Hiding More of Hidden Algebra

Joseph Goguen and Grigore Roşu*

Department of Computer Science & Engineering
University of California at San Diego

1 Introduction

Behavioral specification is a rapidly advancing area of algebraic semantics that supports practical applications by allowing models (implementations) that only behaviorally satisfy specifications, infinitary data structures (such as streams), behavioral refinements, and coinduction proof methods. This paper generalizes the hidden algebra approach to allow: (P1) operations with multiple hidden arguments, and (P2) defining behavioral equivalence with a subset of operations, in addition to the already present (P3) built-in data types, (P4) nondeterminism, (P5) concurrency, and (P6) non-congruent operations. All important results generalize, but more elegant formulations use the new institution in Section 5. Behavioral satisfaction appeared 1981 in [20], hidden algebra 1989 in [9], multiple hidden arguments 1992 in [1], congruent and behavioral operations in [1, 18], behavioral equivalence defined by a subset of operations in [1], and non-congruent operations in [5]; all this was previously integrated in [21], but this paper gives new examples, institutions, and results relating hidden algebra to information hiding. We assume familiarity with basics of algebraic specification, e.g., [11, 13].

2 Basic Hidden Algebra

Definition 1. A *hidden signature* is (Ψ, D, Σ), often written just Σ, where Ψ is a V-sorted signature, D is a Ψ-algebra called the *data algebra*, Σ is a $(V \cup H)$-sorted signature extending Ψ such that each operation in Σ with both its arguments and result in V lies in Ψ, and V and H are disjoint sets, called *visible sorts* and *hidden sorts*, respectively. For technical reasons (e.g., see [12]), we assume each element d in D is denoted by exactly one constant in Ψ, also denoted d.

A *hidden subsignature of* Σ is a hidden signature (Ψ, D, Γ) with $\Gamma \subseteq \Sigma$. A *behavioral*, or *hidden*, Σ*-specification* or *theory* is (Σ, Γ, E), where Σ is a hidden signature, Γ is a hidden subsignature of Σ, and E is a set of Σ-equations. Operations in $\Gamma - \Psi$ may be called *behavioral* [6] or *observational* [1, 2].

A *hidden* Σ*-algebra* is a many sorted Σ-algebra A such that $A|_\Psi = D$. □

An adequate discussion of the complex historical and technical relations among the many approaches to behavioral specification is not possible in this

* On leave from Fundamentals of Computer Science, Faculty of Mathematics, University of Bucharest, Romania.

J. Wing, J. Woodcock, J. Davies (Eds.): FM'99, Vol. II, LNCS 1709, pp. 1704–1719, 1999.

short paper, but we do our best to be accurate, if not comprehensive. We drop the restriction of [9, 12] to operations with at most one hidden argument. Operations with hidden arguments may be called *attributes* if the result is visible, and *methods* if it is hidden; those with visible arguments and hidden result are called *hidden constants*. Behavioral operations are used in experiments to distinguish states; i.e., they define behavioral equivalence. Note that our models do not require all operations to be congruent (see Definition 5) as in [15, 18], since non-congruent operations are needed for applications like `length` for lists implemented as sets, and the `push` operation in Example 2. Example 3 gives a spec equivalent to that in Example 1, with `in` as its only behavioral operation, thus illustrating the need for (P2). Our models also satisfy (P3), having a fixed subalgebra of data values, as distinct from observational logic ([1, 2], etc.) and CafeOBJ [6]. This is desirable because real applications use standard Booleans and integers rather than arbitrary models of some theory; however, all results of this paper still hold for the fully loose semantics, and there are applications where it is useful, although we feel these are better handled by parameterization. Coalgebra is an elegant related approach (e.g., [16]) that lacks nondeterminism, multiple hidden arguments, and other features; one symptom of the difference is that final algebras no longer exist for our generalization. Set union is one natural example motivating (P1); there are many others. If sets are objects with hidden state, then operations like union have two hidden arguments:

Example 1. We can specify sets using CafeOBJ syntax[1] [6] as follows:

```
mod* SET {  *[ Set ]*  pr(NAT)
  bop _in_  : Nat Set -> Bool ** attribute
  op empty  : -> Set          ** hidden const
  bop add   : Nat Set -> Set  ** method
  bop _U_   : Set Set -> Set  ** 2 hidden args
  bop _&_   : Set Set -> Set  ** 2 hidden args
  bop neg   : Set -> Set      ** method
  vars N N' : Nat  vars X X' : Set
  eq N in empty = false .
  eq N in add(N',X) = (N == N') or (N in X) .
  eq N in (X U X')  = (N in X)  or (N in X') .
  eq N in (X & X')  = (N in X) and (N in X') .
  eq N in neg(X)    =  not (N in X) . }
```

Here "`*[Set]*`" declares `Set` a hidden sort, "`bop`" declares behavioral operations, and "`pr(NAT)`" imports the module `NAT` of natural numbers in "protecting" mode, i.e., so that the naturals are not compromised. The constant `empty` is the only non-behavioral operation, indicated by the keyword "`op`", and `neg` is complement with respect to the set of all natural numbers. □

Definition 2. Given an equivalence $\sim$ on A, an operation σ in $\Sigma_{s_1 \ldots s_n, s}$ is *congruent for* $\sim$ iff $A_\sigma(a_1, ..., a_n) \sim A_\sigma(a'_1, ..., a'_n)$ whenever $a_i \sim a'_i$ for $i = 1, ..., n$. A *hidden Γ-congruence on A* is an equivalence on A which is the identity on visible sorts and is congruent for each operation in Γ. □

[1] But CafeOBJ prohibits behavioral operations with more than one hidden argument.

The following result from [21] is the basis for generalizing coinduction and other results to operations with multiple hidden arguments:

Theorem 3. *Given a hidden subsignature Γ of Σ and a hidden Σ-algebra A, there exists a largest hidden Γ-congruence on A, called* Γ-behavioral equivalence *and denoted* $\equiv^{\Gamma}_{\Sigma}$. □

Definition 4. A hidden Σ-algebra A *Γ-behaviorally satisfies* a conditional Σ-equation $e = (\forall X)\ t = t'$ **if** $t_1 = t'_1, ..., t_n = t'_n$ iff for each $\theta\colon X \to A$, if $\theta(t_i) \equiv^{\Gamma}_{\Sigma} \theta(t'_i)$ for $i = 1, ..., n$, then $\theta(t) \equiv^{\Gamma}_{\Sigma} \theta(t')$; in this case we write $A \models^{\Gamma}_{\Sigma} e$. If E is a set of Σ-equations, we write $A \models^{\Gamma}_{\Sigma} E$ if A Γ-behaviorally satisfies each equation in E. When Σ and Γ are clear from context, we may write $\equiv$ and $\models$ instead of $\equiv^{\Gamma}_{\Sigma}$ and $\models^{\Gamma}_{\Sigma}$ respectively. We say that A *behaviorally satisfies* (or *is a model of*) a behavioral specification $\mathcal{B} = (\Sigma, \Gamma, E)$ iff $A \models^{\Gamma}_{\Sigma} E$, and in this case we write $A \models \mathcal{B}$; also $\mathcal{B} \models e$ means $A \models \mathcal{B}$ implies $A \models^{\Gamma}_{\Sigma} e$. □

Example 2. *Nondeterministic Stack:* The following example after [12] motivates non-congruent operations. We specify a random number generator for a distributed system, as a process that puts generated numbers on a stack, where numbers are consumed with exactly one call by one process, since multiple access to a single number is wrong. We consider two stack states equivalent iff they have the same numbers in the same order; then `top` and `pop` are congruent for this equivalence, but `push` is not, since its behavior should not be determined by what is on the stack.

```
mod* NDSTACK { *[ Stack ]*  pr(NAT)
  bop top   : Stack -> Nat     ** attribute
  bop pop   : Stack -> Stack   ** method
  op empty : -> Stack          ** hidden constant
  op push  : Stack -> Stack    ** not congruent!
  var S : Stack
  beq pop(empty)   = empty .
  beq pop(push(S)) = S . }
```

An implementation might use a function $f\colon$ `Nat` $\to$ `Nat` where $f(n)$ is the nth randomly generated number. To ensure that n changes with each new call, we can keep it as a variable with the stack, incremented whenever a new number is pushed. Such an implementation is equivalent to the following model: Let $A_{\texttt{Nat}} = \omega$, where ω is the natural numbers, and let $A_{\texttt{Stack}} = \omega \times \omega^*$, where ω^* is lists of naturals. Using [head,tail] list notation with [] for the empty list, let $A_{\texttt{empty}} = (0, [])$, $A_{\texttt{top}}((n, [])) = 0$, $A_{\texttt{top}}((n, [h, t])) = h$, $A_{\texttt{pop}}((n, [])) = (n, [])$, $A_{\texttt{pop}}((n, [h, t])) = (n, t)$, and $A_{\texttt{push}}((n, l)) = (n + 1, [f(n), l])$. Then two states are behaviorally equivalent iff for every sequence of `pop`s followed by a `top` they give the same number, that is, they store the same elements in the same order; in other words, $(n, l) \equiv (n', l')$ iff $l = l'$. `push` is not behaviorally congruent for this model, because $f(n)$ can be different from $f(n')$. □

2.1 Coinduction

Example 3. We prove that union in Example 1 is commutative, i.e., that
$(\forall X, X')\ X\ \mathtt{U}\ X' = X'\ \mathtt{U}\ X$
is behaviorally satisfied by all models of `SET`. For A a model of `SET`, we use infix notation, $\_ \in \_$ instead of $A_{\mathtt{in}}$ and $\_ \cup \_$ instead of $A_{\_\mathtt{U}\_}$.

Let $\_R\_$ be a binary relation on A, called the *candidate relation*, defined by $a\ R\ a'$ if and only if $n \in a$ iff $n \in a'$ for all natural numbers n. We claim R is a hidden congruence. We show only that union is congruent for R, the other cases being similar. Suppose $a_1\ R\ a_1'$ and $a_2\ R\ a_2'$, i.e., $(n \in a_1$ iff $n \in a_1')$ and $(n \in a_2$ iff $n \in a_2')$ for all natural numbers n. Then $n \in a_1 \cup a_2$ iff $n \in a_1' \cup a_2'$, i.e., $a_1 \cup a_2\ R\ a_1' \cup a_2'$.

Since R is a hidden congruence, it is included in behavioral equivalence. We now show $(a \cup a')\ R\ (a' \cup a)$ for all $a, a' \in A$. This is equivalent to $n \in a \cup a'$ iff $n \in a' \cup a$, i.e., to $(n \in a$ or $n \in a')$ iff $(n \in a'$ or $n \in a)$, which is obvious. Thus we conclude that $a \cup a'$ is behaviorally equivalent to $a' \cup a$ for all $a, a' \in A$. Therefore $A \models^{\Sigma}_{\Sigma} (\forall X, X')\ X\ \mathtt{U}\ X' = X'\ \mathtt{U}\ X$, and since A was arbitrary, $\mathtt{SET} \models (\forall X, X')\ X\ \mathtt{U}\ X' = X'\ \mathtt{U}\ X$. Here is a CafeOBJ proof score for this reasoning (but see footnote 2):

```
mod* COINDUCTION { pr(SET)
  op _R_ : Set Set -> Bool
  op n : -> Nat
  vars X X' : Set
  eq X R X' = (n in X) == (n in X') . }
open COINDUCTION .
  ops a1 a1' a2 a2' : -> Set .
  eq n in a1 = n in a1' .          ** assume that a1 R a1'
  eq n in a2 = n in a2' .          ** assume that a2 R a2'
  red (a1 U a2) R (a1' U a2') .    **> should be true
  red (a1 & a2) R (a1' & a2') .    **> should be true
  red neg(a1) R neg(a1) .          **> should be true
  op m : -> Nat .
  red add(m, a1) R add(m, a1') .   **> should be true
  eq m = n .
  red add(m, a1) R add(m, a1') .   **> should be true
close
open COINDUCTION .
  ops a a' : -> Set .
  red (a U a') R (a' U a) .        **> should be true
close
```

□

3 Eliminating Behavioral Operations

The fewer operations in Γ, the easier it is to do coinduction, because fewer operations need be shown congruent for the candidate relation. This section follows [21], using the notion of behaviorally equivalent specifications and conditions for a specification to be behavioral equivalent to another with fewer behavioral

operations. The first definition of operations congruent over behavioral equivalence defined by a subset of operations seems to have been [1]; similar ideas also appear in [18, 19, 21], and in [6, 5] as well as in [15], which use the term *behavioral coherence*. We prefer the term "congruent" because the congruence rule of equational deduction is sound in hidden logic for an operation iff that operation is behaviorally congruent.

Definition 5. An operation σ is *Γ-behaviorally congruent* for A iff σ is congruent for $\equiv_\Sigma^\Gamma$ on A; we will often say just "congruent". An operation $\sigma \in \Sigma$ is *behaviorally congruent* for a specification $\mathcal{B}$ iff it is behaviorally congruent for every $A \models \mathcal{B}$. □

Proposition 6. *If $\mathcal{B} = (\Sigma, \Gamma, E)$ is a behavioral specification, then all operations in Γ and all hidden constants are behaviorally congruent for $\mathcal{B}$.* □

Corollary 7. Congruence Criterion: *Let $\mathcal{B} = (\Sigma, \Gamma, E)$ be a hidden specification and let $\sigma : v_1 ... v_m h_1 ... h_k \to h$ be an operation in Σ, where $v_1, ..., v_m$ are visible sorts and $h_1, ..., h_k, h$ are hidden sorts. If $W = \{y_1 : v_1, ..., y_m : v_m, x_1 : h_1, ..., x_k : h_k\}$ is a set of variables, then let $\sigma(W)$ denote the term $\sigma(y_1, ..., y_m, x_1, ..., x_k)$. If for each appropriate $\delta\colon\ s_1 ... s_n \to s$ in Γ and each $j = 1, ..., n$ such that $s_j = h$ there is some γ in $T_\Gamma(Z_j \cup W)$ such that the Σ-equation $(\forall Z_j, W)\ \delta(Z_j, \sigma(W)) = \gamma$ is in E (modulo renaming of variables), then σ is behaviorally congruent for $\mathcal{B}$.* □

Example 4 uses this criterion to show all the set operations congruent for the behavioral equivalence generated by `in`. The above is a special case of Theorem 16 of [21], which is further generalized in [4], although the result in [4] also follows from Theorem 16 of [21].

Definition 8. Hidden specifications $\mathcal{B}_1 = (\Sigma, \Gamma_1, E_1)$ and $\mathcal{B}_2 = (\Sigma, \Gamma_2, E_2)$ over the same hidden signature are *equivalent* iff for any hidden Σ-algebra A, $A \models \mathcal{B}_1$ iff $A \models \mathcal{B}_2$, and in this case $\equiv_\Sigma^{\Gamma_1} = \equiv_\Sigma^{\Gamma_2}$ on A. □

The rest of this section assumes $\mathcal{B}_1 = (\Sigma, \Gamma_1, E)$ and $\mathcal{B}_2 = (\Sigma, \Gamma_2, E)$ are two hidden specifications over the same signature with the same equations and with $\Gamma_1 \subseteq \Gamma_2$; we also assume that the Σ-equations in E have no conditions of hidden sort. The result below gives a method for eliminating behavioral operations from a specification. If a behavioral operation can be shown congruent for the behavioral specification that takes that operation as non-behavioral, then the two specs are equivalent.

Theorem 9. *$\mathcal{B}_1$ and $\mathcal{B}_2$ are equivalent iff all operations in Γ_2 are behaviorally congruent for $\mathcal{B}_1$.* □

Example 4. We modify the specification `SET` so that its only behavioral operation is `in`, and call the result `SETH`. By the congruence criterion (Corollary 7), `add`, `_U_`, `_&_` and `neg` are all behaviorally congruent for `SETH`. Thus by Theorem 9, `SET` and `SETH` are equivalent. This greatly simplifies the coinductive proof in Example 3:

```
mod* COINDUCTION { pr(SETH)
  op _R_ : Set Set -> Bool
  op n : -> Nat
  vars X X' : Set
  eq X R X' = (n in X) == (n in X') . }
open COINDUCTION .
  ops a a' : -> Set .
  red (a U a') R (a' U a) .  **> should be true
close
```

□

Example 5. Lists of Semaphores: The use of semaphores for scheduling and protecting resources is well known. A flag is associated with each non-preemptive resource. When the resource is allocated to a process, its semaphore is put up, and access is prohibited. When the process releases the resource, its semaphore is put down and it can be allocated to another process. Many modern processors support semaphores to speed up operating systems, and often include a function to reverse a flag. Here is a CafeOBJ specification:

```
mod* FLAG {  *[ Flag ]*  pr(QID)
  bop resource_ : Flag -> Id
  bop up?_ : Flag -> Bool
  ops  up down rev : Flag -> Flag
  var F : Flag
  eq up? up(F)   = true .      eq resource up(F)   = resource F .
  eq up? down(F) = false .     eq resource down(F) = resource F .
  eq up? rev(F)  = not up? F . eq resource rev(F)  = resource F . }
```

It is intended that all operations are behavioral, but by the congruence criterion (Corollary 7) and Theorem 9, the spec with only `up?` and `resource` declared behavioral is equivalent, because the others are obviously congruent.

When many resources of the same type are available (e.g., printers), their flags are kept in a list (an array is undesirable since the number of resources varies dynamically) from which the scheduler chooses the first unallocated resource when a request is received. We want all operations behavioral, that is, to preserve the intended behavior of flags and lists of flags. Here is a CafeOBJ spec (but see footnote 2, noting that `cons` has two hidden arguments):

```
mod* FLAG-LIST {  *[ List ]*  pr(FLAG)
  bop car_ : List -> Flag
  bop cdr_ : List -> List
  bop cons : Flag List -> List
  var F : Flag    var L : List
  beq car cons(F, L) = F .
  beq cdr cons(F, L) = L . }
```

The behavioral equations here allow more flexible implementation. For example, an operating system can allocate at its discretion software or hardware implementations for flags, so that `car cons(F, L)` is only behaviorally equivalent to `F`. The congruence criterion can again be applied with Theorem 9 to show that `FLAG-LIST` is equivalent to the spec where `cons` is not behavioral. (We have left

some details unspecified, such as car and cdr of the empty list, to make the spec easier to understand.)

Now consider a new spec where lists of flags can be put up and down. This is useful for operating systems to put resources in a safe state for system shutdown, or when hardware or software anomalies are detected.

```
mod* FLAG-LIST'{ *[ Flag < List ]*  pr(FLAG)
  bop car_ : List -> Flag
  bop cdr_ : List -> List
  op cons : Flag List -> List
  var F : Flag    var L : List
  beq car cons(F, L) = F .   beq cdr cons(F, L) = L .
  ops up down : List -> List
  beq car up(L)   =   up(car L) .   beq cdr up(L)   = up(cdr L) .
  beq car down(L) = down(car L) .   beq cdr down(L) = down(cdr L) . }
```

The congruence criterion and Theorem 9 again justify having only `car` and `cdr` behavioral. Now we use coinduction to prove that `up(cons(F, L))` is behaviorally equivalent to `cons(up(F), up(L))` for all flags `F` and lists of flags `L`:

```
mod* COINDUCTION { pr(FLAG-LIST')
  op _R_ : Flag Flag -> Bool
  op _R_ : List List -> Bool
  vars F F' : Flag   vars L L' : List
  eq F R F  = true .  eq L R L = true .
  eq F R F' = (up? F == up? F') and (resource F == resource F') .
  eq L R L' = ((car L) R (car L')) and ((cdr L) R (cdr L')) . }
```

Notice that we didn't completely define the candidate relation, but rather gave axioms it should satisfy, saying R is a hidden congruence (but without symmetry and transitivity, since we don't need these properties); we know such relations exist, because behavioral equivalence is one. This code is a bit dangerous, because of its (co)recurrent definition of the candidate relation, which can lead to nonterminating rewriting; but it works in this case, because the equation `eq L R L = true` is applied before the last equation. We now demonstrate two interesting properties:

```
open COINDUCTION .
  op f : -> Flag .  op l : -> List .
  red up(cons(f, l))   R cons(up(f), up(l)) .      **> should be true
  red down(cons(f, l)) R cons(down(f), down(l)) .  **> should be true
close
```

CafeOBJ does 29 rewrites and 114 matches for each reduction. □

4 Behavioral Abstraction Is Information Hiding

This section shows that any behavioral specification B over a hidden signature Σ can be translated to an ordinary algebraic specification $\tilde{B}$ over a signature $\tilde{\Sigma}$ containing Σ, such that a hidden Σ-algebra behaviorally satisfies B iff it strictly satisfies $\Sigma\Box\tilde{B}$ (which is the set of all Σ-theorems of $\tilde{B}$, see [7] for more detail). The specification $\tilde{B}$ can be generated automatically from B. This result allows using an equational logic theorem prover (such as OBJ3) for behavioral equations. Constructions in Definitions 12, 13 and 17 were inspired by work in [2, 3, 17].

Definition 10. For each hidden sort h, let $\star_h$ be a special variable of sort h different from every other variable appearing in this paper. Given a hidden Σ-algebra A and an element a_h of sort h of A, let $\tilde{a}_h : T_\Gamma(A \cup \{\star_h\}) \to A$ denote the unique extension of the function from $A \cup \{\star_h\}$ to A which is the identity on A and takes $\star_h$ to a_h. If t, t' are terms in $T_\Gamma(A \cup \{\star_h\})$ and $T_\Sigma(A \cup X)$ respectively, let $t[t']$ denote the term in $T_\Sigma(A \cup X)$ obtained by substituting $\star_h$ for t' in t. Let $LT_\Gamma(A \cup \{\star_h\})$ be the $(V \cup H)$-sorted subset of terms in $T_\Gamma(A \cup \{\star_h\})$ over the behavioral operations in Γ having exactly one occurrence of $\star_h$, whose proper subterms are either elements of A or else hidden terms in $LT_\Gamma(A \cup \{\star_h\})$. □

In other words, there are only hidden valued operations on the path from $\star_h$ to the root of any term in $LT_\Gamma(A \cup \{\star_h\})$, except that the operation at the top may be visible, and all other proper subterms which do not contain $\star_h$ are elements of A. The following can be seen as an alternative proof of Theorem 3:

Proposition 11. *Given a hidden Σ-algebra A and $a, a' \in A_h$ then $a \equiv^\Gamma_{\Sigma,h} a'$ iff $\tilde{a}(c) = \tilde{a}'(c)$ for each $v \in V$ and each $c \in LT_{\Gamma,v}(A \cup \{\star_h\})$.*

Proof. We show that the relation $\sim$ defined by $a \sim_h a'$ iff $\tilde{a}(c) = \tilde{a}'(c)$ for each $v \in V$ and each $c \in LT_{\Gamma,v}(A \cup \{\star_h\})$ is the largest hidden Γ-congruence.

Let $\sigma : h_1 ... h_k v_{k+1} ... v_n \to s$ be any operation in Γ (with its first k arguments hidden), let $a_i, a'_i \in A_{h_i}$ such that $a_i \sim_{h_i} a'_i$ for $i = 1, ..., k$, let $d_i \in A_{v_i} (= D_{v_i})$ for $i = k+1, ..., n$, and let $v \in V$ and $c \in LT_{\Gamma,v}(A \cup \{\star_s\})$ (if the sort s is visible, delete all occurrences of c in the proof that follows and replace terms of the form $c[t]$ by just t). Let c_i be the term $c[\sigma(a'_1, ..., a'_{i-1}, \star_{h_i}, a_{i+1}, ..., a_k, d_{k+1}, ..., d_n)]$ for each $i = 1, ..., k$. Because $a_i \sim_{h_i} a'_i$ one gets $\tilde{a}_i(c_i) = \tilde{a}'_i(c_i)$. Letting a and a' denote the elements $A_\sigma(a_1, ..., a_k, d_{k+1}, ..., d_n)$ and $A_\sigma(a'_1, ..., a'_k, d_{k+1}, ..., d_n)$ respectively, notice that $\tilde{a}(c) = \tilde{a}_1(c_1)$, $\tilde{a}'_i(c_i) = \tilde{a}_{i+1}(c_{i+1})$ for $i = 1, ..., k-1$, and $\tilde{a}'_k = \tilde{a}'(c)$. Therefore $\tilde{a}(c) = \tilde{a}'(c)$, and since c is arbitrary, we obtain $a \sim_s a'$, i.e., $\sim$ is preserved by σ, and so $\sim$ is a hidden Γ-congruence.

Because all operations in Γ preserve hidden Γ-congruences, so do the terms in $LT_\Gamma(A \cup \{\star_h\})$. In particular, terms in $LT_{\Gamma,v}(A \cup \{\star_h\})$ take congruent elements to identities. Therefore any hidden Γ-congruence is included in $\sim$. □

Definition 12. Given a hidden signature (Ψ, D, Σ) (where $S = V \cup H$), let $(\tilde{S}, \tilde{\Sigma})$ be the ordinary signature with $\tilde{S} = S \cup (H \to S)$, where $(H \to S) = \{(h \to s) \mid s \in S, h \in H\}$ is a set of new sorts, and where $\tilde{\Sigma}$ adds to Σ:

- a new operation $\diamond_h : \to (h \to h)$ for each $h \in H$,
- a new operation $\sigma^k_h : s_1 \ ... \ s_{k-1} \ (h \to h_k) \ s_{k+1} \ ... \ s_n \to (h \to s)$ for each behavioral operation $\sigma : s_1 \ ... \ s_{k-1} \ h_k \ s_{k+1} \ ... \ s_n \to s$ in Γ, for each $k = 1, ..., n$ such that $h_k \in H$ and each $h \in H$, and
- a new operation $\_[\_] : (h \to s) \ h \to s$ for each $h \in H$ and $s \in S$.

□

Definition 13. Given hidden Σ-algebra A, define an ordinary $\tilde{\Sigma}$-algebra $\tilde{A}$ by:

1. $\tilde{A}|_\Sigma = A$, so $\tilde{A}$ extends A,
2. $\tilde{A}_{(h \to s)} = LT_{\Gamma,s}(A \cup \{\star_h\})$,

3. $\tilde{A}_{\diamond_h} = \star_h$,
4. $\tilde{A}_{\sigma_h^k} \colon A_{s_1} \times \cdots A_{s_{k-1}} \times \tilde{A}_{(h \to h_k)} \times A_{s_{k+1}} \times \cdots \times A_{s_n} \to \tilde{A}_{(h \to s)}$ for each behavioral operation $\sigma : s_1 \ldots s_{k-1}\ h_k\ s_{k+1} \ldots s_n \to s$ in Γ and $h \in H$, by $\tilde{A}_{\sigma_h^k}(a_1, ..., a_{k-1}, t, a_{k+1}, ..., a_n) = \sigma(a_1, ..., a_{k-1}, t, a_{k+1}, ..., a_n)$ for $a_i \in A_{s_i}$ for $i \in \{1, ..., k-1, k+1, ..., n\}$, $t \in LT_{\Gamma,h_k}(A \cup \{\star_h\})$, and
5. $\tilde{A}_{\_[\_]} \colon \tilde{A}_{(h \to s)} \times A_h \to A_s$ for $s \in S$ and $h \in H$, by $\tilde{A}_{\_[\_]}(t, a_h) = \tilde{a}_h(t)$.

□

Proposition 14. *Given a hidden Σ-algebra A, then*

1. $\tilde{A} \models_{\tilde{\Sigma}} (\forall x : h)\ \diamond_h[x] = x$ *for each* $h \in H$, *and*
2. $\tilde{A} \models_{\tilde{\Sigma}} (\forall Y_k, z : (h \to h_k), x : h)\ \sigma_h^k(Y_k, z)[x] = \sigma(Y_k, z[x])$, *where* Y_k *is the set of variables* $\{y_1 : s_1, ..., y_{k-1} : s_{k-1}, y_{k+1} : s_{k+1}, ..., y_n : s_n\}$, $\sigma_h^k(Y_k, z)$ *is a shorthand for the term* $\sigma_h^k(y_1, ..., y_{k-1}, z, y_{k+1}, ..., y_n)$ *and* $\sigma(Y_k, z[x])$ *for the term* $\sigma(y_1, ..., y_{k-1}, z[x], y_{k+1}, ..., y_n)$, *for all behavioral operations* $\sigma : s_1 \ldots s_{k-1}\ h_k\ s_{k+1} \ldots s_n \to s$ *in* Γ *and all* $h \in H$.

Proof. 1. Let $\theta \colon \{x\} \to \tilde{A}$ *be any assignment and let* a_h *be* $\theta(x)$. *Then*

$$\tilde{\theta}(\diamond_h[x]) = \tilde{A}_{\_[\_]}(\tilde{A}_{\diamond_h}, a_h) = \tilde{a}_h(\star_h) = a_h = \tilde{\theta}(x)\ ,$$

where $\tilde{\theta} \colon T_{\tilde{\Sigma}}(\{x\}) \to \tilde{A}$ *is the unique* $\tilde{\Sigma}$*-algebra morphism extending* θ.

2. Let $\theta \colon Y_k \cup \{s : (h \to h_k), x : h\} \to \tilde{A}$ *be any assignment and let* $a_i = \theta(y_i)$ *for all* $i \in \{1, ..., k-1, k+1, ..., n\}$, $t = \theta(z)$, *and* $a_h = \theta(x)$. *Then*

$$\begin{aligned}
\tilde{\theta}(\sigma_h^k(Y_k, z)[x]) &= \tilde{A}_{_[_]}(\tilde{A}_{\sigma_h^k}(a_1, ..., a_{k-1}, t, a_{k+1}, ..., a_n), a_h) \\
&= \tilde{a}_h(\sigma(a_1, ..., a_{k-1}, t, a_{k+1}, ..., a_n)) \\
&= A_\sigma(a_1, ..., a_{k-1}, \tilde{a}_h(t), a_{k+1}, ..., a_n) \\
&= \tilde{A}_\sigma(a_1, ..., a_{k-1}, \tilde{A}_{_[_]}(t, a_h), a_{k+1}, ..., a_n) \\
&= \tilde{\theta}(\sigma(Y_k, z[x]))\ .
\end{aligned}$$

□

The rest of this section assumes equations have no conditions of hidden sort.

Definition 15. For each Σ-equation $e = (\forall X)\ t = t'$ **if** $t_1 = t'_1, .., t_n = t'_n$, let $\tilde{e}$ be the set of $\tilde{\Sigma}$-equations where $\tilde{e}$ is either the set containing only e regarded as a $\tilde{\Sigma}$-equation if the sort of t and t' is visible, or the set

$$\{(\forall X, z : (h \to v))\ z[t] = z[t']\ \textbf{if}\ t_1 = t'_1, .., t_n = t'_n\ \mid\ v \in V\}$$

if the sort h of t and t' is hidden. □

Proposition 16. *Given a hidden Σ-algebra A and Σ-equation e, then* $\tilde{A} \models_{\tilde{\Sigma}} \tilde{e}$ *iff* $A \models_\Sigma^\Gamma e$.

Proof. Let e be the Σ-equation $(\forall X)\ t = t'$ **if** $t_1 = t'_1, .., t_n = t'_n$. *If the sort of* t, t' *is visible then the result is easy, so we assume the sort h of* t, t' *is hidden.*

Suppose $\tilde{A} \models_{\tilde{\Sigma}} \tilde{e}$ *and let* $\theta \colon X \to A$ *be any assignment such that* $\theta(t_i) = \theta(t'_i)$ *for* $i = 1, ..., n$. *Let* $v \in V$ *and* $c \in LT_{\Gamma,v}(A \cup \{\star_h\})$. *Define* $\varphi \colon X \cup \{z : (h \to v)\} \to \tilde{A}$ *to be θ on X, with* $\varphi(z) = c$. *Then* $\tilde{A} \models_{\tilde{\Sigma}} \tilde{e}$ *implies* $\tilde{\varphi}(z[t]) = \tilde{\varphi}(z[t'])$, *where*

$\tilde{\varphi}\colon T_{\tilde{\Sigma}}(X \cup \{z : (h \to v)\}) \to \tilde{A}$ *is the unique extension of* φ *to a* $\tilde{\Sigma}$*-homomorphism. But* $\tilde{\varphi}(z[t]) = \tilde{A}_{\_[\_]}(\varphi(z), \theta(t)) = \theta\tilde{(t)}(c)$ *and similarly* $\tilde{\varphi}(z[t']) = \theta\tilde{(t')}(c)$*, so by Proposition 11,* $\theta(t) \equiv^{\Gamma}_{\Sigma,h} \theta(t')$*. Thus* $A \models_{\Sigma} e$*.*

Conversely, suppose $A \models_{\Sigma} e$ *and let* $v \in V$ *and* $\varphi\colon X \cup \{z : (h \to v)\} \to \tilde{A}$ *such that* $\varphi(t_i) = \varphi(t'_i)$ *for* $i = 1, ..., n$*. Then* $A \models_{\Sigma} e$ *implies* $\varphi(t) \equiv^{\Gamma}_{\Sigma,h} \varphi(t')$*, so by Proposition 11,* $\varphi\tilde{(t)}(\varphi(z)) = \varphi\tilde{(t')}(\varphi(z))$*. But* $\varphi\tilde{(t)}(\varphi(z)) = \tilde{\varphi}(z[t])$ *and* $\varphi\tilde{(t')}(\varphi(z)) = \tilde{\varphi}(z[t'])$*, so* $\tilde{\varphi}(z[t]) = \tilde{\varphi}(z[t'])$*. Therefore* $\tilde{A} \models_{\tilde{\Sigma}} \tilde{e}$*.* □

Definition 17. Given $B = (\Gamma, \Sigma, E)$, let $\tilde{B} = (\tilde{\Sigma}, \tilde{E})$ be the ordinary specification with $\tilde{E}$ adding to $\bigcup_{e \in E} \tilde{e}$ the equations, for each $h \in H$

$(\forall x : h)\ \diamond_h[x] = x$,

$(\forall Y_k, z : (h \to h_k), x : h)\ \sigma^k_h(Y_k, z)[x] = \sigma(Y_k, z[x])$,

for all behavioral operations $\sigma : s_1 \ ... \ s_{k-1} \ h_k \ s_{k+1} \ ... \ s_n \to s$ in Γ. (See the notation of Proposition 14). □

Notice that $\tilde{B}$ is finite whenever B is finite, and that if B has no conditional equations then neither does $\tilde{B}$.

Example 6. If B is the specification `SETH` of Example 4, then $\tilde{B}$ is:

```
mod! SET! { [ Set ]  pr(NAT)
  op _in_  : Nat Set -> Bool
  op empty : -> Set
  op add   : Nat Set -> Set
  op _U_   : Set Set -> Set
  op _&_   : Set Set -> Set
  op neg   : Set -> Set
  vars N N' : Nat  vars X X' : Set
  eq N in empty = false .
  eq N in add(N',X) = (N == N') or (N in X) .
  eq N in (X U X')  = (N in X)  or (N in X') .
  eq N in (X & X')  = (N in X) and (N in X') .
  eq N in neg(X)    = not (N in X) . }
mod! SET~ { [ Set->Set Set->Bool ]  pr(SET!)
  op <>   : -> Set->Set
  op _IN_ : Nat Set->Set  -> Set->Bool
  op _[_] : Set->Set Set  -> Set
  op _[_] : Set->Bool Set -> Bool
  var Z : Set->Set    var X : Set    var N : Nat
  eq       <> [ X ] = X .
  eq (N IN Z) [ X ] = N in Z [ X ] . }
```

Here `SET!` is just `SETH` with behavioral features removed, extended with sorts `Set->Set` and `Set->Bool` (we don't add the sort `Set->Nat` because there is no behavioral operation of sort `Nat` in `SETH`), a constant `<>` of sort `Set->Set` which stands for $\diamond_{\texttt{Set}}\colon \to$ (`Set->Set`), an operation `_IN_` which stands for $(\_in\_)^2_{\texttt{Set}}$: `Nat` (`Set->Set`) $\to$ (`Set->Bool`), two operations `_[_]` defined from `Set->Set` and `Set` to `Set` and from `Set->Bool` and `Set` to `Bool` respectively, and the two equations required by Definition 17. □

Corollary 18. *For any hidden Σ-algebra A, $\tilde{A} \models \tilde{B}$ iff $A \models\!\!\models B$.*

Proof. From Propositions 14 and 16. □

Example 7. Proposition 16 and Corollary 18 can help prove behavioral properties equationally, such as commutativity of union in the spec `SETH` of Example 4. We claim it suffices to show that `SET~` satisfies

($\star$) $\quad (\forall X, X' : \texttt{Set}, Z : (\texttt{Set} -> \texttt{Bool}))\ Z[X\ \texttt{U}\ X'] = Z[X'\ \texttt{U}\ X]$.

Indeed, if A behaviorally satisfies `SETH`, then Corollary 18 implies $\tilde{A}$ satisfies ($\star$), so by Proposition 16, A behaviorally satisfies $(\forall X, X' : \texttt{Set})\ X\ \texttt{U}\ X' = X'\ \texttt{U}\ X$.

We prove that $(\forall X, X' : \texttt{Set}, Z : \texttt{Set} -> \texttt{Bool})\ Z[X\ \texttt{U}\ X'] = Z[X'\ \texttt{U}\ X]$ is an equational consequence of `SET~`. First open `SET~` and introduce two constants of sort `Set` and another of sort `Set->Bool`:

```
open SET~ .
  ops x x' : -> Set .  op  z : -> Set->Bool .  op  p : -> Bool .
  eq p = (z [ x U x' ] == z [ x' U x ]) .
```

Our goal is to prove that `p` reduces to `true`. Since `_IN_` is the only operation of sort `Set->Bool`, the only way for `z` as above to exist is for it to be a term of the form `n IN s`, where `n` is a natural number and `s` is of sort `Set->Set`:

```
  op n : -> Nat .  op s : -> Set->Set .
  eq z = n IN s .
```

Because the only operation of sort `Set->Set` is `<>`, we can reduce `p` as follows:

```
  eq s = <> .
  red p .    **> should be true
```

CafeOBJ does 12 rewrites and 64 matches. This proof was simple because there were no behavioral operations of hidden sort, but in general such proofs would need induction on the structure of terms of sorts $(h \rightarrow h')$, and thus would be as awkward as are proofs by context induction [14]. □

Proposition 19. *If A is a Σ-algebra (not necessary hidden) such that $A \models_\Sigma (\forall X)\ t = t'$ **if** C, then:*

1. *$A \models_\Sigma (\forall X')\ t = t'$ **if** C for each $X \subseteq X'$;*
2. *$A \models_\Sigma (\forall X)\ t' = t$ **if** C;*
3. *$A \models_\Sigma (\forall X)\ t = t''$ **if** C whenever $A \models_\Sigma (\forall X)\ t' = t''$;*
4. *$A \models_\Sigma (\forall Y)\ \rho(t) = \rho(t')$ **if** $\rho(C)$ for any substitution $\rho\colon X \to T_\Sigma(Y)$, where $\rho(C)$ is the set $\{\rho(t_i) = \rho(t_i') \mid t_i = t_i' \in C\}$.*

□

Theorem 20. *For any hidden Σ-algebra A and any behavioral specification B, $A \models\!\!\models B$ iff $A \models \Sigma\Box\tilde{B}$.*

Proof. If $A \models\!\!\models B$ then Corollary 18 gives $\tilde{A} \models \tilde{B}$, so that $\tilde{A} \models \Sigma\Box\tilde{B}$, and thus $A \models \Sigma\Box\tilde{B}$.

*Suppose $A \models \Sigma\Box\tilde{B}$, let e be any Σ-equation $(\forall X)\ t = t'$ **if** C in B, and let $\theta\colon X \to A$ be any assignment such that $\theta(C)$. If the sort of t, t' is visible then $A \models_\Sigma e$, so $A \models\!\!\models^{\Gamma}_{\Sigma} e$. If the sort h of t, t' is hidden then let $v \in V$*

and $c \in LT_{\Gamma,v}(A \cup \{\star_h\})$. *Then* c *has the form* $\sigma_1(\alpha_1, \sigma_2(\alpha_2, ..., \sigma_m(\alpha_m, \star_h)...))$, *where* $\sigma_j(\alpha_j, t)$ *indicates* $\sigma_j(a_{1,j}, ..., a_{k-1,j}, t, a_{k+1,j}, ..., a_{n_j,j})$ *for some appropriate elements* $a_{1,j}, ..., a_{k-1,j}, a_{k+1,j}, ..., a_{n_j,j}$ *in* A, *such that the sort of* σ_1 *is* v *and the sorts of* $\sigma_2, ..., \sigma_m$ *are hidden. Let* $c_h \in T_{\tilde{\Sigma},(h \to v)}(A)$ *be the term* $(\sigma_1)_h^{k_1}(\alpha_1, \sigma_2)_h^{k_2}(\alpha_2, ..., (\sigma_m)_h^{k_m}(\alpha_m, \diamond_h)...))$. *Using the special equations in* $\tilde{B}$ *(see Definition 17) and Proposition 19, it can be shown that* $\tilde{B} \models (\forall X, A)\ c_h[t] = c[t]$ *and* $\tilde{B} \models (\forall X, A)\ c_h[t'] = c[t']$. *On the other hand, since the equation*

$(\forall X, z : (h \to v))\ z[t] = z[t']$ **if** C

is in $\tilde{E}$ *and* c_h *is a* $\tilde{\Sigma}$*-term, Proposition 19 gives* $\tilde{B} \models (\forall X, A)\ c_h[t] = c_h[t']$ **if** C. *Also Proposition 19 gives* $\tilde{B} \models (\forall X, A)\ c[t] = c[t']$ **if** C, *i.e.,* $(\forall X, A)\ c[t] = c[t']$ **if** C *belongs to* $\Sigma \Box \tilde{B}$. *Therefore* $A \models_\Sigma (\forall X, A)\ c[t] = c[t']$ **if** C. *Letting* $\varphi\colon X \cup A \to A$ *be* θ *on* X *and the identity on* A, *we get* $\theta(\tilde{t})(c) = \theta(\tilde{t'})(c)$. *Since* c *was arbitrary, Proposition 11 gives* $\theta(t) \equiv_\Sigma^\Gamma \theta(t')$. *Thus* $A \models_\Sigma^\Gamma e$, *so that* $A \models B$. □

5 Two Institutions for Hidden Algebra

We give two institutions [10] for the generalization of hidden algebra to multiple hidden arguments and fewer behavioral operations. The first follows the institution of basic hidden algebra [9] and the approach earlier in this paper, while the second seems more promising for future research. A similar adaptation (but without the citation) of the result in [9] to the observational logic framework appears in [15]; our approach also avoids the infinitary logic used in observational logic. We fix a data algebra D, and proceed as follows:

Signatures: The category **Sign** has hidden signatures over D as objects. A morphism of hidden signatures $\phi\colon (\Gamma_1, \Sigma_1) \to (\Gamma_2, \Sigma_2)$ is the identity on the visible signature Ψ, takes hidden sorts to hidden sorts, and if a behavioral operation δ_2 in Γ_2 has an argument sort in $\phi(H_1)$ then there is some behavioral operation δ_1 in Γ_1 such that $\delta_2 = \phi(\delta_1)$. **Sign** is indeed a category, and the composition of two hidden signature morphisms is another. Indeed, let $\psi\colon (\Gamma_2, \Sigma_2) \to (\Gamma_3, \Sigma_3)$ and let δ_3 be an operation in Γ_3 having an argument sort in $(\phi;\psi)(H_1)$. Then δ_3 has an argument sort in $\psi(H_2)$, so there is an operation δ_2 in Γ_2 with $\delta_3 = \psi(\delta_2)$. Also δ_2 has an argument sort in $\phi(H_1)$, so there is some δ_1 in Γ_1 with $\delta_2 = \phi(\delta_1)$. Therefore $\delta_3 = (\phi;\psi)(\delta_1)$, i.e., $\phi;\psi$ is a morphism of hidden signatures.

Sentences: Given a hidden signature (Γ, Σ), let $\mathbf{Sen}(\Gamma, \Sigma)$ be the set of all Σ-equations. If $\phi\colon (\Gamma_1, \Sigma_1) \to (\Gamma_2, \Sigma_2)$ is a hidden signature morphism, then $\mathbf{Sen}(\phi)$ is the function taking a Σ_1-equation $e = (\forall X)\ t = t'$ **if** $t_1 = t'_1, ..., t_n = t'_n$ to the Σ_2-equation

$\phi(e) = (\forall X')\ \phi(t) = \phi(t')$ **if** $\phi(t_1) = \phi(t'_1), ..., \phi(t_n) = \phi(t'_n)$,

where X' is $\{x : \phi(s) \mid x : s \in X\}$. Then $\mathbf{Sen}\colon$ **Sign** $\to$ **Set** is indeed a functor.

Models: Given a hidden signature (Γ, Σ), let $\mathbf{Mod}(\Gamma, \Sigma)$ be the category of hidden Σ-algebras and their morphisms. If $\phi\colon (\Gamma_1, \Sigma_1) \to (\Gamma_2, \Sigma_2)$ is a hidden signature morphism, then $\mathbf{Mod}(\phi)$ is the usual reduct functor, $\_|_\phi$. Unlike [1, 15], etc., this allows models where not all operations are congruent.

Satisfaction Relation: behavioral satisfaction, i.e., $\models_{(\Gamma,\Sigma)} = \models^{\Gamma}_{\Sigma}$.

Theorem 21. Satisfaction Condition: *Given* $\phi\colon (\Gamma_1,\Sigma_1) \to (\Gamma_2,\Sigma_2)$ *a hidden signature morphism,* $e = (\forall X)\ t = t'$ **if** $t_1 = t'_1, ..., t_n = t'_n$ *a* Σ_1*-equation, and* A *a hidden* Σ_2*-algebra, then* $A \models^{\Gamma_2}_{\Sigma_2} \phi(e)$ *iff* $A|_\phi \models^{\Gamma_1}_{\Sigma_1} e$.

Proof. There is a bijection between $(A|_\phi)^X$ *and* $A^{X'}$ *that takes* $\theta\colon X \to A|_\phi$ *to* $\theta'\colon X' \to A$ *defined by* $\theta'(x:\phi(s)) = \theta(x:s)$*, and takes* $\theta'\colon X' \to A$ *to* $\theta\colon X \to A|_\phi$ *defined by* $\theta(x:s) = \theta'(x:\phi(s))$*. Notice that for every term* t *in* $T_{\Sigma_1}(X)$*, we have* $\theta(t) = \theta'(\phi(t))$ *where* $\phi(t)$ *is the term* t *with each* $x:s$ *replaced by* $x:\phi(s)$ *and each operation* σ *replaced by* $\phi(\sigma)$*. It remains to prove that* $a \equiv^{\Gamma_1}_{\Sigma_1,h} a'$ *iff* $a \equiv^{\Gamma_2}_{\Sigma_2,\phi(h)} a'$ *for each* $a, a' \in A_{\phi(h)}$*, where* $\equiv^{\Gamma_1}_{\Sigma_1}$ *is behavioral equivalence on* $A|_\phi$ *and* $\equiv^{\Gamma_2}_{\Sigma_2}$ *is behavioral equivalence on* A*. Since* $\phi(c_1) \in LT_{\Gamma_2}(A|_\phi \cup \{\star_{\phi(h)}\})$ *whenever* $c_1 \in LT_{\Gamma_1}(A|_\phi \cup \{\star_h\})$*, one gets* $a \equiv^{\Gamma_2}_{\Sigma_2,\phi(h)} a'$ *implies* $a \equiv^{\Gamma_1}_{\Sigma_1,h} a'$*. Now if* $c_2 \in LT_{\Gamma_2}(A \cup \{\star_{\phi(h)}\})$ *then because for every operation* δ_2 *in* Γ_2 *having an argument sort in* $\phi(H_1)$ *there is some* δ_1 *in* Γ_1 *with* $\delta_2 = \phi(\delta_1)$*, we iteratively get a term* $c_1 \in LT_{\Gamma_1}(A|_p hi \cup \{\star_h\})$ *such that* $c_2 = \phi(c_1)$*. Therefore* $a \equiv^{\Gamma_1}_{\Sigma_1,h} a'$ *implies* $a \equiv^{\Gamma_2}_{\Sigma_2,\phi(h)} a'$. □

Our second institution views the declaration of a behavioral operation as a new kind of sentence, rather than part of a hidden signature. The notion of model also changes, adding an equivalence relation as in [1]. This is natural for modern software engineering, since languages like Java provide classes with an operation denoted `equals` which serves this purpose. Sentences in [1] are pairs $\langle e, \Delta\rangle$, where Δ is a set of terms (pretty much like a cobasis over the derived signature), which are satisfied by $(A,\sim)$ iff $(A,\sim)$ satisfies e as in our case below (actually e is a first-order formula in their framework) and $\sim \subseteq \equiv_\Delta$. Fix a data algebra D, and proceed as follows:

Signatures: The category **Sign** has hidden signatures over D as objects. A morphism of hidden signatures $\phi\colon \Sigma_1 \to \Sigma_2$ is identity on the visible signature Ψ and takes hidden sorts to hidden sorts.

Sentences: Given a hidden signature Σ, let $\mathbf{Sen}(\Sigma)$ be the set of all Σ-equations unioned with Σ. If $\phi\colon \Sigma_1 \to \Sigma_2$ is a hidden signature morphism, then $\mathbf{Sen}(\phi)$ is the function taking a Σ_1-equation $e = (\forall X)\ t = t'$ **if** $t_1 = t'_1, ..., t_n = t'_n$ to the Σ_2-equation $\phi(e) = (\forall X')\ \phi(t) = \phi(t')$ **if** $\phi(t_1) = \phi(t'_1), ..., \phi(t_n) = \phi(t'_n)$, where X' is the set $\{x : \phi(s) \mid x : s \in X\}$, and taking $\sigma : s_1\ ...\ s_n \to s$ to $\phi(\sigma) : \phi(s_1)\ ...\ \phi(s_n) \to \phi(s)$. Then $\mathbf{Sen}\colon \mathbf{Sign} \to \mathbf{Set}$ is indeed a functor.

Models: Given a hidden signature Σ, let $\mathbf{Mod}(\Sigma)$ be the category of pairs $(A,\sim)$ where A is a hidden Σ-algebra and $\sim$ is an equivalence relation on A which is identity on visible sorts, with morphisms $f\colon (A,\sim) \to (A',\sim')$ with $f\colon A \to A'$ a Σ-homomorphism such that $f(\sim) \subseteq \sim'$. If $\phi\colon \Sigma_1 \to \Sigma_2$ is a hidden signature morphism, then $\mathbf{Mod}(\phi)$, often denoted $\_|_\phi$, is defined as $(A,\sim)|_\phi = (A|_\phi, \sim|_\phi)$ on objects, where $A|_\phi$ is the ordinary many-sorted algebra reduct and $(\sim|_\phi)_s = \sim_{\phi(s)}$ for all sorts s of Σ_1, and as $f|_\phi\colon (A,\sim)|_\phi \to (A',\sim')|_\phi$ on morphisms. Notice that indeed $f|_\phi(\sim|_\phi) \subseteq \sim'|_\phi$, so **Mod** is well defined.

Satisfaction Relation: A Σ-model $(A, \sim)$ satisfies a conditional Σ-equation $(\forall X)\ t = t'$ **if** $t_1 = t'_1, ..., t_n = t'_n$ iff for each $\theta\colon X \to A$, if $\theta(t_1) \sim \theta(t'_1)$, ..., $\theta(t_n) \sim \theta(t'_n)$ then $\theta(t) \sim \theta(t')$. Also $(A, \sim)$ satisfies a Σ-sentence $\gamma \in \Sigma$ iff γ is congruent for $\sim$.

Theorem 22. Satisfaction Condition: *Let $\phi\colon \Sigma_1 \to \Sigma_2$ be a morphism of hidden signatures, let e be a Σ_1-sentence and let $(A, \sim)$ be a model of Σ_2. Then $(A, \sim) \models_{\Sigma_2} \phi(e)$ iff $(A, \sim)|_\phi \models_{\Sigma_1} e$.*

*Proof. First suppose e is a Σ-equation $(\forall X)\ t = t'$ **if** $t_1 = t'_1, ..., t_n = t'_n$. Notice that there is a bijection between functions from X to $(A|_\phi)$ and functions from X' to A taking $\theta\colon X \to A|_\phi$ to $\theta'\colon X' \to A$ defined by $\theta'(x : \phi(s)) = \theta(x : s)$ and taking $\theta'\colon X' \to A$ to $\theta\colon X \to A|_\phi$ defined by $\theta(x : s) = \theta'(x : \phi(s))$. Because for every term t in $T_{\Sigma_1}(X)$ we have $\theta(t) = \theta'(\phi(t))$ where $\phi(t)$ is the term t with each $x : s$ replaced by $x : \phi(s)$ and each operation σ replaced by $\phi(\sigma)$, the result is immediate.*

Second, suppose e is an operation $\gamma \in \Sigma$. Then $(A, \sim)$ satisfies $\phi(\gamma)$ iff $\phi(\gamma)$ is congruent for $\sim$, which is equivalent to γ being congruent for $\sim|_\phi$. □

This institution justifies our belief that asserting an operation behavioral is a kind of sentence, not a kind of syntactic declaration as in the "extended hidden signatures" of [5][2]. Coinduction now appears in the following elegant guise:

Proposition 23. *Given a hidden subsignature Γ of Σ, a set of Σ-equations E and a hidden Σ-algebra A, then*

- $(A, \sim) \models_\Sigma E, \Gamma$ *implies* $(A, \equiv^\Gamma_\Sigma) \models_\Sigma E, \Gamma$.
- $(A, \equiv^\Gamma_\Sigma) \models_\Sigma \Gamma$.
- $A \models^\Gamma_\Sigma E$ *iff* $(A, \equiv^\Gamma_\Sigma) \models_\Sigma E$ *iff* $(A, \equiv^\Gamma_\Sigma) \models_\Sigma E, \Gamma$.

□

There is a natural relationship between our two institutions:

- since congruent operations are declared with sentences, any signature in the first institution translates to a specification in the second;
- any model A of (Σ, Γ) in the first institution gives a model of the second, namely $(A, \equiv^\Gamma_\Sigma)$;
- any (Σ, Γ)-sentence is a Σ-sentence;

and we can see that for any (Σ, Γ)-sentence e and any hidden Σ-algebra A, we get $A \models^\Gamma_\Sigma e$ iff $(A, \equiv^\Gamma_\Sigma) \models_\Sigma e$. This relationship suggests a new (as far as we know) kind of relationship between institutions (here $\mathbf{Th}(\mathcal{I})$ denotes the category of theories over $\mathcal{I}$, see [10]):

Definition 24. Given two institutions $\mathcal{I} = (\mathbf{Sign}, \mathbf{Mod}, \mathbf{Sen}, \models)$ and $\mathcal{I}' = (\mathbf{Sign}', \mathbf{Mod}', \mathbf{Sen}', \models')$, then an *institution theoroidal forward morphism*[3], *from $\mathcal{I}$ to $\mathcal{I}'$* is (Φ, β, α) where:

[2] However, the most recent version of [8] treats coherence assertions as sentences.

[3] This terminology is a preliminary attempt to bring some order to the chaos of relationships among institutions, by using names that suggest the nature of the relationship involved.

- $\Phi\colon \mathbf{Sign} \to \mathbf{Th}(\mathcal{I}')$ is a map such that $\Phi;\mathcal{U}'\colon \mathbf{Sign} \to \mathbf{Sign}'$ is a functor, where $\mathcal{U}'\colon \mathbf{Th}(\mathcal{I}') \to \mathbf{Sign}'$ is the forgetful functor; we ambiguously let Φ also denote the functor $\Phi;\mathcal{U}'$,
- $\beta\colon \mathbf{Mod} \Rightarrow \Phi;\mathbf{Mod}'$ is a natural transformation, and
- $\alpha\colon \mathbf{Sen} \Rightarrow \Phi;\mathbf{Sen}'$ is a natural transformation,

such that for any signature $\Sigma \in \mathbf{Sign}$, any sentence $e \in \mathbf{Sen}(\mathbf{Sign})$ and any model $m \in \mathbf{Mod}(\mathbf{Sign})$, the *satisfaction condition*, $m \models_\Sigma e$ iff $\beta(m) \models_{\Phi(\Sigma)} \alpha(e)$, holds. □

Proposition 25. *There is an institution theoroidal forward morphism from the first to the second institution defined above.* □

We thank the anonymous referees for their comments, which have helped us to piece together aspects of the relationship of our work with that of other groups, and to conclude that a convergence of viewpoints may be occurring within the broad area that might be called behavioral algebra.

References

[1] Gilles Bernot, Michael Bidoit, and Teodor Knapik. Observational specifications and the indistinguishability assumption. *Theoretical Computer Science*, 139(1-2):275–314, 1995. Submitted 1992.

[2] Michael Bidoit and Rolf Hennicker. Behavioral theories and the proof of behavioral properties. *Theoretical Computer Science*, 165(1):3–55, 1996.

[3] Michael Bidoit and Rolf Hennicker. Modular correctness proofs of behavioural implementations. *Acta Informatica*, 35(11):951–1005, 1998.

[4] Michael Bidoit and Rolf Hennicker. Observer complete definitions are behaviourally coherent. Technical Report LSV-99-4, ENS de Cachan, 1999.

[5] Răzvan Diaconescu. Behavioral coherence in object-oriented algebraic specification. Technical Report IS–RR–98–0017F, Japan Advanced Institute for Science and Technology, June 1998. Submitted for publication.

[6] Răzvan Diaconescu and Kokichi Futatsugi. *CafeOBJ Report: The Language, Proof Techniques, and Methodologies for Object-Oriented Algebraic Specification*. World Scientific, 1998. AMAST Series in Computing, volume 6.

[7] Răzvan Diaconescu, Joseph Goguen, and Petros Stefaneas. Logical support for modularization. In Gerard Huet and Gordon Plotkin, editors, *Logical Environments*, pages 83–130. Cambridge, 1993.

[8] Răzvan Diaconescu and Kokichi Futatsugi. Logical foundations of CafeOBJ. Submitted for publication.

[9] Joseph Goguen. Types as theories. In George Michael Reed, Andrew William Roscoe, and Ralph F. Wachter, editors, *Topology and Category Theory in Computer Science*, pages 357–390. Oxford, 1991. Proceedings of a Conference held at Oxford, June 1989.

[10] Joseph Goguen and Rod Burstall. Institutions: Abstract model theory for specification and programming. *Journal of the Association for Computing Machinery*, 39(1):95–146, January 1992.

[11] Joseph Goguen and Grant Malcolm. *Algebraic Semantics of Imperative Programs*. MIT, 1996.

[12] Joseph Goguen and Grant Malcolm. A hidden agenda. *Theoretical Computer Science*, to appear 1999. Also UCSD Dept. Computer Science & Eng. Technical Report CS97–538, May 1997.

[13] Joseph Goguen, James Thatcher, and Eric Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In Raymond Yeh, editor, *Current Trends in Programming Methodology, IV*, pages 80–149. Prentice-Hall, 1978.

[14] Rolf Hennicker. Context induction: a proof principle for behavioral abstractions. *Formal Aspects of Computing*, 3(4):326–345, 1991.

[15] Rolf Hennicker and Michel Bidoit. Observational logic. In *Algebraic Methodology and Software Technology (AMAST'98)*, volume 1548 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 1999.

[16] Bart Jacobs and Jan Rutten. A tutorial on (co)algebras and (co)induction. *Bulletin of the European Association for Theoretical Computer Science*, 62:222–259, 1997.

[17] Seikô Mikami. Semantics of equational specifications with module import and verification method of behavioral equations. In *Proceedings, CafeOBJ Symposium.* Japan Advanced Institute for Science and Technology, 1998. Numazu, Japan, April 1998.

[18] Peter Padawitz. Swinging data types: Syntax, semantics, and theory. In *Proceedings, WADT'95*, volume 1130 of *Lecture Notes in Computer Science*, pages 409–435. Springer, 1996.

[19] Peter Padawitz. Towards the one-tiered design of data types and transition systems. In *Proceedings, WADT'97*, volume 1376 of *Lecture Notes in Computer Science*, pages 365–380. Springer, 1998.

[20] Horst Reichel. Behavioural equivalence – a unifying concept for initial and final specifications. In *Proceedings, Third Hungarian Computer Science Conference.* Akademiai Kiado, 1981. Budapest.

[21] Grigore Roşu and Joseph Goguen. Hidden congruent deduction. In Ricardo Caferra and Gernot Salzer, editors, *Proceedings, First-Order Theorem Proving - FTP'98*, pages 213–223. Technische Universitat Wien, 1998. Full version to appear in Lecture Notes in Artificial Intelligence, 1999.