our original definition to that case. Under this specialization, Definitions 36 and 38 apply to the MSA case, and we use these MSA forms in showing that MSA *cannot* solve the constructor–selector problem. Although the definition of selector only requires a one-sided inverse, Lemma 40 below shows that we get the other inverse equation for free in any reachable Σ-algebra satisfying Γ.

In the stack example, the subsignature Π containing `push` and `empty` is a signature of canonical constructors. Theorem 43 below shows that `top` and `pop` are the first and second selectors for `push`, using the equations

(i) $\texttt{top(push(x}_1\texttt{,x}_2\texttt{))} = \texttt{x}_1\texttt{.}$

(ii) $\texttt{pop(push(x}_1\texttt{,x}_2\texttt{))} = \texttt{x}_2\texttt{.}$

Thus constructors "put data elements together," and selectors decompose data elements into their components, and these two processes are inverse to each other. The lemma below shows that the identities that we could intuitively expect of selectors as functions that yield the different components of a data structure combined by the corresponding constructor are a semantic consequence of our definition.

LEMMA 40. *Let Π be a signature of canonical constructors for Σ with respect to Γ, let $\kappa \in \Pi_{w,s}$ be a canonical constructor with $w = s_1 \ldots s_n$ and let $\sigma_i \in \Sigma_{s,s_i}$ for $1 \leqslant i \leqslant n$ be a family of selectors for κ with respect to Π and Γ. Then every reachable Σ-algebra that satisfies Γ also satisfies the equations*

$$(\forall X)\ \sigma_i(\kappa(x_1, \ldots, x_n)) = x_i$$

for $1 \leqslant i \leqslant n$ and for X consisting of the variables $x_1, \ldots, x_n$ of sorts $s_1, \ldots, s_n$, respectively.

Proof. As mentioned in the proof of Lemma 39, it is enough to prove that $\mathscr{T}_{\Sigma,\Gamma}$ satisfies the equations

$$(\forall X)\ \sigma_i(\kappa(x_1, \ldots, x_n)) = x_i$$

for $1 \leqslant i \leqslant n$. In other words, we have to show that for each $t_1 \in \mathscr{T}_{\Sigma,s_1}, \ldots, t_n \in \mathscr{T}_{\Sigma,s_n}$ the equations

$$(\forall \varnothing)\ \sigma_i(\kappa(t_1, \ldots, t_n)) = t_i$$

for $1 \leqslant i \leqslant n$ follow from Γ by order-sorted equational deduction. By (*) this is equivalent to showing the term identities

$$\mathrm{can}(\sigma_i(\kappa(t_1, \ldots, t_n))) = \mathrm{can}(t_i)$$

for $1 \leqslant i \leqslant n$.

By hypothesis, for each choice of $t_1 \in \mathcal{T}_{\Sigma,s_1}, \ldots, t_n \in \mathcal{T}_{\Sigma,s_n}$ the equation

$$(\forall \varnothing)\ \kappa(\sigma_1(\kappa(t_1, \ldots, t_n)), \ldots, \sigma_n(\kappa(t_1, \ldots, t_n))) = \kappa(t_1, \ldots, t_n)$$

is provable from Γ using rules (1)–(5). By (*) we therefore have the term identity

$$\mathrm{can}(\kappa(\sigma_1(\kappa(t_1, \ldots, t_n)), \ldots, \sigma_n(\kappa(t_1, \ldots, t_n)))) = \mathrm{can}(\kappa(t_1, \ldots, t_n))$$

which, because can is a Π-homomorphism, can be rewritten as

$$\kappa(\mathrm{can}(\sigma_1(\kappa(t_1, \ldots, t_n))), \ldots, \mathrm{can}(\sigma_n(\kappa(t_1, \ldots, t_n)))) = \kappa(\mathrm{can}(t_1), \ldots, \mathrm{can}(t_n))$$

which, by injectivity of κ as an operation in $\mathcal{T}_\Pi$, yields the identities

$$\mathrm{can}(\sigma_i(\kappa(t_1, \ldots, t_n))) = \mathrm{can}(t_i) \qquad \text{for} \quad 1 \leqslant i \leqslant n,$$

as desired. ∎

As further motivation for Theorem 43, we give a somewhat more complex example:

EXAMPLE 41: Aritmetic Expressions:

```
obj EXPRESSION is
  sorts Expression Sum Product .
  subsorts Sum Product < Expression .
  ops 0 1 : -> Expression .
  op _+_ : Expression Expression -> Sum .
  op _*_ : Expression Expression -> Product .
  ops 1s 2s : Sum -> Expression .
  ops 1p 2p : Product -> Expression .
  vars E1 E2 : Expression .
  eq 1s(E1 + E2) = E1 .
  eq 2s(E1 + E2) = E2 .
  eq 1p(E1 * E2) = E1 .
  eq 2p(E1 * E2) = E2 .
endo
```

∎

Here the signature Π of canonical constructors contains `0`, `1`, `+`, `*`, while `1s` and `2s` are (respectively) first and second selectors for `+`, and `1p` and `2p` are (respectively) first and second selectors for `*`. Note that the domain of `1s` and `2s` is the sort `Sum` of trees having a `+` on top, while the domain of `1p` and `2p` is `Product`, the trees having a `*` on top. Of course, we could have given a simpler one-sorted signature of constructors, as in

```
obj EXPRESSION-0 is
  sorts Expression .
  ops 0 1 : -> Expression .
  op _+_ : Expression Expression -> Expression .
  op _*_ : Expression Expression -> Expression .
endo
```

but then there is no satisfactory way to define the selectors. For example, if

```
s1 s2:Expression -> Expression
```

were the first and second selectors for $\_+\_$, then we would have to have

```
s1(0) + s2(0) = 0,
```

contradicting that both + and 0 are canonical constructors.

The above argument generalizes to show the following, which is more precise statement of the inherent impossibility of expressing selectors in MSA:

THEOREM 42. *In an MSA signature Σ having a subsignature Π of constructors relative to equations Γ, if $\kappa: s_1 \ldots s_n \to s$ for $n \geqslant 1$ is a constructor, then if there is any other constructor with the same codomain, $\kappa': s'_1 \ldots s'_m \to s$ for $m \geqslant 0$, and with T_{Σ, s'_j} nonempty for $1 \leqslant j \leqslant m$, then κ cannot have selectors.*

Proof. By hypothesis there is a term $\kappa'(t_1, \ldots, t_n) \in T_{\Sigma, s}$. If σ_i for $1 \leqslant i \leqslant n$ were a family of selectors for κ, then by definition of selectors the equation

$$(\forall \varnothing)\ \kappa(\sigma_1(\kappa'(t_1, \ldots, t_n)), \ldots, \sigma_n(\kappa'(t_1, \ldots, t_n))) = \kappa'(t_1, \ldots, t_n)$$

would be provable by the rules of many-sorted equational deduction (i.e., by rules (1)–(5) specialized to the MSA case). Therefore, by (*) we would obtain the term identity

$$\mathrm{can}(\kappa(\sigma_1(\kappa'(t_1, \ldots, t_n)), \ldots, \sigma_n(\kappa'(t_1, \ldots, t_n)))) = \mathrm{can}(\kappa'(t_1, \ldots, t_n))$$

which because can is a Π-homomorphism can be rewritten as the identity

$$\kappa(\mathrm{can}(\sigma_1(\kappa'(t_1, \ldots, t_n))), \ldots, \mathrm{can}(\sigma_n(\kappa'(t_1, \ldots, t_n)))) = \kappa'(\mathrm{can}(t_1), \ldots, \mathrm{can}(t_n)),$$

which is impossible in the term algebra T_Π, since by hypothesis κ and κ' are different operators. ∎

That is, the only possible MSA selectors are untupling operators for the rather trivial case of a tupling operator that is the only constructor of its value sort.

Note that any EXPRESSION-algebra $A = \{A_{\texttt{Expression}}, A_{\texttt{Sum}}, A_{\texttt{Product}}\}$ can be viewed as a EXPRESSION-0-algebra by forgetting about the operators `1s`, `2s`, `1p`, `2p` and extending the range of + to `Expression` using $A_{\texttt{Sum}} \subseteq A_{\texttt{Expression}}$ and similarly for $*$. This motivates the following general method of associating an order-sorted signature Σ with constructors and selectors for operators to each many-sorted signature Σ^0 without overloading,[9] with many-sorted canonical constructors Π^0 relative to a set Γ^0 of Σ^0-equations:

1. Define a partially ordered set S^1 to have the sorts S^0 of Π^0 together with a new sort $s_\kappa \leqslant s$ for each constructor $\kappa: s_1 \ldots s_n \to s$ in Π^0 with $n \geqslant 1$.

2. Define an order-sorted signature Π^1 over S^1 to have the same constants as Π^0, plus a new operator $\kappa: s_1 \ldots s_n \to s_\kappa$ for each $\kappa: s_1 \ldots s_n \to s$ in Π^0 with $n \geqslant 1$. (In Example 41, starting with Π^0 as the signature of EXPRESSION-0 yields Π^1 as the subsignature of canonical constructors of EXPRESSION.)

3. Now enlarge Π^1 to a signature Π by adding a new selector operator $i_\kappa: s_\kappa \to s_i$ for $1 \leqslant i \leqslant n$ for each operator $\kappa: s_1 \ldots s_n \to s$ in Π^0. (In Example 41, Π is the signature of EXPRESSION.)

4. Finally, define SEL to be the set of Π-equations $(\forall X)$ $i_\kappa(\kappa(x_1, \ldots, x_n)) = x_i$ for $i = 1, \ldots, n$, where X consists of x_i of sort s_i for $s_1 \ldots s_n$ the arity of κ in Π with $n \geqslant 1$.

The following says that this construction works as desired.

THEOREM 43. *Let Σ^0 be a many-sorted signature (without overloading) having $\Pi^0 \subseteq \Sigma^0$ as canonical constructors with respect to Γ^0. Then for Π^1, Π, SEL defined as above, the following properties hold:*

(1) *Π^1 is a signature of canonical constructors for the order-sorted signature $\Sigma = (\Sigma^0 - \Pi^0) \cup \Pi$ relative to $\Gamma = SEL \cup \Gamma^0$, and for each non-constant κ in Π^1, i_κ is its ith selector with respect to Σ and Γ.*

(2) *By extending the range of its operators and forgetting about the selectors, each Σ-algebra A can be viewed as a Σ^0-algebra $U(A)$ in such a way that initial algebras are preserved, i.e., such that $U(\mathcal{T}_{\Sigma,\Gamma}) = T_\Sigma^{0,\Gamma^0}$.*

Proof. To prove (1), we decompose the problem into two smaller subproblems by realizing that Σ is obtained adding to Π^1 two different--and

[9] This means that the sets $\Sigma_{w,s}$ are pairwise disjoint.

in a sense independent—sets of operators, namely the operators from Σ^0 not in Π^0, as well as the selectors. We therefore define $\Sigma^1 = (\Sigma^0 - \Pi^0) \cup \Pi^1$ and consider the OSA specification inclusions

$$(S^1, \leqslant, \Pi^1, \varnothing) \subseteq (S^1, \leqslant, \Pi, \mathrm{SEL}) \subseteq (S^1, \leqslant, \Sigma, \Gamma)$$

and

$$(S^1, \leqslant, \Pi^1, \varnothing) \subseteq (S^1, \leqslant, \Sigma^1, \Gamma^0) \subseteq (S^1, \leqslant, \Sigma, \Gamma).$$

We have to show that

(i) the unique Π^1-homomorphism $h_{\mathrm{i}}: \mathscr{T}_{\Pi^1} \to \mathscr{T}_{\Sigma,\Gamma}|_{\Pi^1}$ is an isomorphism.

If we had already proved that

(ii) $h_{\mathrm{ii}}: \mathscr{T}_{\Pi^1} \to \mathscr{T}_{\Pi,\mathrm{SEL}}|_{\Pi^1}$ is an isomorphism,

(iii) $h_{\mathrm{iii}}: \mathscr{T}_{\Pi^1} \to \mathscr{T}_{\Sigma^1,\Gamma^0}|_{\Pi^1}$ is an isomorphism,

then we could establish (i) by the following argument: Without loss of generality, we may assume that h_{ii} and h_{iii} are identity isomorphisms, and define a Σ-algebra structure $\mathscr{T}_{\Pi,\mathrm{SEL}} + \mathscr{T}_{\Sigma^1,\Gamma^0}$ with $(\mathscr{T}_{\Pi,\mathrm{SEL}} + \mathscr{T}_{\Sigma^1,\Gamma^0})_s = (\mathscr{T}_{\Pi,\mathrm{SEL}})_s = (\mathscr{T}_{\Sigma^1,\Gamma^0})_s$ for each sort s and each operation in Σ coinciding with the one in $\mathscr{T}_{\Pi,\mathrm{SEL}}$ or in $\mathscr{T}_{\Sigma^1,\Gamma^0}$. Therefore, the equations Γ are satisfied by construction since $\Gamma = \mathrm{SEL} \cup \Gamma^0$. We claim that $\mathscr{T}_{\Pi,\mathrm{SEL}} + \mathscr{T}_{\Sigma^1,\Gamma^0}$ is an initial Σ, Γ-algebra. Let B be any Σ, Γ-algebra; then we have at most one homomorphism $h: \mathscr{T}_{\Pi,\mathrm{SEL}} + \mathscr{T}_{\Sigma^1,\Gamma^0} \to B$ since as an S^1-sorted function, h must coincide with the unique Π^1-homomorphism $h: (\mathscr{T}_{\Pi,\mathrm{SEL}} + \mathscr{T}_{\Sigma^1,\Gamma^0})|_{\Pi^1} = \mathscr{T}_{\Pi^1} \to B|_{\Pi^1}$. But by (ii) and (iii), h is also a Π-homomorphism $h: \mathscr{T}_{\Pi,\mathrm{SEL}} \to B|_{\Pi}$ and a Σ^1-homomorphism $h: \mathscr{T}_{\Sigma^1,\Gamma^0} \to B|_{\Sigma^1}$ and therefore a $\Pi \cup \Sigma^1 = \Sigma$-homomorphism, as desired; thus (i) follows.

To prove (ii), it is enough to show that the equations SEL are confluent and terminating as order-sorted rewrite rules and such that for each $t \in \mathscr{T}_{\Pi}$ the normal form term $\mathrm{norm}_{\mathrm{SEL}}(t)$ is in $\mathscr{T}_{\Pi^1}$. Indeed, since the terms in $\mathscr{T}_{\Pi^1}$ do not contain any selectors, no equations in SEL apply to them so that they are all in normal form. Therefore by Theorem 31, for each $t \in \mathscr{T}_{\Pi}$ there is a unique $t' \in \mathscr{T}_{\Pi^1}$ (namely, $t' = \mathrm{norm}_{\mathrm{SEL}}(t)$) such that the equation $(\forall \varnothing)$ $t = t'$ is derivable from SEL by the rules of order-sorted equational deduction. By Fact 28 we have in addition $LS(\mathrm{norm}_{\mathrm{SEL}}(t)) \leqslant LS(t)$ and therefore Π^1 is a signature of canonical constructors for Π, which is equivalent to (ii) by Lemma 37. Therefore, proving (ii) has been reduced to proving the following:

LEMMA 44. *The equations SEL are confluent and terminating order-sorted rewrite rules such that for each $t \in \mathscr{T}_{\Pi}$ the normal form term* $\mathrm{norm}_{\mathrm{SEL}}(t)$ *is in* $\mathscr{T}_{\Pi^1}$.

Proof. The equations in SEL are of the form $i_\kappa(\kappa(x_1, ..., x_n)) = x_i$ for $i = 1, ..., n$, where X consists of x_i of sort s_i for $s_1 ... s_n$ the arity of κ in Π with $n \geq 1$. Under any substitution θ, the least sort of the lefthand side will remain constant (namely s_i), whereas the righthand side may possibly decrease to a sort smaller than s_i; therefore, since in addition the lefthand sides have more variables than the righthand sides, the equations in SEL are indeed order-sorted rewrite rules. Termination can easily be proved by taking the number of selector occurrences in a term as a "measure" and noting that each application of a rule in SEL decreases that measure exactly by 1. To see that the normal forms of ground terms are in $\mathscr{T}_{\Pi^1}$, note that in a ground term with selector occurrences we can always find a subterm with a selector occurring at its top and no selectors in any of its subterms, and that such a subterm must necessarily be of the form $i_\kappa(t)$ with $t \in \mathscr{T}_{\Pi^1}$, which by the definition of the signature Π^1 forces t to be of the form $\kappa(t_1, ..., t_n)$. Therefore, an equation in SEL always applies to a subterm of this kind and a selector symbol disappears after rewriting with such an equation. This shows that a term with no selectors (and therefore in normal form and an element of $\mathscr{T}_{\Pi^1}$) can be reached after N rewriting steps for N the number of selector occurrences in the original ground term. Since the rules are terminating, to establish confluence it is enough by Fact 30 to establish local confluence, which in turn can, thanks to Theorem 32, be established by analysis of the critical pairs of overlaps. But it is trivial to check that there are no overlaps for the rules in SEL. ∎

Before proving (iii) we prove two auxiliary lemmas that establish a very close connection between terms and proofs at the order-sorted and many-sorted levels.

LEMMA 45. *For each* $s \in S^0$, $\mathscr{T}_{\Sigma^1, s} = T_{\Sigma^0, s}$.

Proof. This can be established using induction on the depth of terms to show that

$$t \in \mathscr{T}_{\Sigma^1, s} \quad \text{iff} \quad t \in T_{\Sigma^0, s}.$$

For terms of depth 0 (i.e., constants) this is clear since Σ^1 and Σ^0 have the same constants, each with the same sort in each signature. Terms of sort s having depth $N+1$ are of the form $\sigma(t_1, ..., t_n)$ with t_i of depth less than or equal to N (thus, the induction hypothesis applies) and σ an operation of appropriate arity and sort s (in the many-sorted case), or sort smaller or equal than s in the order-sorted case. Now, note the following:

(1) The sorts in S^0 are exactly the tops of each of the connected components of S^1.

(2) The arities of an operator σ in Σ^1 and in Σ^0 coincide and are always in $(S^0)^*$.

(3) The coarity of an operator σ in Σ^1 is smaller than or equal to its coarity in Σ^0.

By the induction hypothesis, this shows that $\sigma(t_1, ..., t_n)$ is a well-formed term in $\mathscr{T}_{\Sigma^1, s}$ iff it is a well-formed term in $T_{\Sigma^0, s}$, as desired. ▮

As a consequence of this lemma, and by remark (1) in its proof, note that $t \in \mathscr{T}_{\Sigma^1}$ iff $t \in T_{\Sigma^0}$. Also, note that, again by remark (1), for X a family of variables whose sorts are all in S^0, $(\forall X)\ t = t'$ is a Σ^1-equation iff it is a Σ^0-equation (we abuse language somewhat by using the same notation for X as an S^1-sorted and an S^0-sorted set).

LEMMA 46. *An equation* $(\forall \varnothing)\ t = t'$ *is provable from the* Σ^1*-equations* Γ^0 *by the rules* (1)–(5) *of order-sorted equational deduction iff it is provable from the* Σ^0*-equations* Γ^0 *by the rules* (1)–(5) *of many-sorted equational deduction* (*the same rules, but specialized to a many-sorted signature*).

Proof. To simplify the argument we will use a somewhat more restrictive version of the rule (4) of congruence, namely the rule

(4′) *Congruence.* If $t_i, t_i' \in \mathscr{T}_\Sigma(X)_{s_i}$ for $i = 1, ..., n$, and if $(\forall X)\ t_i = t_i'$ is derivable for $i = 1, ..., n$, then for any $\sigma: s_1 ... s_n \to s$ in Σ the equation $(\forall X)$ $\sigma(t_1, ..., t_n) = \sigma(t_1', ..., t_n')$ is also derivable.

For a proof of the completeness of the rules of deduction after replacing (4) by (4′) see (Goguen and Meseguer, 1987); however, a direct proof of the equivalence of the two sets of rules is an easy exercise by induction on the depth of the term t in rule (4) to which the substitution is applied.

We prove the lemma by induction on the depth of proofs, where proofs that only use reflexivity have depth 0, and proofs whose last step used one of the rules (2)–(3), (4′), or (5) have depth 1 plus the maximum depth of the proofs used in the premises of the rule. The case of depth 0 is trivial. Assuming the induction hypothesis for depth N, let us consider proofs of depth $N + 1$. If the last rule used is symmetry or transitivity, the argument is also trivial and is left to the reader. If the last rule used is (4′), recall remarks (1)–(3) in the proof of Lemma 45 (as well as that lemma itself) to conclude that the terms are well-formed in either case and that therefore, by the induction hypothesis, rule (4) applies in the order-sorted case iff it applies in the many-sorted case. If the last rule used is rule (5), the argument reduces in essence to showing, for X a family of variables whose sorts are in S^0 (as is the case for the families of variables quantifying equations in Γ^0), a bijective correspondence between many-sorted substitutions $X \to T_{\Sigma^0}(X)$ and order-sorted substitutions $X \to \mathscr{T}_{\Sigma^1}(X)$ (note again the

slight abuse of language regarding X, which is viewed as S^0-sorted in the first case and as S^1-sorted in the other). Since in X there are no variables of sorts outside S^0, Lemma 45 shows that we obtain such a bijective correspondence by adding in or throwing out empty functions $\varnothing \to \mathscr{T}_{\Sigma^1}(X)_{s'}$ for the sorts $s' \in S^1 - S^0$. ∎

We are now ready to prove (iii), which is equivalent to proving that for each $t \in \mathscr{T}_{\Sigma^1}$ there is a unique $t' \in \mathscr{T}_{\Pi^1}$ such that the equation $(\forall \varnothing)\ t = t'$ is derivable from Γ^0 by OSA deduction and, in addition, that $LS(t') \leqslant LS(t)$. Provability of such an equation for a unique such t' follows from Lemma 45 (which in particular shows that $t' \in \mathscr{T}_{\Pi^1}$ iff $t' \in T_{\Pi^0}$) and Lemma 46, by the hypothesis that Π^0 is a signature of canonical constructors for Σ^0. To show that in addition $LS(t') \leqslant LS(t)$, we reason by cases. The term t' is either a constant κ in Π^0 or a term of the form $\kappa(t_1, ..., t_n)$ for $n \geqslant 1$. If t' is a constant κ in Π^0 then it has sort $s \in S^0$ and by Π^0 being a signature of canonical constructors t must be either κ itself—in which case the least sorts of t and t' obviously coincide—or a term of the form $\sigma(u_1, ..., u_m)$ with $\sigma \in \Sigma^0 - \Pi^0$ (otherwise, using property (*) after Lemma 37 we reach a contradiction); therefore, the coarity of σ is the same in Σ^1 and Σ^0, namely s, and so we have $LS(t) = LS(t')$. If t' is of the form $\kappa(t_1, ..., t_n)$ for $n \geqslant 1$ with κ a constructor, then it has sort $s_\kappa < s$, and using the same argument again we must conclude that t is either of the form $\kappa(v_1, ..., v_n)$, in which case the least sorts of t and t' coincide, or t is of the form $\sigma(u_1, ..., u_m)$ with $\sigma \in \Sigma^0 - \Pi^0$ and therefore has sort s so that we have $LS(t) = s \geqslant s_\kappa = LS(t')$, as desired.

To prove (2) of Theorem 43, note that, since we first forget about the selectors, the functor U factors through the functor $-|_{\Sigma^1}$ so that we have $U(A)_s = (A|_{\Sigma^1})_s = A_s$ for each $s \in S^0$, and for each $\sigma \in \Sigma^1$ we have $U(A)_\sigma(a_1, ..., a_n) = A_\sigma(a_1, ..., a_n)$. Note that we have

$$\mathscr{T}_{\Sigma, \Gamma}|_{\Sigma^1} = (\mathscr{T}_{\Pi, \mathrm{SEL}} + \mathscr{T}_{\Sigma^1, \Gamma^0})|_{\Sigma^1} = \mathscr{T}_{\Sigma^1, \Gamma^0}.$$

Now, by Lemmas 45 and 46 we have, for each $s \in S^0$,

$$(\mathscr{T}_{\Sigma^1, \Gamma^0})_s = (T_{\Sigma^0, \Gamma^0})_s$$

with the operations coinciding in the obvious way. Therefore we have $U(\mathscr{T}_{\Sigma, \Gamma}) = T_{\Sigma^0, \Gamma^0}$, as desired. ∎

Remark 47. Definitions 36 and 38 for constructors and selectors can be made more general by dropping the requirement that the constructors be "canonical." Thus, we just define a signature Π of *constructors* for Σ relative to equations Γ by dropping the requirement that for each term t the Π-term t_0 such that $(\forall \varnothing)\ t = t_0$ is derivable be *unique* (i.e., canonical), leaving the rest of the definition unchanged. Using this weaker condition,

the concatenation operation on lists is a constructor for lists, though not canonical. The notion of selector also generalizes, so that `head` and `tail` become first and second selectors (in the generalized sense) for list concatenation. In addition, selectors can be made relative to a specialization of subsorts; that is, given a constructor $\kappa: s_1 \ldots s_n \to s$ with $n \geqslant 1$ in a signature Π of constructors for Σ relative to Γ, a string $w' = s'_1 \ldots s'_n$ with $s'_i \leqslant s_i$, and a sort $s' \leqslant s$, we can define the operators $\sigma_i: s' \to s'_i$ for $1 \leqslant i \leqslant n$ as a family of $\langle w', s' \rangle$-*selectors* for κ if all reachable Σ-algebras that satisfy Γ also satisfy the equation

$$(\forall x) \qquad \kappa(\sigma_1(x), \ldots, \sigma_n(x)) = x$$

for x of sort s'. ∎

5. Multiple Representations, Coercions, and Degenerate Cases

This section applies order-sorted algebra to give simple explications of some problems that have long vexed the algebraic specifications and functional programming communities. These problems include multiple representation, coercion, and degeneracy in data representation. We begin with the latter, which is very straightforward given the work already accomplished earlier in the paper. (Although the following discussion is based on (Goguen, 1988), it is entirely self-contained.)

5.1. *Degenerate Cases*

A degenerate case is simply a constructor that has a lower dimension, i.e., that involves fewer parameters, than some other constructor. It is well known that degenerate cases can be a particular nuisance in computational geometry. For example, in a `LINE` object, the sort `Line` might have two cases, one `General` and the other `Vertical`, with the latter being degenerate, since the first involves two parameters and the latter only one, its intercept on the x-axis.

The theory of constructors and selectors that we have already developed provides a fully adequate solution of this problem, but it seems helpful to use a more compact syntax which more clearly shows the relationships involved. Thus, consider the following specification for lists of integers:

```
obj LIST-INT is
  protecting INT .
  rep List by case(Empty) is: { }
  or by case(NonEmpty) is: _·_ with head : Int and
  tail : List .
endo
```

Here the keyword rep indicates that we are defining a data representation, in this case List, by "cases"; the first case is called Empty, and has the constructor { }, while the second case is Nonempty, with the constructor _·_. The functions head and tail following the keyword with are the selectors for _·_; the constructor { } is degenerate and has no selectors. Thus, the intended order-sorted semantics for this is given by the following:

```
obj LIST-INT is
  protecting INT .
  sorts Empty NonEmpty List .
  subsorts Empty < List .
  subsorts NonEmpty < List .
  op { } : -> Empty .
  op _·_ : Int List -> NonEmpty .
  op head : NonEmpty -> Int .
  op tail : NonEmpty -> List .
  var I : Int .
  var L : List .
  eq head(I·L)=I .
  eq tail(I·L)=L .
endo
```

We know that OSA is *needed* here, because MSA is not strong enough to define the selectors. The above code reveals that Empty and Nonempty are *subsorts* of List (for this purpose, they could perhaps be more clearly designated EmptyList and NonEmptyList). This translation makes good use of the machinery of order-sorted algebra, including *sort predicates*, which are Bool-valued operations that are true iff their argument has the appropriate sort. Thus,

```
Empty({ })=true.
```

while

```
NonEmpty({ })=false.
```

(Such predicates are easily defined using constructors.)

"Case analysis" can also be very useful in the righthand sides of equations. For example, given the representation

```
rep S by case(A) is: ...
   or by case(B) is: ... .
```

we can consider an equation of the form

```
t = t'(case(A X) is: t1 or case(B X) is: t2).
```

where t, t'(_), t1, and t2 are expressions, and where X is a variable of sort S occurring in t, to be "syntactic sugar" for the following two conditional equations, which use the sort predicates A and B:

```
t = t'(t1) if A(X).
t = t'(t2) if B(X).
```

Here, t1 can use the selectors associated with the constructor for case A, while t2 can use the selectors associated with B, but t1 cannot use those associated with B and t2 cannot use those associated with A. Case analyses may also involve more than one variable, with syntax of the form

```
case(A X,Y ; B Z) is: ...
```

which is shorthand for a conditional equation of the form

```
... if A(X) and A(Y) and B(Z) .
```

The rep construction is also useful when there are no degenerate cases, as illustrated in the specification of Cartesian coordinates in the object POINT below.

5.2. *Multiple Representation*

There are many problems in which it is very desirable to represent some data in more than one way, because some computations are much easier in one representation, while others are much easier in another. For example, if we represent points by Cartesian *and* by Polar coordinates, then it is easier to compute distance in the Cartesian representation, but easier to compute angles in the polar representation. In such a situation, using a selector does not *require* the corresponding representation, since its value could be computed from any underlying representation where it is defined.

We first explain what we mean by multiple representation using an example in a rather rich notational extension of OBJ; later, we give a precise translation of it into a standard initial order-sorted algebra semantics. An alternative approach with a nicer operational semantics, though with less theoretical elegance, is given in the next subsection, The module POINT below declares both of these representations, together with *coercions*, which are implicit conversions or translations between them; each representation is given a name to help distinguish between the two coercions involved. Two new subsorts of Real are introduced (namely, Pos and Angle) for positive elements and for angles, and are defined by sort constraints, which were explained in Section 3.5. Also, it is assumed that a number of auxiliary functions, including square root, some trigonometric functions, and the constant pi, have been defined in the imported real arithmetic module REAL.

```
obj POINT is
  protecting REAL .
  sort Pos < Real if P > 0 .
  sort Angle < Real if 0 <= A < 2 * pi .
  rep(Cart) Point by ⟨_;_⟩ with x : Real and y : Real .
  rep(Polar) Point by case(General) is:
  [_;_] with theta : Angle
        and    rho : Pos
                 or by case(Origin) is: * .
  coerce(Cart to Polar) by
  if x =/= 0 then [ arctan(y/x) ; sqrt((x **2)+(y **2))]
    else if y == 0 then * else if y > 0
    then [pi / 2 ; y] else [3 * pi/2 ; -y] fi fi fi .
  coerce(Polar to Cart) by
    case(General)is: ⟨rho * cos(theta);rho * sin(theta)⟩
    or case(Origin) is: ⟨0 ; 0⟩ .
  op d : Point Point -> Real .
  vars P Q : Point .
  eq d(P,Q) = sqrt(((x(P) - x(Q))**2)+((y(P) - y(Q))**2)) .
endo
```

Thus, Cart and Polar are the names of the two representations, and the latter has two cases, called General and Origin. There is no need for variables in the coercion definitions: because there is just one item involved that has the indicated representation, the selectors declared in the representations implicitly take it as their argument. An interesting point about this example is that one representation has a degenerate case, while the other does not; however, the two representations are completely intertranslatable.

Given this POINT object, any of the four selectors can be used anywhere in a program, without having to worry about what the underlying representation might actually be. Semantically, each representation of a given sort defines a new subsort of it, and each case of a given representation defines a new subsubsort. Thus, Point has subsorts Cart and Polar, and Polar has subsorts General and Origin, as shown in Fig. 1. Operations involving points should use the sort Point, rather than the sort of one of the representations, in order to obtain the full benefits of this facility. (If one wants to use the same name for cases that belong to different representations, a qualification notation can be used to distinguish them; for example, if we had a General case of Cart, the two different General cases would be named General.Cart and General.Polar, respectively.)

Although this example has just two representations and one coercion

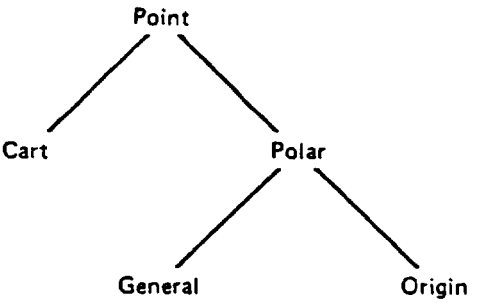

FIG. 1. Subsorts involved in the two representations of points.

from each representation to the other, the construction also allows any number of representations, and it also allows *one-way coercions*, whereby some representations can be destructively translated to others; however, the construction syntactically enforces that there is *at most* one coercion from any given representation to another. In addition, we must require that the coercions *compose correctly*, in the sense that if there is a coercion c from representation $R1$ to representation $R2$, and another c' from $R2$ to $R3$, and a third c'' from $R1$ to $R3$, then the composition $c' \circ c$ must equal c''. In particular, if c goes from $R1$ to $R2$ and c' goes from $R2$ to $R1$, then $c \circ c'$ must be the identity, and in fact, $R1$ and $R2$ must be isomorphic representations; this case corresponds to what are called *views* in (Wadler, 1987). (In fact, our coercions resemble the default views of (Goguen, 1984), in that they form a category with at most one morphism from any given object to another.) This imposes a preorder structure on the sort set, and ensures that each item has at most one translation, i.e., that the implicit conversions are always consistent.

It might seem that this construction assumes that the representations are "free" (or one might say, "anarchic"), in the sense that they do not satisfy any non-trivial equations. However, the only disadvantage to adding equations for constructors is that it may be harder to ensure the consistency of operations defined over them, including coercions; in fact, it might require some theorem proving, whereas it is automatic in the anarchic case.

We now describe the initial OSA semantics for coercions, which simply regards the coercion definitions as equations. This semantics will ensure, for our example, that the sorts `Cart`, `Polar`, and `Point` have *identical* carriers in the initial algebra, and it will thus guarantee *representation independence*, since the different representations are identified, and appear in the same equivalence class. Under this view, it is necessary to use different constructors for each representation, and non-injective one-way coercions cannot be handled (however, these restrictions are relaxed in an alternative semantics given below). For our example, the intended semantics is the initial algebra of the following OSA specification, which uses a slight

variant of the rep notation for single constructors, in which selectors follow the keyword with, and the corresponding subsorts and equations remain implicit:

```
obj POINT is
  protecting REAL .
  sort Pos < Real if P < 0 .
  sort Angle < Real if 0 <= A < 2 * pi .
  sorts Point Cart Polar General Origin .
  subsorts Cart Polar < Point .
  op <_ ; _> : Real Real -> Cart with x and y .
  op [_;_] : Angle Pos -> General with theta and rho .
  op * : -> Origin .
  op d : Point Point -> Real .
  vars X Y X' Y' : Real .
  var Rho : Pos .
  var Theta : Angle .
  eq <X ; Y> = if X =/= 0 then
    [arctan(Y/X) ; sqrt((X**2) + (Y**2))]
    else if Y == 0 then * else if Y > 0 then [pi/2 ; Y]
    else [3 * pi/2 ; -Y] fi fi fi .
  eq[Rho ; Theta]=<Rho * cos(Theta) ; Rho * sin(Theta)> .
  eq *= <0 ; 0> .
  eq d(<X ; Y>,<X' ; Y'>)=sqrt(((X-X')**2)+((Y-Y')**2)) .
endo
```

Operationally, the coercion equations are not treated as rewrite rules, but rather, the remaining equations should be matched modulo the coercion equations. For example, the equation

```
d(<X ; Y>,<X' ; Y'>) = sqrt(((X - X')**2) + ((Y - Y')**2)) .
```

matches the expression d(<7 ; 0>,[pi ; 6]) modulo the equation

```
  [Rho ; Theta] = <Rho * cos(Theta) ; Rho * sin(Theta)>.
```

and rewrites to

```
sqrt(((7 - (6 * cos(pi))) **2)+((0 - (6 * sin(pi))h) **2)).
```

which eventually simplifies to 13.

Let us now consider a more complex example, the ASCII coercion from natural numbers to characters. Here, neither Nat nor Char is a subsort of the other, and the coercion function, let us denote it ascii, is defined only on a subset of the naturals, those between 0 and 127 (inclusive); let us

denote this subsort Nat[0..127], and assume it is alredy defined by a sort constraint in a module NATS. Then the code involved looks as follows:

```
obj CHAR is
  protecting NATS .
  sort Char .
  sort Nat[0..127] < Char .
  rep Char by nul, ........
  coerce(Nat[0..127] to Char) by
    0 is: nul
    .........
endo
```

Thus, the sort Char is defined by a large list of cases, and the coercion is, in effect, defined by a large table.

5.3. *Coercions via Retracts*

Coercions are also meaningful in more general cases than multiple representation, but here the simple approach of regarding the coercion definitions as equations may break down. For example, suppose that Int < Rat and that we want a coercion from rationals to integers, taking the integer part of a rational; since this operation (let us denote it int-part:Rat → Int) is not injective, dividing by the congruence relation generated by R = int-part(R) would simply destroy the rationals! What we really want here is to apply int-part iff some function needs an integer but has been supplied a rational. This kind of behavior is familiar from the retracts that we have already considered in Section 3.4: the parser inserts a special function symbol $r_{Rat,Int}$ when it gets a subexpression of sort Rat but needs one of sort Int. All we have to do then, is provide the conditional equation

$$r_{Rat,Int}(R) = \texttt{int-part}(R) \texttt{ if not } \texttt{Int}(R) \ .$$

in addition to the equation that we already have for retracts, namely

$$r_{Rat,Int}(I) = I \ .$$

where I has sort Int; actually, in this case, we do not need the second equation at all, or the condition in the first equation, since int-part(I) = I by its definition. Thus, in the most general case, we may regard coercions as operation symbols having empty syntax that are invoked iff run-time type checking fails to yield an argument of the right sort; they are defined by conditional equations whose lefthand side is a retract and whose righthand side is a coercion definition. In fact, this gives a fully general way

to handle coercions, since retracts will only arise inside of expressions that do not, properly speaking, parse anyway. This guarantees correctness, since the coercion equations only affect expressions outside the initial algebra of well-sorted terms. It also provides a more efficient operational semantics. However, it is less satisfying when applied to the multiple representation case, since the corresponding representations are not actually identified. Nevertheless, for such cases we can just add the equations that do the identifications, and thus combine the advantages of both approaches.

5.4. *Related Work*

Our coercions resemble Wadler's "views" (Wadler, 1987), except that our coercions can be one-way, i.e., mutual inter-tranlability is not required; also, we use the order-sorted algebra to get selectors, we allow any number of representations, and we do not restrict to anarchic representations. The work of Reynolds (1980) on "category-sorted algebra" is also closely related; however, Reynolds does not support user-defined coercions (although Wadler does), does not use order-sorted algebra to get selectors, and his category-sorted algebra may be too general, since it allows any number of coercions between a given pair of representations.[10] Our work on coercions is also related to work of Bruce and Wegner (1986, 1987). All these works have influenced our approach.

6. Conclusions

We have presented some basics of order-sorted universal algebra, and shown how it generalizes the usual many-sorted algebra. Moreover, we have shown that order-sorted algebra permits an elegant solution to the constructor-selector problem, whereas this problem cannot be solved in many-sorted algebra, and we have given some examples to illustrate the issues involved. Finally, we have discussed and given examples for the multiple representation and coercion problems, and shown how they can be solved and given an initial algebra semantics using order-sorted algebra.

Acknowledgments

We thank the referees for their valuable suggestions for improving a previous version; their suggestions have led us to introduce substantial revisions in the original manuscript that we

[10] It may be worth noting that Reynolds' category-sorted algebra is a special case of ordinary many-sorted algebra, in the sense that there is a simple translation which preserves the models, and that no special rules of deduction are required. The case of order-sorted algebra is quite different, since new rules of deduction really are needed to account for the requirement that some carriers *must* be contained in others.

believe make the exposition clearer and more accessible. We also thank Narciso Martí-Oliet for his careful reading of the manuscript and his many suggestions for improving it. This research was supported in part by Office of Naval Research Contracts N00014-85-C-0417, N00014-86-C-0450, N00014-88-C-0618, and N00014-90-C-0086, by National Science Foundation Grant MCS8201380, and by a gift from the System Development Foundation.

RECEIVED August 28, 1987; FINAL MANUSCRIPT RECEIVED March 25, 1991

REFERENCES

BRUCE, K., AND WEGNER, P. (1986), An algebraic model of subtypes in object-oriented languages (draft), *SIGPLAN Notices* **21** (10), 163–172.

BRUCE, K., AND WEGNER, P. (1987), "An Algebraic Model of Subtype and Inheritance," Technical Report CS-87-21, Computer Science Department Brown University; to appear *in* "Proceedings of an International Workshop on Database and Programming Languages, Roscoff, France, Septembers 1987."

EHRIG, H., AND MAHR, B. (1985), "Fundamentals of Algebraic Specification 1: Equations and Initial Semantics," Springer-Verlag, Berlin/New York.

GOGUEN, J. (1978), "Order Sorted Algebra," Technical Report, UCLA Computer Science Department (Semantics and Theory of Computation Report 14).

GOGUEN, J. (1984), Parameterized programming, *Trans. Software Engrg.* **SE-10**(5); 528–543; early versions *in* "Proceedings, Workshop on Reusability in Programming" (T. Biggerstaff and T. Cheatham, Eds.), pp. 138–150, ITT, (1983), and *in* Report Number CSLI-84-9, Center for the Study of Language and Information, Stanford University (1984).

GOGUEN, J. (1988), Modular algebraic specification of some basic geometrical constructions, in *Artifi. Intell.* **37**, 123–153 (Special Issue on Computational Geometry, J. Mundy, Ed.); also as Report CSLI-87-87, Center for the Study of Language and Information, Stanford University (1987).

GOGUEN J., AND MESEGUER, J. (1986), Remarks on remarks on many-sorted equational logic, *Bull. Eur. Assoc. Theoret. Comput. Sci.* **30**, 66–73; also in *SIGPLAN Notices* **22** (4), 41–48 (1987).

GOGUEN J., AND MESEGUER, J. (1987), Models and equality for logical programming, *in* "Proceedings, TAPSOFT '87" (H. Ehrig, G. Levi, R. Kowalski, and U. Montanari, Eds.), pp. 1–22, Lecture Notes in Computer Science, Volume 250, Springer-Verlag, Berlin/New York; also as Report CSLI-87-91, Center for the Study of Language and Information, Stanford University (1987).

GOGUEN, J., AND MESEGUER, J. (1992), "Order-Sorted Algebra I: Equational Deduction for Multiple Inheritance, Overloading, Exceptions and Partial Operations," *Theoret. Comput. Sci.* **105**, 217–273; also as Technical Report SRI-CSL-89-10, SRI International, Computer Science Lab (1989).

GOGUEN, J., WINKLER, T., MESEGUER, J., FUTATSUGI, K., AND JOUANNAUD, J.-P. (1993), Introducing OBJ, *in* "Applications of Algebraic Specification Using OBJ" (J. Goguen, Ed.) Cambridge University Press; also Technical Report SRI-CSL-92-03, SRI International, Computer Science Lab (1992).

GOGUEN, J., JOUANNAUD, J.-P., AND MESEGUER, J. (1985), Operational semantics of order-sorted algebra, *in* "Proceedings, 1985 International Conference on Automata, Languages an Programming" (W. Brauer, Ed.), pp. 221–231, Lecture Notes in Computer Science, Volume 194, Springer-Verlag, Berlin/New York.

GOGUEN, J., KIRCHNER, C., KIRCHNER, H., MÉGRELIS, A., MESEGUER, J., AND WINKLER, T.

(1988), An introduction to OBJ3, *in* "Proceedings, Conference on Conditional Term Rewriting, Orsay, France, July 1987" (J.-P. Jouannaud and S. Kaplan, Eds.), pp. 258–263, Lecture Notes in Computer Science, Vol. 308, Springer-Verlag, Berlin/New York.

GOGUEN, J., THATCHER, J., AND WAGNER, E. (1976), "An initial algebra approach to the specification, correctness and implementation of abstract data types," Technical Report RC 6487, IBM Watson Research Center, October, 1976; appears *in* "Current Trends in Programming Methodology, IV" (R. Yeh, Ed.), pp. 80–149, Prentice-Hall, Englewood Cliffs, NJ (1978).

HUET, G. (1980), Confluent reductions: Abstract properties and applications to term rewriting systems, *J. Assoc. Comput. Mach.* **27**, 797–821; preliminary version in 18th Symposium on Mathematical Foundations of Computer Science, 1977."

KIRCHNER, C., KIRCHNER, H., AND MESEGUER, J. (1987), Operational semantics of OBJ3, *in* "Proceedings, ICALP '87" (T. Lepistö and A. Salomaa, Eds.), pp. 287–301, Lecture Notes in Computer Science, Vol. 317, Springer-Verlag, Berlin/New York.

MESEGUER, J., AND GOGUEN, J. (1985), Initiality, induction and computability, *in* "Algebraic Methods in Semantics" (M. Nivat and J. C. Reynolds, Eds.), pp. 459–541, Cambridge Univ. Press, London/New York; also as SRI Computer Science Lab Technical Report CSL-140 (1983).

MESEGUER, J., AND GOGUEN, J. (1993), "Order-Sorted Algebra II," Technical Report, SRI International, Computer Science Lab (to appear).

MESEGUER, J., GOGUEN, J., AND SMOLKA, G. (1989), Order-sorted unification, *J. Symbolic Comput.* **8**, 383–413; also as Technical Report CSLI-87-86, Center for the Study of Language and Information, Stanford University (1987).

REYNOLDS, J. (1980), Using category theory to design implicit conversions and generic operators, *in* "Semantics Directed Compiler Generation" (N. D. Jones, Ed.), pp. 211–258, Lecture Notes in Computer Science, Vol, 94, Springer-Verlag, Berlin/New York.

SMOLKA, G., NUTT, W., GOGUEN, J., AND MESEGUER, J. (1989), Order-sorted equational computation, *in* "Resolution of Equations in Algebraic Structures" (M. Nivat and Hassan Ait-Kaci, Eds.), Vol. 2, pp. 297–367. Academic Press, San Diego; preliminary version *in* "Proceedings, Colloquium on the Resolution of Equations in Algebraic Structures, Lakeway, Texas, May 1987."

WADLER, P. (1987), Views: A way for pattern matchings to cohabit with data abstraction, *in* "Proceedings, 14th Symposium on Principles of Programming Languages" (S. Munchnik, Ed.), pp. 307–312, Assoc. Comp. Mach., New York.

Printed in Belgium
Uitgever: Academic Press, Inc.
Verantwoordelijke uitgever voor België:
Hubert Van Maele
Altenastraat 20, B-8310 Sint-Kruis