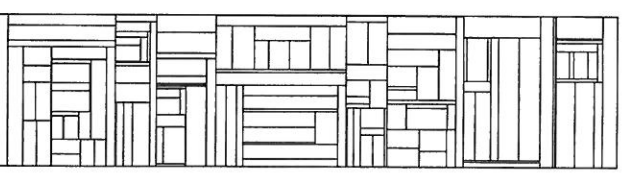
ISSN 0105-8517

Unified Algebras and Institutions

Peter D. Mosses

DAIMI PB – 274
February 1989

AARHUS UNIVERSITY
COMPUTER SCIENCE DEPARTMENT
Ny Munkegade 116 – DK 8000 Aarhus C – DENMARK
Telephone: + 45 86 12 71 88 *Telex: 64767 aausci dk*



Abstract

A novel framework for algebraic specification of abstract data types is introduced. It involves so-called "unified algebras", where sorts are treated as values, so that operations may be applied to sorts as well as to the elements that they classify.

An institution for unified algebras is defined, and shown to be liberal. However, the ordinary "forgetful" functor does not forget any values in unified algebras, so the usual data constraints do not have any models. A "more forgetful" functor is introduced and used to define so-called "bounded" data constraints, which have the expected models.

©IEEE. A reformatted version of this report is to be presented at *LICS'89, Fourth Annual Symposium on Logic in Computer Science*, to be held 5–8 June, 1989, in Asilomar, California. Citations should refer to the Proceedings.

The author welcomes comments on the framework of unified algebras, and collaboration in its further development. The address for electronic mail is: `pdm@daimi.DK`.

1 Introduction

The reader is assumed to be familiar with the basic notions of Category Theory (category, functor, initiality, left adjoint [1,16]) and Lattice Theory (distributive lattice, bottom, atom [13]). Familiarity with the many-sorted and order-sorted frameworks for algebraic specification [2,4,6,9,11,30], and with the notion of an institution [3,7,8] is advantageous, but not essential.

A *unified algebra* is a kind of total homogeneous algebra, i.e., a set equipped with some total functions. Particular unified algebras represent *data types*; classes of unified algebras represent *abstract* data types. This paper establishes an institution, i.e., a logical specification framework, for specifying abstract data types as classes of unified algebras.

The carrier of a unified algebra is a distributive lattice with a bottom. The functions of the algebra always include the join and meet of the lattice, and a constant denoting the bottom of the lattice. All functions are required to be monotone with respect to the partial order of the lattice.

The main idea is that the values in the carrier of a unified algebra represent not only *elements* of data, but also *classifications* of elements into *sorts*. For instance, a unified algebra representing a data type of numbers and lists would have values not only for particular numbers and particular lists, but also for the sort of all numbers and the sort of all lists.

The lattice partial order of the carrier represents sort inclusion; the join and meet operations represent sort union and intersection. The bottom of the lattice represents the empty sort. The empty sort, shunned in conventional algebraic frameworks, provides a particularly natural way of representing the lack of result of partial operations—avoiding the need to introduce special "error" elements. Operations need not preserve the empty sort. Readers familiar with Scott's Domain Theory [27] should note that the lattice partial order does *not* correspond to computational approximation, nor does the bottom of the lattice correspond to divergence.

When the carrier of a unified algebra is a power set, the algebra is called a *power algebra*. Thus the carrier of a power algebra is a complete lattice, although in general the carriers of unified algebras need not be even chain-complete. In a power algebra, there are values representing all

possible classifications, and the values that represent the elements (the singletons) are "atoms", i.e., just above the bottom of the lattice. Note that neither of these properties hold for unified algebras in general.

As sorts are values in unified algebras, they may be the arguments and results of operations. Ordinary operations on elements extend monotonically ("element-wise") to sorts. For example, the operation of doubling a number extends to map the sort of all integers to the sort of all even integers—and to map the empty sort to itself.

The extension of operations to sorts allows many useful classifications of elements to be expressed directly, without naming them by constants. For example, applying the successor operation to the sort of natural numbers gives the sort of positive integers; applying negation to the positive integers gives the negative integers; applying sort union to the natural numbers and the negative integers gives all the integers; and so on.

Now consider operations that represent the sort constructors of "dependent" and "generic" data types. A simple example is provided by the operation that maps each integer (element) i to the sort of all integers up to i. More interesting is the operation that maps a natural number n and a sort s to the sort of all lists of length n with components in s: the sort of unbounded lists with components in s is given simply by applying this operation to the entire *sort* of natural numbers, instead of to a particular element n. The sort of lists of length n is a subsort of the sort of unbounded lists, by monotonicity. "Polymorphic" operations on arbitrary lists specialize to operations on lists with particular sorts of components, just by the restriction of arguments to particular subsorts.

There is a further consequence of treating sorts as values: a sort that classifies several elements may be regarded as a *nondeterministic choice* between those elements. Sort union corresponds to (binary) choice; the empty sort corresponds to the impossible choice; and single elements correspond to "Hobson's choices". Notice that the monotonicity of operations is essential: adding further choices for an argument must not remove choices for the result! (The development of unified algebras originated from the observation that sorts correspond to nondeterministic values. For further discussion of the origins of unified algebras, see [21,22].)

From the above discussion, it is evident that unified algebras should provide a rather general framework for algebraic specification of abstract data types—especially regarding the treatment of polymorphic operations and generic data types. Also, it has been shown elsewhere [22] that uni-

fied algebras can be provided with a particularly simple form of modular specification, such that instantiation of generic data types is simply specialization, and mutually-dependent modules are allowed. Thus the detailed study of the foundations of unified algebras seems well-motivated.

This paper proceeds as follows. Section 2 establishes notation, providing an institution for homogeneous structures with first-order specifications. Section 3 defines an institution for power algebras with first-order specifications. Power algebras do not support the "initial algebra approach" to specification at all: in general, initial power algebras satisfying specifications do not exist—even when specifications are restricted to Horn clauses. Section 4 defines an institution for unified algebras with Horn clause specifications. This institution is shown to be "liberal"; however, the homogeneity of unified algebras prevents the straightforward use of "data constraints" in specifications of generic data types. It is necessary to introduce a novel kind of constraint, which simulates heterogeneity by exploiting the structure of unified algebras. Section 5 provides some (very basic) examples of unified algebraic specifications. Section 6 concludes by relating unified algebras to other frameworks, and pointing out some unresolved questions about unified algebras.

2 Notation

Following Goguen and Burstall [7,8], a logical specification framework may be formalized as an "institution". An institution I consists of

- a category $\mathbf{Sign}_I$ (of so-called *signatures*);
- a functor $\mathbf{Mod}_I : \mathbf{Sign}_I \to \mathbf{Cat}^{op}$ (giving a category of *models* for each signature, and a functor between categories of models for each signature morphism);
- a functor $\mathbf{Sen}_I : \mathbf{Sign}_I \to \mathbf{Set}$ (giving a set of *sentences* for each signature, and a translation function between sentences for each signature morphism); and
- a relation $\models_I$ (of *satisfaction* between models and sentences)

such that the following *Satisfaction Condition* holds:

- for any signature morphism $\phi : \Omega \to \Omega'$ in $\mathbf{Sign}_I$, for any sentence γ in $\mathbf{Sen}_I(\Omega)$ and model M' in $\mathbf{Mod}_I(\Omega')$,

$$M' \models_I \mathbf{Sen}_I(\phi)(\gamma) \iff \mathbf{Mod}_I(\phi)(M') \models_I \gamma \ .$$

(Sometimes we write just $\phi(\gamma)$ for $\mathbf{Sen}_I(\phi)(\gamma)$.)

An institution HFO for homogeneous first-order specifications (without equality) is defined below. HFO could also be obtained (essentially) by restricting the many-sorted first-order institution FO [8] to singleton sort sets, but let us take the chance to simplify notation by eradicating sort-indexed sets altogether.

First, let **Symbol** be the set of *symbols* used to name functions and predicates, partitioned by *rank* into disjoint subsets $\mathbf{Symbol}_n, n \geq 0$. We refer to functions of rank 0 as *constants*, and to functions of positive rank as *operations*. It is convenient to use "mix-fix" notation for operation symbols, indicating the number and positions of arguments by occurrences of a place-holder written '_'. Let **Variable** be a set of *variables*, disjoint from **Symbol**.

A *homogeneous first-order signature* Ω is a pair $\langle \Sigma, \Pi \rangle$ where $\Sigma \subseteq \mathbf{Symbol}$ is a set of function symbols, and $\Pi \subseteq \mathbf{Symbol}$ is a set of predicate symbols. We write Σ_n for $\Sigma \cap \mathbf{Symbol}_n$, for $n \geq 0$; similarly for Π_n. A *homogeneous first-order signature morphism* $\phi : \Omega \to \Omega'$ is a pair $\langle \sigma : \Sigma \to \Sigma', \pi : \Pi \to \Pi' \rangle$ of rank-preserving maps. The obvious category of homogeneous first-order signatures gives $\mathbf{Sign}_{\mathrm{HFO}}$.

A *homogeneous Ω-structure* A consists of a carrier set $|A|$ together with for each $\sigma \in \Sigma_n$ a function $\sigma_A : |A|^n \to |A|$, and for each $\pi \in \Pi_n$ a predicate $\pi_A \subseteq |A|^n$. A *homogeneous Ω-homomorphism* $h : A \to B$ is a function from $|A|$ to $|B|$ such that for any $n \geq 0$ and $a_1, \ldots, a_n \in |A|$,

- $h(\sigma_A(a_1, \ldots, a_n)) = \sigma_B(h(a_1), \ldots, h(a_n))$
  for all $\sigma \in \Sigma_n$, and

- $\pi_A(a_i, \ldots, a_n)$ implies $\pi_B(h(a_1), \ldots, h(a_n))$
  for all $\pi \in \Pi_n$.

This gives for each Ω a category $\mathbf{Mod}_{\mathrm{HFO}}(\Omega)$ of homogeneous Ω-structures, i.e., models.

Henceforth, all signatures and structures are assumed to be homogeneous.

For any signature morphism $\phi : \Omega \to \Omega'$ and Ω'-structure A', the *ϕ-reduct* of A', written $A' \dagger \phi$, is the Ω-structure A defined by taking $|A|$ to be $|A'|$, σ_A to be $\phi(\sigma)_{A'}$ for $\sigma \in \Sigma$, and π_A to be $\phi(\pi)_{A'}$ for $\pi \in \Pi$. The ϕ-reduct of a Ω'-homomorphism $h' : A' \to B'$ is just h' regarded as an Ω-homomorphism from $A' \dagger \phi$ to $B' \dagger \phi$. Defining $\mathbf{Mod}_{\mathrm{HFO}}(\phi)$ to be the mapping $\_ \dagger \phi : \mathbf{Mod}_{\mathrm{HFO}}(\Omega') \to \mathbf{Mod}_{\mathrm{HFO}}(\Omega)$ makes $\mathbf{Mod}_{\mathrm{HFO}}$ into a functor, called the *forgetful functor* induced by ϕ. Notice that it doesn't actually "forget" any values at all—only functions and predicates.

For any signature Ω, the set of *homogeneous first-order Ω-sentences* $\mathbf{Sen}_{\mathrm{HFO}}(\Omega)$ is the set of closed first-order formulae with function and predicate symbols from Ω and variables from **Variable**. Signature morphisms $\phi : \Omega \to \Omega'$ extend naturally to maps $\mathbf{Sen}_{\mathrm{HFO}}(\phi) : \mathbf{Sen}_{\mathrm{HFO}}(\Omega) \to \mathbf{Sen}_{\mathrm{HFO}}(\Omega')$, leaving variables unchanged, which makes $\mathbf{Sen}_{\mathrm{HFO}}$ into a functor.

The satisfaction relation $\models_{\mathrm{HFO}}$ between homogeneous first-order sentences and structures is defined as usual. Verification of the Satisfaction Condition is essentially just a special case of that for the many-sorted first-order institution FO [8], since it corresponds to restricting attention to one-sorted signatures.

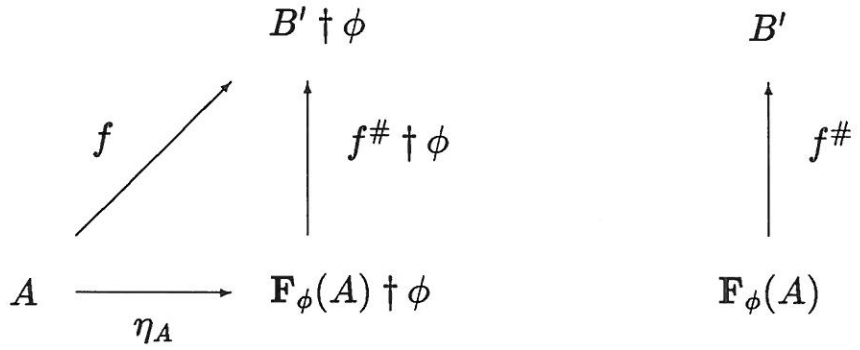
Now recall the following notions concerning arbitrary institutions I. An *Ω-theory presentation* is a pair $\langle \Omega, \Gamma \rangle$ where Ω is in $\mathbf{Sign}_I$ and Γ is a set of sentences in $\mathbf{Sen}_I(\Omega)$. The *closure* of $\langle \Omega, \Gamma \rangle$ is $\langle \Omega, \Gamma' \rangle$ where Γ' is the set of those Ω-sentences that are satisfied by every Ω-model that happens to satisfy all the sentences in Γ. An *Ω-theory* T is an Ω-theory presentation $\langle \Omega, \Gamma \rangle$ that is its own closure. When T is an Ω-theory, let $\mathbf{Mod}_I(T)$ denote the full subcategory of $\mathbf{Mod}_I(\Omega)$ of all Ω-models that satisfy Γ.

In practice we use (finite) theory presentations to specify theories, and hence classes of models.

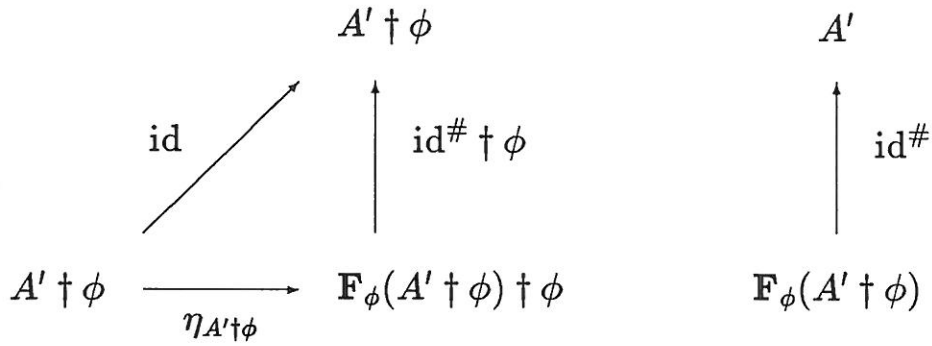
A *theory morphism* $\phi : T \to T'$ (where $T = \langle \Omega, \Gamma \rangle$ and $T' = \langle \Omega', \Gamma' \rangle$) is a signature morphism $\phi : \Omega \to \Omega'$ such that $\phi(\Gamma) \subseteq \Gamma'$. The *forgetful functor* $\_ \dagger \phi : \mathbf{Mod}_I(\Omega') \to \mathbf{Mod}_I(\Omega)$ cuts down to $\_ \dagger \phi : \mathbf{Mod}_I(T') \to \mathbf{Mod}_I(T)$ (by the Satisfaction Condition).

An institution is called *liberal* when for every theory morphism $\phi : T \to T'$ the forgetful functor $\_ \dagger \phi : \mathbf{Mod}_I(T') \to \mathbf{Mod}_I(T)$ has a left adjoint, which we denote by $\mathbf{F}_\phi$. I.e., for every A in $\mathbf{Mod}_I(T)$ there is $\mathbf{F}_\phi(A)$ in $\mathbf{Mod}_I(T')$ with a morphism $\eta_A : A \to \mathbf{F}_\phi(A) \dagger \phi$ in $\mathbf{Mod}_I(T)$ having the property that for each B' in $\mathbf{Mod}_I(T')$ and morphism f :

$A \to B' \dagger \phi$ in $\mathbf{Mod}_I(T)$, there is a *unique* morphism $f^\# : \mathbf{F}_\phi(A) \to B'$ in $\mathbf{Mod}_I(T')$ such that the left part of the following diagram commutes:



A model A' of T' is called *ϕ-free* if it is naturally isomorphic to $\mathbf{F}_\phi(A' \dagger \phi)$, the free model of T' generated by $A' \dagger \phi$. I.e., ${\mathrm{id}_{A' \dagger \phi}}^\#$ is an isomorphism:



An *Ω-data constraint* is a pair $\langle \phi : T'' \to T',\ \theta : \Omega' \to \Omega \rangle$, where ϕ is a theory morphism, θ is a signature morphism, and Ω' is the signature of T'. An Ω-structure A *satisfies* this constraint iff $A \dagger \theta$ is a model of T' and is ϕ-free. Translation of such a constraint $\langle \phi, \theta \rangle$ by a signature morphism $\psi : \Omega \to \hat{\Omega}$ is defined to be $\langle \phi : T'' \to T',\ \psi \circ \theta : \Omega' \to \hat{\Omega} \rangle$.

3 Power Algebras

An institution POW of first-order power algebra specifications is defined as follows.

A *power signature* is a homogeneous first-order signature $\langle \Sigma, \Pi^\circ \rangle$ where $\Sigma \supseteq \Sigma^\circ = \{\mathsf{nothing}, \_|\_, \_\&\_\}$, and $\Pi^\circ = \{\_=\_, \_\leq\_, \_:\_\}$. Let $\mathbf{Sign}_{\mathrm{POW}}$ be the subcategory of $\mathbf{Sign}_{\mathrm{HFO}}$ whose objects are the power signatures, with all the morphisms that preserve the given symbols. Henceforth, let Σ be a power signature (leaving the predicate symbols Π° implicit).

A *power Σ-algebra* A is a homogeneous Σ-structure A such that:

- $|A| = \mathcal{P}(U_A)$, for some set U_A (called the *universe* of A).
- $\mathsf{nothing}_A = \emptyset$; $\_ \mid_A \_$ is set union; and $\_ \&_A \_$ is set intersection.
- σ_A preserves set inclusion, for each $\sigma \in \Sigma$.
- $x =_A y$ holds iff x is the same set as y.
- $x \leq_A y$ holds iff x is a subset of y.
- $x :_A y$ holds iff x is a singleton subset of y.

Let $\mathbf{Mod}_{\mathrm{POW}}(\Sigma)$ be the full subcategory of $\mathbf{Mod}_{\mathrm{HFO}}(\Sigma)$ whose objects are the power Σ-algebras. Notice that a homomorphism between power algebras maps singletons to singletons (since x is a singleton iff $x : x$ holds) and preserves subset inclusions.

For sentences, let $\mathbf{Sen}_{\mathrm{POW}}(\Sigma)$ be $\mathbf{Sen}_{\mathrm{HFO}}(\Sigma)$, and let satisfaction $\models_{\mathrm{POW}}$ be the restriction of $\models_{\mathrm{HFO}}$ to power algebras.

The singleton subsets in the carrier of a power algebra represent elements of a data type; *all* subsets represent classifications of these elements. Thus with power algebras, operations may be applied to classifications as well as to elements.

It might seem that the institution POW of power algebras provides an appropriate framework for specification of data types, and that further generality would be superfluous. However, POW does not support the "initial algebra" approach, which is advocated in [2,4,6,9,11,30] and generalized to the use of data (or "initial") constraints in [3,7,8,24,25].

According to the initial algebra approach, so-called "junk" (unnecessary values) and "confusion" (unnecessary identifications between values) are prohibited by stipulating that models have to be *initial* among those that satisfy the specified sentences.

An immediate reason why POW does not support initial algebras is that the sentences are arbitrary first order sentences, whereas universal Horn clauses are the most general sentences that allow initial models [17]. But even if the sentences in POW were restricted to be universal Horn clauses, the resulting institution still would not provide initial algebras, as the following Proposition shows.

Proposition 1 *There are presentations $\langle \Sigma, \Gamma \rangle$, where Γ is a set of ground formulae, such that the class of power Σ-algebras satisfying Γ does* not *have an initial algebra.*

Proof: Take $\Sigma = \Sigma^\circ \cup \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ and $\Gamma = \{\mathrm{a{:}a}, \mathrm{b{:}b}, \mathrm{c{:}a} \mid \mathrm{b}\}$. Suppose A satisfies Γ; then a_A, b_A, and c_A must all be singletons, with $\mathrm{c}_A \subseteq \mathrm{a}_A \cup \mathrm{b}_A$. This forces (at least one of) $\mathrm{c}_A = \mathrm{a}_A$ or $\mathrm{c}_A = \mathrm{b}_A$. Suppose (w.l.o.g.) that $\mathrm{c}_A = \mathrm{a}_A$; then A cannot be initial, as there is no power Σ-homomorphism from A to any model B with $\mathrm{c}_A \neq \mathrm{a}_A$. ∎

By the way, notice that it is the mixture of the extensionality of sets with the "no confusion" that gives the problem with initial power algebras: the "no junk" part of initiality is all right.

The lack of initial power algebras satisfying even very basic specifications does not prevent the use of power algebras for specifying abstract data types: instead of using initiality to eliminate "junk" and "confusion", one may use "reachability constraints" [26] to dispose of "junk", and then use first-order sentences (inequalities, etc.) to eliminate "confusion".

Instead, however, let us *generalize* the structure of power algebras enough to ensure the existence of initial algebras, so as to support the initial algebra approach. The appropriate generalization seems to be unified algebras, as defined in the next section.

4 Unified Algebras

An institution UNI for specification of unified algebras is defined as follows.

A *unified signature* is just a power signature. So let $\mathbf{Sign}_{\mathrm{UNI}}$ be $\mathbf{Sign}_{\mathrm{POW}}$.

A *unified* Σ*-algebra* A is a homogeneous Σ-structure such that:

- $|A|$ is a *distributive lattice* with $\_\mid_A\_$ as join, $\_\&_A\_$ as meet, and $\mathsf{nothing}_A$ as bottom.
- There is a distinguished set of values $E_A \subseteq |A|$ (the *elements* of A).
- σ_A is monotone with respect to the partial order of the lattice, for each $\sigma \in \Sigma$.
- $x =_A y$ holds iff x is identical to y.
- $x \leq_A y$ holds iff $x \mid_A y =_A y$ (i.e., $\leq_A$ is the partial order of the lattice).

- $x :_A y$ holds iff $x \in E_A$ and $x \leq_A y$.

Note that the elements E_A need *not* be the "atoms" of the lattice, i.e., "just above" the bottom. In practice E_A is usually discrete (as in power algebras); but there are cases (e.g., lists with nondeterministic components) where elements may be included in other elements, so we leave the structure of E_A open.

Let $\mathbf{Mod}_{\mathrm{UNI}}(\Sigma)$ be the full subcategory of $\mathbf{Mod}_{\mathrm{HFO}}(\Sigma)$ whose objects are the unified Σ-algebras.

For sentences, let $\mathbf{Sen}_{\mathrm{UNI}}(\Sigma)$ be the restriction of $\mathbf{Sen}_{\mathrm{HFO}}(\Sigma)$ to universal Horn clauses. Letting $\models_{\mathrm{UNI}}$ be the restriction of $\models_{\mathrm{HFO}}$ to unified algebras and universal Horn clauses completes the definition of the institution UNI.

Proposition 2 *UNI is a liberal institution.*

Proof: It is known that the institution HORN of many-sorted Horn clause specifications with equality is liberal [8]. We may restrict HORN specifications to one sort, giving a liberal institution of Horn clause specifications of (essentially) homogeneous structures, with $\_=\_$ interpreted as identity. The liberality of this institution transfers to UNI by characterizing the structure of unified algebras by Horn clauses, as follows.

Let Σ be a unified signature. Let Γ° be the set of Horn clauses shown in Table 1 (where conjunction is denoted by ';', and the universal quantifiers are left implicit, and conjunction of hypotheses is denoted by ';'). Then for any set Γ of unified Σ-sentences, the unified algebras satisfying Γ correspond to the homogeneous $\langle \Sigma, \Pi^\circ \rangle$-structures satisfying $\Gamma \cup \Gamma^\circ$. ∎

Corollary 3 *For any unified signature Σ and set of unified Σ-sentences Γ, the class of unified Σ-algebras that satisfy Γ has an initial algebra.*

Proof: Let T° be the theory given by the closure of the presentation $\langle \Sigma^\circ, \emptyset \rangle$. Any one-point unified Σ°-algebra I is an initial algebra in the class of models of T°. Now let T' be the closure of the presentation $\langle \Sigma, \Gamma \rangle$ and consider the inclusion $\phi : T^\circ \to T'$. Clearly ϕ is a unified theory morphism. By the liberality of UNI, the forgetful functor $\_ \dagger \phi : \mathbf{Mod}_{\mathrm{UNI}}(T') \to \mathbf{Mod}_{\mathrm{UNI}}(T^\circ)$ has a left adjoint, $\mathbf{F}_\phi$. Let A be $\mathbf{F}_\phi(I)$. The initiality of A in $\mathbf{Mod}_{\mathrm{UNI}}(T')$ follows from the initiality of I in $\mathbf{Mod}_{\mathrm{UNI}}(T^\circ)$. ∎

$x \leq y; y \leq x \Longrightarrow x = y$	$x \leq y; y \leq z \Longrightarrow x \leq z$
$x \leq x$	nothing $\leq x$
$x \leq z; y \leq z \Longrightarrow x \mid y \leq z$	$x \leq x \mid y \qquad y \leq x \mid y$
$z \leq x; z \leq y \Longrightarrow z \leq x \,\&\, y$	$x \,\&\, y \leq x \qquad x \,\&\, y \leq y$
$x \,\&\, (y \mid z) = (x \,\&\, y) \mid (x \,\&\, z)$	$x \mid (y \,\&\, z) = (x \mid y) \,\&\, (x \mid z)$
$x : x; x \leq y \Longrightarrow x : y$	
$x : y \Longrightarrow x : x$	$x : y \Longrightarrow x \leq y$
for each $\sigma \in \Sigma_n$, $n \geq 1$, the n clauses:	
$x_i \leq x_i' \Longrightarrow \sigma(x_1, \ldots, x_i, \ldots, x_n) \leq \sigma(x_1, \ldots, x_i', \ldots, x_n)$	

Table 1: The structure of unified Σ-algebras

Now reconsider the specification $\langle \Sigma, \Gamma \rangle$ from Proposition 1. With unified algebras, the relation $c{:}a \,|\, b$ between the elements does not force any of the constants to be identified. (It is an instructive exercise to draw the carrier of the initial unified algebra that satisfies the specification.)

But the liberality of UNI is not much help for data constraints, as the ϕ-reduct functor *never forgets any values*—only functions. Models that are ϕ-free (i.e., *naturally* isomorphic to the free model over their ϕ-reduct) *do not exist*, except in trivial cases. Next we consider a solution to this problem, to obtain a useful version of data constraints for unified algebras.

For each $\phi : \Sigma \to \Sigma'$ in $\mathbf{Sign}_{\mathrm{UNI}}$ we define a functor $\_\ddagger\phi : \mathbf{Mod}_{\mathrm{UNI}}(\Sigma') \to \mathbf{Mod}_{\mathrm{UNI}}(\Sigma)$. The idea is that $\_\ddagger\phi$ is "more forgetful" than $\_\dagger\phi$, in that it forgets not only functions but also (some) values. This is done essentially by regarding ground terms as "sorts", and considering the elements that are included in them.

Suppose for simplicity that ϕ is merely an inclusion of Σ in Σ': then the only elements of a Σ'-model that $\_ \ddagger \phi$ retains are those that are included in the values of ground Σ-terms; values that are not elements are retained only if they are denotable by ground Σ-terms, or by applying Σ-operations to retained values.

Now let $\phi : \Sigma \to \Sigma'$ be any unified signature morphism. The general definition of the *more forgetful functor* $\_ \ddagger \phi$ is as follows. Let A' be in $\mathbf{Mod}_{\mathrm{UNI}}(\Sigma')$. We take $A' \ddagger \phi$ to be the unified Σ-algebra A determined by:

- $E_A = \{x \in E_{A'} \mid x \leq_{A'} \phi(t)_{A'}$
  for some ground Σ-term t $\}$.

- $|A|$ is the least set that includes E_A such that when $\sigma \in \Sigma_n$ and $x_1, \ldots, x_n \in |A|$ then also $\phi(\sigma)_{A'}(x_1, \ldots, x_n) \in |A|$.
- $\sigma_A = \phi(\sigma)_{A'}$ for each $\sigma \in \Sigma$.

Further, for each unified Σ'-homomorphism $h' : A' \to B'$, let $h' \ddagger \phi$ be h' regarded as a Σ-homomorphism from $A' \ddagger \phi$ to $B' \ddagger \phi$. This makes $\_ \ddagger \phi$ into a functor.

The next proposition shows what happens when ϕ is not just a signature morphism but a unified theory morphism.

Proposition 4 *If $\phi : T \to T'$ is a unified theory morphism then $\_ \ddagger \phi$ maps models of T' to models of T.*

Proof: Let A' be a model of T'. We know that $A' \dagger \phi$ is a model of T. But $A' \ddagger \phi$ is as $A' \dagger \phi$, except that some values have been forgotten. Thus the universally-quantified Horn clauses in T hold *a fortiori* for $A' \ddagger \phi$, which is therefore a model of T. ∎

We now exploit $\_ \ddagger \phi$ to define new constraints, called "bounded data constraints". A Σ-structure A *satisfies* the *bounded data constraint* $\langle \phi : T'' \to T', \theta : \Sigma' \to \Sigma \rangle$ iff $A \ddagger \theta$ is a model of T' and naturally isomorphic to $\mathbf{F}_\phi(A \ddagger \theta \ddagger \phi)$. Translation of bounded data constraints by signature morphisms is defined as for ordinary data constraints.

Finally, we show that satisfaction of bounded data constraints is invariant under change of signature, so that we may get an institution for specifying unified algebras where sentences may be bounded data constraints as well as universal Horn clauses.

Lemma 5 *If $\psi : \Sigma \to \hat{\Sigma}$, and $\langle \phi, \theta \rangle$ is a bounded data constraint (as above), then for any $\hat{\Sigma}$-unified algebra $\hat{B}$, $\hat{B} \ddagger (\psi \circ \theta) = (\hat{B} \dagger \psi) \ddagger \theta$.*

Proof: By calculation. We have

$$\begin{aligned} E_{\hat{B} \ddagger (\psi \circ \theta)} &= \{x \in E_{\hat{B}} \mid x \leq_{\hat{B}} (\psi \circ \theta)(t)_{\hat{B}}\} \\ &= \{x \in E_{\hat{B}} \mid x \leq_{\hat{B}} \psi(\theta(t))_{\hat{B}}\} \\ &= \{x \in E_{\hat{B} \dagger \psi} \mid x \leq_{\hat{B} \dagger \psi} \theta(t)_{\hat{B} \dagger \psi}\} \\ &= E_{(\hat{B} \dagger \psi) \ddagger \theta}. \end{aligned}$$

Similarly for the values and functions. ∎

Lemma 6 *If $\psi : \Sigma \to \hat{\Sigma}$ is a unified signature morphism, if $\langle \phi : T'' \to T', \theta : \Sigma' \to \Sigma \rangle$ is a bounded data constraint, and if $\hat{B}$ is a $\hat{\Sigma}$-unified algebra, then $\hat{B}$ satisfies $\langle \phi, \psi \circ \theta \rangle$ iff $\hat{B} \dagger \psi$ satisfies $\langle \phi, \theta \rangle$.*

Proof: $\hat{B}$ satisfies $\langle \phi, \psi \circ \theta \rangle$ iff $\hat{B} \ddagger (\psi \circ \theta)$ is a model of T' and ϕ-free. $\hat{B} \dagger \psi$ satisfies $\langle \phi, \theta \rangle$ iff $(\hat{B} \dagger \psi) \ddagger \theta$ is a model of T' and ϕ-free. By the previous lemma, $\hat{B} \ddagger (\psi \circ \theta) = (\hat{B} \dagger \psi) \ddagger \theta$. ∎

Theorem 7 *The extension of UNI by allowing bounded data constraints as sentences gives an institution, UNICON.*

Proof: Lemma 6 verifies the Satisfaction Condition for bounded data constraints, and the result follows from a general theorem about adding constraints as sentences [8, Proposition 40] ∎

Some examples of UNICON specifications are given in the next section.

5 Examples

The following simple examples should give an idea of what unified specifications look like. For examples of modular unified specifications using a more practical notation, see [22].

Truth Values

$$\Sigma^{Tr} = \Sigma^{\circ} \cup \{\mathsf{Tr}, \mathsf{true}, \mathsf{false}, \mathsf{if\_then\_else\_}\}$$

$$\begin{aligned}
\Gamma^{Tr} = \{&\mathsf{true} : \mathsf{Tr}, \quad \mathsf{false} : \mathsf{Tr},\\
&\mathsf{Tr} = \mathsf{true} \mid \mathsf{false},\\
&\mathsf{nothing} = \mathsf{true} \,\&\, \mathsf{false},\\
&\mathsf{if\ true\ then}\ X\ \mathsf{else}\ Y = X,\\
&\mathsf{if\ false\ then}\ X\ \mathsf{else}\ Y = Y,\\
&\mathsf{if\ nothing\ then}\ X\ \mathsf{else}\ Y = \mathsf{nothing},\\
&\mathsf{if}\ (T \mid U)\ \mathsf{then}\ X\ \mathsf{else}\ Y =\\
&\quad (\mathsf{if}\ T\ \mathsf{then}\ X\ \mathsf{else}\ Y) \mid\\
&\quad (\mathsf{if}\ U\ \mathsf{then}\ X\ \mathsf{else}\ Y),\\
&(T \,\&\, \mathsf{Tr}) = \mathsf{nothing} \implies\\
&\quad (\mathsf{if}\ T\ \mathsf{then}\ X\ \mathsf{else}\ Y) = \mathsf{nothing}\}
\end{aligned}$$

When ϕ is the inclusion of (the closure of) $\langle \emptyset, \emptyset \rangle$ in (the closure of) $\langle \Sigma^{Tr}, \Gamma^{Tr} \rangle$, the constraint $\langle \phi, id \rangle$ restricts models to those where the only elements are true and false.

Natural Numbers

$$\Sigma^{Nat} = \Sigma^{\circ} \cup \{\mathsf{Nat}, 0, \mathsf{succ\_}, \mathsf{pred\_}, [0 \,..\, \_]\}$$

$$\begin{aligned}\Gamma^{Nat} = \{&0 : \mathsf{Nat}, \quad n : \mathsf{Nat} \Longrightarrow \mathsf{succ}\, n : \mathsf{Nat},\\ &\mathsf{succ\,nothing} = \mathsf{nothing},\\ &\mathsf{Nat} = 0 \mid \mathsf{succ\,Nat},\\ &\mathsf{nothing} = 0 \,\&\, \mathsf{succ\,Nat},\\ &\mathsf{pred}\, 0 = \mathsf{nothing}, \quad \mathsf{pred\,nothing} = \mathsf{nothing},\\ &N \leq \mathsf{Nat} \Longrightarrow \mathsf{pred\,succ}\, N = N,\\ &[0 \,..\, \mathsf{nothing}] = \mathsf{nothing},\\ &[0 \,..\, (M \mid N)] = [0 \,..\, M] \mid [0 \,..\, N],\\ &N \leq \mathsf{Nat} \Longrightarrow [0 \,..\, N] = N \mid [0 \,..\, \mathsf{pred}\, N]\}\end{aligned}$$

When ϕ is the inclusion of (the closure of) $\langle \emptyset, \emptyset \rangle$ in (the closure of) $\langle \Sigma^{Nat}, \Gamma^{Nat} \rangle$, the constraint $\langle \phi, id \rangle$ restricts models to those where the only elements are 0, $\mathsf{succ}\, 0$, $\mathsf{succ\,succ}\, 0$, ...

Lists of Data

$$\Sigma^{List} = \Sigma^{\circ} \cup \{\mathsf{List}, \mathsf{Data}, \_(\mathsf{of}\_), \mathsf{nil}, \mathsf{cons}(\_,\_)\}$$

$$\begin{aligned}\Gamma^{List} = \{&\mathsf{List} = \mathsf{List(of\ Data)}, \quad \mathsf{nil} : \mathsf{List},\\ &(d : \mathsf{Data}; l : \mathsf{List}) \Longrightarrow \mathsf{cons}(d,l) : \mathsf{List},\\ &\mathsf{cons}(d,\mathsf{nothing}) = \mathsf{nothing},\\ &\mathsf{cons}(\mathsf{nothing},l) = \mathsf{nothing},\\ &D \leq \mathsf{Data} \Longrightarrow\\ &\quad \mathsf{List(of}\ D) = \mathsf{nil} \mid \mathsf{cons}(D,\mathsf{List(of}\ D)),\\ &\mathsf{nil(of}\ D) = \mathsf{nil},\\ &\mathsf{cons}(d,l)(\mathsf{of}\ D) = \mathsf{cons}(d \,\&\, D, l(\mathsf{of}\ D))\}\end{aligned}$$

When ϕ is the inclusion of (the closure of) $\langle \{\mathsf{Data}\}, \emptyset \rangle$ in (the closure of) $\langle \Sigma^{Tr}, \Gamma^{Tr} \rangle$, the constraint $\langle \phi, id \rangle$ restricts models to those where the only elements of List are (finite) lists of Data elements.

6 Comparison with Related Work

Sort inclusions are used in the order-sorted framework of Goguen and Meseguer [6,10,30]. Unified algebras generalize order-sorted algebras by allowing operations to be applied to sorts as well as elements, and by allowing (Horn clause) axioms involving sort inclusion.

It seems that any order-sorted specification can be translated to an analogous unified specification: the unified signature is the union of the sort, constant, and operation symbols of the order-sorted signature; there is a unified clause for each sort inclusion, for each order-sorted constant and operation declaration, and for each order-sorted conditional equation (extra hypotheses are needed in these latter clauses: to restrict variables to be elements of the declared sorts).

The other way round, one could probably simulate unified algebras using order-sorted algebras: by introducing values that are *tokens* for sorts, defining truth-valued operations on these values corresponding to inclusion and classification. But it is not clear that this simulation would be convenient enough for practical use.

Smolka [28] has given a reduction of order-sorted Horn logic to unsorted Horn logic using tokens for sorts, and treating inclusion and classification as predicates. Recently he has also developed an unsorted Horn clause "type logic" [29] which is closely related to unified algebras. One difference is that his framework is based on partial algebras, so only strict operations are considered; also, he leaves union and intersection of types to be specified by the user, rather than building them into the framework (one could do that with unified algebras too, but that would make unified specifications more tedious). Another difference is that his typing relation ':' is not necessarily reflexive, so that there may be types of types—in unified algebras, the only way of classifying classifications is as subclassifications of a classification. It would be interesting to see whether a useful notion of data constraint can be provided for this partial algebra framework.

Manca and Salibra [18], also together with Scollo [19], have proposed a homogeneous partial algebra framework rather similar to that of Smolka. They make do without the inclusion relation, '$\leq$', and do not assume any properties of the typing relation. More recently they have investigated the use of total algebras for the foundations of their framework [20], but full details are not yet available.

Nondeterminism has been represented in algebraic specifications by using multi-algebras [15,23]. The power algebras introduced here (Section 3) are a simple generalization of multi-algebras.

Scott's Domain Theory [27] was originally based on complete lattices and continuous functions. Operations on domains are usually limited to universally-characterized categorical constructions. Nondeterministic domains (with linear operations) have been studied by Hennessy and Plotkin [14].

In logic programming, sorts are typically represented as predicates. (Fitting [5] shows how computability theory can be based on such a treatment.) By letting sorts be values, unified algebras allow (first-order) operations on sorts.

Future Work

Let us conclude by posing various questions about unified algebras:

1. Is UNI actually *equivalent* to the institution of order-sorted specifications?
2. What is the precise relationship between UNI and the frameworks recently proposed by Smolka and by Manca et. al.?
3. Can unified algebras be generalized to deal with higher-order operations? (What is the appropriate ordering on operations?)
4. What restrictions on UNI specifications are required to allow implementations in the style of OBJ [12]?
5. Can extra sentences always be added to unified specifications so that values are identified (only) when
   (a) they classify the same sets of elements, or
   (b) their distinction is incompatible with regarding elements as indivisible (as in power algebras)?
6. What is the appropriate notion of "correct implementation" for UNI?
7. Can bounded data constraints be relaxed so that (e.g.) the subsequent identification of distinct non-elements in constrained parts does not conflict with constraints?

The author would be grateful for any assistance with elucidating the answers to these questions.

Acknowledgements. The author is indebted to Joseph Goguen, José Meseguer, Gert Smolka, Andrzej Tarlecki, Don Sannella, Hartmut Ehrig, Bernd Mahr, Hassan Aït-Kaci, Brian Mayoh, and Michael Schwartzbach for constructive suggestions and encouragement regarding unified algebras.

References

[1] M. A. Arbib and E. G. Manes. *Arrows, Structures, and Functors.* Academic Press, 1975.

[2] R. M. Burstall and J. A. Goguen. Putting theories together to make specifications. In *Proc. 5th Int. Joint Conf. on Artificial Intelligence*, 1977.

[3] R. M. Burstall and J. A. Goguen. The semantics of CLEAR, a specification language. In *Proc. Winter School on Abstract Software Specifications, Copenhagen, 1979*, number 86 in Lecture Notes in Computer Science. Springer-Verlag, 1980.

[4] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics.* Number 6 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, 1985.

[5] M. Fitting. *Computability Theory, Semantics, and Logic Programming.* Number 13 in Oxford Logic Guides. Oxford University Press, 1987.

[6] J. A. Goguen. Order sorted algebra. Semantics and Theory of Computation Report 14, UCLA Computer Science Dept., 1978.

[7] J. A. Goguen and R. M. Burstall. Introducing institutions. In *Proc. Logics of Programming Workshop*, number 164 in Lecture Notes in Computer Science, pages 221–256. Springer-Verlag, 1984.

[8] J. A. Goguen and R. M. Burstall. Institutions: Abstract model theory for computer science. Report CSLI-85-30, Center for the Study of Language and Information, Stanford University, 1985.

[9] J. A. Goguen and J. Meseguer. Order-sorted algebra: Algebraic theory of polymorphism. *Journal of Symbolic Logic*, 51:844–845, 1986. Abstract.

[10] J. A. Goguen and J. Meseguer. Order-sorted algebra I: Partial and overloaded operators, errors and inheritance. Technical report, Computer Science Lab., SRI International, 1987.

[11] J. A. Goguen, J. W. Thatcher, and E. G. Wagner. An initial algebra approach to the specification, correctness, and implementation of abstract data types. In R. T. Yeh, editor, *Current Trends in Programming Methodology*, volume IV. Prentice-Hall, 1978.

[12] J. A. Goguen and T. Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, Computer Science Lab., SRI International, 1988.

[13] G. Grätzer. *Lattice Theory: First Concepts and Distributive Lattices.* W. H. Freeman & Co., 1971.

[14] M. C. B. Hennessy and G. D. Plotkin. Full abstraction for a simple parallel programming language. In *MFCS'79, Proc. Symp. on Math. Foundations of Computer Science*, number 74 in Lecture Notes in Computer Science. Springer-Verlag, 1979.

[15] W. H. Hesselink. A mathematical approach to nondeterminism in data types. *ACM Trans. Prog. Lang. Syst.*, 10:87–117, 1988.

[16] S. Mac Lane. *Categories for the Working Mathematician.* Number 5 in Graduate Texts in Mathematics. Springer-Verlag, 1971.

[17] B. Mahr and J. A. Makowsky. Characterizing specification languages which admit initial semantics. *Theoretical Comput. Sci.*, 31:49–59, 1984.

[18] V. Manca and A. Salibra. On the power of equational logic: Applications and extensions. Technical Report TR–19/88, Dip. di Informatica, Università di Pisa, 1978. To appear in Proc. 1st Int. Conf. on Algebraic Logic (Budapest, August 1988).

[19] V. Manca, A. Salibra, and G. Scollo. On the nature of TELLUS. Personal communication, 1988.

[20] V. Manca, A. Salibra, and G. Scollo. DELTA: A deduction system integrating equational logic and type assignment. Personal communication, 1989.

[21] P. D. Mosses. Unified algebras and action semantics. In *STACS'89, Proc. Symp. on Theoretical Aspects of Computer Science, Paderborn*, number 349 in Lecture Notes in Computer Science. Springer-Verlag, 1989.

[22] P. D. Mosses. Unified algebras and modules. In *POPL'89, Proc. 16th Ann. ACM Symp. on Principles of Programming Languages*, pages 329–343. ACM, 1989.

[23] T. Nipkow. Nondeterministic data types: Models and implementations. *Acta Inf.*, 22:629–661, 1986.

[24] H. Reichel. Initially-restricting algebraic theories. In *MFCS'80, Proc. Symp. on Math. Foundations of Computer Science*, number 88 in Lecture Notes in Computer Science, pages 504–514. Springer-Verlag, 1980.

[25] H. Reichel. *Initial Computability, Algebraic Specifications, and Partial Algebras.* Number 2 in The International Series of Monographs on Computer Science. Oxford University Press, 1987.

[26] D. Sannella and A. Tarlecki. Specifications in an arbitrary institution. *Information and Computation*, 76:165–210, 1988.

[27] D. S. Scott. Data types as lattices. *SIAM J. Comput.*, 5:522–587, 1976.

[28] G. Smolka. Order-sorted Horn logic: Semantics and deduction. SEKI Report SR–86–17, FB Informatik, Universität Kaiserslautern, 1986.

[29] G. Smolka. Type logic. Abstract for 6th Workshop on Abstract Data Types, Berlin, Aug. 1988.

[30] G. Smolka, W. Nutt, J. A. Goguen, and J. Meseguer. Order-sorted equational computation. SEKI Report SR–87–14, FB Informatik, Universität Kaiserslautern, 1987.